Verifying and enforcing network paths with ICING

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

ABSTRACT

We describe a new networking primitive, called a Path Verification Mechanism (PVM). There has been much recent work about how senders and receivers *express* policies about the paths that their packets take. For instance, a company might want fine-grained control over which providers carry which traffic between its branch offices, or a receiver may want traffic sent to it to travel through an intrusion detection service.

While the ability to express policies has been well-studied, the ability to *enforce* policies has not. The core challenge is: if we assume an adversarial, decentralized, and high-speed environment, then when a packet arrives at a node, how can the node be sure that the packet followed an approved path? Our solution, ICING, incorporates an optimized cryptographic construction that is compact, and requires negligible configuration state and no PKI. We demonstrate ICING's plausibility with a NetFPGA hardware implementation. At 93% more costly than an IP router on the same platform, its cost is significant but affordable. Indeed, our evaluation suggests that ICING can scale to backbone speeds.

Categories and Subject Descriptors: C.2.1 [Computer-Comunication Networks]: Network architecture and design; C.2.0 [Computer-Communication Networks]: General—Security and protection; C.2.2 [Computer-Communication Networks]: Network protocols; D.4.6 [Operating Systems]: Security and Protection—Access controls; Authentication; Cryptographic controls;

General Terms: Algorithms, Design, Experimentation, Performance, Security

Keywords: Path enforcement, consent, NetFPGA, default-off

1 INTRODUCTION

The current Internet provides a simple delivery mechanism: we put destination addresses in packets and launch them into the network. We leave the network to decide the path that our packets take and the intermediate providers that the path passes through. Even network operators have little control over the paths that packets take toward them, or after leaving them. There are times, however, when senders, receivers, and operators would prefer to control packets' paths—and be sure that their preferences are enforced.

For example, an enterprise might want the packets that it receives to pass through several services, such as an accounting service and a packet-cleaning service. Or a company might want fine-grained control over which providers carry which traffic between its branch offices, yet the network paths must respect the providers' pairwise business relationships. Or providers might want to make sure that they are carrying traffic only from friendly nations.

The functionality above does not exist in the Internet today, though there are proposals that address various aspects of the problem. However, there is no general-purpose mechanism that *enforces* these policies (short of allocating dedicated connections, which is expensive).

This paper tries to fill that void. We describe a new primitive that we call a PVM (Path Verification Mechanism). A PVM is a protocol and mechanism for *forwarding* (sending a packet to its next hop), as distinct from topology discovery and path selection, or *routing*. Enriching routing policy [8, 28, 33, 47, 57, 59, 64, 67, 68, 70] has received much attention. Our focus, in contrast, is enforcing those policies during forwarding, which has received less attention and which we view as complementary: if it is important that routing produce policy-compliant paths, then it is important that packets actually be forwarded along those paths. (Note that this is orthogonal to the problem of securing routing protocols [6, 16, 36–38, 40, 43, 70].)

A PVM provides two properties:

1. **Path Consent**: Before a communication, every entity on the path of the communication (including the sender and receiver) or a delegate of that entity consents to the use of the whole path, based on the entity's or the delegate's particular policy [61].
2. **Path Compliance**: On receiving a packet, every entity can verify (1) that it or its delegate had approved the packet's purported path, and (2) that all previous entities on the path have already forwarded the packet in the order specified by the path.

Realizing a PVM is a challenging technical problem: when a packet arrives at a node, how can the node be sure that the packet followed an approved path? Many "first-guess" solutions are ruled out by our target environment, which we assume is:

- **Adversarial**: Nodes may try to thwart the mechanism.
- **High-speed**: To work at Internet backbone speeds, a PVM cannot rely on per-packet public key operations (ruling out a signed log in every packet [18]), per-flow state in forwarders (which stymies fail-over), and ideally not even per-flow public key operations.

ACM CoNEXT 2011, December 6–9 2011, Tokyo, Japan.

- **Decentralized**: There is no authority that could configure every node per-flow (ruling out Ethane [19], which provides Path Consent but only inside an enterprise).

Our solution, **ICING**, has three key aspects. First, we develop a protocol whereby a node can tell, given a packet, whether the packet's stated path is approved and whether the prior nodes on that path actually handled the packet. The protocol condenses what appears to be quadratic information into linear space, without requiring a PKI or any significant configuration state; it composes cryptographic constructions (e.g., MAC aggregation [39]) with security techniques (e.g., self-certifying names [7, 49, 51]). Second, the protocol does not require each entity to approve every flow; it allows an entity to selectively delegate its authority (for instance, a provider delegates to its customers, who delegate to their customers, etc.). Third, since we aim for high speeds, and since processing protocols cost-effectively at backbone speeds requires hardware, we apply careful engineering to make our design amenable to implementation in hardware.[1]

To validate our design, we have implemented ICING in hardware, on NetFPGA [2]. This implementation achieves a minimum throughput of 3.2 Gbits/s at an equivalent gate cost of 54% more than a simple IP forwarder running at 4 Gbits/s; thus, per unit of throughput, our ICING implementation costs 93% more than IP. Our evaluation further suggests that, if implemented on a custom ASIC (as in a modern router), ICING would scale to backbone speeds at acceptable cost (§5–§6).

We have also implemented an overlay, ICING-ON, which uses ICING for forwarding and is incrementally deployable.

We treat related work later in the paper (§7). For now we just note that some of ICING's components are reminiscent of or inspired by prior mechanisms [7, 10, 11, 13, 17, 19, 24, 28–31, 49, 51, 57, 58, 65, 69], and ICING can enforce many previously proposed policies. However, we know of only one proposal that offers both Path Consent and Path Compliance [18] and none that offers these two properties in an environment that is adversarial, high-speed, and federated. More specifically, this paper's contributions are:

- A new networking primitive, the PVM, and the design of an efficient PVM, called ICING.
- A fast and affordable hardware implementation of ICING, and a software implementation in an overlay.
- A packet header format and an optimized construction that demonstrate the plausibility of rich per-packet cryptography at network line rates.

Motivating scenarios and caveats We gave several motivating scenarios above (controlling inbound packet paths for packet cleaning, etc.), and Section 4 goes into more detail. Here, we just note that a PVM complements the many works on routing policy [8, 28, 33, 47, 57, 59, 64, 67, 68, 70] by verifying that packets take the routes expressed by policy.

We now give three caveats. First, a PVM lets principals restrict paths to specific providers but does not guarantee that those providers are trustworthy. In particular, to guarantee that a packet *hasn't* transited a particular network requires trusting every provider on the path not to tunnel through that network or copy packets there. Second, many of the applications that we give can individually be addressed by special-purpose mechanisms—at the expense of the other examples. A PVM, in contrast, provides these functions together.

Third, we are not trying to convince the reader that a PVM is advantageous on all axes; indeed, our PVM has disadvantages (including complexity and some cost). However, before this study, researchers did not know whether it was possible to build a PVM, let alone how to build one or what it would cost. Indeed, a routinely deferred item in papers on routing security is "data plane security", and we found that realizing a PVM required careful design.

2 OVERVIEW OF ICING

We now describe ICING at a high level, including its threat model, deferring design details to Section 3.

2.1 Architecture and components

ICING is a packet forwarding mechanism that allows a forwarder to verify that a packet is following its pre-approved path—and hence that packet forwarding is complying with the path policies of all entities on the path—before letting the packet consume further network resources. An ICING network comprises ICING ***nodes***, which may include end-hosts.

There are two natural deployment scenarios for ICING: (1) at layer 3 (the network layer) and (2) in an overlay. In the first case, transit providers would deploy ICING nodes at the ingress of their networks. Internal forwarders need not run ICING, just as today's internal forwarders may implement a different protocol from that of the border router. This scenario is default-off [12, 22, 34, 46, 65, 69], which necessitates careful bootstrapping; the details are beyond this paper's scope but can be found in [52, §4], and similar problems have been treated elsewhere [9, 46, 69]. In the overlay case, the ICING nodes are ***waypoints*** interconnected by the regular IP network. In this case, ICING's guarantees refer to the waypoint-level paths, not IP-level paths. This section will be agnostic about the deployment scenario.

To communicate with a receiver, the sender first chooses a ***path*** of nodes. How senders find paths depends on the scenario; the sender might query DNS to get a path, purchase access to a remote ISP via its Web site, statically configure paths, etc. This paper mostly assumes that the sender has candidate paths in hand, as ICING is concerned with forwarding and is orthogonal to path retrieval (routing).

Figure 1 summarizes ICING's forwarding. For each node on the path, the sender requests from the node's provider a ***Proof of Consent*** (***PoC***), which certifies the provider's consent to carry packets along that path. The PoC is cre-

[1] Recent work [23] has examined high-speed software forwarders. However, backbone forwarders seem likely to continue to require dedicated hardware for the medium-term future. Besides, our design, being parallelizable, can run efficiently on multicore general-purpose processors.

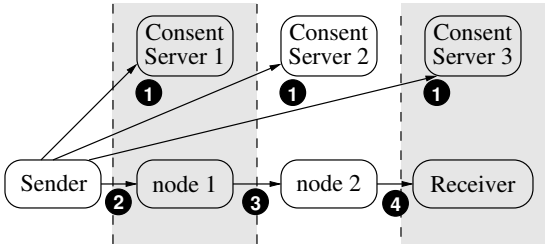

Figure 1—Forwarding in ICING. ❶ The sender logically gets PoCs from the consent servers of all nodes on the path (a consent server can delegate PoC-issuing, making this step lightweight). ❷ The sender creates and sends the packet to the first ICING node, having used the PoCs to construct tokens that ❸ each forwarder verifies and transforms for its successors until ❹ it arrives at the receiver.

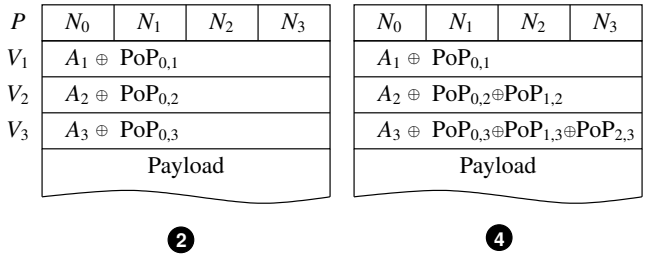

Figure 2—Simplified ICING packet at steps ❷ and ❹ from Figure 1. Two crucial header fields are the path (P) and the verifiers (V_j's). The sender (N_0) initializes the verifiers with path authenticators (A_j's) derived from the PoCs and the packet content. Each node N_i checks its own verifier (V_i) and updates the verifiers for downstream nodes (V_j for $j > i$) to prove that it passed the packet. $\text{PoP}_{i,j}$ is a proof to N_j that N_i has carried the packet. $\oplus$ represents XOR.

ated by a ***consent server*** owned by the provider or acting on its behalf. A consent server is a general-purpose server separate from a provider's forwarding nodes; as articulated in [17, 19, 30, 31], this separation lets policies be flexible, fine-grained, and evolvable. We note that a provider's nodes and consent servers trust each other (they are a trust domain).

As a packet travels through the network, each node verifies that the packet is following its approved path. This job decomposes into three tasks: (1) the node checks that the path is approved; (2) it checks that the path has been followed so far; and (3) it proves to downstream nodes that it has seen the packet. To perform these functions, ICING nodes use the construction depicted in Figure 2. The construction relies on PoCs and on ***Proofs of Provenance***, or ***PoPs***. PoPs allow upstream nodes to prove to downstream nodes that they carried the packet. These proofs require pairwise ***PoP keys***, but these keys do not require significant configuration or coordination, as nodes derive keys on demand from each other's IDs.

2.2 Goals and non-goals

ICING seeks to provide a PVM's two properties, Path Consent and Path Compliance. We refine these properties into the following requirements for ICING:

- **Delegation**: A consent server must be able to delegate its path approval function.
- **Path Consent**: When a node receives a packet with path P, it must be able to verify that its consent server, or a delegate, approved P.
- **Path Compliance**: When a node N_i with index i in path P receives a packet with path P, the node must be able to verify that the packet was sent by the purported sender (index 0) and has been forwarded by each of the nodes at indices $1, 2, \ldots, i-1$, in that order.

ICING is designed to meet the above requirements while being amenable to an affordable high-speed hardware implementation and while not requiring a central authority, PKI, or significant configuration state. Our threat model, which is strongly adversarial, gives us further constraints. We describe this model in the next subsection.

There are several functions that ICING is *not* designed for (these are either seemingly infeasible or outside the scope of our problem of path enforcement):

The statement of Path Compliance does not guarantee a packet's future. After a packet departs a node, any downstream node can send it anywhere. It seems extremely hard to *prevent* such misbehavior. However, ICING can *contain* it: honest nodes drop packets that do not contain a proof of having passed through every prior node on the path.

ICING *nodes can copy packets and send them elsewhere, or pass packets through hidden nodes.* This, too, seems very hard to prevent in a federated environment. However, unlike in the status quo, ICING senders and receivers can restrict the path to providers that they trust not to leak their packets. (This choice is an alternative to onion routing, which is not always feasible [21].)

ICING *does not try to provide authenticated information about the location of silent errors or failures on the path.* An ICING node can signal an error to the sender (Section 3.4); however, the sender cannot discover the location of a fault if none of the nodes on the path initiates this signal.

ICING *does not provide information about whether a packet received any contracted-for services at a node.* For instance, if a sender chooses to send a packet through a particular node because the node advertised a virus-scanning service, the receiver can check if the packet was forwarded through the node but not that it was actually scanned.

ICING *makes a binary decision about whether a path is acceptable; it does not regulate the* amount *of traffic sent along a path, or associated to a PoC.* Other work [69] has shown how to perform such accounting with little forwarder state, and ICING could be extended with this technique.

2.3 Threat model

Machines that obey the protocol we term *honest*. We assume that some providers, nodes (including end-hosts), and consent servers are not honest and specifically that they are controlled by attackers. These machines can engage in Byzantine behavior that deviates arbitrarily from ICING's specified packet handling. For instance, the attacker can send arbitrary packets or try to flood links to which it connects. The attacker can also observe legitimate data packets that pass through

it. We make no assumptions about how malicious nodes are implemented: they may connect to one another and be controlled by a single attacker, or they may collude, potentially bracketing honest nodes on paths. Furthermore, even honest machines may give service to malicious parties; for instance, a consent server can grant PoCs to an attacker.

The attacker tries to make ICING fail some of its goals (Section 2.2), for instance by trying to abuse the delegation mechanism, or trying to make an honest node N_i accept a packet whose path was not approved by N_i's consent server (or a delegate), or whose actual path skipped some of the honest nodes upstream of N_i in the approved path.

2.4 Naming

Each ICING node assigns itself an identifier, called a ***node ID***, that is a unique public key. The node keeps secret the private key. The identifier space is large enough to make the probability of a collision negligible. With such *self-certifying names* [7, 49, 51], a central naming authority or PKI is not needed. This fits the Internet's federated structure.

A path is a list of $\langle \text{nodeID}, \text{tag} \rangle$ pairs. The ***tag*** identifies a specific set of local actions that a node performs on packets with this tag. For example, a tag can describe a priority level for queuing, identify a customer to bill, select an output link, request virus-scanning services, or specify a combination of these. It can be thought of as a generalized MPLS label [25] (and shares some functionality with the vnode mechanism in [28]). The provider conveys the particular meaning of a tag on a node to the users of that tag through out-of-band means, such as an agreement with the user or a Web page.

2.5 Proofs of consent (PoCs)

After a sender has determined a path, it contacts a consent server for each node N on the path to obtain PoCs for that path. Each consent server is preconfigured with its provider's policy, so it can check whether paths are compliant. To aid its checks, the consent server may use external information (billing, authentication, etc.). If the check passes, the consent server creates a PoC and returns it to the sender.

For a node N, the PoC is a cryptographic token indicating that N's provider consents to the full path, including N's tag in that path. This token is computed under a ***tag key***, which is unique to the $\langle N, \text{tag} \rangle$ pair. This key is known to both the consent server and the local node.

Consent serving is flexible. A provider with multiple ICING nodes can deploy a single consent server. It can also delegate the ability to create PoCs for a particular node and tag (by divulging that tag's key to the delegate) or disintermediate itself altogether (by disclosing all of its tag keys).

2.6 Packet creation and proofs of provenance (PoPs)

The sender uses the PoCs it obtains to construct the packet header; the construction is such that when a packet arrives at node N, N can tell whether the sender held a PoC issued by N's consent server. The sender also computes PoPs for each of the nodes on the path. These PoPs prove to the nodes that the sender created the packet; each PoP includes a MAC of the packet under a shared symmetric PoP key.

These shared PoP keys do not require the network to be configured with pairwise keys; ICING *nodes derive PoP keys on demand.* By using non-interactive Diffie Hellman key exchange, an ICING node (such as the sender) derives the PoP key that it shares with any other node N from its own private key and N's node ID (which is a public key). Nodes cache PoP keys after deriving them.

2.7 Packet processing: Verification and forwarding

Each node that receives the packet does the following:

1. It computes the PoC from the path and the tag key that the node shares with its consent server;
2. For each upstream node in the path:
   (a) The node derives the PoP key it shares with this upstream node (using the upstream's ID, from the path);
   (b) It computes the PoP (a MAC) under the PoP key.
3. It checks that the PoC and PoPs are correct.

The PoPs computed in step 2 prove to the node that the packet has passed through all the upstream nodes. If the PoC is correct and the PoPs are all correct, then the packet has been following an approved path. Otherwise, the node drops the packet. If the checks pass, the node proves to downstream nodes that it has seen the packet, as follows:

4. For each downstream node in the path:
   (a) It derives the PoP key that it shares with this downstream node (from the downstream's ID, in the path);
   (b) It computes the PoP under this PoP key.
5. It inserts these PoPs into the header.
6. It forwards the packet to the next node.

As so far described, packet header size appears quadratic in the length of the path. However, the header size is in fact *linear* in path length. As illustrated in Figure 2, the PoC and the PoPs that a node inspects in steps 1–3 above are XORed together in an aggregate MAC [39]. The cost of this reduction in space is that a node cannot tell, in the case of tampering, which of the PoPs or PoC is incorrect. However, the probability of successful tampering is unaffected [39].

3 DESIGN DETAILS OF ICING

This section details ICING's design, which aims to meet the requirements stated in Section 2.2. Figure 3 describes the notation that we use throughout our design discussion and our pseudocode, while Figure 4 summarizes the secret cryptographic material used in ICING.

ICING's packet format is shown in Figure 5. Each packet includes three types of information for every node in its path other than the sender. The first is per-node information: a node ID, N_i, and a corresponding tag, tag_i (Section 2.4). The second is the Proof of Consent (PoC) showing that the tag's owner authorized the path (Section 2.5). The third is the Proofs of Provenance (PoP), which allow a

M	$\{vers, counter, proto, path\text{-}len, pkt\text{-}len, error\text{-}path\text{-}idx, payload\}$. A packet's static contents.
P	$\langle N_0{:}tag_0, N_1{:}tag_1, \ldots, N_n{:}tag_n\rangle$. A packet's path: a list of node identifiers and corresponding tags.
N_i	A node's identifier: a public key.
x_i	A node's private key: satisfies $N_i = g^{x_i}$.
tag_i	The tag corresponding to node N_i on path P: an opaque 32-bit string.
m_{N_i}	A node's *master tag key*: used in a key-derivation function (GET-TAG-KEY) to associate key material (*tag keys*) to any tag.
$m_{N_i:t/p}$	The *p-bit-prefix key* for tag t at node N_i: an intermediate key, created by GET-TAG-KEY, that enables calculation of tag keys at node N_i for any tag t' with the same p-bit prefix as t's. Note that $m_{N_i:t/0}$ equals m_{N_i} for any t.
$s_{N_i:tag_i}$	The *tag key* : used by a consent server for node N_i to create a PoC for a path that includes $N_i{:}tag_i$. Amounts to $m_{N_i:tag_i/32}$.
PoC_i	$(\text{PoC}_i.expire, \text{PoC}_i.proof)$. *Proof of Consent* to path P by node N_i.
$\text{PoC}_i.expire$	A PoC's 16-bit expiration time indicator.
$\text{PoC}_i.proof$	$\text{VPRF}(s_{N_i:tag_i}, P \parallel \text{PoC}_i.expire)$.
A_i	$\text{PRF-96}(\text{PoC}_i.proof, -1 \parallel \text{HASH}(P \parallel M))$. A packet's *path authenticator* for node N_i.
$k_{i,j} (= k_{j,i})$	$\text{NIDH}(x_i, N_i, N_j) (= \text{NIDH}(x_j, N_j, N_i))$. The *PoP key*: a symmetric key shared by nodes N_i and N_j, used for PoP computations. Soft state, derivable from nodes' identifiers.
$\text{PoP}_{i,j}$	$\text{PRF-96}(k_{i,j}, i \parallel \text{HASH}(P \parallel M))$. *Proof of Provenance* designated for N_j: A MAC by which N_i attests that it had approved a packet's path and handled it accordingly.
V	$\langle V_1, \ldots, V_n\rangle$. A packet's *verifier-vector*.
V_i	$(V_i.expire, V_i.proofs, V_i.hardener)$.
$V_i.expire$	Same as $\text{PoC}_i.expire$.
$V_i.proofs$	$A_i \oplus \text{PoP}_{0,i} \oplus \ldots \oplus \text{PoP}_{i-1,i}$. Aggregate MAC by which N_i checks out A_i (and hence PoC_i), as well as $\text{PoP}_{j,i}$, for $j < i$.
$V_i.hardener$	$\text{PRF-32}(\text{PoC}_i.proof, 0 \parallel \text{HASH}(P \parallel M))$. Hardens forwarder slow path against DoS.
$\text{NIDH}(x_i, N_i, N_j)$	$\text{HASH2}(\text{SORT}(N_i, N_j) \parallel N_j^{x_i})$. (Hashed) Non-interactive DiffieHellman key exchange.
$\text{VPRF}(s, d)$	A keyed function that maps variable-length data d to 128-bit pseudorandom outputs. The current implementation uses PMAC [15].
$\text{PRF}(k, d)$	A keyed function that maps 256-bit data to 128-bit pseudorandom outputs. The current implementation uses an optimized AES-CBC-MAC.
$\text{PRF-96}(k, d)$	First 12 bytes of $\text{PRF}(k, d)$. Suitable as a 128-bit message authentication code for d.
$\text{PRF-32}(k, d)$	Last 4 bytes of $\text{PRF}(k, d)$.
$\text{HASH}(d)$	A collision-resistant hash function that maps variable-length data d to a 248-bit digest. Based on CHI [35]. Future versions of ICING will use the final SHA-3.
$\text{HASH2}(d)$	A collision-resistant hash function that maps variable-length data d to a 128-bit digest. Based on SHA-1. Future versions of ICING will use the final SHA-3.

Figure 3—Symbols and notation used in the pseudocode.

	node i ($i \geq 0$)	consent server i ($i > 0$)	delegate of node i	sender
x_i	x			
$k_{i,j}$	o			
m_{N_i}	x	x		
$m_{N_i:t/p}$	o	o	x	
$s_{N_i:tag_i}$	o	o	o	
$\text{PoC}_i.proof$	o	o	o	x

Figure 4—Cryptographic keys in ICING (rows), and holders of these keys (columns). The key material is relative to the i-th entry in a packet's path (which is the sender, if $i = 0$). x denotes a key that the entity is given; o denotes a key that the entity can derive.

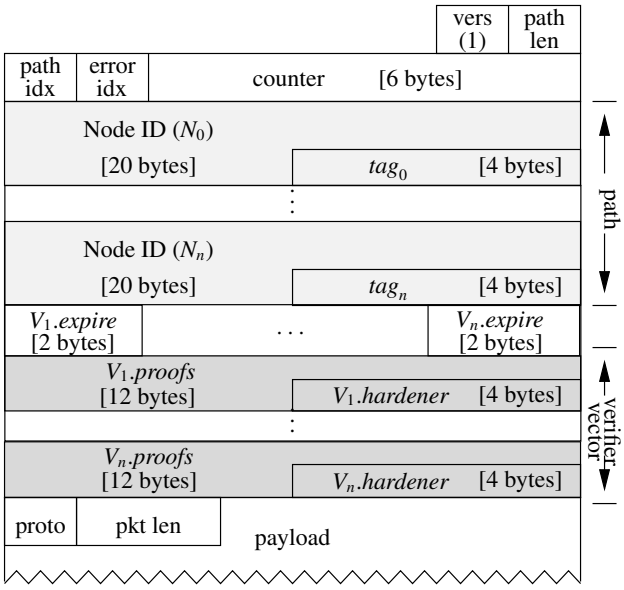

Figure 5—ICING header. The per-node overhead is 42 bytes; we estimate the total overhead on the avg. packet to be $< 25\%$ (§6.1).

node to verify that every previous node has approved the path and forwarded the specific packet (Sections 2.6–2.7). The PoC—more precisely, an authenticator derived from it—and PoPs are aggregated into a constant-length verifier ($V_i.proof$). PoCs and PoPs allow ICING to meet its requirements of Path Consent and Path Compliance. We discuss Delegation later.

Because packets carry node IDs, and because node IDs are public keys, our design needs small public keys. Thus, we use elliptic curve cryptography (ECC): every node ID, N_i, is a point on NIST's B-163 binary-field elliptic curve group [5], which gives roughly 80-bit security, similar to 1024-bit RSA keys [5]. The private key, x_i, corresponding to N_i, is the 163-bit number such that $N_i = g^{x_i}$ in B-163, where g is the group generator. To make the approach amenable to hardware implementation, we reduce the representation of N_i from 163 to 160 bits.[2] We do so by requiring the top three bits to equal

[2]Each version of ICING hard-codes particular cryptographic algorithms. As algorithms are later broken or require longer key lengths, we expect the choices of algorithms and key lengths to change but not the primitives (pseudo-random functions (PRFs), collision-resistant hash functions, etc.)

```
1: function INITIALIZE(pkt, PoC_1, ..., PoC_n)
2:     P, M, V = pkt.P, pkt.M, pkt.V
3:     H = HASH(P ‖ M)
4:     for 1 ≤ j ≤ n do
5:         Set V_j.expire = PoC_j.expire
6:         A_j = PRF-96(PoC_j.proof, −1 ‖ H)
7:         Set V_j.proofs = A_j
8:         Set V_j.hardener = PRF-32(PoC_j.proof, 0 ‖ H)
9:         N_j = j-th node in P
10:        k_{0,j} = DHCache[N_j] or NIDH(x_0, N_0, N_j)
11:        PoP_{0,j} = PRF-96(k_{0,j}, 0 ‖ H)
12:        Set V_j.proofs = V_j.proofs ⊕ PoP_{0,j}
```

Figure 6—Pseudocode for packet initialization. The sender initializes the verifiers before sending the packet to the first node.

a cryptographic hash of the lower 160 bits; this does not diminish the strength of the keys, though it increases expected key generation time by a factor of 8.

3.1 Creating a packet

Once a sender assembles a path P (a process largely orthogonal to ICING, as discussed in Section 2.1), it must obtain PoCs for each N_i:tag_i in P. To do so, it contacts N_i's consent server. The PoC consists of a 16-bit expiration time indicator, $\text{PoC}_i.expire$, and a cryptographic token, $\text{PoC}_i.proof = \text{vPRF}(s_{N_i:tag_i}, P \parallel \text{PoC}_i.expire)$. $s_{N_i:tag_i}$ is a tag-specific secret key shared between node N_i and its consent server. Because managing keys separately for their 2^{32} tags would be cumbersome for a node and its consent server, they instead share one master tag key, m_{N_i}, that pseudorandomly generates many tag keys. This process is encapsulated by GET-TAG-KEY(m_{N_i}, tag_i), and we unpack it below, in Section 3.3. The master tag key m_{N_i} is changed periodically to prevent cryptanalytic attacks; a PoC's *expire* indicator is relative to the time when m_{N_i} was last re-keyed. The granularity of each *expire* time-unit is a per-node parameter; we envision typical choices between a few seconds and several minutes.

Given these PoCs, the sender calls INITIALIZE (Figure 6), which creates a verifier V_j for each other node j on the path. The sender initializes V_j with A_j (which binds the PoC_j to the packet contents) and $\text{PoP}_{0,j}$; given this V_j, a downstream node can verify that the packet's path is approved by its consent server and that the packet has been created by the packet's purported sender.

3.2 Forwarding and receiving a packet

On receiving a packet, a node N_i with index i in the packet's path processes it according to the pseudocode in Figure 7. The node must ensure that Path Consent and Path Compliance (Section 2.2) are met for *each* packet that it passes. (Delegation will be discussed in Section 3.3.)

First, for Path Consent, N_i must verify that the PoC implicit in the packet, PoC_i, is correct. Second, for Path Compliance, the node must verify the PoPs created by upstream nodes $N_0, \ldots, N_{i-1}$ (that is, it must verify $\text{PoP}_{j,i}$ for $j < i$). The node executes both checks by validating the verifier V_i. To do so, it derives an expected verifier V'_i, which requires deriving the expected PoC'_i (based on the path and the relevant tag key, *cf.* Figure 7, lines 9–10), the expected A'_i, and the expected PoPs. If V'_i does not match the verifier in the packet (V_i), the packet is dropped (Figure 7, lines 11–24). To perform its required duty with respect to Path Compliance, the node modifies the verifiers for downstream nodes (V_j for $j \geq i$) by XORing V_j with $\text{PoP}_{i,j}$ (Figure 7, lines 26–30).

Computing PoPs may require deriving PoP keys $k_{i,j}$ ($= k_{j,i}$). Being resource-intensive (it takes about 4 msec in our implementation), this calculation happens on a node's ***slow path*** (meaning on a general-purpose processor). An attacker may try to attack a node's slow path by sending many pack-

```
1: function RECEIVE(pkt)
2:     P, M, V = pkt.P, pkt.M, pkt.V
3:     i = pkt.path-idx
4:     N_i:tag_i = i-th entry in P
5:     T = current time
6:     PoC'_i.expire = V_i.expire
7:     if PoC'_i.expire < T or N_i ≠ my node ID then
8:         Drop pkt
9:     s_{N_i:tag_i} = GET-TAG-KEY(m_{N_i}, tag_i)
10:    PoC'_i.proof = vPRF(s_{N_i:tag_i}, P ‖ PoC'_i.expire)
11:    H = HASH(P ‖ M)
12:    A'_i = PRF-96(PoC'_i.proof, −1 ‖ H)
13:    V'_i.proofs = A'_i
14:    V'_i.hardener = PRF-32(PoC'_i.proof, 0 ‖ H)
15:    if V'_i.hardener ≠ V_i.hardener then
16:        Drop pkt
17:    // verify upstream PoPs (check Path Compliance)
18:    for 0 ≤ j < i do
19:        N_j = j-th node in P
20:        k_{j,i} = DHCache[N_j] or NIDH(x_i, N_j, N_i)
21:        PoP_{j,i} = PRF-96(k_{j,i}, j ‖ H)
22:        V'_i.proofs = V'_i.proofs ⊕ PoP_{j,i}
23:    if V'_i.proofs ≠ V_i.proofs then
24:        Drop pkt
25:    // verify downstream PoPs (prove Path Compliance)
26:    for i ≤ j ≤ n do
27:        N_j = j-th node in P
28:        k_{i,j} = DHCache[N_j] or NIDH(x_i, N_i, N_j)
29:        PoP_{i,j} = PRF-96(k_{i,j}, i ‖ H)
30:        Set V_j.proofs = V_j.proofs ⊕ PoP_{i,j}
31:    Set pkt.path-idx = i + 1
32:    Add all calculated k_{x,y} to DHCache
33:    Perform any special handling prescribed by tag_i
34:    Transmit pkt to next node (or accept if destination)
```

Figure 7—Pseudocode for packet forwarding. The node validates the packet and transforms verifier entries before honoring the tag specified in the packet's header and sending the packet to the next node. Note that P is 0-indexed and V is 1-indexed. For brevity, generating and handling error packets (§3.4) is not depicted.

and hence not ICING's design. To smooth the changeover, hardware would support old and new versions (as with IPv4 to IPv6).

ets with invented node IDs. To defend against such an attack, the node requires a valid ***hardener*** (V_i.*hardener*) in the packet, which it checks on the *fast* path (Figure 7, lines 14–16). V_i.*hardener* is only 32 bits, so it does not fully rule out such attacks, but it decreases their effectiveness by a factor of 2^{32}, which is sufficient to avoid denial-of-service.

3.3 Deriving tag keys and controlled delegation

We now detail GET-TAG-KEY(m_{N_i}, tag_i). Let t/p denote the p-bit prefix of tag t, and define $m_{N_i:t/p}$ to be the corresponding p-bit ***prefix key***. We take $m_{N_i:t/0} = m_{N_i}$ for any tag t. Then, GET-TAG-KEY(m_{N_i}, tag_i) iteratively computes $m_{N_i:tag_i/p} = \text{PRF}(m_{N_i:tag_i/(p-1)}, tag_i/p)$ to get $m_{N_i:tag_i/32} = s_{N_i:tag_i}$, the tag key associated to N_i:tag_i. This approach is inspired by a technique in [57].

As so far described, this technique derives $s_{N_i:tag_i}$ from $m_{N_i:tag_i/0} = m_{N_i}$ using 32 serial rounds of PRF, which is too many for high speed packet processing. Three modifications that significantly reduce costs are described in [52, §2.3.1].

Controlled delegation. Given the above approach to tag key derivation, ***tag prefix delegation*** is easy to implement. To delegate the tag block with prefix t/p (i.e., 2^{32-p} tags), the node's provider shares $m_{N_i:t/p}$. The delegate can further sub-delegate tags by sharing $m_{N_i:t/k}$, where $k \geq p$. Delegation lets a provider give customers control over particular tags on the provider's nodes. A customer with such control can, with no provider intervention, act as a consent server on behalf of the provider (creating PoCs for its own traffic if it is an end-host or for its customers if it is itself a provider) or give its customers their own tag keys (to disintermediate itself).

Expiration and revocation. The PoC.*expire* timestamp allows consent servers to mint time-limited PoCs. This requires that a consent server and its nodes be loosely time synchronized (e.g., via NTP), within a tolerance commensurate to the *expire*-granularity adopted at the node.

The master tag key m_{N_i} and other prefix keys $m_{N_i:t/p}$ are changed periodically to guard against chosen-message cryptanalytic attacks and to prevent an old timestamp that has wrapped from appearing valid.

To revoke PoCs before the expiration interval, a provider can (1) change m_{N_i} at the node and consent server; or (2) change only a prefix key (or tag key), no longer deriving it from m_{N_i}. The latter option requires a small, longest-prefix-match override table mapping prefixes to prefix keys; the details are described in [52, §2.3.1].

3.4 Signaling errors and failures

Because ICING packets are source-routed, a network using ICING needs to report errors and other failures back to the sender so that the sender can use a different path if necessary. Note that a sender can hold pre-approved backup paths, so failures need not require the sender to obtain new paths.

On encountering an error, an ICING node generates an ICING error packet that travels *backward* along a given path toward the sender. A slight complication is Path Consent: for some nodes, consenting to a path's forward direction may not imply consent to carry error packets in reverse. In that case, the sender has to rely on end-to-end failure detection.

For nodes that do consent to carry error packets, ICING handles errors as follows. To create an error packet, a node sets the ***error index*** field in the header to the current index and replaces the payload with the original packet's hash, followed by optional error-specific information (analogous to ICMP error code and data). A node recognizes packets with non-zero error index fields as error packets and handles them differently: most importantly, the node forwards such packets to the previous node (rather than the next) and decrements (rather than increments) the path index field.

The V_i in an error packet contain all the forward-direction PoPs from the original packet, in addition to PoPs for the error packet itself. Because the error payload begins with the original packet's hash, nodes can verify these forward-direction PoPs despite not having the original packet. In particular, node i drops an error packet unless V_i.*proofs* includes a PoP under $k_{i,i}$. This ensures that a node will not forward an error packet if it did not previously forward the original.

3.5 Attacks

Attacks against the verification algorithm. The algorithm guards against the following attacks.

- Using incorrect or expired PoCs: This attack fails because each node checks the expiry and recalculates its expected PoC (Figure 7, lines 7–10).
- Skipping an honest node i: When the packet is received at honest node j downstream of node i, V_j will lack $\text{PoP}_{i,j}$ and will be flagged (Figure 7, lines 11–24).
- Flooding a node's slow path: The attack is mitigated because the node checks V_i.*hardener* before calculating any PoP keys (Figure 7, lines 14–16).

Attacks that compromise secrets. How should a node handle the inevitable compromises of its cryptographic material (Figure 4)? We have discussed PoC revocation (Section 3.3): a node changes prefix or tag keys ($m_{N_i:t/p}$). If an $m_{N_i:t/p}$ itself is compromised, a node can simply change it. A more serious concern is the compromise of a private key, x_i, or any derived PoP key, $k_{i,j}$. In that case, the node must generate a fresh public/private key pair so must also change its node ID. This requirement is inconvenient but not disastrous; other entities that express policy in terms of the renamed node must be notified about the change.[3]

Attacks that attempt packet replay. An attacker who has observed a valid packet may inject a duplicate copy along a suffix of a path. At low rates, such attacks are not problematic: the layer using ICING presumably handles duplicates anyway. Meanwhile, an attack that aggressively floods using a few packets can be defeated by a modestly sized replay cache at each node; this cache would store $\langle \text{PoC}, \text{counter} \rangle$ pairs (the counter, from the packet header field, is chosen by the sender to be unique over the flow). A difficult case is if

[3]This inconvenience could be mitigated by modifying ICING to add a level of indirection in naming.

the attacker can amass packets from many flows within a single PoC validity window and then replay each packet a small number of times. Defending against this case is future work; it may require both reducing the PoC validity window and compressing the information in the replay cache.

Attacks on availability. What if an attacker overwhelms a consent server [9, 69]? One option is to locate the consent server at a high-bandwidth denial-of-service mitigator (e.g., [56]). Another option, if the receiver already knows who should be allowed to reach it (for example, employees or customers), is to give these senders their own tag keys $s_{N_i:tag_i}$ so that they can mint their own PoCs without the consent server. Third, if PoC requests travel in ICING packets, then ICING's mechanisms themselves provide a foundation for defense. These mechanisms apply not just to an overloaded consent server but also to any receiver wishing not to hear from a sender. For instance, if senders can be identified at a useful granularity (e.g., "employees", "paying customers", "unknown senders who solved a CAPTCHA"), then the victim can assign each category to a different tag. When overloaded, the victim deprioritizes categories by not renewing expired PoCs for their tags; downgrading service to them; or, in an emergency, changing tag keys. If senders cannot be assigned to categories, we can follow TVA [69], ensuring roughly fair bandwidth consumption among senders by applying Hierarchical Fair Queueing [14] to a packet's path. While attackers can weaken this defense under TVA by faking path identifiers, ICING does not have this vulnerability.

Other attacks. Section 2.2 mentioned attacks that ICING does not defend against.

4 APPLICATIONS OF ICING

We first describe three applications of ICING and then say a few words about the interface to ICING.

First, ICING receivers can request services for incoming packets (e.g., outsourced intrusion detection service (IDS) or denial-of-service mitigation [56]) and then verify that received packets actually traversed the services. ICING also enables these services themselves to specify other intermediate services (e.g., the IDS can specify an accounting service that drops traffic for non-customers). Unlike previous work [62, 63], ICING provides integrity and authentication in the forwarding mechanism, even for intermediaries.

A second application is enforcing routing policy. Today, Internet providers run a policy routing protocol (BGP). Yet, forwarding in the current Internet can undermine policy routing: packets can (and do [48]) deviate from the paths determined by BGP [16, 26]. Under ICING, if a consent server agrees only to BGP-compliant paths, then its nodes will carry only packets that follow such paths.

Finally, ICING lets an enterprise protect itself against a flooding attack, by allowing it to control traffic toward it in a remote provider's network. The enterprise purchases delegated tags from a provider and mints PoCs for those tags on behalf of the provider only for authenticated senders.

Interface to ICING. We have implemented ICING both in an overlay network (called ICING-ON [50], [52, §3]) and at layer-3 (see [52, §4]) and have deployed ICING-ON on virtual machines in Amazon's EC2. We have no space for details so here just say a few words. These implementations demonstrate how bootstrapping, path retrieval, and topology discovery work. For ICING-ON, bootstrapping is easier since senders can always use the underlying network to reach consent servers. At layer 3, there are some details regarding how senders can use the default-off network itself to get consent to request consent [9, 46, 69]. For path building, how can the sender build a path that has the approval of all nodes? In the overlay scenario, one can iteratively build a path by querying consent servers that return patterns of allowed paths. At layer 3, allowed patterns are distributed in a routing protocol.

5 IMPLEMENTATION

This section describes our implementation of the ICING node's hardware and software. Our prototype node accepts ICING packets carried in Ethernet frames and implements the algorithm in Figure 7. The implementation has a fast path that runs in hardware, and a slow path that is executed in software if a PoP key ($k_{i,j}$) is not cached in hardware or if an exception occurs. The fast path is implemented on the NetFPGA programmable hardware platform [2], which is a PCI card with 4 GigE ports, a field programmable gate array (FPGA), SRAM, and DRAM. The slow path, implemented in Click [42], 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 [60]. All of the node's software runs on Linux 2.6.25.

We have not yet implemented PoC expiry or the handling of error packets. However, we do not expect these features to change our evaluation, as reported in the next section.

The hardware image uses support modules from the NetFPGA project. We implemented the ICING-specific logic, including cryptographic modules. The forwarder uses 89% of the total FPGA logic area and has a total equivalent gate count (EGC) of 13.4M. (EGC estimates how many gates a design would use on an ASIC, as reported by the Xilinx ISE synthesis tool, v10.1.) The area breakdown is: 38% to the AES, CHI, and PMAC modules, 28% to all other ICING-specific logic, and 34% to the NetFPGA support modules.

By comparison, NetFPGA's reference IP router has an EGC of 8.7M and uses 50% of the total FPGA logic area.

6 EVALUATION

ICING introduces space and time overhead from per-packet cryptographic objects and operations. Our principal question in this section is whether these overheads are practical at the speeds of Internet backbone links. In this section, we assume that ICING is deployed at the network layer. We begin by estimating ICING's total space overhead (Section 6.1). Sections 6.2 and 6.3 present microbenchmarks of our prototype node and supporting software. In Section 6.4, we extrapolate

Average increase in packet overhead: 23.3%	§6.1
Throughput: 80-100% of IP on NetFPGA	§6.2
Normalized hardware cost: 193% of IP on NetGPA	§6.4

Figure 8—Summary of main evaluation results.

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

Figure 9—Machines for measuring ICING overhead.

from our results to assess ICING's future feasibility at Internet backbone scale. Our results are summarized in Figure 8.

Setup and parameters. Figure 9 lists the 3 machine classes that we use for evaluation. The NetFPGA is in the *slow* machine. Our experiments often vary packets' path lengths, path indices, and sizes; Figure 10 gives the fixed and variable parameters for these experiments.

6.1 Packet overhead

Relative to IP, ICING requires larger packet headers so would consume more bandwidth. We now roughly quantify this overhead. An ICING header includes 13 bytes that do not depend on the packet's path length (see Figure 5). 42 bytes are needed for each node in the path other than the sender: 24 bytes for the node ID and tag, $N_j{:}tag_j$, and 18 bytes for the verifier, V_j. For a packet whose path length is 5—a pessimistic estimate of the average provider-level path length from [40] and [7]—the header is 205 bytes. To translate this overhead to a total increase in bandwidth, we look at a sample trace from CAIDA [3]. The total number of packets observed for about 15 minutes was 37,571,701 with a total size of 28,475 MB. For each packet, ICING adds relative to an IP header (of 20 bytes) $205 - 20 = 185$ bytes (assuming path lengths have the same distribution across packet sizes). So the total increase in bandwidth for this dataset would be $37,571,701 \times 185/(28,475 \times 2^{20}) = 23.3\%$ relative to IP.

6.2 ICING hardware

We now measure the performance of the (fast path) hardware in our prototype ICING node, described in Section 5.

From Figure 7, one might expect the cost of processing a packet to depend on the path length because the work of verifying Path Compliance and proving it seems proportional to the path length. However, the results of the various PRF-96 operations are XORed, so they can be parallelized in a pipeline and thus removed from the critical path. The only other heavily serialized function in the design is the hash function (HASH), 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 path index and the verifiers V_j, we expect throughput to be lower when the V_j's 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.

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	—

Figure 10—Parameters used throughout experiments. Packet size includes header.

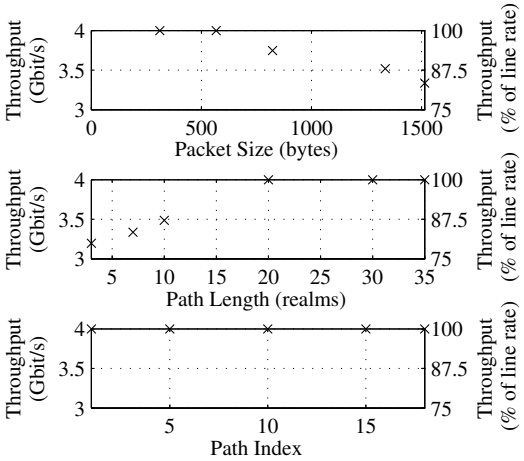

Figure 11—Avg. throughput as a function of packet size (Fig. 10, row 1), path length (Fig. 10, row 2), and path index (Fig. 10, row 3). Percentages are relative to maximum possible throughput on the NetFPGA. Standard deviations are less than 0.02% of the means. The forwarder's throughput is lowest for packets with large payloads and small path lengths: such packets send the most bits through the hash function, which is the bottleneck.

We measure our prototype's fast path throughput by connecting the four ports of an ICING node to a NetFPGA packet generator [20]. The ICING node loops ingress packets back to the generator, which measures the average bit rate. We take 5 10-second samples, using the parameters in Figure 10.

Figure 11 plots the measured throughput. (Note that we do not report goodput; instead we report packet header overhead in Section 6.1.) The minimum aggregate throughput is 3.2 Gbit/s. The path index has no effect on performance because it doesn't affect the number of bits hashed.

6.3 ICING software

We now measure the performance of the (slow path) software in our prototype ICING node. We also measure end-host and consent server operations. Figure 12 summarizes.

Shared key ($k_{i,j}$) derivation. A packet invokes our prototype's slow path when the hardware does not have the required shared keys cached (Figure 7). We measure the cost of deriving $k_{i,j}$ by running 3,000 iterations of the calculation function in a tight loop on the *slow* machine. On average, a single calculation takes 4 ms.

End-host. An end-host must also perform cryptographic operations: senders initialize all the verifier entries, and receivers validate and modify some of these entries. To understand these costs, we seek a linear function from path length

Action	Processing time	Throughput (1/Proc. time)
Calculate $k_{i,j}$	4 ms ($\sigma = .043$ ms)	250 keys/s
Generate PoC	$0.4x + 1.3$ μs	$2.6 \cdot 10^6/(x + 3.5)$ PoC/s
Create packet (w/c)	$2.6x + 40.1$ μs	$3.9 \cdot 10^5/(x + 15.4)$ pkt/s
Verify packet (w/c)	$2.6x + 24.4$ μs	$3.9 \cdot 10^5/(x + 9.5)$ pkt/s
Create packet (n/c)	$33796.1x - 32758.4$ μs	$29.6/(x - 0.9)$ pkt/s
Verify packet (n/c)	$34875.1x - 33647.1$ μs	$28.6/(x - 0.9)$ pkt/s

Figure 12—Processing time and throughput for software operations. x is the path length. Packet creation and verification costs are measured both with and without the use of cached shared keys (w/c and n/c resp.). For the last four rows, processing time is derived by linear regression, and $R^2 > 0.99$ in all three cases.

to processing time. To infer such a function, we vary path length per Figure 10, take packet size to be 1,514 bytes, and collect 1,000 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 Figure 12. Each entry in the path increases packet creation and verification times by 2.6 μs. For an average path length of 5, packet verification can be performed at 23K pkt/s.

Packet generation takes longer than verification because senders are so far unoptimized and compute HASH($P \parallel M$) 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 recomputing the local PoC.

Consent server. To measure the cost of generating PoCs in software, we run the calculation function in a tight loop, varying path length per Figure 10. We use the *fast* machine. We observe that cost is proportional to path length (Fig. 12), as expected from the definition of PoC.*proof* (§3). For a path length of 7, the consent server can generate ≈248k PoCs/s, within the range of rates handled by a fast DNS server.

6.4 Scaling

We now give a rough assessment of whether an ICING node could meet the demands of the Internet backbone.

Throughput and cost. In assessing whether ICING could scale to backbone speeds, our metric is *normalized cost*: it measures the hardware cost, reported as equivalent gate count, per unit of throughput. As a baseline, we consider a simple IP router on the NetFPGA. We obtain gate counts as described in Section 5.

Figure 13 summarizes the comparison. Using our normalized cost metric, our ICING forwarder is ∼93% more expensive than the NetFPGA IP router. However, the IP router is a pessimistic baseline because it is bare bones: it has only a 32-entry TCAM for longest-prefix matching (commercial routers have far more, and the TCAM is a big consumer of logic area), and it does not have the functionality of commercial-grade routers (packet filtering, tunneling, etc.). On the other hand, almost all of ICING's processing can be parallelized, so it seems that there is no fundamental obstacle

	NetFPGA ICING	NetFPGA IP
Min Throughput (Gbits/s)	3.2 (from §6.2)	4
(Eq.) Gate Count (Gates)	13.4M (from §5)	8.7M (from §5)
Normalized Cost (Gates/(Gbits/s))	4.2M	2.2M

Figure 13—Normalized costs of the NetFPGA ICING forwarder and the NetFPGA IP reference router.

to scaling ICING to backbone speeds (around 100 Gbits/s).

A rough estimate of the die size of an ASIC running ICING is in [52, §2.5.4]. In summary, by moving to an ASIC with today's technology (say 40 nm), our design would be approximately 300 times smaller and three times faster (10 Gbps). Then, we could "spend" some of the area saved replicating processing logic to reach 100 Gbit/s.

PoP key cache. An ICING node N_i stores a table of $(N_j, k_{i,j})$ pairs. Would this cache be too expensive? There are fewer than 40k advertised AS numbers, and the total is growing at less than 3.2k/year [1]. If we assume that each AS owns on average 10 nodes, the key cache would need to be approximately 400k entries. With 160-bit node IDs and 128-bit PoP keys, fitting all of N_i's PoP keys would require less than 14 MB, which is within today's SRAM capabilities [4]. Moreover, we believe that, to ease key management and policy configuration, providers would use the same node ID for many or all of their nodes. For further analysis of a nearly identical question, see [7, §4].

7 RELATED WORK

We divide related work into: (1) secure routing and secure forwarding, (2) related mechanisms, and (3) policy routing. As noted in the introduction, ICING has many debts, but no work that we are aware of offers Path Consent and Path Compliance under our three environmental constraints.

Secure routing and forwarding. Routing security [6, 16, 36–38, 40, 43, 70] ensures the authenticity and correctness of topology propagation and route computation. For instance, S-BGP [40] protects BGP against spurious messages. However, these works do not ensure that the resulting routes are actually used in packet forwarding, which is ICING's focus and which we view as complementary.

Rule-Based Forwarding (RBF) [55], in which end-hosts get some control over forwarding functionality, is also complementary to ICING: the forwarder-specific rules could be named and invoked by ICING's tags, but by itself RBF does not provide Path Consent or Compliance.

Like ICING, other works bind packets to their purported paths. However, they do not target an environment that is high-speed and adversarial and federated. For instance, [18, 54] use per-packet signatures and assume some centralization; [10, 54] require large configuration state in the network and a packet header quadratic in path length; Ethane [19] is centralized; and MPLS [58] is not robust to misbehavior (two nodes on the path can collude to skip a third; more generally there is no proof that a packet follows its path).

Other work on forwarding security is geared toward secrecy or isolation. Virtual Private Networks (VPNs) (ei-

ther point-to-point IPSec tunnels or isolated "slices" of a provider's network) provide neither Path Consent nor Path Compliance. Onion routing [21] provides anonymity by onion-encrypting a source-routed packet, with one layer of encryption removed at every hop. ICING is reminiscent of onion routing's per-hop cryptography, but onion routing uses encryption and decryption at every hop to provide secrecy and anonymity while ICING uses PRFs and MACs to enforce Path Consent and Path Compliance.

Localizing faults [13, 53, 71] and providing availability in a Byzantine network [10, 54] are also complementary. Other related works include denial-of-service (DoS) protection and allowing receivers to control which senders reach them [12, 22, 27, 34, 41, 46, 65, 66, 69]. These mechanisms can be enhanced to securely identify packets' senders [7, 44, 45], enabling accountability. ICING likewise provides support for DoS protection and verifies not only *sources* but also *paths*.

Related mechanisms. Aspects of ICING are inspired by prior works. PoCs generalize network capabilities [57, 65, 69] and Visas [24]. For instance, under Platypus [57], senders choose overlay paths, and providers associate packets with accountable entities. Under TVA [69], receivers control which senders reach them. However, none of these works provides Path Consent or Path Compliance.

Other related mechanisms are as follows. Node IDs resemble the self-certifying [49] ADs in AIP [7] and HIP [51]. PoPs are reminiscent of constructions in [10, 13]. However in those works, the number of PoP-like things in a packet is quadratic in path length, whereas an ICING packet carries a linear number of PoPs. Using Diffie-Hellman key exchanges for creating pairwise keys between nodes (in analogy with ICING's PoP keys) is proposed in [11, 29], but they suggest using a global directory or PKI, which ICING does not need (since node IDs are public keys). Also, ICING's hierarchical delegation generalizes a technique in Platypus [57], ICING's tags are reminiscent of Pathlets' vnodes [28] and MPLS labels [58], and expressing policy in general-purpose servers apart from forwarding hardware echoes [17, 19, 30, 31].

Policy routing. Like BGP, many works [8, 28, 32, 33, 47, 59, 62–64, 67, 68, 70] allow entities to express path preferences. Under NIRA [67], for instance, senders choose the path into the Internet core, and receivers choose the path out. Indeed, even default-off and filtering can be regarded as policy routing, in that the receiver exercises control over the first path component, the sender (e.g., [12, 22, 46, 65, 66, 69]).

These proposals are either orthogonal to ICING (because they concern only path computation, not enforcement), or else they incorporate a mechanism that enforces something less general than Path Consent and Path Compliance. For example, under Pathlets [28], senders choose paths, and providers specify policies based on the previous hop and a suffix of the path. But Pathlets does not provide verification that a path was actually followed. As another example, [68] gives provider-approved path diversity but does not allow senders or receivers to determine paths. ICING can be regarded as providing an enforcement mechanism that is general enough to enforce many of the policies in the works cited above. The next section discusses the price of this generality.

8 DISCUSSION

Does ICING restrict communication, as it empowers each node to enforce policy unilaterally? We note, first, that regardless of ICING, *any* carrier of a communication can exercise control over it. It is free to drop packets, deprioritize them, or corrupt them. Second, providers in the current Internet sometimes sever transit between each other to gain leverage in contract negotiations, effectively partitioning the Internet at users' expense (e.g., [72]). Under ICING, end-points get multiple options about paths and providers, which could create competition where today monopoly reigns.

The generality of ICING (§7) certainly has a price, as it is more expensive than many of the individual mechanisms that we have surveyed. However, we are solving a different problem than these other mechanisms: our goal is to provide a mechanism that can enforce a wide range of policies.

Moreover, it is possible to imagine cheaper PVMs than ICING, if we relax our requirements. As examples, if we assume a central authority (a reasonable assumption: the current Internet has IANA) that distributes a map from short identifiers to public keys, packets need not carry public keys, yielding smaller packet headers; or a PVM could check a fraction of the packets at a fraction of the cost; or if we allow per-flow state in nodes, then packets need not carry the full path, only a token that corresponds to it [19, 69].

We must also consider ICING's use complexity. At the network layer, there is some complexity from bootstrapping [52, §4]. In an overlay, the interface to ICING is relatively simple (though obviously more involved than IP). Moreover, the overlay scenario may provide a path to deployment.

But this is looking ahead. Looking back, our animating question was whether it was possible to design a feasible PVM and, if so, what it would cost. This paper has attempted to answer that question.

Acknowledgments

Naous's work was done at Stanford; Walfish's was done partially at Stanford and UCL. ICING has a long list of debts, beginning with one to Scott Shenker, who gave many and much-needed crucial expository suggestions. Dave Andersen, Tom Anderson, Russ Cox, Brandon Heller, and Jennifer Rexford read drafts carefully and kindly spent hours to help improve the presentation. We are likewise grateful to the CoNEXT reviewers, who read our work carefully. Comments and conversations that improved our presentation were given by Hari Balakrishnan, Andrew Blumberg, Dan Boneh, Mike Dahlin, Dawson Engler, Nick Feamster, Sanjam Garg, Sharon Goldberg, Mark Handley, Steve Keckler, Ramesh Johari, Josh Leners, Nick McKeown, Guru Parulkar, Ivo Popov, Srinath Setty, and Emmett Witchel. Hao Wu assisted in the evaluation of ICING. We thank our shepherd Laurent Mathy for his support and critiques. For their support, advocacy, and generosity, we thank Lorenzo Alvisi, Mike Dahlin, Nick McKeown, and Jonathan Smith.

This work was supported by NSF grants 1040083 (FIA),

1040784 (FIA), 1040190 (FIA), 0716806, 1052985, 0627112, and 1117679; by AFOSR grant FA9550-10-1-0073; by ONR grant N00014-09-10757; by the Stanford Clean Slate program; and by Intel Corporation, whose gift to Brad Karp supported Walfish and Mazières while they visited Karp at UCL in Autumn 2008.

Source for our hardware and software is available at: `http://www.cs.utexas.edu/icing`.

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] Packet traces from wide backbone. `http://mawi.wide.ad.jp/mawi/samplepoint-F/2011/201101231400.html`.

[4] Sync SRAMs overview. `http://www.cypress.com/?id=95`.

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

[6] W. Aiello, J. Ioannidis, and P. McDaniel. Origin authentication in interdomain routing. In *ACM CCS*, Oct. 2003.

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

[8] K. Argyraki and D. R. Cheriton. Loose source routing as a mechanism for traffic policies. In *SIGCOMM Wkshp. on Future Directions in Net. Arch.*, Sept. 2004.

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

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

[11] A. Aziz, M. Patterson, and G. Baehr. Simple key-management for Internet Protocol (SKIP). In *Proc of the INET Conference*, June 1995.

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

[13] B. Barak, S. Goldberg, and D. Xiao. Protocols and lower bounds for failure localization in the Internet. In *Proc. EUROCRYPT*, Apr. 2008.

[14] J. C. R. Bennett and H. Zhang. Hierarchical packet fair queueing algorithms. *ACM/IEEE Trans. on Networking*, 5(5):675–689, Oct. 1997.

[15] J. Black and P. Rogaway. A block-cipher mode of operation for parallelizable message authentication. In *Proc. EUROCRYPT*, Apr. 2002.

[16] K. Butler, T. Farley, P. McDaniel, and J. Rexford. A survey of BGP security issues and solutions. *Proceedings of the IEEE*, 98(1):100–122, Jan. 2010.

[17] 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.

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

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

[20] G. A. Covington, G. Gibb, J. W. Lockwood, and N. McKeown. A packet generator on the NetFPGA platform. *IEEE Symposium on Field-Programmable Custom Computing Machines*, 2009.

[21] R. Dingledine, N. Mathewson, and P. Syverson. Tor: the second-generation onion router. In *USENIX SECURITY*, 2004.

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

[23] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy. Routebricks: exploiting parallelism to scale software routers. In *SOSP*, 2009.

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

[25] A. Farrel, A. Ayyangar, and J. Vasseur. Inter-domain MPLS and GMPLS traffic engineering – resource reservation protocol-traffic engineering (RSVP-TE) extensions. RFC 5151, Feb. 2008.

[26] N. G. Feamster. *Proactive Techniques for Correct and Predictable Internet Routing*. PhD thesis, M.I.T., Sept. 2005.

[27] P. Ferguson and D. Senie. Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing. RFC 2827, May 2000.

[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 *SIGCOMM Wkshp. on Future Directions in Net. Arch.*, 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] Y. Hu and A. Perrig. A survey of secure wireless ad hoc routing. *IEEE Security and Privacy Magazine*, 2:28–39, 2004.

[37] Y.-C. Hu, A. Perrig, and D. Johnson. Efficient security mechansims for routing protocols. In *NDSS*, Feb. 2003.

[38] Y.-C. Hu, A. Perrig, and M. Sirbu. SPV: Secure path vector routing for securing BGP. In *SIGCOMM*, Sept. 2004.

[39] 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.

[40] S. Kent, C. Lynn, and K. Seo. Secure border gateway protocol (S-BGP). *IEEE JSAC*, 18(4), Apr. 2000.

[41] A. D. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure overlay services. In *SIGCOMM*, Aug. 2002.

[42] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click modular router. *ACM TOCS*, 18(4):263–297, Nov. 2000.

[43] M. Lad, D. Massey, D. Pei, Y. Wu, B. Zhang, and L. Zhang. PHAS: A prefix hijack alert system. In *USENIX SECURITY*, July 2006.

[44] A. Li, X. Liu, and X. Yang. Bootstrapping accountability in the Internet we have. In *NSDI*, Apr. 2011.

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

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

[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] M. Miller. PoComON: A POlicy-COMpliant Overlay Network. Technical Report HR-11-04 (honors thesis), CS Dept, UT Austin, Oct. 2011.

[51] R. Moskowitz and P. Nikander. Host identity protocol (HIP) architecture. RFC 4423, May 2006.

[52] J. Naous. *Path-policy Compliant Networking and a Platform for Heterogeneous IAAS Management*. PhD thesis, Mar. 2011.

[53] V. N. Padmanabhan and D. R. Simon. Secure traceroute to detect faulty or malicious routing. In *SIGCOMM*, Aug. 2003.

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

[55] L. Popa, N. Egi, S. Ratnasamy, and I. Stoica. Building extensible networks with rule-based forwarding. In *OSDI*, Oct. 2010.

[56] Prolexic Technologies, Inc. `http://www.prolexic.com`.

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

[58] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol label switching. RFC 3031, Network Working Group, Jan. 2001.

[59] RouteScience PathControl. `http://www.networkworld.com/reviews/2002/0415rev.html`.

[60] M. Scott. Miracl library. `https://www.shamus.ie/index.php?page=Downloads`.

[61] A. Seehra, J. Naous, M. Walfish, D. Mazières, A. Nicolosi, and S. Shenker. A policy framework for the future Internet. In *HotNets*, Oct. 2009.

[62] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet Indirection Infrastructure. In *SIGCOMM*, Aug. 2002.

[63] M. Walfish, J. Stribling, M. Krohn, H. Balakrishnan, R. Morris, and S. Shenker. Middleboxes no longer considered harmful. In *OSDI*, Dec. 2004.

[64] W. Xu and J. Rexford. MIRO: Multi-path interdomain routing. In *SIGCOMM*, Sept. 2006.

[65] A. Yaar, A. Perrig, and D. Song. SIFF: A stateless Internet flow filter to mitigate DDoS flooding attacks. In *IEEE Symposium on Security and Privacy*, May 2004.

[66] 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.

[67] X. Yang, D. Clark, and A. W. Berger. NIRA: A new inter-domain routing architecture. *ACM/IEEE Trans. on Networking*, 15(4), Aug. 2007.

[68] X. Yang and D. Wetherall. Source selectable path diversity via routing deflections. In *SIGCOMM*, Sept. 2006.

[69] X. Yang, D. Wetherall, and T. Anderson. TVA: A DoS-limiting network architecture. *ACM/IEEE Trans. on Networking*, 16(6):1267–1280, Dec. 2008.

[70] X. Zhang, H.-C. Hsiao, G. Hasker, H. Chan, A. Perrig, and D. G. Andersen. SCION: Scalability, control, and isolation on next-generation networks. In *IEEE Symposium on Security and Privacy*, May 2011.

[71] X. Zhang, A. Jain, and A. Perrig. Packet-dropping adversary identification for data plane security. In *CoNEXT*, Dec. 2008.

[72] E. Zmijewski. You can't get there from here. `http://www.renesys.com/blog/2008/03/you-cant-get-there-from-here-1.shtml`, Mar. 2008.