Network Security Via Explicit Consent

Jad Naous*, Michael Walfish†, David Mazières*, Antonio Nicolosi‡, and Arun Seehra†
*Stanford †UT Austin ‡Stevens Institute of Technology

Abstract

Securing real-world operating systems is hard; dropping packets headed to those systems is easy. Thus, network-layer defenses have become indispensable to end-host security. Unfortunately, most defenses inflict collateral damage, require hardware modification, or necessitate coordination between organizations' administrators. Yet, for all that, each defense addresses only a subset of attacks. This paper describes ICING, a network layer that allows all stakeholders (senders, receivers, and providers) to deploy new network defenses unilaterally, with enough precision to avoid collateral damage, and without further hardware modification. ICING captures many prior network-layer defenses within a coherent framework: for a packet to flow from sender to receiver, every entity along the path must have consented to the *entire path*. To enforce this property, ICING's data plane must address a key challenge: how mutually distrustful realms that cannot rely on per-packet or per-flow public key cryptography ensure that packets follow their purported paths. We demonstrate ICING's technical feasibility with a prototype that forwards at over 2 Gbit/s.

1 Introduction

Although the Internet architecture is commonly regarded as a disaster for end-host security, it does offer some security benefits. The assignment of functions to layers allows people to sidestep intractable *systems* problems by relying on the *network* as a "choke point": rather than secure real-world operating systems, people filter packets headed to those systems.

But of course Internet security *is* a disaster. First, the architecture enables attacks (malware epidemics, denial-of-service, route hijacking, etc.). Second, it inhibits defenses: conceptually simple actions, like identifying and blocking a troublesome traffic transmitter, are research problems. More generally, network-layer defenses—ACLs, VPNs, NATs, firewalls, probabilistic signature matching, pleading with your ISP at midnight, pleading with your attacker's ISP at midnight, whitelisting, blacklisting, redirecting traffic to a DoS defender, securing BGP, network capabilities, proof-of-work, upgrading your router, etc.—inflict collateral damage, require hardware modifications, or necessitate coordination between organizations' administrators.

Worse, each defense addresses only a subset of attacks, so people incur the above costs repeatedly. Worse still, not every defense can coexist with every other. And worst of all, attacks continue to evolve. Is a better world possible? We note that despite the continual evolution of *attacks*, the network *stakeholders*—senders, providers, and receivers—remain constant.

This paper presents a new network layer, **ICING**. ICING's deployment requires hardware modification—but only once. In an ICING network, each stakeholder can deploy new network defenses unilaterally, with enough precision to avoid collateral damage, and without further hardware modification. Many proposed defenses that so far require different, overlapping mechanisms can, under ICING, coexist in a coherent framework.

ICING divides the network into mutually distrustful administrative *realms*. To communicate with a destination, the sender must identify an appropriate sequence of realms, which we call a *path*, between it and the destination. ICING's approach to "path finding" involves senders contacting path servers that are akin to today's DNS servers (an approach inspired by [43]), but ICING also works with any other approach to path finding.

Before sending a packet along a path, a sender must get explicit *consent* from each realm. (For now, we take the two endpoints to be their own realms, but we revisit that simplification in §3.4.) To get a realm's consent, a sender communicates with a general-purpose server that is physically separate from the realm's forwarding hardware (a decomposition inspired by [14, 15, 23, 37]). The sender proposes the path. In making its decision, the server can incorporate arbitrary factors besides the proposed path (billing relationships [36], authentication, etc.). Upon consent, the server issues a *proof-of-consent* (PoC) that authenticates the proposed path. PoCs generalize network capabilities [8, 36, 42, 44] and Visas [18] (see §8). On the data path, an honest forwarder drops packets with invalid PoCs; it also drops packets that have deviated from their authorized paths.

In practice, many "control plane" steps can be combined; for example, the sender may, in one step, obtain a path and PoCs for a large subsequence of that path.

ICING upholds a coherent security property. We say that the path, P, taken by a packet is *conforming* when P itself conforms to the security policies of the entities along P—the sender, the receiver, and all transited realms. The process described above ensures that packets flow from senders to receivers only along conforming

paths, a property that we call *consent-to-connectivity*.

Consent-to-connectivity subsumes or generalizes the high-level goals of many prior proposals (§7.3). In those proposals, some stakeholder along a packet's path decides whether to drop the packet. The decision is based on various factors, but one frequent factor is the path itself, or rather, *some* component of the path—the source [6, 8, 13, 25, 32, 42, 44], the two endpoints [18, 24, 36], a suffix [41], a prefix [9], an arbitrary subsequence [22], the whole path [34], etc. A network that lets entities grant consent on an arbitrary basis in the control plane and that enforces consent-to-connectivity in the data plane serves to unify these proposals.

Consent-to-connectivity also leads to security innovations (§7.3). For example, under ICING, receivers and providers can control paths toward them. Such control allows them to invoke network services selectively (e.g., forcing the traffic of certain sources or classes to go through a third-party deep packet inspector).

In explicitly empowering intermediate providers to reject flows, consent-to-connectivity apparently violates the IPv4 ideal of global connectivity. But we observe that today providers can and do drop packets with no recourse for the endpoints. Under ICING, providers can exercise their legitimate interests but can do so explicitly, while endpoints can choose any conforming path. And, because decisions are made in general-purpose servers, prior to packet flow, they can incorporate arbitrary evidence, permitting them to avoid the coarse-grained rules (e.g., "block all traffic on particular ports from particular areas of the network") that cause collateral damage.

This separation of control plane software from forwarding hardware facilitates innovation, as articulated by others [14, 15, 23]. ICING's contribution to this separation is to permit a realm to unilaterally express security policies in its *own* software but to have *other realms'* forwarding hardware participate in enforcing those policies (§3.4,§7). More generally, deploying new security measures under ICING is closer to updating one's DNS servers than one's routers (§7.1). And, ICING encourages other kinds of innovation; for example, it permits coexisting inter-domain routing protocols (§7.1).

Our main focus in this paper is on upholding consent-to-connectivity, which requires addressing some challenges (§3). First, it is not clear how a sequence, S, of mutually distrustful realms can, without prior coordination or prohibitive per-packet public key cryptography, ensure that a packet follows S only if the realms along S consented to S (§3.2). Second, to avoid the overhead of executing the consent-granting process for every flow that it carries, a realm needs a way to *selectively delegate* its authority (§3.4). Third, senders must request consent, but how do they get consent to request the consent? In prior work, such bootstrapping requires great care [10, 32, 44]; ICING must address a similar challenge (§3.5).

ICING is technically viable: our prototype forwarder, built on an FPGA (§5), forwards at over 2 Gbit/s for all packet sizes. A production ASIC implementation of ICING would of course be far faster and could handle backbone links. We believe that ICING's space overhead is heavy but acceptable: for 1514-byte packets, the average header is 18% of the packet; for IP it is 1% (§6).

However, technical viability alone does not address the questions of how to gain adoption and how ICING coexists with other proposals of similar scope. We make only passing reference to these questions (§8, §9) and for now note that ICING promotes the coexistence of naming, routing and access control mechanisms. Ultimately, we believe that these questions are premature. Our purpose in this paper is to give an existence proof that ICING's goals are realizable; that proof must precede discussion about backwards compatibility with the Internet or cohabitability with other proposals, because without knowing if something is possible, we certainly cannot figure out how to deploy it.

Many other "clean slate" architectures have been proposed recently, some of which partially inspire this work (§8). ICING's contributions are its overall architecture (§2), particularly that mutually distrustful realms cooperate to enforce each other's policies; the articulation of consent-to-connectivity as a unifying property (§7); a protocol that upholds this property (§3); and a prototype that serves as an existence proof (§5,§6).

2 Overview of ICING and rationale

We now give an overview of ICING and describe the rationale for our high-level approach. §3–§4 detail ICING's design. This paper's focus is ICING's data plane, specifically interrealm transit. §7 gives example uses of ICING.

2.1 Overview

ICING divides the network into **realms**. Realms are defined by trust boundaries, and no two realms need trust each other. ICING does not change the basic network topology and peering model: today's ASes map naturally to realms. However, the granularity of a realm is variable. An organization that centrally administers its forwarders and hosts may form a single realm. When different machines belong to different people, each host, or virtual machine, or even process can be its own realm. (In §3.4, we describe a more convenient way to name endpoints.) For deploying ICING, it may be useful to regard the current Internet as one realm.

A packet's route brings it from the source, through intermediate realms, to the destination. We call this sequence of realms a **path**. The essence of ICING is as follows: Each realm decides whether a path, or set of paths, conforms to its security policy. These decisions

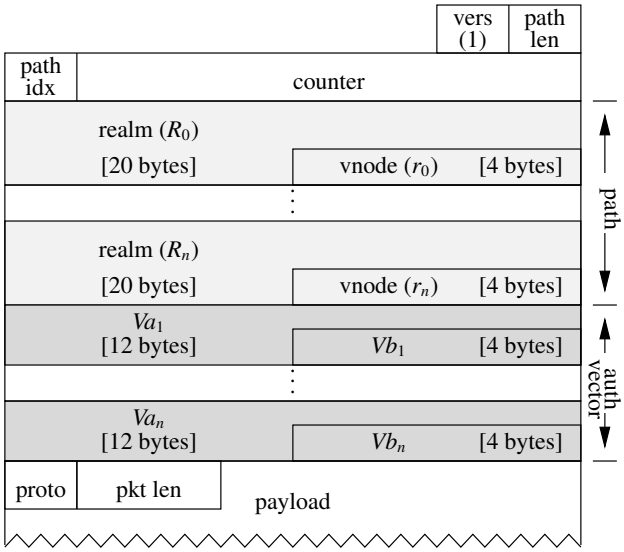

Figure 1—Format of an ICING packet (which follows a 14-byte Ethernet header).

are made in general-purpose servers, prior to a packet flowing, and can incorporate arbitrary information that the realm finds relevant (e.g., billing [36]). Forwarders collectively enforce these decisions, a property that we defined as consent-to-connectivity (§1). Below, in §2.2, we explain why ICING is built around this essence.

Figure 1 depicts the packet format. The packet contains its path as well as cryptographic values that allow the forwarders to validate the path.

The forwarders within a realm share two keys: a symmetric key, called the *PoC key*, and a public/private key pair, called the *realm key*. The PoC key allows the entity that makes policy decisions on behalf of a realm to indirectly communicate those decisions to the realm's forwarders. The way it does so is by issuing a *proof-of-consent* (PoC) to the sender. Because the cryptographic values in the sender's packets are bound to the PoC, the forwarders can verify that the realm issued consent (§3.2). We note that any machine that knows the PoC key can issue consent; the machine need not be physically located in the realm and can be an explicit *delegate* of the realm (§3.4).

A realm's name is its public realm key. This key allows realms earlier in the path to prove to later realms that they have processed a packet. Thus, forwarders can verify not only that the local realm consented to the path but also that the packet has actually followed its path. As ICING is so far described, this verification is conceptually simple: every realm that touches a packet could theoretically sign the packet with its private realm key. However, in practice, per-packet public key operations are prohibitive. Our actual approach is given in §3.2.

To send a packet, an application running on the source passes the path, a set of appropriate PoCs, and a payload to its local networking stack.

To obtain a path and PoCs, a sender, *S*, exercises *control plane* functions. These functions are conceptually distinct but in practice can happen simultaneously:

1. Configuration (when *S* attaches to a network);
2. Resolution of a name (e.g., a human-friendly name) to a realm name, R_{dest};
3. Resolution of R_{dest} to a suitable path, *P*, that leads from *S* to R_{dest}; and
4. Retrieval of PoCs that authenticate *P*.

Control plane traffic travels in ICING packets. Thus, only after presenting the data plane do we describe the details of the functions above (§4) and address bootstrapping questions (e.g., how senders get PoCs to send packets that request PoCs). We give a simplified and very brief summary now: *S* uses a broadcast configuration protocol (ICING's equivalent of DHCP) when joining the network to obtain its first PoCs. These PoCs allow *S* to communicate with a *path server* (an approach inspired by NIRA [43]). *S* queries the path server, supplying a name, and receives a path and PoCs for that path.

2.2 Rationale

We now explain how our high-level approach derives from three requirements.

The security benefits of network-layer defenses stem from the ability to drop packets. While different proposals invest this ability in different stakeholders (e.g., the receiver), our view is that if we have a clean slate opportunity, we should design in the interests of all stakeholders. For a given packet, the stakeholders are the source, the destination, and the intermediate providers.[1] Each of these stakeholders has legitimate reasons for dropping traffic (as also argued in [24]). Thus, our first requirement is *(1) give all stakeholders a way to refuse to carry traffic*.

Of course, any entity along a packet's path has always had the ability to drop traffic. But today entities make silent decisions. There is little transparency and no opportunity for endpoints to use alternate paths. Worse, entities make bad decisions. Because they lack information, they cause damage from drops that they commit (e.g., blocking traffic from Web sites that share IP prefixes with spammers) and drops that they omit (e.g., not shutting down denial-of-service attacks because doing so would deny service to legitimate hosts). Thus, our next requirements are that *(2) allow/deny decisions be explicit and prior to packet flow*, allowing senders and receivers to avoid providers that would drop their flows, and that *(3) these decisions be informed*, based on the precise infor-

[1] Certainly, there are other higher-level stakeholders, such as the human whose medical records are traveling from source to destination, but we are concerned with network-level security here so can uphold the interests only of stakeholders that are visible at that level.

mation that is needed, rather than heuristics.

These three requirements imply that all entities in a packet's path should make an allow/deny decision in the control plane, prior to packets flowing, based on arbitrary factors. But why is ICING's data plane designed to enforce consent-to-connectivity? While any entity's policies are likely to be idiosyncratic, *any* policy based in part on the path requires that the forwarding infrastructure constrain packet flow to meet the policy. And, as mentioned in §1, such policies are common; in the allow/deny decisions in prior work, entities often use *some* component of the path as an input, such as the source [6, 8, 13, 25, 32, 42, 44], the two endpoints [18, 24, 36], a suffix [41], a prefix [9], an arbitrary subsequence [22], the whole path [34], etc.

We further discuss consent-to-connectivity in §7.

3 Design of ICING's data plane

We start with a simplified version of the protocol that assumes that each realm consists of a single forwarder. We then augment the protocol with *vnodes*, a mechanism that enables delegation of PoC-minting ability, bootstrapping, intrarealm forwarding, and more.

3.1 Threat model

We assume that some realms (end-hosts and providers) are controlled by attackers who are subject to standard cryptographic hardness assumptions. We refer to the realms they control as *malicious*. These realms can deviate arbitrarily from the protocols that we describe, including sending arbitrary packets or flooding the links they have direct access to.

Realms that obey the protocol we term *honest*. Honest realms can carry packets between other realms. We make no assumptions on how malicious realms are implemented (they may directly connect to one another and be controlled by a single attacker). ICING concerns the behavior of honest realms, in particular determining when they have carried or should carry a packet.

3.2 Basic protocol

To uphold consent-to-connectivity, the data plane ensures that packets may transit an honest realm R only under the following conditions:

1. [Path Validity] The path P in the packet's header must have been previously approved by R; and
2. [Provenance Verification] The packet must verifiably have transited all honest realms before R in P.

Note that neither condition explicitly constrains the trajectory packets take *after* leaving a given realm R. Taken together, however, they guarantee that packets that skip any honest realm will be dropped at the *next* honest realm that they traverse. In other words, the two conditions together imply the following:

P	$\langle R_0, R_1, R_2, \ldots, R_{n-1}, R_n \rangle$. A packet's path.
M	$\{\text{vers}, \text{ctr}, \text{proto}, \text{len}, \text{payload}\}$. Its contents.
R_i	A public key which is also the realm name.
x_i	The private key of realm R_i.
s_i	The symmetric *PoC key* used by R_i's forwarders to verify packets.
$k_{i,j}$	SHA-1$(R_i, R_j, g^{x_i x_j})$. A symmetric key shared by realms R_i and R_j, and derived from their names using non-interactive Diffie-Hellman key exchange.
$\mathsf{poc}_{P,i}$	CMAC(s_i, P). *Proof of consent* (PoC) to path P by realm R_i.
V^i	$\langle Va^i_1, Vb_1, Va^i_2, Vb_2, \ldots, Va^i_n, Vb_n \rangle$. Auth vector when packet leaves R_i; allows downstream realms to verify provenance.
A_j	$H'\left(\mathsf{poc}_{P,j}, H([P,]\, M)\right)$.
Va^0_j	PRF-96 $(k_{0,j}, \{[H(P,M),] \text{ first 96 bits of } A_j\})$.
Va^i_j	PRF-96$(k_{i,j}, \{[H(P,M),]\, Va^{i-1}_j\})$. Proves to R_j that packet has transited path through R_i. Unused if $i \geq j$.
Vb_j	Last four bytes of A_j. Guards the forwarder slow path from DoS attacks.

Figure 2—Cryptographic values in ICING. PRF-96 is AES-XCBC-MAC-96 [21]. H' is CMAC (but may change in the future). Hash H and the bracketed quantities in A and Va were not implemented for the benchmarks in §6.

3. [No Path Deviation] Packets that fork off their declared path P by skipping an honest realm R_{skip} cannot traverse any honest realm that succeeds R_{skip} in P.

Designing a viable protocol that enforces the above conditions requires some care: in particular, per-packet public-key operations would induce an unacceptable performance penalty. Our protocol avoids this penalty by sharing symmetric keys between each pair of realms. Establishing the shared keys without adding hard state to forwarders requires public-key cryptography but only the first time the two given realms appear on the same path (or after a symmetric key has been evicted from cache).

Preliminaries and notation. As mentioned above, the name of a realm, R_i, is a public key. Because public keys are carried in packets, we wanted them to be as small as possible. Thus, we chose to use elliptic curve cryptography. Every R_i is a point on NIST's B-163 [5], which is a binary-field elliptic curve group. The corresponding private key, x_i, is the discrete logarithm of the public key: $g^{x_i} = R_i$, where g is a globally agreed upon generator. [2]

To make our protocol more amenable to a hardware

[2]The elliptic curve literature uses additive notation, but here, for readability, we use the more familiar multiplicative notation.

function SENDPACKET(P, pocs, m)
// $P = \langle R_0, R_1, \ldots, R_n \rangle$
// pocs = $\{\mathsf{poc}_{P,i} = \text{CMAC}(s_i, P) \mid 1 \le i \le n\}$
// $m = \{proto, packet\text{-}len, [return\ path + PoCs,] data\}$
// $M = vers \| ctr \| m$
for $(i = 1 \ldots n)$ **do**
$A_i = H'(\mathsf{poc}_{P,i}, H([P,]\ M))$
$Va_i = \text{PRF-96}(k_{0,i}, \{[H(P,M),]\ \text{first 96 bits of } A_i\})$
$Vb_i =$ last 4 bytes of A_i
$V^0 = \langle Va_1, Vb_1, Va_2, Vb_2, \ldots, Va_n, Vb_n \rangle$
$path\text{-}idx = 1$
$pkt = ver \| path\text{-}len \| path\text{-}idx \| cntr \| P \| V^0 \| m$
transmit pkt to R_1 // may need intrarealm forwarding
$cntr$++

Figure 3—Pseudocode for packet construction: $S = R_0$ constructs a packet to send payload m along path P. Note that P is 0-indexed and V^0 is 1-indexed.

implementation, we reduce the representation of the key from 163 bits to 160 bits. We do so by requiring the top 3 bits of R_i to equal a hash of the rest of R_i; the cost is a factor of 8 in expected key generation time. The security attained is roughly 80-bit security, comparable to the security of RSA-based systems with 1024-bit moduli [5].

We label the source of a packet R_0, the destination R_n, and the path $P = \langle R_0, R_1, ..., R_n \rangle$. For each R_i, assume the source holds a PoC that proves to forwarders in R_i that R_i has consented to P. For the purpose of creating PoCs, every realm R_i has a symmetric key, s_i, known to the equipment in that realm. A PoC is set as $\mathsf{poc}_{P,i} = \text{CMAC}(s_i, P)$. (We use CMAC-AES-128 [4], a deterministic message authentication code with pseudorandom outputs.) Realms change their PoC keys, s_i, periodically to safeguard against chosen-message cryptanalytic attacks.

Each realm implicitly has a pairwise shared key with every other realm. The keys are generated by non-interactive Diffie-Hellman key exchanges: realms R_i and R_j share $k_{i,j} = \text{SHA-1}(R_i, R_j, g^{x_i x_j})$.[3] The purpose of these keys is for realms along a packet's path to provide each other with cryptographic evidence of packet provenance.

Figure 2 summarizes the protocol's constructs.

Packet construction by sources. To send a packet, a source follows the pseudocode in Figure 3. Sources do not place PoCs in packets directly (see Figure 1): the presence of a PoC, say $\mathsf{poc}_{P,i}$, in the packet would only inform realm R_i's forwarders that the realm consented to the purported path, P, listed in the packet. However, there would be no guarantee that the packet actually took this path, and nothing to constrain the packet to continue to follow the path.

[3]Realms also use their private keys to sign certificates (§4). Such "dual purposing" of key material is wisely discouraged by folklore because the resulting interplay among distinct cryptographic functions might void their individual guarantees. A careful security analysis indicates that ICING's protocol is safe in this regard, but the details are outside this paper's scope.

function RECEIVE(pkt)
// $pkt = vers \| path\text{-}len \| path\text{-}idx \| cntr \| P \| V^{i-1} \| m$
// $M = vers \| ctr \| m$
$\mathsf{poc}_{P,i} = \text{CMAC}(s_i, P)$
$A_i = H'(\mathsf{poc}_{P,i}, H([P,]\ M))$
// extract components in V^{i-1} that we need to verify
let $\langle Va_i^{i-1}, Vb_i \rangle$ = the ith entry in V^{i-1}
// following line protects slow path from spurious calls
check that Vb_i equals last 4 bytes of A_i: if not, drop
// following line may require slow path invocation
compute $k_{0,i}, k_{1,i}, \ldots, k_{i-1,i}$
// simulate what earlier forwarders should have done to
// the ith component of the authorization vector
$W =$ first 96 bits of A_i
for $0 \le j \le i-1$ **do**
$W = \text{PRF-96}\,(k_{j,i}, \{[H(P,M),]\ W\})$
check that $W = Va_i^{i-1}$: if not, drop
// following line may require slow path invocation
compute $k_{i,i+1}, \ldots, k_{i,n}$
// construct V^i
$V^i = V^{i-1}$
for $i+1 \le j \le n$ **do**
$Va_j^i = \text{PRF-96}\left(k_{i,j}, \left\{[H(P,M),]\ Va_j^{i-1}\right\}\right)$
increment $pkt.path\text{-}idx$ to $i+1$
transmit pkt to R_{i+1} // may need intrarealm forwarding

Figure 4—Pseudocode for packet forwarding: R_i validates pkt and transforms V^{i-1} to V^i before sending pkt to the next realm. Note that P is 0-indexed and V^{i-1}, V^i are 1-indexed.

To provide such a guarantee, the packet contains an *authorization vector*, V^i. Each realm R_i receives a packet with V^{i-1} and transforms the vector to V^i. The source constructs V^0 from a set of packet- and realm-specific authenticators, $A_1, \ldots, A_n$. The authenticators bind the PoC, path, and payload together in a concise value that can be certified by each realm on the packet's path. Intuitively, V_j^0 assures realm R_j that realm R_0 claims to have sent this packet because only realms R_0 and R_j know the key $k_{0,j}$ used to certify this value.

Packet verification and transformation by forwarders. Figure 4 depicts the handling of packets by a realm, here represented as a single forwarder. Consider the ith entry of V^{i-1}, the vector at the time that R_i receives the packet. It is in two pieces: Va_i^{i-1} and Vb_i. The first piece, Va_i^{i-1}, should, in the absence of attack, be a recursive application of PRF-96 to A_i under keys $k_{0,i}, \ldots, k_{i-1,i}$. R_j can verify Va_i^{i-1} by recomputing it; if the value is correct, R_i knows all previous realms in the packet claim to have carried the packet, so at least the subset of them that is honest will have carried the packet.

To verify the integrity of Va_i^{i-1}, R_i's forwarder must derive any of $\{k_{0,i}, \ldots, k_{i-1,i}\}$ that it is not already caching. This derivation entails a non-interactive Diffie-Hellman key exchange, which is performed[4] in the forwarder's slow path and requires $\sim$ 4 msec in our environment (see §6.3).

The purpose of the second piece, Vb_i, is to guard the forwarder's slow path. Vb_i is always verified on the fast path and the packet dropped if the verification fails. Without this check, an attacker could invent realms and bogus paths to force spurious slow path operations on forwarders. Vb_i is only 32 bits, so it does not rule out such attacks altogether, just decreases their effectiveness by a factor of 2^{32}, sufficient to avoid DoS.

Return paths. Most network applications require bi-directional communication. While a destination, R_n, can reply to a source, R_0, by resolving R_0 at path servers and obtaining its own PoCs, it is more efficient for R_0 to negotiate the return path at the same time as it is negotiating the forward one. Thus, the payload of a packet begins with a return path and set of *return PoCs* that the recipient can use to reply. This offloads return path negotiation to clients, which helps in settings where it is important to minimize load on servers.

3.3 Analysis and limitations

ICING allows realms to innovate and deploy a variety of new security policies in the network (§7), but it is also important to realize what ICING does not achieve. While realms can enforce policy based on where a packet has been and on where it claims to be going, they cannot control where a packet actually goes downstream. If, for instance, realm B consents to path $\langle A, B, C, D\rangle$ but not to $\langle A, B, C, D'\rangle$, it is still possible for A, C, and D' to collude to use the latter path (which could potentially result in B charging D for traffic that actually went to D').

ICING also cannot meaningfully enforce negative policies against edge realms. While the Internet's IP address scarcity makes it somewhat reasonable to blacklist bad IP addresses, edge realms can easily generate new keys. However, a bad edge realm's provider (and the vnodes assigned to that edge realm, as discussed in the next section) may be harder to change, in which case one could achieve similar results by blacklisting the provider (or one of its vnodes).

More generally, the policies ICING serves best are those that can be expressed in terms of either authentication or network topology. For example, the policy embodied by a typical corporate VPN, whereby employees authenticate themselves to access a network, is easily accommodated by ICING, with the added bonus that ICING can stop unauthorized traffic upstream. In contrast, a policy to deny network access to all machines in the US would not be enforceable without a database of all US realms with international links.

ICING does not offer confidentiality, except as far as it allows realms to conceal communication patterns from hostile realms by routing around them. Proper confidentiality requires transport-layer encryption, a simpler project we have undertaken separately and hope to combine with ICING.

Finally we note that ICING cannot solve some control plane problems that are primarily social or business issues. For instance, providers can still drop packets that they have consented to carry. Or, ICING can secure hostname lookups with certificates, but deployment would run into the same political problems as DNSSEC.

3.4 Vnodes

Thus far we have described ICING as though each realm consisted of a single forwarder. In reality, ICING divides realms into vnodes [22] denoted by 32-bit identifiers. As shown in Figure 5, a packet's path actually consists of a series of (realm, vnode) pairs.

Vnodes provide end-host addressing. For ordinary unicast packets, the vnode of the last element in a packet's path designates the packet's destination within that realm. We also intend to use vnodes for intra-domain routing. Vnode identifiers are large enough to contain IPv4 addresses so as to facilitate using ICING to connect networks that run IPv4 rather than ICING internally. Vnodes need not correspond to individual devices, however. They can, for instance, be used for anycast (in particular to reach a realm's control-plane servers).

Every vnode is not necessarily reachable from every other vnode in a realm. Given two vnodes r_1 and r_2, it may be possible for packets to flow from r_1 to r_2, or from r_2 to r_1, or both, or neither. Similarly, neighboring realms can only put packets onto a subset of a realm's vnodes and receive packets from a (generally different) subset. We expect typical providers to assign a different vnode or set of vnodes to each customer.

The virtual topology connecting a realm's vnodes is an important part of its configuration. Others [22] have shown how vnodes can enforce a variety of inter-domain routing policies such as valley-free routing. We envisage prioritizing traffic by vnode and charging for it by vnode.

When two realms peer at multiple points, each peering point may connect to a different set of vnodes. For instance, to charge more for long-haul traffic, a provider can offer each customer two cheap vnodes, one limited to East-cost peering points and the other to West-coast ones, while a third, more expensive, vnode reaches all networks connected to the provider.

[4] An additional level of caching in software (not currently implemented) would ensure that a forwarder incurs the cost of the NIDH key derivation only the first time it encounters another realm.

P	Path is now $\langle T_0, T_1, T_2, \ldots, T_{n-1}, T_n \rangle$.
T_i	A pair (R_i, r_i). R_i is a realm; r_i is a vnode within R_i.
s_{T_i}	The PoC key for vnode $T_i = (R_i, r_i)$.
$POC_{P,i}$	CMAC(s_{T_i}, P).

Figure 5—Modification to cryptographic constructs in §3; compare to Figure 2.

Each vnode has its own PoC key, which simplifies changing PoC keys—one can gradually migrate traffic to a new vnode before changing the old one's key. A realm may chose to divulge the PoC key for one of its vnodes to another realm, thereby delegating the ability to create PoCs. We expect providers to give PoC keys to customers who run their own path servers. Customers may then deny upstream PoCs according to their own policy, stopping unwanted traffic before it transits their provider's networks.

A backbone provider will likely give PoC keys to the ISPs it sells transit to. Those ISPs may wish to subdelegate the backbone's PoC key. To allow controlled delegation, a backbone provider may delegate a block of vnodes to each ISP, who can then further subdelegate them individually or in smaller blocks. Like Platypus [36], we generate all vnode keys for a given prefix from a single master key in a way that allows further subdelegation along bit boundaries. (If s_p is the master key for prefix p, then $s_{p\|b} = H(s_p, b)$.)

3.5 Bootstrapping

One particularly important application of delegation is bootstrapping. Realms will want to host their own control-plane servers, in which case potential senders need a PoC to contact the server that grants PoCs. For this purpose, each realm can create a *bootstrap vnode* that connects only to its PoC server. Bootstrap vnode keys can be published in naming service glue records (much like DNS). Anyone possessing such a key can create PoCs to reach the realm's PoC server. Since the bootstrap vnode does not connect to any other networks, knowledge of its PoC key does not allow non-bootstrap traffic to transit the realm. We discuss potential "Denial of PoC" attacks [10] in §4.1.

4 Control plane functions

Our focus in this paper is ICING's data plane. For context and concreteness, we now mention some control plane functions. We have space to give only a sketch. The sketch covers only the "common case"; many other uses are possible.

Configuration. When an ICING host joins a network, it receives from a local *configuration server* the following state: (1) *partial interrealm paths to and from upstream realms*. For example, the configuration server might give a host at a university partial paths to a local provider that peers with the university, to the Internet2 network, to the university's commercial ISP, and to the top-tier ISP from which the commercial ISP buys service (as well as the reverse directions of those partial paths). (2) For each realm in the partial path, a *PoC key that allows the host to mint PoCs for those realms*. The idea here is that each provider would have sub-delegated (§3.4) a slice of vnode space to each of its customers. At the extreme, an end-host has permission to mint PoCs that allow the end-host's traffic to flow over, for example, a "slice" of a backbone network. (3) *The full path, and accompanying bootstrap PoC keys, to at least one* path server, PS_0.

Path construction. When an ICING host wants to reach a desired destination (steps 2–4 at the end of §2.1), it submits a human-friendly name to PS_0. The host and PS_0 perform a negotiation. If it is successful, they will have identified a path that goes through an intermediate realm I such that: (a) the host can mint PoCs to reach I and (b) the path server can mint PoCs to get from either I to the destination or from I to the next path server that the ICING host should query, at which point the process continues. (Here, the path server has the ability to mint PoCs on behalf of intermediate realms just as the sender does.) This approach is inspired by NIRA [43].

Consent servers and consent certificates. So far, we have described senders and path servers as minting PoCs. But in fact, each realm has a *consent server* (CS) that takes as input proposed paths and ancillary information, and issues PoCs on behalf of the realm. The ancillary information is arbitrary. The option that we envision (and have implemented) is for the CS to require a set of *consent certificates* from each realm along the proposed path. Such certificates are signed by a realm's private key. *An honest CS never issues a PoC unless it gets a set of consent certificates from each realm along the proposed path.* Thus, packets with non-conforming paths will not travel further than the first honest realm along the path.

Topology propagation. The topology propagation protocol that we have designed "pushes information to the edges" (i.e., to path servers and configuration servers). We do not have space for more than a few words. Under this protocol, advertisements flow *downward*, from transit providers to customers. A provider, say T, collects the advertisements that it has heard, appends a group of certificates that express information about T's interrealm links, and forwards the augmented set of advertisements to its transit customers. Thus, when a realm receives an advertisement, it gets information about its provider, its provider's peers, its providers' providers, etc. This process can handle topology changes at coarse grain time scales. There are several possibili-

ties for handling dynamic topology changes at fine grain time scales, but we have not implemented them, and there are some issues that require care (as also observed in [11, 12]) so leave this to future work for now; see §9.

4.1 Flooding attacks

If a realm wishes to receive traffic from anyone in the world (for instance because it runs a Web server), that realm will be vulnerable to packet-flooding DoS attacks. In the general case, such attacks are inevitable and impossible to defend against. ICING cannot solve the general problem, but the specifics of a scenario often leave room for effective defense measures; ICING allows victims to capitalize on those specifics.

We give three scenarios in which ICING can help. The DoS victim may have user accounts, only a small fraction of which are compromised (even if many machines attack). The attacks may be coming through a small number of providers. Or the attacker's machines may mostly reside behind honest forwarders and path servers.

However, we must consider two types of DoS attack—a flooding attack against an application server, and a "**denial of PoC**" attack against the victim's consent server, which may be located on a bootstrap vnode with a publicly known PoC key.

DoS attacks on application servers can be mitigated by withholding consent (not granting PoCs), a common approach [8, 32, 42, 44]. ICING contributes the ability to base decisions on the full path, allowing victims to recognize and deny a malicious provider emulating many customers. If the victim has some notion of authenticated users, each user can be redirected to a different vnode and fair queuing used among vnodes to ensure a small number of bad users cannot crowd out the honest ones.

Several measures can combat "denial of PoC" attacks in which attackers flood a realm's consent server with traffic. We first note that a consent server can be reachable though multiple vnodes. Some will likely be public, while others may be reserved for employees (who might cache PoC keys on their laptops), and so with fair queuing at least an attack on the public vnode will not crowd employees out from reaching the company.

Alternatively, an organization need not host its own public consent server. Today, third-party DoS defenders exist that mediate all traffic to customers [35]. With ICING, such services need only host (or mediate access to) the public consent server. Authenticated users can then communicate directly with the organization.

5 Implementation

We describe our implementation of the data plane and then our (partial) control plane implementation.

Data plane. Our prototype ICING forwarder accepts ICING packets carried in Ethernet frames. The core of the prototype is an image that runs on the NetFPGA programmable hardware platform [3], which has 4 Gigabit Ethernet ports. The hardware executes the fast path (see Figure 4 in §3.2). When an ICING 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 $k_{i,j}$ (§3.2) cached in hardware, the hardware sends the packet to a software slow path (again, see Figure 4) over the PCI bus to an x86 processor running Linux (versions 2.6.18 and 2.6.25 in our experiments).

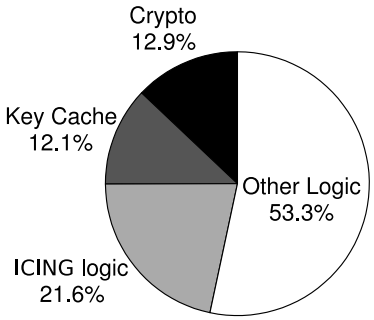

Figure 6—Hardware logic area costs.

The slow path, implemented in the Click modular software router, calculates the needed keys and installs them in the hardware's key cache, possibly evicting old keys. The Diffie-Hellman is implemented with the publicly available MIRACL cryptographic library. Together, the fast path and software implement the protocol described in §3.2. We have not yet implemented per-vnode PoC keys (§3.4) and plan to do so soon.

The hardware image is based on the reference base package provided by the NetFPGA project. We implemented the ICING-specific logic, including the cryptographic modules and re-used the reference package's support modules—Ethernet logic, DMA logic, queueing, and the interface to host software.

The total equivalent gate count for our 2 Gbit/s NetFPGA forwarder design is 11.1M gates. In comparison, the 4 Gbit/s reference IP router from the NetFPGA project [3] has an equivalent gate count of 8.65M. The implementation uses 78% of the total FPGA logic area. The relative costs in terms of logic area, or lookup tables (LUTs) used, of the main components are shown in Figure 6. Of the 78% total, the ICING-specific logic uses approximately 46.7%, while the support modules mentioned earlier use the other 53.3%. Figure 7 shows the major hardware blocks.

We now address an important performance concern. From Figure 4, one might expect the cost of processing a packet to depend on the length of a packet's path and on a realm's position in the path (the value of i): a larger path length means more iterations of AES-XCBC-MAC-96, and the value of i affects whether the forwarder is doing more "verification" or more "MACing". However, the number of AES-ENCRYPT operations—the slowest

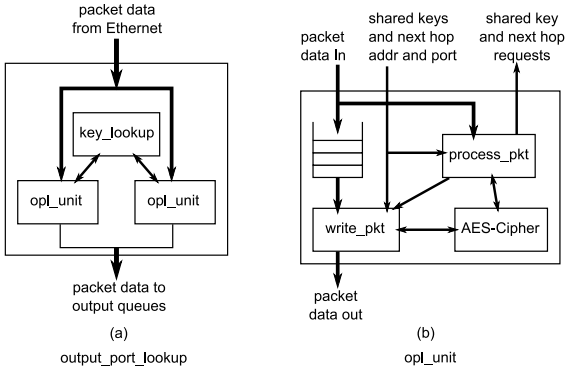

Figure 7—Block diagram for ICING forwarder fast path showing (a) the *output_port_lookup* module and (b) the *opl_unit* module. *output_port_lookup* encapsulates two *opl_unit*s that process data arriving from the Ethernet interface and that share a single *key_lookup* module. Inside *opl_unit*, the *process_pkt* module pre-processes the packet and verifies V^{i-1} while the *write_pkt* module transforms it into V^i (§3.2).

operations used in our protocol—performed per packet is proportional to the packet size. The result is that *our prototype* ICING *forwarder's per-packet processing cost depends only on the total size of the packet, not on the path length, the forwarder's position in the path, or the other header values*; that is, its throughput (in bits per second) is well-defined (and very close to the AES-ENCRYPT throughput without pipelining). We validate this claim experimentally in §6.3.

Control plane. Our control plane software currently runs over IP; our near-term work is to make it run over ICING, as well as to address the other incomplete items that we are about to describe. So far, we have implemented a combined consent server and path server (§4). It is implemented in C++, exposes a `getpath` call over XDR RPC, and runs on Linux 2.6.18 and 2.6.27. It can participate in a DNS-like hierarchy by supporting non-recursive queries that return partial paths to a desired destination or to other path servers. We emulated the results of a routing protocol by using configuration files to seed the path server with paths to realms. Clients requesting paths from the path server were manually bootstrapped with paths to intermediate realms.

Endpoints. We implemented ICING-aware sender and receiver applications. After communicating with the path server, a sender calls a user-level library function, which creates ICING packets (per SENDPACKET in Figure 3 in §2.1), embeds them in dummy IP packets, and directs them to a local Click instance using `tun`. Click strips the IP header and transmits the ICING packet in an Ethernet frame. The receiver application gets the ICING packet encapsulated in another dummy IP packet from a local Click instance through `tun`. The application then strips the IP header and calls another user-level library function that verifies the packet (the function is similar to RECEIVE in Figure 4). We ran this software on Linux 2.6.25 and 2.6.27.

The average packet overhead is 18.4%.	§6.2
Our prototype forwarder processes packets at 2.1 Gbit/s for all packets. The same hardware platform processes IP packets at 4 Gbit/s using 78% of the logic used in the ICING implementation, suggesting that the hardware cost of ICING is tolerable.	§6.3
Over the WAN, the link delays are the major contributor to end-to-end latency and control path latency is negligible relative to the end-to-end latency.	§6.4

Table 1—Summary of main evaluation results.

6 Evaluation

The ICING protocol (§3.2) introduces a number of per-packet cryptographic operations. One of our principal questions is the degree to which it is practical to perform such checks at today's Internet backbone link speeds.

We began by collecting microbenchmarks (§6.3). We also deployed our ICING prototype in the Internet2 backbone and used it to carry traffic between hosts at geographically distant locations. We used this deployment to evaluate the feasibility and performance of a potential ICING backbone network (§6.4). In §6.5 we extrapolate from our findings to assess ICING's future feasibility in the Internet core. Table 1 summarizes our main results.

6.1 Setup and Parameters

We used four types of machines when evaluating ICING: a *slow* machine, with an Intel Core 2 Duo 1.86 GHz processor and 2 GB of RAM; a *medium1* machine, equipped with an Intel Core 2 Quad 2.40 GHz CPU and 4 GB RAM, also running Linux 2.6.25; a *medium2* machine, with an Intel Core 2 Duo 2.33 GHz CPU and 2 GB RAM, running Linux 2.6.27; and 3 *fast* Intel quad Xeon 3.0 GHz machines with 2 GB RAM and, running Linux 2.6.18. The *slow* and *medium1* machines were setup in one lab, while the *medium2* machine was located in a geographically distant lab. The *fast* machines are installed in 3 Internet2 Point-of-Presence (PoP) locations: Houston, Los Angeles, and New York. All machines, except the *medium2*, are configured with NetFPGA cards.

The NetFPGAs in the Internet2 nodes were connected in a full mesh by dedicated 100Mbit/s circuits. The nodes themselves were accessible from the Internet.

In our measurements, we often vary packets' path lengths, path indices, or sizes. Here, we present three sets of points that we have used in §6.3 and §6.4. When the *path length* was a parameter, it varied over {3, 7, 10, 20, 30, 37}, with the path index set to one. For throughput measurement the payload was set to give 1514-byte

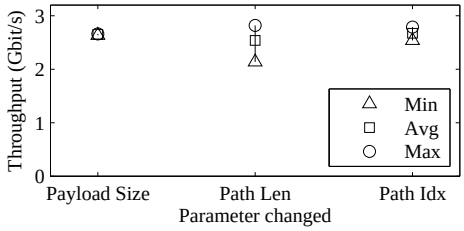

Figure 8—Maximum, average, and minimum throughputs when each parameter was varied as described in §6.1. For each 30-second sample, the standard deviation was less than 5% of the mean. The throughput varies slightly as packet parameters are changed.

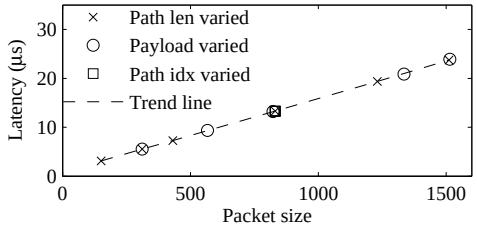

Figure 9—Summary of the three latency measurement results using the points described in §6.1. The standard deviation was 2% less than the mean in all measurements. The prototype's forwarding latency is independent of the ratio of path size to payload size or current path index. The latency depends only on total packet size; the trend line has slope 15 ns/byte.

packets, while for latency measurements the payload size was set to zero. When the *path index* was a parameter, it varied over {1, 5, 10,15, 18} and the path length was set to 20 and the packet size to 831 bytes. When the *packet size* was a parameter, it varied over {311, 567, 823, 1335, 1514} while keeping the path length and path index constant at 7 and 3 respectively.

6.2 Packet overhead

The ICING header size is significant. The header fields that are not dependent on the packet's path length use 13 bytes (see Figure 1). Each (R_i, r_i) is 24 bytes, and each component of V uses 16 bytes (the notation is from Figures 2 and 5). Thus, the total overhead in bytes is:

$$13 + 24 + 40 \cdot (path\_length - 1)$$

For a packet whose path is 7 realms long—the average length of an AS level path found in [28]—the header is 277 bytes or 18.3% of a 1514-byte packet. By comparison, IP's 20-byte header is 1.3% of a 1514-byte packet.

6.3 Microbenchmarks

ICING forwarder We began with our prototype's throughput and per-packet latency, for both the fast path and the slow path. As mentioned in §5, we expect the first two metrics to be independent of the path length and of the forwarder's position, i or the path index, in the path (except as far as these factors affect the total packet length).

Forwarder fast path throughput. In our lab, we connected the four ports of the ICING forwarder to a NetFPGA packet generator that can send ICING packets at line-rate. We varied the packet's path length, path index, and the payload size and measured the throughput over 20 30-second samples at the measurement points described in §6.1. The ICING forwarder looped ingress packets back into the packet generator, which measured the average rate of packet arrivals. All required symmetric keys were already in the hardware cache.

Figure 8 plots the measurement results.The standard deviation of each of the 20 samples we collected was less than 5% of the mean. The minimum aggregate throughput—over the four ports and including the Ethernet preamble and inter-packet gap—was 2.1 Gbit/s.

Forwarder fast path latency. To measure latency, again we employed the packet generator and used the points described in §6.1. The packet generator can accurately send packets simultaneously and timestamp packet arrivals (with errors of a few nanoseconds). In our lab, we connected two ports of the packet generator together through the ICING forwarder and the other two ports directly to each other. All required symmetric keys were again in the hardware cache.

The packet generator simultaneously sent packets through the ICING forwarder and through the loop-back cable and timestamped packet arrivals. The difference between the timestamps is the forwarder latency.

We sampled 100 single packet latencies for each measurement point. Figure 9 shows that latency grew linearly with packet size and was independent of the forwarder's position in the packet path, as hypothesized in §5.

Forwarder slow path latency. We do not consider the time a packet takes to travel from the fast path to the slow path via the PCI bus or vice versa because the PCI bus latency is negligible (tens of microseconds) compared to the shared key calculation as will be shown.

The primary function of an ICING forwarder's slow path is the calculation of the shared symmetric keys ($k_{i,j}$) that are not currently in the hardware cache. To measure the cost of this operation, we ran the shared key calculation function in a tight loop to calculate 3,000 keys. We ran the calculations on the *slow* machine (§6.1) because both end-hosts and forwarders will need to get shared keys in an ICING network. On average, a single key calculation took 4 ms, with a standard deviation of 43 μs. Writing this new key to hardware cache was negligible in comparison, taking approximately 9 μs.

Because elliptic curve cryptography (ECC) is

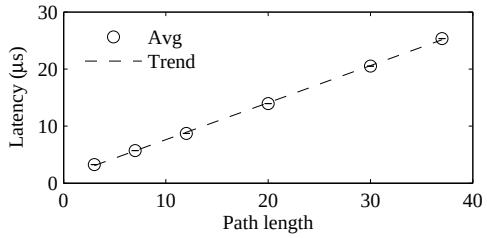

Figure 10—Average latency of the example consent server as the requested paths' lengths is varied. Standard deviation was less than 2% of the mean. The trend line has slope 0.648 μs/realm

amenable to a hardware implementation, it is possible to considerably speed up the slow path using a hardware ECC co-processor.

Path server and consent server To better represent the hardware that would run ICING control plane servers, we used the *fast* Internet2 nodes (§6.1) to run server benchmarks. We measured the throughput of our prototype path server and consent server by isolating the two bottleneck functions: `getpath` and `getpocs`.

Our prototype `getpath` function generates complete paths by finding a common intermediate realm (as described in §4). We bootstrapped the path server with 20 paths from intermediate realms to destination realms, and called its `getpath` function providing it a random number between 1 and 20 of paths from sources to intermediate realms. We measured the throughput of `getpath` by calling it in a tight loop over 20 30-second intervals. The prototype path server generated approximately 6,990 paths/s, with a standard deviation of 42.6 paths/s.

We measured the throughput of the `getpocs` function of a consent server by calling it in a tight loop and varying the path length in each request. Since the consent server is single-threaded, we plot the latency of a single request as the inverse of the throughput. The plot in Figure 10 shows that the latency to generate a PoC is proportional to the path length, which is expected because the PoC is a CMAC of the path.

End-host We measured the throughput at which senders can create and receivers can verify packets. For these experiments, we used the *medium1* machine (S6.1).

Sender. We measured the sender latency as the path length varied when it already had all the required PoCs. The sender generated 1000 1514-byte sized packets in a tight loop with variable path length as in §6.1.

Receiver. We also measured receiver latency as the path length varied using the same parameters. We sent the same packet to the receiver (in the same memory location) in a tight loop and measured the latency for 1000 packets. The results of both the sender and receiver latency measurements are in Figure 11.

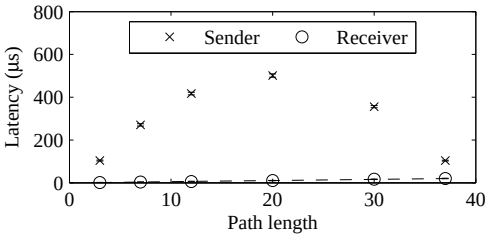

Figure 11—Average latency of senders and receivers when the total packet size was constant at 1514 bytes and the path length changes using the path length measurement points from §6.1. Standard deviation was less than 2% of the mean at each point. The sender has the worst latency when the number of encryptions is maximized at path length = 20. The receiver latency is proportional to the path length with a slope of 0.55 μs/realm.

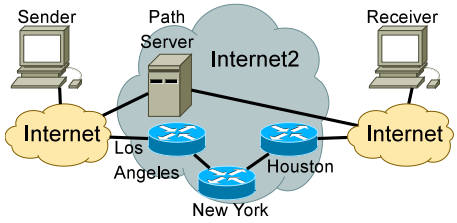

Figure 12—Our Internet2 experimental deployment. Two end-hosts in geographically distant locations are connected over the Internet to a path server and to Internet2 nodes running as ICING forwarders.

The receiver is similar to a forwarder and incurs latency proportional to the path length for a constant packet size. The sender, on the other hand, CMACs the payload *path_length* times. And since the payload size is the difference of the packet size and path size, it follows that, as the path length increases, the latency follows a parabola as shown in Figure 11.

6.4 End-to-End

In our end-to-end experiment, each node is a unique realm in an attempt to emulate a microcosm of three peering network providers sharing a single path and consent server. Figure 12 shows our deployment. We ran forwarders on the *fast* Internet2 machines and used the *medium1* and *medium2* machines as sender and receiver respectively (see §6.1). The Los Angeles Internet2 node ran a path server that could be queried by ICING-enabled end-hosts over IP. The end-hosts connected to the backbone using a software IP tunnel.

The first packet of a new flow experiences two ICING delays: path/PoC query and slow path invocation. To measure these two operations, we use an ICING ping application on the end-hosts. The sender initiates the ping by first querying the path server for a path to the end-host at the other end of the Internet2 network. The path server acts as both a path and consent server, returning a complete path and the necessary PoCs for communi-

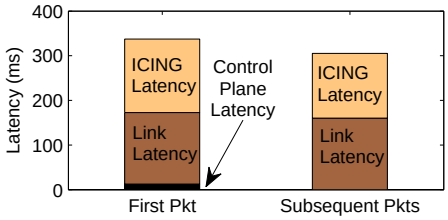

Figure 13—End-to-end latency by component for the first packet in a new path and the subsequent packets going over Internet2. The ICING latency includes the IP tunnel processing. The path query adds negligible delay, and the link delay is approximately as large as the total processing delays.

cation in *both* directions. The sender generates a packet, and includes the "return" PoCs in the payload. We measured the round trip time as the time from when a path server request is sent by an end-host to the time when a ping echo reply is received. The measurement results are summarized in Figure 13.

The latency to query the path server is 3.4% of the total latency, and thus negligible. Note that because the forwarder's fast path latency (Figure 9) is on the order of tens of microseconds, and the sender and receiver latencies are on the order of hundreds of microseconds (Figure 11), while the ICING processing latency from Figure 13 is on the order of hundreds of milliseconds, we conclude that most of the ICING processing latency is due to the IP tunnel and the overhead of using Click in user space. In future work, we will move the IP encapsulation/decapsulation into the hardware and use kernel-level Click.

6.5 Scaling

In this section, we extrapolate the measured performance of our ICING prototype to assess whether a production implementation could meet the Internet's backbone rates. We examine throughput and key cache size.

Throughput. Current backbone links run at 40 Gbit/s, which should increase to 100 Gbit/s by 2010 [27]. Our NetFPGA implementation gets 2 Gbit/s with older technology—the latest Virtex6 FPGAs are more than 10 times as dense and can run at a much higher clock speeds. Thus, we expect ICING hardware to be neither expensive nor difficult to build to meet backbone requirements.

Key cache size. The key cache (§5) stores the calculated symmetric shared keys in the forwarder's hardware. To ensure that traffic through an ICING forwarder runs at line rate, the cache must fit the symmetric shared keys for all realms in all packet paths in the traffic. Here, we estimate the required size.

If we assume that each realm corresponds roughly to an autonomous system, we can set an upper bound on the maximum size of the key cache by looking at the number of advertised Autonomous System Numbers (ASNs). As of March 3rd, 2009 this number is less than 32k and growing at less than 3.2k/year [1], so the key cache size for an ICING forwarder that can handle today's traffic and the traffic for at least the next 5 years—assuming the growth rate remains constant—is less than 100k entries. Note that a forwarder will almost certainly never be receiving flows passing through *every* realm on the Internet, so the actual required number is less. The estimate found above, however, does suggest that it might be possible to eliminate the slow path by precalculating the symmetric keys for all realms.

Current IP routers and switches already have tables in the order of hundreds of thousands of entries[2], and so we do not believe the key cache will be an obstacle. In fact, most packets will tend to follow very similar paths since backbone link topologies are sparse, and, hence, the number of keys that need to be stored in the key cache will be much less than 100k.

7 Implications, applications, unifications

A benefit of ICING is that, once deployed, it allows people to meet the goals of many other proposals without further data plane modifications. To illustrate ICING's versatility, we begin with some high-level observations (§7.1), move onto specific uses of ICING that are not explicitly related to security (§7.2), and then discuss ICING's potential to unify many previous proposals and current mechanisms (§7.3). As part of this discussion, we give a (crude) taxonomy that references a sizeable quantity of related work. We discuss other related work in §8.

7.1 High-level implications

Much of ICING's flexibility stems from the separation of path and consent servers from the forwarding hardware. Updating these servers is akin to updating one's DNS servers today—and far easier than changing one's routers. And, just as one could introduce a new naming facility to the Internet without disabling DNS, ICING allows multiple naming, routing, and access control frameworks to coexist on the same forwarding infrastructure. This decomposition promotes innovation (as argued by others [14, 15, 23]): it allows people to evaluate new ideas on production infrastructure without the global coordination from tier-1 ISPs and router vendors that would be required to test, say, a BGP replacement.

Moreover, because ICING is specifically geared toward enforcing a coherent security property—namely that packets not take non-conforming paths—people can experiment with new control plane functions without having to worry that doing so violates the security policies of any realms. In other words, ICING treats routing and security as the same problem, in the same framework. (With few exceptions [15, 37] these two problems are handled with different mechanisms in the literature, despite their natural connection.)

7.2 Non-security applications

Under ICING, the existence of multiple paths from clients to servers is not hidden. One consequence is that ICING provides a natural solution to "multipath", i.e., permitting sources to choose from multiple paths to a destination or to use several paths simultaneously.

Another consequence is that ICING facilitates anycast. Path multiplicity is already explicit in ICING, and the mechanism works whether the paths terminate at the same or different physical instances. Thus, clients and servers can negotiate between them to devise optimal routes to nearby replicas. Today, commercial content distribution networks approximate this functionality with DNS tricks, but doing so requires detailed, proprietary knowledge of the Internet's topology.

7.3 Potential unifications

We now try to give some intuition for how ICING can express the high-level goals—except for backward compatibility—of current mechanisms and prior work.

Consider blacklisting. One can implement it unilaterally in ICING's control plane servers by withholding consent and not returning PoCs for any path that contains a blacklisted realm. More interesting, using delegation via vnodes [22] (see §3.4), a destination realm can get other realms to drop blacklisted packets closer to the source. As discussed in §3.4, if providers delegate PoC creation for each vnode to the path server of the customer billed for that vnode, then a destination's path server can withhold upstream consent for blacklisted paths.

Consider VPNs, in which an organization wants only employees to send packets to its network. Under ICING, the organization simply sets its consent and path servers not to return paths or PoCs to non-employees. Doing so requires a way for employees to authenticate themselves to the company's consent servers (e.g., single-sign-on authentication systems). But ICING can also permit richer and more precise policies than VPNs. For instance, each employee may be given a separate vnode which can reach only an employee-specific set of servers.

Note that in this example, the organization's consent server is likely to be a target of denial-of-service attacks, but it can be located off-site, possibly hosted by a DoS-prevention company [35]. Or an organization might dispense with consent servers and instead issue employees tamper-resistant hardware devices containing present and future PoC keys. So long as each employee is given a separate vnode, revocation is simple.

A number of recent proposals have shown how to derive security benefits from scenarios in which edge routers are honest even if the hosts behind them are compromised [8, 13, 25, 32, 42, 44]. ICING offers similar benefits when a host's local path resolver (§4) is honest, as follows. If a host is "undesirable" in the view of some destination, then its local resolver will not be able to negotiate a valid path from it to the destination. With no valid path, an honest resolver will refuse to generate PoCs that allow the host to send traffic *out* of the local network along that path; this refusal "contains" the "undesirable" host, in the view of the given destination.

Crude taxonomy. We now give a taxonomy of proposals (which is necessarily incomplete) and show how ICING captures this taxonomy. By upholding consent-to-connectivity, ICING captures the cross product of *which entity along the path gets to make a decision* with *which entities along the path form the basis of the decision*. That is, consent-to-connectivity captures senders controlling the path [34], receivers deciding which hosts can send them traffic [8, 13, 25, 32, 42, 44], providers controlling inbound paths [9], providers controlling onward paths [41], providers controlling subsequences of paths [22], middleboxes controlling each other [24], etc.

And, this cross-product directly suggests new security notions that are not expressible or enforceable under the status quo. For example, receivers should be able to control the path toward them. This would allow an organization (financial, medical, intelligence, legal) that handles sensitive data to ensure that only providers that it trusts carry data to it or between two remote offices.[5]

Of course, many allow/deny decisions today are based on packet contents rather than some component of the path (e.g., "packet washing", intrusion detection, or malware detection). ICING permits such defenses. Within an organization, it suffices to configure the vnode topology to force traffic through these middleboxes. More interestingly, however, such defenses need not be implemented inside the realm that wants them. A company might consent only to inbound traffic whose path transits some third-party security service. The service may be many hops away from the company on the network, but if the service appears in the path, then any honest realm between the service and the company will drop forged traffic that has not transited the service.

Note that we do not discuss "control plane" security proposals (e.g., [26, 29, 40]); the issues addressed by these proposals don't arise under ICING. Also, given our focus on the network layer, we do not discuss proposals that work at other layers (e.g., DNS, overlays [7, 17, 30]).

8 Other related work

In §7.3, we referenced many related works during our argument that ICING unifies the high-level goals of many prior proposals. Here we just single out a few research strands that ICING borrows from or builds on.

ICING's PoCs generalize network capabilities [8, 36,

[5] Such an organization might not want to rely on end-to-end encryption alone because, even with encryption, network eavesdroppers can sometimes learn useful information from packet sizes and timing [38].

42, 44] and Visas [18]. In all cases, some logical entity "mints" a permission that a sender can use for packet transit. At a very high level, the principal differences between a PoC and a capability or a Visa are that (1) PoCs explicitly consent to the *entire path* that a packet is supposed to take and (2) the forwarding infrastructure constrains the packet to take this path. These requirements permit ICING to uphold consent-to-connectivity but lead to a very different design that, compared to these proposals, is a harder "deployment sell".

ICING's approach to delegation is inspired by Pathlet routing [22] (from which we borrow vnodes) and Platypus [36] (whose hierarchical subdelegation technique ICING uses for delegating "slices" of vnode space). ICING also borrows some of its control plane techniques from NIRA [43]. And ICING's instantiation of policy in general-purpose servers apart from forwarding hardware echoes [14, 15, 23].

As mentioned in §7.1, a natural consequence of ICING is that packets' paths respect providers' policies. Of course, there has been much work in policy routing over the years from early frameworks like [16, 19, 20] through more recent attention to BGP (e.g., [41]), but none of these proposals upholds consent-to-connectivity.

ICING is trying to solve a variant of the "secure forwarding" problem [11, 12, 33, 34]. In contrast to this work, ICING does not rely on a trusted entity or centralized administration (as in [33, 34]). ICING's recursive application of PRF-96 (§3.2) is reminiscent of a construction in [12], but ICING operates under a different trust model and cannot assume pairwise coordination. Also, in contrast to these proposals, ICING explicitly upholds all stakeholders' interests, not just the source's.

Of course, many "clean slate" proposals have emerged recently. ICING's relationship to this work is that ICING respects traditional layering (mostly) and is a network layer so is mostly orthogonal to proposals that focus on other layers (e.g., [31, 39], etc.).

9 Summary and future work

Summary. Our purpose in this work was exploratory: to identify a coherent security property and to find a way to uphold it with a viable implementation; we believe this effort was successful.

Near-term work. In the data plane, we will enhance our prototype to implement the bracketed quantities in Figure 2 and per-vnode keys (§3.4). In the control plane, we will run control traffic over ICING, improve our path server, and handle route failures and topology changes at fine grain. Last, we will consider how to use the momentum behind various clean slate efforts to identify potential ICING deployments.

References

[1] The 32-bit autonomous system number report. `http://www.potaroo.net/tools/asn32/index.html`. Last accessed on 3/2/2009.

[2] Integrated device technology (IDT) quick reference guide. `http://www.idt.com/products/getDoc.cfm?docID=18640144`. Last accessed on 1/30/2009.

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

[4] Recommendation for block cipher modes of operation: The CMAC mode for authentication. Special Publication (800-series), May 2005. DRAFT FIPS PUB 186-3.

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

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

[7] D. G. Andersen. Distributed filtering for Internet services. In *Proc. USENIX Symposium on Internet Technologies and Systems (USITS)*, Mar. 2003.

[8] T. Anderson, T. Roscoe, and D. Wetherall. Preventing Internet denial-of-service with capabilities. In *HotNets*, Nov. 2003.

[9] 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.

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

[11] I. Avramopoulos, H. Kobayashi, R. Wang, and A. Krishnamurthy. Amendment to: Highly secure and efficient routing. Feb. 2004.

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

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

[14] 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.

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

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

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

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

[19] D. Estrin, Y. Rekhter, and S. Hotz. Scalable inter-domain routing architecture. In *SIGCOMM*, Aug. 1992.

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

[21] S. Frankel and H. Herbert. The AES-XCBC-MAC-96 algorithm and its use with IPsec. RFC 3566, Network Working Group, September 2003.

[22] P. B. Godfrey, S. Shenker, and I. Stoica. Pathlet routing. In *HotNets*, Oct. 2008.

[23] 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.

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

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

[26] Y.-C. Hu, A. Perrig, and M. Sirbu. SPV: Secure path vector routing for securing BGP. In *SIGCOMM*, Sept. 2004.
[27] IEEE 802.3ba Working Group. IEEE P802.3ba 40Gb/s and 100Gb/s Ethernet task force. `http://grouper.ieee.org/groups/802/3/ba/index.html`.
[28] 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.
[29] S. Kent, C. Lynn, and K. Seo. Secure border gateway protocol (S-BGP). *IEEE JSAC*, 18(4), Apr. 2000.
[30] A. D. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure overlay services. In *SIGCOMM*, Aug. 2002.
[31] 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.
[32] X. Liu, X. Yang, and Y. Lu. To filter or to authorize: Network-layer DoS defense against multimillion-node botnets. In *SIGCOMM*, Aug. 2008.
[33] A. T. Mizrak, Y.-C. Cheng, K. Marzullo, and S. Savage. Fatih: Detecting and isolating malicious routers. In *IEEE DSN*, June 2005.
[34] R. Perlman. Routing with byzantine robustness. Technical Report TR-2005-146, Sun Microsystems, Aug. 2005.
[35] Prolexic Technologies, Inc. `http://www.prolexic.com`.
[36] B. Raghavan and A. C. Snoeren. A system for authenticated policy-compliant routing. In *SIGCOMM*, Sept. 2004.
[37] T. Roscoe, S. Hand, R. Isaacs, R. Mortier, and P. Jardetzky. Predicate routing: Enabling controlled networking. In *HotNets*, Oct. 2002.
[38] D. X. Song, D. Wagner, and X. Tian. Timing analysis of keystrokes and timing attacks on SSH. In *USENIX SECURITY*, Aug. 2001.
[39] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. In *SIGCOMM*, Aug. 2002.
[40] L. Subramanian, V. Roth, I. Stoica, S. Shenker, and R. H. Katz. Listen and whisper: Security mechanisms for BGP. In *NSDI*, Mar. 2004.
[41] W. Xu and J. Rexford. MIRO: Multi-path interdomain routing. In *SIGCOMM*, Sept. 2006.
[42] 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.
[43] X. Yang, D. Clark, and A. W. Berger. NIRA: A new inter-domain routing architecture. *ACM/IEEE Transactions on Networking*, 15(4), Aug. 2007.
[44] X. Yang, D. Wetherall, and T. Anderson. A DoS-limiting network architecture. In *SIGCOMM*, Aug. 2005.