Securing User-controlled Routing Infrastructures

Karthik Kalambur Lakshminarayanan
Daniel Giannico Adkins
Adrian Perrig
Ion Stoica

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2007-37
http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-37.html

March 22, 2007

Copyright © 2007, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Securing User-controlled Routing Infrastructures

Karthik Lakshminarayanan, UC Berkeley
Daniel Adkins, UC Berkeley
Adrian Perrig, CMU
Ion Stoica, UC Berkeley

***Abstract*— Designing infrastructures that give untrusted third-parties (such as end-hosts) control over routing is a promising research direction for achieving flexible and efficient communication. However, serious concerns remain over the deployment of such infrastructures, none less than the new security vulnerabilities they introduce. The flexible control plane of these infrastructures can be exploited to launch many types of powerful attacks with little effort.**

In this paper, we make several contributions towards studying security issues in forwarding infrastructures. We present a general model for a forwarding infrastructure, analyze potential security vulnerabilities, and present techniques to address these vulnerabilities. The main technique that we introduce in this paper is the use of simple, light-weight, cryptographic constraints on forwarding entries. We show that it is possible to prevent a large class of attacks on end-hosts, and bound the flooding attacks that can be launched on the infrastructure nodes to a small constant value. Our mechanisms are general and apply to a variety of earlier proposals such as $i3$, DataRouter and Network Pointers.

I. Introduction

Several recent proposals have argued for giving third-parties and end-users control over routing in the network infrastructure. Some examples of such routing architectures include TRIAD [6], $i3$ [30], NIRA [39], DataRouter [33], and Network Pointers [34]. While exposing control over routing to third-parties departs from conventional network architecture, these proposals have shown that such control significantly increases the flexibility and extensibility of these networks. Using such control, hosts can achieve many functions that are difficult to achieve in the Internet today, such as support for mobility, multicast, content routing, and service composition. Another somewhat surprising application is that such control can be used by hosts to protect themselves from packet-level denial-of-service (DoS) attacks [18], since, at the extreme, these hosts can remove the forwarding state that malicious hosts use to forward packets to them. While each of these specific functions can be achieved using a specific mechanism—for example, mobile IP allows host mobility—we believe that these FIs provide architectural simplicity and uniformity in providing several functions that makes them worth exploring.

Forwarding infrastructures typically provide user control by either allowing source-routing (such as [6], [30], [39]) or allowing users to insert forwarding state in the infrastructure (such as [30], [33], [34]). Allowing forwarding entries enables functions like mobility and multicast that are hard to achieve using source-routing alone.

While there seems to be a general agreement over the potential benefits of user-controlled routing architectures, the security vulnerabilities that they introduce has been one of the important concerns that has been not addressed fully. The flexibility that the FIs provide allows malicious entities to attack both the FI as well as hosts connected to the FI. For instance, consider $i3$ [30], an indirection-based FI which allows hosts to insert forwarding entries of the form (id, R), so that all packets addressed to id are forwarded to R. An attacker A can eavesdrop or subvert the traffic directed to a victim V by inserting a forwarding entry (id_V, A); the attacker can eavesdrop even when it does not have access to the physical links carrying the victim's traffic. Alternatively, consider an FI that provides multicast; an attacker can use such an FI to amplify a flooding attack by replicating a packet several times and directing all the replicas to a victim. These vulnerabilities should come as no surprise; in general, the greater the flexibility of the infrastructure, the harder it is to make it secure [1], [36].

In this paper, we aim to push the envelope of the security that truly flexible communication infrastructures, that provide a diverse set of operations including packet replication, allow. Our main goal in this paper is to show that FIs are no more vulnerable than traditional communication networks such as IP, which do not export control on forwarding. To this end, we present several mechanisms that make these FIs achieve certain specific security properties, yet retain the essential features and efficiency of the original design. Our main defense technique, which is based on light-weight cryptographic constraints on forwarding entries, prevents several attacks including eavesdropping, loops, and traffic amplification attacks. From earlier work, we leverage some techniques, such as challenge-responses and erasure coding techniques, to thwart other attacks.

The organization of the rest of the paper is as follows:

- To abstract away the details of the several forwarding infrastructures, we propose a simple model for FIs in Section II.
- We present the desirable security properties of a FI that can be roughly summarized as follows (Section IV): (a) an attacker should not be able to eavesdrop on the traffic to an arbitrary host, (b) an attacker should not be able to amplify its attack on end-hosts using the FI, (c) an attacker can only cause a small bounded attack on the FI, and (d) an attacker that has compromised an FI node

can only affect traffic that it forwards. For each of these properties, we also present examples of attacks that show why a naive FI design violates these properties.

- We describe a set of security mechanisms that achieve these properties (Section V). The most important contribution, *light-weight cryptographic constraints on forwarding entries*, allows the construction of only acyclic topologies, thus preventing malicious hosts from using packet replication of the infrastructure to multiply flooding attacks. For example, to prevent loops, we leverage the difficulty in finding short loops in the mapping defined by cryptographic hash functions [22]. To the best of our knowledge, this is the first system that exploits the difficulty in finding short loops in cryptographic hash functions for designing a secure routing system.

II. Forwarding Infrastructure Model

Since the designs of various FIs proposals vary greatly, we present a simplified model that abstracts the forwarding operations of these proposals. The following FI model we present is similar to MPLS [27]; in summary, the model tries to abstract the forwarding operation performed at an FI node to an update of the identifier that is contained in the packet header.

A. Identifiers and Forwarding Entries

Each packet header contains an identifier *id*, that contains both the next-hop that the packet is addressed to (*id.node*), and a flat label used to match the routing table at the next-hop (*id.key*). The structure of *id.node* depends on the underlying routing used by the particular FI; for example, it could represent the IP address of the node (e.g. DataRouter [33]), or the DHT identifier of the node (e.g. $i3$ [30]). When a host A wishes to communicate with host B using the FI, the host A sends a packet containing an identifier *id* that would eventually be routed to host B.

Each FI node maintains a table of *forwarding entries*. A forwarding entry is a pair *(id, finfo)*, where *id* has the same structure and semantics as the packet identifier, and *finfo* (shorthand for *forwarding information*) is additional information that is used to modify the header before forwarding the packet.

In the simplest case, the *finfo* is just the identifier to which the packet is next forwarded to, but it could also represent other types of forwarding information such as a source route or a stack of identifiers. The notion of *finfo* is introduced here just to show how we can accommodate several FIs; in the update function we specify later, we abstract out the *finfo* and only worry about the final identifier that the packet header is updated to.

The scope of the *key* of an identifier is local an FI node, and there may be several entries with the same key at a node to allow multicast. While precluding replication would eliminate

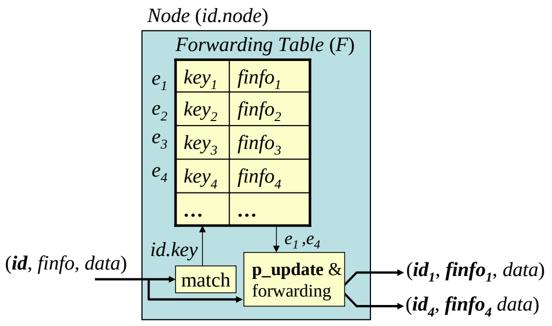


Fig. 1. The operations performed by an FI node upon the arrival of a packet with identifier id.

several of the attacks that we discuss in this paper, we believe that multicast is a key functionality that future FIs will provide. These forwarding entries are maintained in the FI as soft-state that must be refreshed periodically.

B. Packet Routing Functions

The three steps in routing a packet are: (1) matching the packet header with forwarding entries at a node, (2) modifying the packet header based on the forwarding entry it matches, and (3) forwarding the packet to the next hop. Figure 1 illustrates the packet processing at an FI node.

Packet Matching. When a packet arrives at node, the packet identifier is matched against the the forwarding table by a matching function:

$$\textbf{match}(id, F) \rightarrow \{e_1, e_2, \ldots, e_k\}, \tag{1}$$

which takes as input a packet's *id* and a forwarding table F (stored at node *id.node*), and outputs a set of entries. For achieving our security properties, we'll later require that the matching operation matches a certain number of bits in the identifier exactly.

Packet Header Update. The header and destination of a packet are based only on the incoming packet's header and the matching entry. If multiple entries are matched, the packet is replicated. The update function:

$$\textbf{update}(p, e) \rightarrow p' \tag{2}$$

takes a packet header p and an entry e, and produces a modified packet header p'.

The security techniques will impose constraints between the input identifier *id* and the output *id'*, hence the semantics of the update function (e.g. how exactly the finfo is used) is irrelevant to our discussion. In the rest of the paper, we denote an entry that changes the ID of a packet from id_1 to id_2 by $[id_1{\rightarrow}id_2]$.

$$\textbf{update}(id_1, [id_1{\rightarrow}id_2]) \rightarrow id_2,$$

where $p.id{=}id_1$ and $p'.id{=}id_2$.

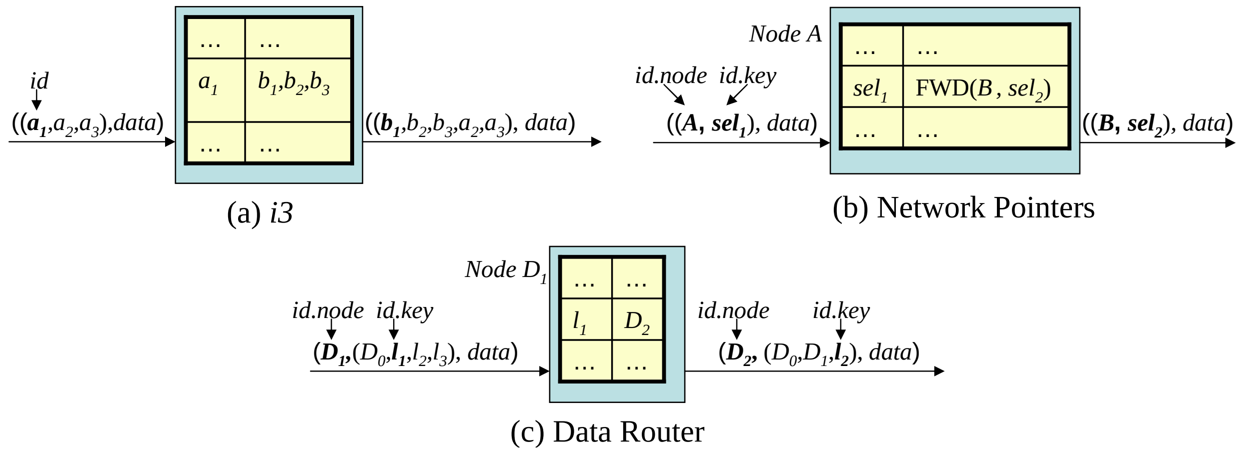

id
((a_1,a_2,a_3),data)
a_1 | b_1,b_2,b_3
((b_1,b_2,b_3,a_2,a_3), data)
(a) i3
Node A
id.node id.key
((A, sel_1), data)
sel_1 | FWD(B, sel_2)
((B, sel_2), data)
(b) Network Pointers
Node D_1
id.node id.key
(D_1,(D_0,l_1,l_2,l_3), data)
l_1 | D_2
(D_2, (D_0,D_1,l_2), data)
(c) Data Router

Fig. 2. The forwarding operation for three forwarding infrastructure proposals: (a) $i3$, (b) Network Pointers, and (c) DataRouter.

C. Examples of FIs

For concreteness, we give some examples of FIs to illustrate how an existing FIs can be instantiated in our model.

1) Internet Indirection Infrastructure: $i3$ is an indirection overlay that allows hosts to specify which packets they want to receive by inserting forwarding entries with the appropriate identifiers [30]. In the simplest case, at an $i3$ node, the incoming packet p contains the identifier id_a (*i.e.,* $p.id = id_a$). The identifier, id_a determines the matching entry in the forwarding table, as well as the next hop of the packet. Let us say that id_a matches an entry $id_a \rightarrow (id_b, id_c)$; here (id_b, id_c) denotes a stack of IDs. Then, the ID id_a is replaced by the ID id_b, id_c is added to *p.finfo*, and the packet is forwarded to id_b.

Both *p.id.node* and *p.id.key* are encoded in the $i3$ ID of the packet. The **match** operation is longest prefix matching. The **p_update** operation swaps the identifier at the top of the stack with the stack in the matching entry *e.finfo*. Note that host addresses can be encoded in *e.finfo* for packets sent to end-hosts.

2) Network Pointers: Network Pointers is a link layer mechanism that gives end-hosts fine-grained control over forwarding in the network by inserting *pointers* [34] (see example in Figure 2(b)). The incoming packet contains the address of the next hop A as well as a selector sel_1 which is used to index the forwarding table at node A. The packet is forwarded to the next hop after its selector is updated to (B, sel_2). The *p.id.node* field is the next-hop address, and *p.id.key* is the selector. The **match** operation is exact matching. The **p_update** operation is specified in the *e.finfo* field. The packet can be either forwarded to a next hop by updating its $p.id$ or delivered to a local application.

3) DataRouter: DataRouter is a forwarding engine that provides generic string matching and rewriting capabilities at the IP layer based on application-specific needs [33]. DataRouter is a high performance generic alternative to application-layer overlays. In addition to the IP header, a packet carries a generalized source route. The source route can contain arbitrary strings, which is used to index the forwarding table. Figure 2(c) shows an example in which the packet with destination address D_1 arrives at the next hop. The source route carried by the packet consists of the IP path traversed by the packet so far (D_0), and a list of string labels, $[l_1, l_2, l_3]$, used to index the forwarding tables of the hops along the path. In this example, when the packet arrives at node D_1, the node swaps the first string label l_1 with its address, and forwards the packet to the next hop, as indicated by the forwarding entry, D_2.

In general, *p.id.node* is the destination address of the packet, and *p.id.key* consists of a *class* that identifies a forwarding table at the next hop (not shown in the example), and a *string* used to search in that forwarding table. The **p_update** operation, in general, updates the destination IP address and the forward information *p.finfo* in the packet. The only constraint is that the node cannot update the prefix of the source route (*i.e., p.info*) that shows the path followed by the packet so far.

D. User Control over Forwarding Entries

We assume that FI nodes allow end-hosts to insert and remove of entries into and from the forwarding tables at the FI nodes.

insert(*n*, *e*); *// insert entry e into n's forwarding table*
remove(*n*, *e*); *// remove entry e from n's forwarding table*

Following $i3$ terminology [30], we assume that there are two types of IDs: *public* and *private*. These IDs differ in their level of "visibility" to end-users: a public ID is publicly known, while a private ID is known only to a trusted set of users. Similarly, we call a forwarding entry whose ID is public/private, a public/private forwarding entry. Public forwarding entries might be used by servers that arbitrary users can contact—all packets delivered to such servers will be relayed through their public forwarding entries. We introduce this distinction now since public and private entries require different security properties, and we exploit that fact to provide slightly different security mechanisms for these types of entries.

III. Threat Model

We describe our assumptions and the attacker threat model, and then derive the attacks that can be launched.

A. Security Assumptions

Our main goal in this paper is to show that the FIs are no more vulnerable than traditional communication networks such as IP, which do not export control on forwarding. To achieve this goal, we rely on several assumptions about the underlying routing layer. We assume that the virtual links between FI nodes as well as the link between the end-hosts and the FI node it is connected to[1] provide secrecy, authenticity, and replay protection—i.e., we do not consider link-level adversaries that can eavesdrop on arbitrary network links. These virtual links represent ISP-ISP relationships, which can be readily secured through standard security protocols (*e.g.*, IPsec [16]), and do not need a public-key infrastructure. The security requirement for the virtual link from hosts to FI nodes stems from the fact that we want to protect against link-level adversaries eavesdropping on the messages that hosts send, and the security requirement for that between FI nodes stems from the fact that we wish to show that attackers that control FI nodes are no worse than attackers that control IP routers today. However, we note that such a requirement might limit the scalability of the system to a few thousand nodes which we believe is in the same ballpark of how much the overlay deployments of such FIs target.

FI proposals rely on an underlying routing protocol that routes packets between FI nodes. For example, DataRouter uses IP routing, and $i3$ uses the Chord lookup protocol [31]. Addressing security issues of these underlying protocols is outside the scope of this paper. We note that there are several ongoing research efforts to address security issues both in the context of IP routing [11], [14], [17], [28], [32], [38] and DHT-routing [5], [29]. Finally, we do not consider processing or state-based attacks (such as insertion of many forwarding entries at an FI node) since these attacks are well-studied in the literature and can be solved using cryptographic puzzles [8], [9], [23].

B. Attacker Threat Model

We consider two attacker types: internal and external attackers. An *external attacker* does not control any compromised FI node but misuses the flexibility given by the FI. An external attacker can perform only the operations that a legitimate host can: insert a forwarding entry and send a packet. An *internal attacker* is an adversary who controls some compromised FI nodes. Ideally, we want to ensure that an external attacker cannot misuse an FI network to amplify the magnitude of a flooding attack[2]. In the case of an internal attack, we want to ensure that an attacker who compromises an FI node cannot affect other traffic that is not forwarded through that compromised FI node.

[1]We assume that in real deployments, end-hosts are connected to one or a few FI nodes that act as the entry point of all packets of the hosts; hence, assuming that a host shares a key with a couple of FI nodes is reasonable.

IV. Properties of a Secure FI

In this section, we precisely state the properties of a secure FI that we seek to achieve, and present some simple examples of how these properties are violated in the naive FI designs.

A. Preventing Eavesdropping and Impersonation

Property 1: Let $[id \rightarrow X]$ be a public forwarding entry inserted by a host. Then, an external attacker cannot insert a forwarding entry with the same identifier id.

This property prevents eavesdropping and impersonation by preventing an external attacker from inserting a forwarding entry with the same ID as that of the victim. The property also covers the case in which the victim has no entry in the FI at the time the attacker inserts its entry. Hence, even if the attacker causes the removal of the victim's entry (*e.g.*, by flooding the victim), it cannot impersonate the victim.

To demonstrate that the basic FI design does not guarantee this property, we list an example each of an eavesdropping attack and an impersonation attack.

Eavesdropping. Consider an end-host R that inserts a public forwarding entry[3] $[id{\rightarrow}R]$ (see Figure 3(d)). An attacker X can eavesdrop on packets sent to R by inserting a forwarding entry $[id{\rightarrow}X]$. All packets that are forwarded via $[id{\rightarrow}R]$ will be replicated and forwarded via $[id{\rightarrow}X]$ to X as well.

Impersonation. A variant of eavesdropping involves an attacker X making an end-host R drop its public entry by flooding it.[4] Then, if attacker X inserts $[id{\rightarrow}X]$, X can not only eavesdrop on R's traffic but also actively respond to it, thus impersonating R.

B. Preventing Flooding Attacks on End-Hosts

The following property prevents an external attacker from using the FI to: (a) amplify the traffic it sends to a victim host, and (b) redirect traffic meant for other hosts to the victim host.

Property 2: An external attacker cannot make a single victim end-host receive more packets than the attacker itself sends or receives.

[2]By flooding attack, we refer to a DoS attack in which the attacker floods the victim's network link by sending data at a large rate.

[3]To improve readability, we simplify the notation: we write $[id{\rightarrow}R]$ to mean $[id{\rightarrow}id_R]$, where $id_R.node = R$.

[4]We assume that R maintains its entry using soft state (since forwarding tables are usually managed using soft-state). We also assume that a host under flooding attack cannot refresh its entries.

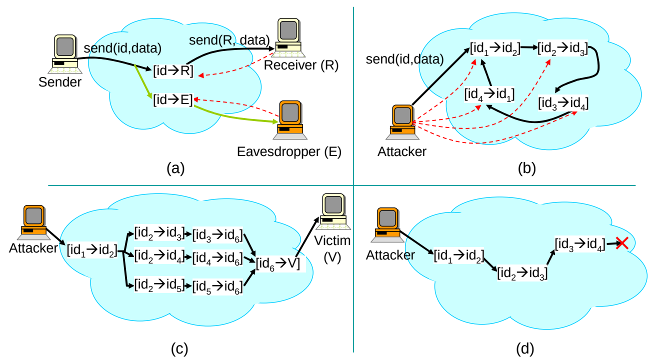


Fig. 3. Attack examples: (a) eavesdropping, (b) cycle, (c) end-host confluence, and (d) dead-end.

In essense, the property bounds the worst-case flooding attack that an external attacker can perform to what the attacker can do in today's Internet: send packets directly to the victim. However, the basic design of FIs do not guarantee the property; we illustrate this using some intuitive examples.

Malicious linking. Consider a forwarding entry $[id_1 \rightarrow X]$ that receives a large number of packets. An attacker can sign up an end-host R, with an existing public forwarding entry $[id \rightarrow R]$, to the high bandwidth traffic stream of the popular entry by inserting the entry $[id_1 \rightarrow id]$.

Cycles involving end-hosts. Consider two benign hosts R_1 and R_2 inserting entries $[id_1 \rightarrow R_1]$ and $[id_2 \rightarrow R_2]$ respectively. An attacker can create a cycle by inserting entries $[id_1 \rightarrow id_2]$ and $[id_2 \rightarrow id_1]$. Packets sent to id_1 and id_2 would be indefinitely replicated, thus overwhelming R_1 and R_2.

End-host confluence. This is a variant of the confluence attack where the target is an end-host rather than an FI node. By making the leaves of the tree point to the public entry of an end-host (see Figure 3(c)), an attacker can overwhelm the host.

C. Limiting Attacks on FI

While the previous two properties *prevent* attacks on end-hosts, the next property only alleviates external attacks on the FI. We state the property after introducing a new metric: *forwarding cost*.

Definition 1: Consider a packet m that traverses l links (*i.e.*, the packet is forwarded l times) in the FI and that is received by k receivers. The forwarding cost of m is then

$$FC(m) = \frac{l}{k+1} \tag{3}$$

The forwarding cost measures the amount of work the FI does for every unit of work performed by end-hosts involved in the communication, where a unit of work is either sending, receiving or forwarding a packet. (The increment by one in the denominator of Eq. (3) accounts for the sender sending a

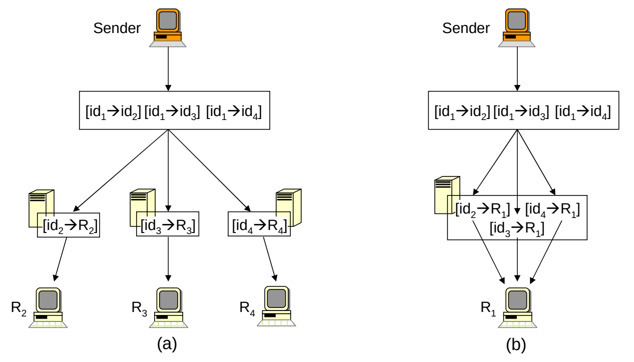


Fig. 4. Example where a seemingly legitimate topology can be exploited for attack.

packet.) For example, the forwarding cost of a unicast packet traversing h hops is $h/2$ if it is delivered to the receiver, and h otherwise. A cycle has infinite forwarding cost; by imposing a TTL of l, the cost of a cycle would be bounded by l.

To reduce the ability of an external attacker to use the FI to amplify its attack, we should make the forwarding cost as small as possible. The following property captures this requirement in rather vague terms, which we shall make more precise in Section V-C.2.

Property 3: The forwarding cost is bounded and small.

Before we give simple examples of why the basic FI design doesn't guarantee this property, we first reason why we only bound the attack on the FI and not prevent it completely.

Consider the subtle class of attacks called *over-subscription* attacks, where an attacker builds a seemingly benign topology; what we mean is that by just looking at the topology, one cannot determine if the topology is used maliciously or not. But an attacker can use this topology to make the FI do extra work by sending packets at a much higher rate than the (colluding) receivers can handle. Consider the legitimate multicast topology in Figure 4(a). An attacker can exploit this topology to mount an attack on an FI node, by having all leaves terminate at a colluding receiver (see Figure 4(b), which is identical to Figure 4(a) where $R_1 = R_2 = R_3$) which has limited receiving capability, and make all the IDs in the penultimate level reside on the same FI node. This will cause all the replicated traffic to be directed to that FI node.

From the above example, it is clear that a defense mechanism can detect such attacks only after the attacks are started since one cannot decide whether it is an attack just looking at the topology. Thus, we can only alleviate such attacks, not prevent them completely. Property 3 achieves this by linking the damage caused by an attacker to how much communication resources the attacker has, *i.e.*, it bounds the ratio between how many packets an FI forwards on the behalf of the attacker and how much traffic the attacker can send/receive directly to/from the FI.

In the basic FI design, an attacker can insert forwardin entries to unboundedly amplify a flooding attack on the F We present examples of such undesirable topologies.

Cycles. An attacker can form a loop by inserting for warding entries $[id_1 \rightarrow id_2]$, …, $[id_{n-1} \rightarrow id_n]$, $[id_n \rightarrow id_1]$ (se Figure 3(a)). A packet with identifier id_i ($1 \leq i \leq n$) woul indefinitely cycle around the loop and consume FI resources

Dead-ends. An attacker can construct a chain of forwardin entries, or even a multicast tree, which do not point to valid end-host (see Figure 3(b)). Data packets sent on such a topology would be forwarded and replicated only to be dropped at the dead ends.

Confluence. An attacker can refine a dead-ends attack by constructing a multicast tree with m leaves, all pointing to a victim FI node. For every packet sent by the attacker, the destination will receive m duplicates.

D. Limiting Internal Attacks

Property 4: An internal attacker should be able to mount only two forms of attacks: (a) drop the packets directed to forwarding entries it is responsible for, (b) a *random* flooding attack, *i.e.*, attacking a host through its forwarding entry *without* knowing the identity of the host.

The above property essentially makes an internal FI attacker no worse than an attacker compromising a router in the Internet today. In fact, in some cases an FI internal attacker is less powerful than an internal attacker in IP today since an FI internal attacker cannot mount an "off-path" attack, *i.e.*, it cannot affect other FI nodes or end-hosts whose packets are not normally forwarded through the compromised node.

V. Defense Mechanisms

We present defense mechanisms that achieve the properties of a secure FI that we enumerated in the previous section. The first technique, *constrained IDs*, is our main technique. We also use two other well-known techniques—challenge-responses and erasure coding. The constrained IDs technique enforces property 1, and together with the challenge response technique, they enforce property 2. By using all three techniques, we can provide property 3. Finally, we discuss the case of internal attackers.

Before we present our main security mechanisms, we briefly note that attackers cannot update or remove entries inserted by other hosts. To remove or update an entry, users need to specify both fields of the entry: the key and the *finfo* fields. Hence, an attacker can modify an entry only by guessing the fields. By allowing the owner of an entry to include a sufficiently long random nonce (80 bits long suffice [19]) in the *finfo* field, we can ensure that guessing the *finfo* field is highly improbable. We also assume that it is infeasible for an attacker to guess the ID of a *private* forwarding entry. As before, we enforce this by including a nonce in the ID of a private entry.

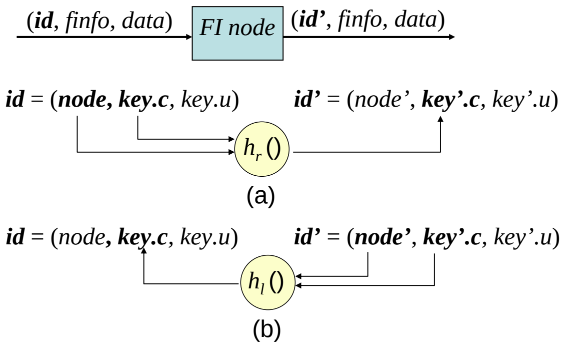

(**id**, *finfo*, *data*)
FI node
(**id'**, *finfo*, *data*)
id = (**node, key.c**, *key.u*)
id' = (*node'*, **key'.c**, *key'.u*)
$h_r()$
(a)
id = (*node*, **key.c**, *key.u*)
id' = (**node'**, **key'.c**, *key'.u*)
$h_l()$
(b)

Fig. 5. An FI node can update the ID of a packet from id to id' iff id and id' are either (a) right constrained (or r-constrained), or (b) left constrained (or l-constrained).

A. Technique 1: Constrained IDs

Constrained IDs is our core technique, which prevents eavesdropping, impersonation, and the construction of topologies that are not trees. Consider an FI node that updates the packet ID from id to id'. We enforce a constraint on the structure of IDs such that the choice of id cryptographically constrains the choice of id' or vice-versa.

To implement the constraints, we divide *id.key* into two sub-fields: a constrained part (*id.key.c*) and an unconstrained part (*id.key.u*). When a packet is matched at an FI node, the constrained part *must* match. The *constrained IDs* rule can be stated as follows (see Figure 5):

Constrained IDs Rule: A packet ID, id, can be updated to id', if and only if either *id'.key.c*=h_r*(id.node, id.key.c)* or *id.key.c*=h_l*(id'.node, id'.key.c)* hold.[5]

Functions h_l and h_r are cryptographic hash functions mapping N-bit strings to n-bit strings, where N is the size of an ID excluding the unconstrained part of the key, and n is the size of the constrained part of the key. The properties we require of the cryptographic hash functions (h_l and h_r) are: (a) strong collision resistance, and (b) computationally infeasibility of finding short cycles. Secure one-way hash functions, such as SHA-256 [24], provide these two properties [22].[6] If it is clear from the context, we use $id'{=}h_r(id)$ and $id{=}h_l(id')$ as a shorthand for *id'.key.c*=h_r*(id.node, id.key.c)* and *id.key.c*=h_l*(id.node, id.key.c)*, respectively. (Recall that an identifier has the form *(id.node, id.key.c, id.key.u)*, where *id.key.c* is constrained by the cryptographic function, and *id.key.u* is unconstrained and can be freely chosen.)

Intuitively, a cryptographic hash function makes it hard for an adversary to construct malicious topologies such as loops. Since h_l and h_r are publicly known hash functions, any FI node or host can check and enforce the constraints. If

[5]We allow both l and r constraints because, as we will show later, l-constraints provide better security properties, whereas r-constraints allow greater functionality.

[6]Given the recent attacks against SHA-1 [35] which reduces the complexity to find a collision to 2^{63}, SHA-256 is required for high security.

packet's ID, id, is updated to id' and $id' = h_r(id)$, we say that packet ID is *right*-constrained (r-constrained); otherwise, we say that it is *left*-constrained (l-constrained). Note that we choose different hash functions h_l and h_r to avoid trivial cycles of length two.

Next, we show that constraining packet IDs allows only topologies that are trees. Note however that since we allow flexibility of choosing *id.node*, one can still construct confluences on end-hosts and FI nodes. We deal with these problems in Sections V-B and V-C.2 respectively.

Theorem 1: With constrained IDs, it is infeasible for a computationally-bounded adversary to create topologies other than trees.

Proof: Refer to Appendix I ■

The rule that we use to constrain IDs results directly from the dual goal of achieving the desirable security properties and at the same time preserving the FI functionality. To illustrate this point, we enumerate several alternatives to constrain IDs we considered. (In Section VII, we show that our constrained IDs rule indeed preserves the functionality of the FIs.)

(a) Constraining the entire ID id' using id (or vice-versa) would imply that *id'.node* would depend on id. This would limit the flexibility of an end-user or third-party in choosing the nodes along a path.

(b) Constraining the entire *id.key* using some part of *id'* would be restrictive, as some FIs require control on the value of *id.key*. For example, $i3$ uses the *id.key*'s suffix to implement anycast [30].

(c) Constraining *id'.key* using only *id.key* would allow an attacker to create confluences on FI nodes by mapping all the leaf IDs to the victim node; e.g., by inserting the entries $[id_1 \rightarrow id_2]$, $[id_1 \rightarrow id_3]$, $[id_2 \rightarrow id_4]$, $[id_3 \rightarrow id_5]$ where all IDs are constrained and $id_4.node = id_5.node$.

Providing Property 1: Constrained IDs (l-constrained IDs in particular) help achieve property 1 (preventing eavesdropping and impersonation) if we enforce all public IDs to be l-constrained, *i.e.,* if a packet matches a public ID id and is replaced by id', then $id{=}h_l(id')$. If a host constrains its public ID id using a secret ID id', then, to eavesdrop, an attacker should insert an entry $[id \rightarrow id'']$ pointing to the attacker. Hence, an attacker needs to find an ID id'' such that $h_l(id'') = h_l(id') = id$ which amounts to finding hash collisions.

A simple technique to ensure l-constraints on public IDs would be to separate the key space for public and private IDs. For instance, the first bit of *id.key* could denote whether the entry is l-constrained or r-constrained. Since the key space of l- and r-constrained entries are separate, an attacker cannot insert an r-constrained entry for eavesdropping.

B. Technique 2: Challenge-Response

To ensure that an attacker cannot insert entries pointing to other benign end-hosts, we use the well-known challenge-response technique. FI nodes challenge the insertion of every forwarding entry using a simple three-way handshake. This mechanism is well-known in the literature and is similar to TCP SYN cookies; we describe it here for completeness.

Consider end-host A inserting an entry $[id \rightarrow B]$ at an FI node I_a. The FI node I_a sends a nonce n to host B, since B is the host contained in the entry $[id \rightarrow B]$. Host A, which attempted the insertion, can respond to I_a with the nonce n only if it receives the the traffic sent to B—this condition is trivially true if a node is inserting an entry pointing to itself (*i.e.,* $A = B$ in this case). However, an attacker that is not in the physical path to B cannot respond to the challenge, and hence the insertion does not succeed. To avoid maintaining state in FI nodes for every insertion, the challenge is computed using a message authentication function h_k on the values id and B, where k is a secret key only known to the FI node. To prevent replay of challenges, the FI node can periodically update the key k.

Providing Property 2: The challenge-response protocol outlined above helps achieve property 2 (preventing amplification attacks on end-hosts) since an attacker cannot insert an entry pointing to an arbitrary end-host it does not control. Hence, to replicate its traffic and direct it towards a particular host E, the attacker must itself create a malicious ID-level topology, and link all leaves with an existing entry already inserted by E. But since we already achieve property 1, such an ID-level topology is not possible.

C. Technique 3: Defense against Over-subscription

As mentioned in Section IV-C, benign topologies can be used by an attacker to launch a flooding attack on a victim FI node. For example, an attacker, by controlling both the sender and the receiver, can construct a tree such that all entries at the last level (*i.e.,* entries of the form $[idv{*} \rightarrow R]$, where R is the receiver) reside at the victim FI node, V. Each packet sent by the sender will be replicated, and all replicas will be sent to the victim FI node. In general, an attacker can amplify its attack N-fold by inserting $O(N)$ forwarding entries. Unfortunately, it is very hard to prevent such an attack since the resulting topology is legitimate, *i.e.,* it is a tree in which each leaf points to an end-host. What enables this attack is the ability of the attacker to insert forwarding entries at an arbitrary FI node. Since this control is critical to the flexibility of many FIs, we choose to use a reactive technique to alleviate this attack (rather than place restrictions on where the entries are stored).

The main observation we make is that such an attack can be alleviated if the attacker cannot make the FI generate more traffic than the attacker can send or receive. In other words, an attacker should not be able to generate more traffic than the receiver R can handle, in which case the attack on node V would be bounded by R's link capacity. The attacker then would not be able to benefit from replicating its traffic, and hence it cannot do better than attacking the victim directly. A simple way to achieve this property is to ensure that the packet loss along *each edge (link)* in the topology is bounded. Consider a forwarding entry $(id, *)$ located at FI node A that forwards packets to FI node or end-host B. If A detects that B receives less than a fraction f of the packets sent by A, then A raises a pushback signal. Now, there are two questions that we need to answer: how is the loss rate measured, and how does the sender react when the loss rate exceeds f.

To detect high losses, we borrow the mechanism based on erasure codes proposed in [12]. FI node A associates a nonce a with every n consecutive packets forwarded via entry $(id, *)$ to next hop B. In particular, node A uses a (k, n) erasure code to encode nonce a, and then piggybacks the erasures into the n consecutive packets forwarded to B. As long as B receives at least k packets, it can reconstruct the nonce a and send it back to A. If node A doesn't receive the nonce, then it implies that B received less than a fraction $\alpha{=}k/n$ of the packets, *i.e.*, the loss is at least[7] $1{-}\alpha$. The additional traffic generated by this mechanism is very low, since only one small-sized packet is sent every n packets. For example, choosing $\alpha{=}3/4$ would help tolerate a loss rate as high as 25% (a loss rate at which TCP would not be able to sustain any reasonable throughput), while worsening a possible attack only by a factor of 1.33. The other important parameter in our design is the block size n. A large value of n makes the test more robust, but increases the vulnerability period during which the attacker can exploit the mechanism. To account for the possible loss of nonce reply messages sent by node B to previous hop A, we require that a loss rate greater than f be observed over c consecutive epochs of encoding before initiating the pushback. In practice we choose $c = 3$.

When an FI node detects that the receivers cannot receive the data packets (since it does not receive a correct nonce), it takes action in the form of a *pushback* to ensure that the topology of forwarding entries is pruned all the way to the source. The pushback can be implemented by simply rate-limiting the traffic, or more aggressively, by removing the forwarding entry. In the latter case, even if there are false positives (*i.e.*, an entry is incorrectly removed), soft-state refreshing of entries would ensure that the topology is restored. In the former case, to ensure that pushback signals propagate up the topology, an FI node should reconstruct the nonce (that it uses to prove to the upstream node that it received at least α packets) based on the packets it successfully sent to its downstream nodes. Instead, if a node reconstructs the nonce based on the packets it receives from its upstream node, then pushback from a bottleneck at its downstream node will not propagate upwards.

1) Providing Property 3: An attacker can exploit the reactive nature of the above mechanism to carry out attacks for short intervals of time. Indeed, the mechanism allows a window of vulnerability from the time the attacker constructs a graph till the time the FI prunes it down. In this section, we bound the damage even when the attacker exploits this window of vulnerability.

Consider an attacker that constructs a tree violating the constraint that the leaves are either dead-ends or end-hosts that receive less than an α fraction of the traffic sent by the FI nodes.[8] Let the maximum height of the tree be h_{max}. This can be enforced using a TTL field. Let $[id{\rightarrow}id']$ be a forwarding entry of this tree stored at node A where id' is a leaf. After receiving the first packet with ID id, A will take t_r time units to remove this entry. If id is a dead-end, t_r is equal to the RTT (τ), since that is how long the pushback mechanism takes to detect a dead-end and propagate a message back one hop. If the leaf is an oversubscribing end-host, t_r is the time it takes the FI node to send $n_c = n{\times}c$ packets of maximum size plus the time it needs to wait for the end-host to send back the nonce: $t_r = n_c l_{max}/r + \tau$, where l_{max} is the maximum packet size.

The only way the attacker can maintain this tree is to replace the leaf edges as soon as they are removed by the pushback mechanism. This attacker strategy would prevent pushback from pruning the rest of the tree. Let λ be the rate at which the attacker can insert new forwarding entries. The maximum number of leaves that an adversary can maintain is then $l = t_r\lambda$. The next result gives a bound on the forwarding cost for this attack scenario.

Theorem 2: For an attacker that can send packets at an aggregate rate of r, and can insert forwarding entries at a rate λ, the average packet forwarding cost is upper-bounded by:

$$\frac{h_{max}\ n_c\ l_{max}\ \lambda}{r + \lambda\ o} + h_{max}\tau\lambda, \tag{4}$$

where o is the overhead incurred by a host (in bits) when inserting a forwarding entry.

We note that mounting an attack that achieves this bound is not trivial. To utilize the resources optimally, an attacker needs to anticipate when an entry is removed, which is hard due to the fact that the attacker does not know the round-trip time between the FI nodes, and the round-trip times can vary significantly.

2) Limiting Forwarding Cost: By inspecting Formula 4, we see that the forwarding cost can be reduced by increasing the overhead of the insertion operation o and limiting the insertion rate λ. We can increase the insertion overhead by either increasing the size of the response packets so that $o\ \simeq l_{max}$, or use multiple challenge-response rounds before inserting an entry. In the latter case, at each round the FI node sends a new challenge containing a nonce based on the nonce sent in the previous round (*e.g.*, by hashing the previous

[7] A dead-end is a special case in which the pushback can be initiated when no forwarding entry matches the packet.

[8] Note that the attacker cannot violate the constraints enforced by the "constrained ID" technique.

nonce). A host will be able to insert an entry only if it answers all challenges sent by the FI node.

Since many systems maintain forwarding entries by soft-state, there is no difference between inserting and refreshing the entries. The refreshing rate can be policed by the first-hop FI node. The rate can be specific to end-hosts, negotiated when hosts sign up for the FI service. However, designing efficient mechanisms for restricting λ for a malicious FI node is a hard problem; initial insights are presented in [15].

Consider an attacker that sends traffic at 5 Mbps. If maximum tree depth of 10, $l_{max} = o = 1400$ bytes, $n_c = 48 \times 3 = 144$, $\tau = 100$ms, and $\lambda = 1$ entries/s[9], we get a forwarding cost of about 2. From the first term in the expression, we also observe that with a higher attack rate, the forwarding cost would only go down.

3) Discussion: The defense against over-subscriptions we presented here is a data plane mechanism as opposed to the control plans mechanisms we presented earlier, and is hence arguably more expensive. While there may be some other control plane approaches that might approximate this solution, we do not explore them further in the scope of this paper for the following reasons. Firstly, any control plane mechanism that needs a complex protocol spanning several FI nodes, because the forwarding entries that comprise the topology can be spread across several FI nodes. In our mechanisms, we perform only local check between pairs of nodes which we show is very efficient to perform. Secondly, and more fundamentally, the problem arises only in the data plane and can be solved “cleanly” only at the data plane (as we explained in Section IV-C).

D. Addressing Internal Attacks

In this section, we consider *internal* attackers. We assume that such an attacker can compromise FI nodes and have complete control over their local state, and over packets received or sent by these FI nodes.

While internal attackers have complete control on the traffic forwarded by the FI nodes they compromise, we show that they have very little power on the traffic forwarded by other FI nodes. In particular, the only attack an internal attacker can mount that an external attacker cannot is a *random* attack, *i.e.,* attacking a host through its private forwarding entry *without* knowing the identity of the host. Unlike routing protocols in today’s Internet (such as BGP [32]), an internal attacker cannot mount an “off-path” attack, *i.e.,* it cannot affect other FI nodes or end-hosts whose packets are not normally forwarded through the compromised node.

We assume that an attacker cannot eavesdrop the payload of the packets it forwards, including the control packets that manipulate the forwarding entries. In other words, the attacker can read only the information in the control packets regarding forwarding entries that are stored locally. This assumption can be enforced by encrypting the payload of both control and data packets. To authenticate the FI nodes, we can use self-certified node IDs like HIP [25], where a node’s ID is computed using an one-way hash function on the node’s public key. This is equivalent with using public keys to identify FI nodes, instead of IDs.

Even if the attacker is not able to eavesdrop the payload of the packets it forwards, the attacker can still learn the IDs of forwarding entries stored at other FI nodes by inspecting the *finfo* field in packets that are matched locally, or the *finfo* field in the FI entry stored at the compromised node. Hence the advantage of an internal attacker is that it can learn about private IDs of other end-hosts while an external attacker cannot. However, the attacker has no direct way to associate that ID with an end-host, since it cannot learn who inserted the forwarding entries at other nodes. Hence, mounting attacks on a private ID is equivalent to mounting an attack on a random end-host.

Finally, a compromised FI node could also violate the l- and r-trigger constraints. However, the key observation is that all replicated traffic will continue to flow through the compromised FI node, *i.e.,* it cannot create a loop not going through itself or an amplification topology not including itself.

The crucial insight behind the argument that no other forms of attacks are possible is that FIs merely forward packets as the forwarding entries dictate; they do *not* run any routing protocol[10]. All the operations that are performed at an FI node use local state and simple packet update rules. Hence, the operations that a compromised FI node can perform—insert or remove forwarding entries and send packets—is fundamentally no different from the ones performed by end-hosts; the only difference is that FI nodes typically have more resources than the end-hosts.

E. Summary of Defense Techniques

The modifications can be classified based on where they are implemented: *data* and *control* plane changes. We first list the data plane modifications.

- Packet IDs should be either l- or r-constrained; *i.e.,* when the packet ID is updated from id to id', then either *id.key.c*$=h_l$(*id’.node*, *id’.key.c*) or h_r(*id.node*, *id.key.c*)$=$*id’.key.c*. The sub-field $id.key.c$ should be long enough (*e.g.,* 128 bit, as discussed in Section V-A) so that it is infeasible for an attacker to guess it. Public IDs should be l-constrained.
- Private IDs should be long enough that it is hard for an external attacker to guess; hence, mounting an attack or eavesdropping on a private entry is hard. The $id.key.c$ field can be re-used for this purpose; it is computed by a cryptographic hash function (which guarantees a pseudo-

[9]A web server may negotiate a much higher rate of inserting entries if needed.

[10]We mean that they do not run any protocol at the forwarding layer; of course, they use underlying routing protocols like Internet routing or DHT routing to forward packets to other nodes.

random string) when id is the target of a constraint and randomly chosen otherwise.

- To limit the forwarding cost, packet headers need to include a TTL field that is decremented at every hop. In practice, a TTL of 8 bits should suffice.
- Packet headers need to include an erasure (about 1-byte) to prevent over-subscription attacks.
- Replicated packets can be delivered to the destination only through a forwarding entry inserted by that destination. This restriction prevents confluences on end-hosts.

Thus, in addition to the fields id and *finfo*, a packet header needs to include two other one-byte fields (one for TTL and one for the erasure). Now, we list the control plane changes.

- The *finfo* field of an entry should include a nonce that is hard to guess. This prevents an attacker from updating and removing such entries. As discussed in Section III-A, this nonce should be at least 80-bits long.
- The insertion of a public entry requires a challenge-response mechanism as described in Section V-B. This mechanism prevents malicious linking, but adds one RTT to the entry insertion operation.
- The FI needs to implement the pushback mechanism described in Section V-C which involves appending an erasure to each packet that is forwarded.

Until now we have implicitly assumed that ID constraints are checked at run-time, *i.e.*, when the ID of a packet is updated. However, in many cases, how the IDs would be updated is known when the forwarding entry is inserted. For example, in $i3$ and Network Pointers, a forwarding entry of the form $[id_1 \rightarrow id_2]$ will update the ID of a packet from id_1 to id_2. In such cases, we can check for constraints when the entry is inserted, rather than when the packet is forwarded. As a result, the overhead due to checking constraints on the data path can be eliminated completely.

VI. Implementation and Evaluation

We have implemented the three main mechanisms—constraints of forwarding entries, response to over-subscription and challenges to forwarding entry insertion—over $i3$ [30], one of the FIs proposed earlier. We used inverted hash tables to implement pushback (needed for implementing response to over-subscription) and used a one-way hash function for generating the constraints as well as the challenges.

For efficiency, our one-way hash function is based on the Advanced Encryption Standard (AES) [7], using the Matyas, Meyer, and Oseas construction [21]. The key is encrypted by the AES cipher and then the output is XORed with the input. We get two different one-way functions, h_l and h_r, by keying the cipher with two different publicly known keys (different from the keys we hash).

We evaluate the three mechanisms in terms of their overhead and effectiveness. Since cryptographic constraints and challenges completely prevent the attacks that they are designed for, we only evaluate the additional overhead that they introduce. To illustrate the effectiveness of the pushback, we use simulations.

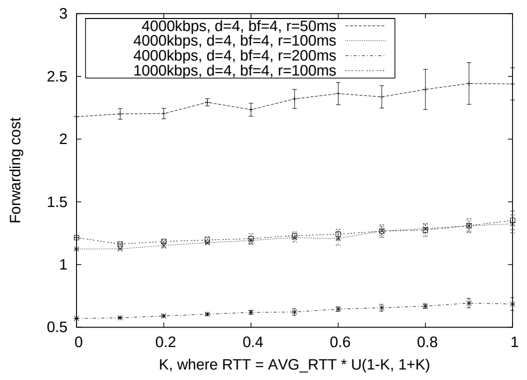


Fig. 6. Effectiveness of pushback as a function of variability of RTTs of links.

A. Cryptographic Constraints and Challenges: Computational Overhead

The two security mechanisms, cryptographic constraints as well as challenge mechanism, require operations on the control path. We show by experiments that the cost of both these operations is minimal.

As mentioned in Section V, checking both cryptographic constraints and challenges involve computation of a one-way hash function. If the challenge checking or constraint checking fails, the forwarding entry is not inserted (or in the case of run-time checking, the packet is not forwarded).

To measure the additional overhead, we ran tests on an $i3$ node on a 866 MHz Pentium III running Linux 2.4.8. The results are averaged over half a million operations. The running time for a hash-computation is less than 3 μs, which implies that constraints can be checked even on the data path while supporting forwarding rates of a few hundred thousand packets per second. Overall, the computational overhead for checking the challenge and the constraints is only about 28%.

B. Defense against Over-subscription

To evaluate the sensitivity of the pushback mechanism, we first performed a set of simple experiments using a 5-node chain topology over Planetlab with an RTT of about 200 ms. In the first experiment, we run a TCP flow across the chain topology; in the subsequent experiments, we run UDP flows of increasing rates. In each experiment, we transferred 3 MB and recorded if the pushback was triggered. We repeat each experiment 25 times.

The TCP transfers experienced an average throughput of 1.6 Mbps and never triggered pushback. Table I shows the fraction

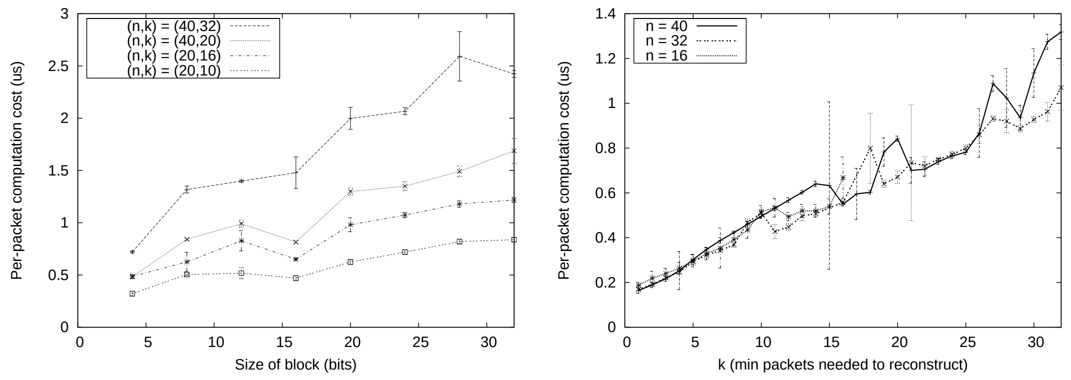


Fig. 7. Overhead of verification mechanisms for preventing over-subscription

of UDP transfers that did not trigger a pushback. As expected, as the rate of the UDP flow increases, the probability that pushback is triggered also increases. When the rate reaches 4 Mbps, pushback is always triggered immediately. We infer that the probability of false positives of the pushback mechanism when the sending rate is close to the TCP sending rate is negligible. A comprehensive evaluation of the interaction between the over-subscription mechanism and congestion control mechanisms is outside the scope of this paper.

Rate (Mbps)	2	2.5	3	3.5	4
Fraction of Successful Transfers	1	0.8	0.6	0.5	0

TABLE I
FRACTION OF SUCCESSFUL UDP TRANSFERS FOR DIFFERENT SENDING RATES.

The analysis of the technique to defend against over-subscription in Section V-C presents an upper bound assuming that all hops have the same RTT. We now present the effect of variation in RTTs on the mechanism using a simple event-driven simulator. We use simulations rather than experiments as it allows the attacker to precisely control the timing of (re)inserting triggers. This precise timing is very hard to achieve in practice due to the RTT variations, thus we expect the simulation results we present here to be an upper bound for the experimental results.

At the beginning of the simulation, we construct a complete tree given a particular depth, d and branching factor b. We let the adversary refresh the forwarding entries at a particular rate r. The adversary also is assumed to have global knowledge so that it can refresh the forwarding entry that would cause *maximum* damage (*i.e.,* the deepest entry).

In the main experiment that we report, we set the branching factor to be 4. Figure 6 shows how the forwarding cost varies depending on how the RTT of the links is chosen in the simulation—randomization of zero corresponds to all links having same RTT (of *MAX_RTT/2*) and randomization of 1 corresponds to RTTs being chosen uniformly between [*0, MAX_RTT*]. The refresh periods are chosen as $50, 100$ and 200 ms. The main inference from the graph is that the variation in RTTs does not affect pushback by much and almost closely mirrors our analysis. Secondly, even when receivers are allowed to refresh every 50 ms, the forwarding cost is only about 2. Finally, varying attacker sending rate had little effect on the forwarding cost.

C. Cost of Erasure Computation

We present the cost of computing the erasures (introduced in Section V-C) for preventing over-subscription attacks. We used the FEC software developed by Rizzo *et al.* [26] for benchmarking the erasure computation. Figure 7(a) shows the cost of per-packet erasure computation by varying the size of the block used for different (n, k) combinations. The increase in cost with the increase in block size is marginal—even for 32-bit blocks, and $(n, k) = (40, 32)$, the cost is under $2.5\mu s$. Figure 7(b) shows the variation in overhead as n/k is varied for three values of n for 8-bit blocks. The increase is almost linear—it is not precisely linear because the implementation is based on Vandermonde matrices and at certain values, low-level issues such as cache hits/misses would cause deviations from the expected trends.

VII. REALIZATION OVER SPECIFIC PROPOSALS

The generic FI model helped us abstract away the details of FIs, and concentrate on fundamental problems. We presented a range of techniques that can be used in specific FI designs. However, we do not advocate a “one-size-fits-all” approach; particular FIs present tradeoffs that need to be considered before making decisions on which techniques are relevant. Here, we present a few examples to illustrate this point.

The FI model, for generality, assumed that FI nodes perform packet replication. Consistent with this assumption, we constrained forwarding entries such that malicious topologies that allow misuse of packet replication—such as confluences—are impossible to construct. However, certain legitimate applications construct *control plane topologies* that are identical to confluences but the *data plane topologies* formed by the IDs that the packets take are benign due to additional operations performed during packet forwarding. Examples of such applications include multipath routing and load balancing; packets can be forwarded either along path $(id_s, id_1, id_2, \ldots, id_e)$, or path $(id_s, id'_1, id'_2, \ldots, id_e)$—the union of the two paths is indeed a confluence. If the FI does not allow packet replication, then we can allow load balancing by constraining the IDs based on the keys alone (*i.e., id.key.c*=h_l(*id'.key.c*) or h_r(*id.key.c*)=*id'.key.c*). In practice, an FI that performs both packet replication as well as load balancing can have separate ID spaces and allow packet replication on one ID space and load balancing on the other.

A. Internet Indirection Infrastructure

We divide the 256-bit identifier in $i3$ into three fields: a 64-bit prefix (roughly corresponds to *id.node*), a 128-bit constrained key (corresponds to *id.key.c*), and a 64-bit suffix (corresponds to *id.key.u*). IDs id and id' are matched based on the longest prefix matching rule, given the constraint that both their keys and prefixes match exactly. If an l-constrained trigger (x, y) points to an end-host, we use only *y.key.c* to constrain *x.key.c*. Ignoring *y.node* and *y.key.u* when computing $h_l(y)$ allows us to preserve support for anycast and mobility.

Since the packet's ID is always replaced with the first ID in the matching trigger's stack, constraints can be checked when the trigger is inserted instead of at run-time, thus avoiding any overhead on the data path. Next, we argue that constrained IDs have limited impact on the functionality provided by $i3$.

Mobility. Since constraints are not computed over the IP addresses of hosts (which is stored in $id.node$), there is no impact on mobility.

Multicast. Applications can still build legitimate multicast trees as in $i3$ by using r-constrained triggers. The triggers that are used to build multicast trees are private triggers and hence having r-constrained triggers would not expose the multicast group to eavesdropping.

Anycast. Anycast functionality is not affected by trigger constraints. However, in an anycast group with l-constrained triggers, each end-host must have the same $id.key.c$; this key needs to be distributed out-of-band.

Service composition. Disallowing insertion of arbitrary triggers still allows sender-driven service composition, but weakens the flexibility of receiver-driven service composition. In particular, it will not be possible for a receiver to redirect packets with a *given* ID x to an intermediate node with a *given* ID y since this would require the receiver to insert a trigger of the form (x, y), where x and y are fixed. However, this situation can be dealt with at the application level by negotiating a private trigger out-of-band. We expect this restriction to be acceptable to a majority of applications.

Apart from the above changes, the main logical change to $i3$ was that hosts have to be explicitly aware of the l- and $r-$constraints on the triggers, which makes the $i3$ client slightly more complicated. In our implementation, we support full-fledged packet replication, limited multi-path routing support where the host explicitly inserts forwarding entries for all the paths to itself (thus removing loops at the ID-level), and no load-balancing support.

B. Network Pointers

Since a node uses exact matching to match the selectors, one can use the entire selector as the key to incorporate ID constraints. However, the length of the selector should be increased as it is only 64 bits long in their design. For supporting the forwarding operation described in [34], the constraints can be checked at the time of inserting the entries. For more complex forwarding operations, one might need to check the constraints at run-time. All proposed uses of Network Pointers involve only chain topologies [10], [34], which will not be affected by constrained IDs.

C. DataRouter

To enforce ID constraints, the strings used to index into the forwarding tables should have a sub-string of bits which are matched exactly and which represent the $id.key.c$ field. Even when alternate matching algorithms are selected by the application (such as range matching), exact matching must be performed on the constrained part before the specific matching algorithm can be invoked on the remainder of the tag. While this does not undermine the specific matching algorithms, it might require additional bits for the field $id.key.c$. Since a packet's ID can be updated based on the packet's *finfo* field, checking the ID constraints needs to be done at run-time. We are not aware of any application in the context of the DataRouter [33] that requires cyclic topologies or confluences, thus constrained IDs will not limit their functionality.

VIII. Related Work

Traditionally, new network architectures have suffered from many security issues. With active networks, achieving security is difficult and has often come at the expense of restricting the flexibility (such as ESP [4]) or use of per-use policy and authentication (such as SANE [1]). In fact, loose source routing is disabled by many ISPs because of security issues [3].

Mechanisms for addressing seemingly simple problems such as loop prevention [37] have involved operations on the data path. Furthermore, loop prevention techniques in literature have been reactive, and do not guarantee loop-free topologies.

Using a time-to-live field is a common technique to prevent persistent routing cycles in networks, with IPv4 networks being the prime example. We believe that TTLs alone aren't sufficient for preventing cycles and confluences in FIs. If we use TTLs alone, then with a TTL of l, an attacker can replicate a packet l times by inserting just two entries (id_1, id_2) and (id_2, id_1). However, the constrained IDs technique makes the construction of short cycles infeasible. Hence an attacker has to insert several forwarding entries to replicate the traffic. By bounding the rate of insertion of new entries, we have shown that we can alleviate attacks effectively.

In the process of designing security mechanisms for FIs, we have leveraged techniques that have been proposed earlier in the literature. Challenge-response protocols have been used for a long time in diverse areas. The idea of using erasure codes to ensure that uncooperative hosts do not oversubscribe to high-bandwidth streams was proposed recently in the context of multicast [12]. Pushback has been proposed for rate-limiting the traffic of IP aggregates by Mahajan et al [20].

Proposals that deal with DoS attacks based on packet floods [2], [13], [18] are orthogonal to ours; we devise mechanisms to prevent end-hosts from *using* the infrastructure to aggravate attacks.

We do not consider the issue of securing the underlying routing layer, since the work in that space is largely orthogonal. We note that there are several ongoing research efforts to address these issues both in the context of IP routing [14], [17], [28], [32] and DHT-routing [5], [29].

IX. Conclusions

Giving hosts control over forwarding in the infrastructure has become one of the promising approaches in designing flexible network architectures. In this paper, we addressed the security concerns of these forwarding infrastructures.

We presented a general FI model, analyzed potential security vulnerabilities and presented several mechanisms to alleviate attacks. Our key defense mechanism, based on light-weight cryptographic constraints, provably prevents a large set of attacks. In contrast to previous efforts that detect and mitigate malicious activity, the cryptographic mechanism prevents attacks altogether. Our mechanisms are applicable to many earlier proposals such as $i3$ [30] and DataRouter [33] while requiring only modest changes. In providing secure forwarding, we make the deployment of these promising architectures much more viable.

References

[1] D. S. Alexander, W. A. Arbaugh, A. D. Keromytis, and J. M. Smith, “A Secure Active Network Environment Architecture,” *IEEE Network*, 1998.

[2] T. Anderson, T. Roscoe, and D. Wetherall, “Preventing Internet Denial-of-Service with Capabilities,” in *Proc. of Hotnets*, 2003.

[3] S. Bellovin, “Security Concerns for IPng,” RFC 1675, 1994.

[4] K. L. Calvert, J. Griffioen, and S. Wen, “Lightweight Network Support for Scalable End-to-End Services,” in *Proc of SIGCOMM*, 2002.

[5] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach, “Secure Routing for Structured Peer-to-peer Overlay Networks,” in *Proc. OSDI*, Dec. 2002.

[6] D. R. Cheriton and M. Gritter, “TRIAD: A New Next Generation Internet Architecture,” Mar. 2000, http://www-dsg.stanford.edu/triad/triad.ps.gz.

[7] J. Daemen and V. Rijmen, “AES proposal: Rijndael,” Mar. 1999.

[8] D. Dean and A. Stubblefield, “Using Client Puzzles to Protect TLS,” in *Proc. of the 10th USENIX Security Symposium*, 2001.

[9] C. Dwork and M. Naor, “Pricing via Processing or Combatting Junk Mail,” in *Advances in Cryptology — CRYPTO ’92*, ser. LNCS, E. Brickell, Ed., vol. 740, International Association for Cryptologic Research. Springer-Verlag, 1993, pp. 139–147.

[10] R. Gold, P. Gunningberg, and C. Tschudin, “A Virtualized Link Layer with Support for Indirection,” in *Proc. of FDNA*, 2004.

[11] G. Goodell, W. Aiello, T. Griffin, J. Ioannidis, P. McDaniel, and A. Rubin, “Working around BGP: An Incremental Approach to Improving Security and Accuracy in Interdomain Routing,” in *Proc. of NDSS*, Feb. 2003.

[12] S. Gorinsky, S. Jain, H. Vin, and Y. Zhang, “Robustness to Inflated Subscription in Multicast Congestion Control,” in *Proc SIGCOMM*, 2003.

[13] M. Handley and A. Greenhalgh, “Steps Towards a DoS-resistant Internet Architecture,” in *Proc. of FDNA*, 2004.

[14] Y.-C. Hu, A. Perrig, and M. Sirbu, “SPV: Secure Path Vector Routing for Securing BGP,” in *Proc. of ACM SIGCOMM*, 2004.

[15] A. Jain, J. Hellerstein, S. Ratnasamy, and D. Wetherall, “A Wakeup Call for Internet Monitoring Systems: The Case for Distributed Triggers,” in *Proc. of Hotnets*, 2004.

[16] S. Kent and R. Atkinson, “Security Architecture for the Internet Protocol,” IETF, Internet RFC 2401, Nov. 1998.

[17] S. Kent, C. Lynn, and K. Seo, “Secure Border Gateway Protocol (S-BGP),” *IEEE JSAC*, vol. 18, no. 4, pp. 582–592, Apr. 2000.

[18] K. Lakshminarayanan, D. Adkins, A. Perrig, and I. Stoica, “Taming IP Packet Flooding Attacks,” in *Proc. ACM HotNets-II*, Cambridge, MA, Nov. 2003.

[19] A. K. Lenstra and E. R. Verheul, “Selecting Cryptographic Key Sizes,” *Journal of Cryptology*, vol. 14, no. 4, pp. 255–293, 2001.

[20] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker, “Controlling High Bandwidth Aggregates in the Network,” *CCR*, vol. 32, no. 3, pp. 62–73, July 2002.

[21] S. Matyas, C. Meyer, and J. Oseas, “Generating Strong One-way Functions with Cryptographic Algorithm,” *IBM Technical Disclosure Bulletin*, vol. 27, pp. 5658–5659, 1985.

[22] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, ser. CRC Press series on discrete mathematics and its applications. CRC Press, 1997, iSBN 0-8493-8523-7.

[23] R. Merkle, “Secure Communication Over Insecure Channels,” vol. 21, no. 4, pp. 294–299, Apr. 1978.

[24] National Institute of Standards and Technology (NIST), Computer Systems Laboratory, “Secure Hash Standard,” Federal Information Processing Standards Publication (FIPS PUB) 180-2, Aug. 2002.

[25] G. O’Shea and M. Roe, “Child-proof authentication for MIPv6 (CAM),” *Computer Communication Review*, vol. Apr., no. 31, p. 2, 2001.

[26] L. Rizzo, “http://info.iet.unipi.it/ luigi/fec.html.”

[27] E. Rosen, A. Viswanathan, and R. Callon, “Multiprotocol Label Switching Architecture,” RFC 3031, Jan. 2001.

[28] Secure Origin BGP (soBGP), ftp://ftp-eng.cisco.com/sobgp.

[29] E. Sit and R. Morris, “Security Considerations for Peer-to-peer Distributed Hash Tables,” in *Proc. of IPTPS*, 2002.

[30] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, “Internet Indirection Infrastructure,” in *Proc. SIGCOMM*, 2002.

[31] I. Stoica, R. Morris, D. R. Karger, M. F. Kaashoek, and H. Balakrishnan, “Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications,” in *Proc. of SIGCOMM*, Aug. 2001.

[32] L. Subramanian, V. Roth, I. Stoica, S. Shenker, and R. H. Katz, “Listen and Whisper: Security Mechanisms for BGP,” in *Proc. of NSDI*, 2003.

[33] J. Touch and V. Pingali, “DataRouter: A Network-Layer Service for Application-Layer Forwarding,” in *Proc. IWAN*, 2003.

[34] C. Tschudin and R. Gold, “Network Pointers,” in *Proc. ACM HotNets-I*, 2002.

[35] X. Wang, Y. Yin, and H. Yu, “Finding collisions in the full sha-1,” in *Proceedings of Crypto*, Aug. 2005.

[36] D. Wetherall, “Active Network Vision and Reality: Lessons from a Capsule-based System,” in *Proc. of SOSP*, 1999.

[37] A. Whitaker and D. Wetherall, “Forwarding Without Loops in Icarus,” in *Proc. of IEEE OPENARCH*, 2002.

[38] R. White, “Deployment Considerations for Secure Origin BGP (soBGP), draft-white-sobgp-bgp-deployment-01.txt, IETF Draft,” June 2003.

[39] X. Yang, “NIRA: A New Internet Routing Architecture,” in *Proc FDNA-03*, 2003.

APPENDIX I
PROOFS OF THEOREMS

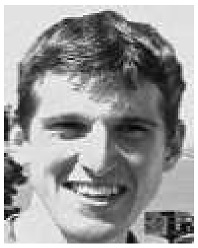
Proof of Theorem 1. *Proof:* Define G_d as the directed graph formed by assigning directions to the edges of G (we simplify the notation by dropping the argument of G and G_d). In particular, for each edge (x, y) in G we associate the direction from x to y if $y = h_r(x)$, and from y to x if $x = h_l(y)$. The proof is by contradiction. Assume G has a cycle. We consider two cases:

Case (i) G_d has at least one vertex with in-degree 2. This implies there are vertices x, y, z such that there are distinct edges $(x \rightarrow z), (y \rightarrow z) \in G_d$. Thus, $h_i(x) = h_j(y)$, for $h_i, h_j \in \{h_l, h_r\}$, such that $x \neq y$ or $h_i \neq h_j$ (otherwise edges $(x,z), (y,z)$ will not be distinct). In both the cases, finding x, y that satisfy these constraints is infeasible as it reduces to finding hash collisions.

Case (ii) All vertices of G_d are of in-degree at most one. We know that underlying graph G has a cycle, say C_u. Consider the sub-graph of G_d induced on the vertices of C_u, call it C_d. We know that $\forall v \in C_d$, $in\_degree(v) \leq 1$. But C_d is a cycle. Hence $\forall v \in C_d$, $in\_degree(v) = 1$. Thus, we have $x = \{h_l, h_r\}^*(x)$. This is equivalent to finding a cycle in the hash function and is hence computationally infeasible. ■

Proof of Theorem 2. *Proof:* The rate of sustained attack is proportional to the number of edges in the tree. Since a tree with l leaves has at most lh_{max} edges, an adversary can exploit the system to amplify its attack rate from r to $rh_{max}t_r\lambda$. Now, the total amount of traffic the attacker sends in the FI is $r+\lambda o$, and thus the damage ratio is $(rh_{max}t_r\lambda)/(r+\lambda o)$. Since the maximum value of t_r is achieved when the leaf is an end-host, we take $t_r = n_c l_{max}/r + \tau$. After some simple algebra, Formula 4 follows. ■

Karthik Lakshminarayanan received his B.Tech. degree in Computer Science and Engineering from the Indian Institute of Technology, Madras in 2001, where he was awarded the President of India's Gold Medal. He is currently pursuing his Ph.D. degree at the University of California at Berkeley. His research interests include overlay and peer-to-peer networks, Internet architecture and Internet security.



Daniel Adkins received the B.S. degree in computer science and mathematics from the Massachusetts Institute of Technology, Cambridge, in 2001. He is currently working toward the Ph.D. degree at the University of California, Berkeley. His research interests include combinatorial algorithms, computational biology, and networking.

Adrian Perrig (M'96) is an Assistant Professor in Electrical and Computer Engineering, Engineering and Public Policy, and Computer Science at Carnegie Mellon University. He earned the Ph.D. degree in Computer Science in 2001 from Carnegie Mellon University, and spent three years during his Ph.D. degree at the University of California, Berkeley. He received the M.S. degree in Computer Science in 1999 from Carnegie Mellon University and the B.Sc. degree in Computer Engineering in 1997 from the Swiss Federal Institute of Technology in Lausanne (EPFL). Professor Perrig's research interests revolve around building secure systems and include Internet security, security for sensor networks and mobile applications.

Ion Stoica received his Ph.D. from the Carnegie Mellon University in 2000. He is an Assistant Professor in the EECS Department at University of California at Berkeley, where he does research on peer-to-peer network technologies in the Internet, resource management, and network architectures. Stoica is the recipient of a Sloan Foundation Fellowship (2003), a Presidential Early Career Award for Scientists & Engineers (PECASE) (2002), and the ACM doctoral dissertation award (2001). He is a member of the ACM.