

# Secure and Policy-Compliant Source Routing

Barath Raghavan, Patrick Verkaik, and Alex C. Snoeren, *Member, IEEE*

**Abstract**—In today’s Internet, inter-domain route control remains elusive; nevertheless, such control could improve the performance, reliability, and utility of the network for end users and ISPs alike. While researchers have proposed a number of source routing techniques to combat this limitation, there has thus far been no way for independent ASes to ensure that such traffic does not circumvent local traffic policies, nor to accurately determine the correct party to charge for forwarding the traffic.

We present Platypus, an authenticated source routing system built around the concept of network capabilities, which allow for accountable, fine-grained path selection by cryptographically attesting to policy compliance at each hop along a source route. Capabilities can be composed to construct routes through multiple ASes and can be delegated to third parties. Platypus caters to the needs of both end users and ISPs: users gain the ability to pool their resources and select routes other than the default, while ISPs maintain control over where, when, and whose packets traverse their networks. We describe the design and implementation of an extensive Platypus policy framework that can be used to address several issues in wide-area routing at both the edge and the core, and evaluate its performance and security. Our results show that incremental deployment of Platypus can achieve immediate gains.

**Index Terms**—Authentication, capabilities, overlay networks, source routing.

## I. INTRODUCTION

NETWORK operators and academic researchers alike recognize that today’s wide-area Internet routing does not realize the full potential of the existing network infrastructure in terms of performance [37], reliability [1], [4], [26], or flexibility [15], [23], [45]. While a number of techniques for intelligent, source-controlled path selection have been proposed to improve end-to-end performance [37], [43], reliability [1], [4], [26], [47], and flexibility [13], [17], [23], [42], [45], they have proven problematic to deploy due to concerns about security and network instability. We attempt to address these issues in developing a scalable, authenticated, policy-compliant, wide-area source routing protocol.

We argue that many of the deficiencies of today’s routing infrastructure are symptoms of the coupling of routing policy and routing mechanism [39]. In particular, today’s primary wide-area routing protocol, the Border Gateway Protocol (BGP), is extraordinarily difficult to describe, analyze, or manage [29].

Manuscript received June 25, 2007; revised January 25, 2008; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor B. Levine. First published December 22, 2008; current version published June 17, 2009. This work was supported in part by the National Science Foundation through NSF CAREER Award CNS-0347949. An earlier version of this manuscript appeared in ACM SIGCOMM 2004.

The authors are with the Department of Computer Science and Engineering, University of California at San Diego, La Jolla, CA 92093-0404 USA (e-mail: barath@cs.ucsd.edu; pverkaik@cs.ucsd.edu; snoeren@cs.ucsd.edu).

Digital Object Identifier 10.1109/TNET.2008.2007949

Autonomous systems (ASes) express their local routing policy during BGP route advertisement by affecting the routes that are chosen and exported to neighbors. Similarly, ASes often adjust a number of attributes on routes they accept from their neighbors according to local guidelines [10], [20], [32]. As a result, configuring BGP becomes an overly complex task, one for which the outcome is rarely certain. BGP’s complexity affects Internet Service Providers (ISPs) and end users alike; ISPs struggle to understand and configure their networks while end users are left to wonder why end-to-end connectivity is so poor.

Our approach to reducing this complexity is to separate the issues of connectivity discovery and path selection. Removing policy constraints from route discovery presents an opportunity for end users and edge networks: routes previously hidden by overly conservative policy filters can be revealed by ASes and traversed by packets. The key challenge becomes determining whether a particular source route is appropriate. ASes have no incentive to forward arbitrary traffic; currently they only wish to forward traffic for their customers or peers. We argue, however, that this is simply a poor approximation of the real goal: ASes want to forward traffic only if they are compensated for it. Henceforth, we will consider traffic *policy compliant* at a particular point in the network if the AS can identify the appropriate party to bill, and that party has been authorized by the AS to use the portion of the network in question.

We present the design and evaluation of Platypus, a source routing system that, like many source-routing protocols before it, can be used to implement efficient overlay forwarding, select among multiple ingress/egress routers, provide virtual AS multi-homing, and address many other common routing deficiencies [39]. The key advantage of Platypus is its ability to ensure policy compliance during packet forwarding. Platypus enables packets to be *stamped* at the source as being policy compliant, reducing policy enforcement to stamp verification. Hence, Platypus allows for management of routing policy independent of route export and path selection.

Platypus uses *network capabilities*, primitives that are placed within individual packets, to securely attest to the policy compliance of source routing requests. Network capabilities are i) transferable: an entity can delegate capabilities to others, ii) composable: a packet may be accompanied by a set of capabilities, and iii) cryptographically authenticated. Capabilities can be issued by ASes to any parties they know how to bill. Each capability specifies a desired transit point (called a *waypoint*), a resource principal responsible for the traffic, and a stamp of authorization. By presenting a capability along with a routing request, end users and ISPs express their willingness to be held accountable for the traffic, and the included authorization ensures the policy compliance of the request.

In addition to its basic design, we also aim to understand how Platypus might be deployed in today’s Internet. To this

end, we detail the design and implementation of a policy framework for managing Platypus in an AS. Incremental deployability is key in our setting, as it would be unreasonable to expect ASes to cooperate in the deployment of a system that affects *local* policy. Thus, we present results from wide-area measurements and performance evaluation of a prototype UNIX-based Platypus router, which indicate that incremental deployment of Platypus is feasible and may yield substantial benefit even using only a few routers.

## II. OVERVIEW AND APPLICATIONS

It is well known that multiple paths often exist between any two points in today's Internet. The central tenet of any source-routing scheme is that no single route will be best for all parties. Instead, sources should be empowered to select their own routes according to whatever criteria they determine. Protocols for efficient wide-area route discovery and selection, however, are beyond the scope of this paper. We assume that the network is configured (using BGP, for example) with a set of default routes and that certain motivated parties become aware of alternative paths, either through active probing [4], [40] or route discovery services [31]. Platypus builds on this basic infrastructure, allowing entities to select paths other than the default. Packets may specify a set of waypoints to be traversed on the way to a destination, but are *not* required to specify each router along the path. A source-routed packet is forwarded using default paths between the specified waypoints; an end-to-end path is therefore a concatenation of default paths provided by the existing routing system.

Platypus is designed to be deployed selectively by ASes at choice locations in their networks. To support incremental deployment, Platypus waypoints are specified using routable IP addresses. When source routing a packet, the routing entity, which may be an end host or a device inside the network, encapsulates the payload and replaces the original destination IP address of the packet with the address of the first waypoint. The original destination IP address is stored in the packet for replacement at the last waypoint. When a Platypus packet arrives at a waypoint, the router updates the Platypus headers and forwards the packet on to the next waypoint.

### A. Sample Applications

We motivate the design of Platypus by describing several possible applications below. These examples are meant to be illustrative, not necessarily comprehensive. Moreover, we do not claim that Platypus is the only (or best) solution to each of the problems; instead, we suggest that it represents a single, elegant approach that addresses them all simultaneously.

1) *Efficient Overlay Routing/On-Demand Transit*: Consider the partial network topology shown in Fig. 1. Nodes  $S_A$ ,  $S_B$ , and  $S_C$  are all willing to cooperate to forward each other's traffic. Assume that  $S_A$  wishes to send a packet to  $S_B$ , but the default route  $S_A \rightarrow R_A \rightarrow R_B \rightarrow S_B$  is unsatisfactory, perhaps because the link  $R_A \leftrightarrow R_B$  is congested or down. With prior overlay systems [4],  $S_A$  could use  $S_C$  as a transit point by tunneling its traffic directly to  $S_C$ , who would then forward it along to  $S_B$ . While effective at avoiding the bad link, this route is clearly sub-optimal for all involved, since:

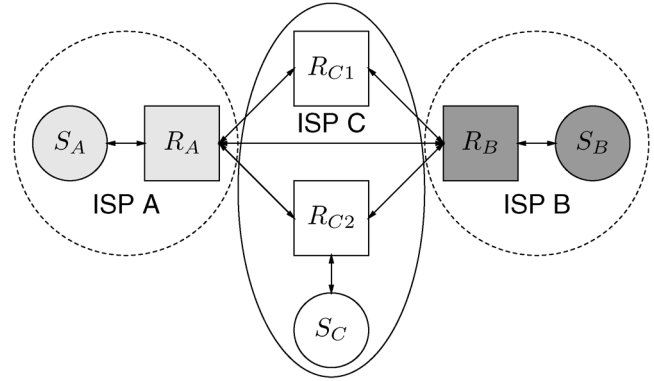


Fig. 1. A simple network topology. Hosts  $S_A$ ,  $S_B$ , and  $S_C$  all have different ISPs, as indicated by the different shading. Note that  $R_A$  and  $R_B$  are directly connected, bypassing ISP  $C$ .

- 1)  $S_C$  is forced to forward each packet itself, consuming both its bandwidth (in both directions) and processor resources. It would prefer that a router forward the traffic instead.
- 2) If avoiding  $R_A \leftrightarrow R_B$  is the objective, an alternate route exists using the  $R_A \leftrightarrow R_{C1} \leftrightarrow R_B$  path. If  $S_C$ 's ISP also owns  $R_{C1}$ ,  $S_C$  should be able to authorize use of the path  $R_A \leftrightarrow R_{C1} \leftrightarrow R_B$ .

Both of these issues could be addressed by traditional source-routing schemes. In the example,  $S_A$  can specify the route  $R_A \rightarrow R_{C1} \rightarrow R_B \rightarrow S_B$ . The challenge is in communicating to ISP  $C$  that such a route request is reasonable. In this case, assuming ISP  $C$  is not a transit provider, it is permissible only because  $S_C$  is a customer of the ISP and is willing to be charged for  $S_A$ 's traffic. With existing source-routing mechanisms, an AS cannot determine whether a forwarding request complies with local policy, and, if so, who to charge for the service. Currently, an AS assumes that packets should arrive at its border only if it advertised a route to their destinations. In our example, ISP  $C$  does not expect  $R_{C2}$  to receive packets from ISP  $A$  that are destined for  $S_B$ . Source-routed packets can obviously be made to explicitly transit any AS, violating this precondition. While ISPs can (and do) use filters to prevent unauthorized traffic from entering their network, filters can only act upon information contained within a packet—source and destination addresses, protocol, type of service, etc.—and current network location. These attributes are insufficient to determine policy compliance or the responsible party in this case. Nothing about the source-routed packet from  $S_A$  to  $S_B$  indicates  $S_C$ 's cooperation (and resulting policy compliance).

In Platypus,  $S_C$ , by virtue of being a customer of its ISP, may have authority to source route through any of the ISP's routers. In that case, ISP  $C$  would issue  $S_C$  a capability and a secret key that can be used to *stamp* packets. The capability would name  $C$  as the *resource principal*—the party responsible for all traffic bearing the capability. Platypus ensures the policy compliance of a given source route by requiring that source-routed packets contain a capability for each waypoint in a packet's source route. Because the secret key needed to stamp packets is known only to the indicated resource principal (or its associates), properly stamped packets certify their policy compliance and allow waypoints to appropriately account for usage.

We posit that ASes conduct *a priori* negotiations with customers and each other to determine mutually agreeable policies about who may source route traffic through which waypoints (similar to today's peering agreements [32]). Efficiently describing or constructing such policies is a complex problem on its own; we do not discuss it here. Instead, we assume the output of this process is a set of rights which can be encoded as a matrix of binary entries: for each waypoint in the network, a given resource principal may or may not forward traffic through it. Capabilities expire periodically and can be revoked, allowing ASes to dynamically update their policies.

Returning to our example,  $S_C$  could transfer a capability to  $S_A$  allowing  $S_A$  to construct a source route that can alleviate both issues enumerated above. In particular, if the capability specifies any router in ISP  $C$  as a waypoint, the first problem is solved— $S_C$  no longer needs to forward the packet itself. Moreover, if  $S_C$  were to transfer a capability specifically naming  $R_{C1}$  as a next hop, the second issue can also be addressed.

While we have described  $S_A$ ,  $S_B$ , and  $S_C$  as end hosts for simplicity, Platypus is designed to allow in-network stamping. Hence, each of these entities could correspond to entire ASes, allowing the example to be recast as a type of secondary transit, where  $S_C$ —a stub domain—can resell its transit privileges to other, non-adjacent stub domains without prior involvement of its provider.

2) *Preferential Ingress Points*: Multi-homed ASes often select multiple upstream providers and send different traffic through each depending on network conditions and destination—so-called policy routing. Unfortunately, an AS remains at the mercy of its upstream providers to control how incoming traffic arrives; there currently exists no widely deployed mechanism to affect ingress points [1], short of assigning multiple addresses to the AS's hosts and advertising different addresses to different providers. Using Platypus, however, an AS can *delegate* multiple capabilities naming waypoints corresponding to its different upstream providers. Just as with toll-free phone numbers, an AS may be willing to be the resource principal responsible for incoming traffic if it can affect how that incoming traffic arrives. In Section IV we demonstrate our design of a mechanism for safely broadcasting delegated capabilities using DNS.

3) *Virtual Multi-Homing*: A stub AS with a single upstream connection is currently limited to the default routes of its provider. Without multi-homing, an AS is incapable of selecting backbone providers to carry its traffic—it must use the backbone selected by its upstream AS. With Platypus, however, a stub AS could request capabilities from providers of its choice, and place these on its out-bound traffic indicating which of its regional provider's upstream backbones to use for particular traffic—in effect making the AS virtually multi-homed. Using delegated capabilities, in-bound traffic can be similarly affected. Thus, a stub AS could implement its own policy routing without the need for any configuration on the part of its upstream provider.

As a concrete example, suppose an AS,  $X$ , wishes to choose between two indirectly upstream providers  $A$  and  $B$ .  $X$ 's ISP,  $Y$ , need not provide Platypus support. At the  $X \leftrightarrow Y$  gateway,  $X$  classifies traffic it wishes to route through either  $A$  or  $B$  and

stamps them with appropriate capabilities. Though  $Y$  does not support Platypus forwarding, it faithfully delivers packets to  $A$ 's or  $B$ 's edge routers, which are aware of Platypus headers, and, thus, deliver the packets as  $X$  desired. In such a scenario  $A$  and  $B$  clearly have a financial motivation to provide such a service since they can bill  $X$ , while  $X$  benefits by having choice in its indirect upstream providers, potentially providing fail-over or optimized routing.  $X$ 's provider,  $Y$ , has no disincentive to allow Platypus-enabled packets to traverse its network since it has an already established relationship with  $X$ .

## B. Challenges

As these examples demonstrate, source routing can be used to address a number of issues with the existing routing infrastructure. We believe, therefore, that the unavailability or limited deployment of source routing protocols stems not from a lack of utility, but, instead, from the omission of two key features: a mechanism for accountable and composable authorization, and the ability for ISPs to effectively manage link utilization. The need for authorization should be clear from the examples. The relationship to load management, however, is a bit more subtle. Recent research indicates that self-interested source routing can achieve performance gains even in wide deployment, but raises concerns about possible negative interactions with traffic engineering—highly reactive sources may make existing traffic engineering mechanisms ineffective by constantly changing fine-grained route requests [34].

## III. NETWORK CAPABILITIES

Platypus addresses both of these issues through the use of network capabilities. Abstractly, a network capability is made up of two fields: a waypoint and a resource principal identifier. The waypoint specifies a topological network location through which the packet should be routed and the resource principal specifies the entity willing to be charged for the routing request. Using intra-AS routing mechanisms, an AS can route packets for a given waypoint to different Platypus routers, thus giving it more control over the effects of source-routed traffic on an ISP's traffic engineering. We return to this issue in Section VII-D. For now, we will consider waypoints to correspond to a specific router within an AS.

In Platypus, packets are stamped with a source-routing request by inserting a Platypus header immediately after the IP header of each packet and including some number of capabilities, encapsulating the existing payload. Fig. 2 shows the Platypus header format with one capability attached. The header contains fields for the protocol version (currently 0), a set of bit flags (whose use is described in Section IV-A.1), a length field (specified in terms of 32-bit words), a pointer to the current capability (also in terms of 32-bit words), and an encapsulated protocol field to facilitate de-encapsulation. Capabilities are appended immediately after the Platypus header. The Platypus header and capabilities may be added by in-network stampers while the packet is in transit.

Since anyone can use a capability to forward packets through the specified waypoint and bill the indicated resource principal, Platypus must ensure that eavesdroppers watching packets in the network cannot use capabilities they observe in flight for their

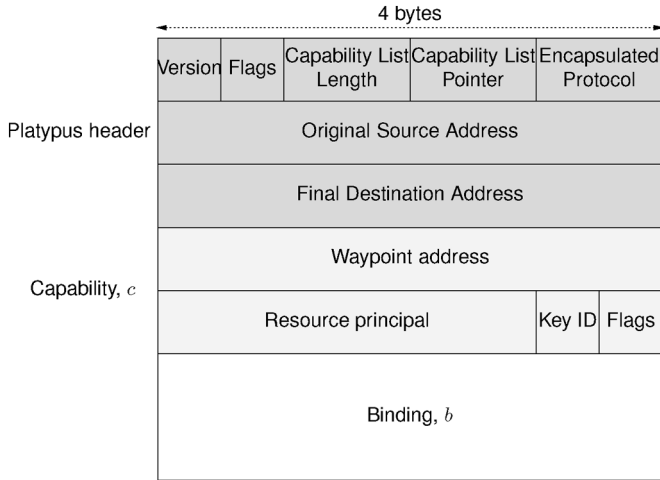


Fig. 2. Platypus header format with a single capability and binding attached.

own packets. Similarly, attackers should not be able to modify capabilities or construct new ones that enable them to use waypoints for which they are not authorized, or to bill other resource principals. To prevent this, each capability in a packet is accompanied by a *binding* that cryptographically ensures the capability is valid and being used by the appropriate party. Bindings are a function of the capability, the packet contents, and a secret known only to the owner of the capability. When a Platypus packet arrives at a waypoint, the Platypus router validates the corresponding capability and its binding. If the capability/binding pair is valid, the router updates the waypoint pointer (indicating the packet has already passed through this waypoint), sets the packet's current destination IP to the waypoint field of the next capability in the capability list, replaces the current source IP with its own (to prevent ingress filters from dropping the packet), and forwards the packet on. If no additional capabilities remain, the router replaces the original destination address.

#### A. MAC-Based Authentication

Platypus prevents forgery of capabilities or their bindings with the *cascade construction* of Bellare *et al.* [7], which is provably secure given an underlying MAC that is a pseudo-random function (PRF), as most modern MACs are believed to be. We define a secret temporal key,  $s = \text{MAC}_k(c)$ , generated from the capability,  $c$ , using a message authentication code (MAC) such as HMAC [24]. The MAC is keyed with  $k$ , the key of the specified waypoint. This value  $s$  is securely transferred to the resource principal (in a manner described in Section IV). In order to use a capability, an individual packet must be stamped with the capability and a binding,  $b = \text{MAC}_s(\text{MASK}(P))$ , where  $\text{MASK}(P)$  is the invariant [16] contents of the packet (not including Platypus headers) with the end-to-end source and destination addresses substituted and the packet length field omitted. Both  $b$  and  $c$  are included in the packet, as shown in Fig. 2. In this way, the binding is dependent upon both the secret key  $s$  and the packet's contents, and thus cannot be reused for other packets. Similarly, any changes to the capability  $c$  would render bindings computed with the secret temporal key  $s$  invalid.

```

R: Revocation set, ID: Current key ID
PROCESS(P: Packet)
  c ← *(P.phdr.ptr)
  if |c.id-ID| > 1 or c ∈ R then
    ICMPERROR(P)
  s ← MACk(c||GETTIME(c.id))
  b' ← MACs(MASK(P))
  if c.b = b' then
    ACCOUNT(c.rp, P)
    P.phdr.ptr ← P.phdr.ptr + |c|
    if P.phdr.ptr ≥ P.phdr.len then
      P.dst ← P.phdr.dst
    else
      c ← *(P.phdr.ptr)
      P.dst ← c.way
    FORWARD(P)
  else
    ICMPERROR(P)

```

Fig. 3. Pseudocode for Platypus forwarding.  $P$  is a packet,  $P.\text{src}$  is the packet's source IP address, and  $P.\text{phdr}$  is the Platypus header in which  $\text{src}(\text{dst})$  is the source (destination) address,  $\text{ptr}$  is the pointer to the current capability and  $\text{len}$  is the length of the capability list.  $c$  is a capability,  $c.\text{way}$  is its waypoint field,  $c.\text{rp}$  is its resource principal field,  $c.\text{id}$  is its key ID, and  $c.b$  is the binding accompanying  $c$ .  $\|$  denotes concatenation.

Fig. 3 presents pseudocode for Platypus packet verification and forwarding. To verify a packet's binding (and, therefore, capability), a Platypus router only needs the local waypoint key,  $k$ , since  $b' = \text{MAC}_{\text{MAC}_k(c)}(\text{MASK}(P)) = \text{MAC}_s(\text{MASK}(P))$ . If  $b \neq b'$ , either the capability or the binding (or both) has been forged and the packet should be discarded. An advantage of this construction is that the router needs to maintain only a constant amount of state irrespective of the number of resource principals. In addition, rejected packets elicit ICMP responses to the sender to quell further invalid transmissions (subject to standard ICMP rate limiting).

#### B. Key Expiration and Timing

If temporal secret keys were never to expire, ASes would have no means to enforce changing policies—resource principals could use their capabilities forever. In addition, if a key were transferred to a third party or compromised, the resource principal would have no way to regain control over its associated capability. To address these issues, Platypus provides automatic key expiration. Once a temporal secret key expires, resource principals must retrieve a new one from the *key server*. To simplify the task of authenticating resource principals to the key server, we introduce the notion of a capability master key,  $c_k$ , which is shared between the resource principal and the key server. The capability master key is not used to generate capabilities or bindings, it is only needed to retrieve a new temporal secret key from the key server.

Platypus is designed to avoid the need for tight time synchronization between stamping parties and Platypus routers. Each capability includes a key identifier (key ID) which is a small (4-bit) integer that identifies the temporal secret used to compute each packet's binding. This key ID value changes on a regular

TABLE I  
CAPABILITY KNOWLEDGE HIERARCHY. • DENOTES THAT THE VALUE IS/CAN BE KNOWN, ○ INDICATES IT IS GENERATED ON THE FLY

	Waypoint Key $k$	Revocation List $R$	Capability Master Key $c_k$	Temporal Secret Key $s$	Binding $b$
Key Server	•	•	•	○	
Platypus Router	•	•		○	○
Resource Principal			•	•	○
Trusted Third Party				•	○
Others					•

basis (e.g., every hour) and a new corresponding temporal secret generated. Since the key ID space is small, the key ID may wrap around often, yielding what would be identical temporal secrets if  $s = \text{MAC}_k(c)$ . We address this issue by incorporating the current time during generation of temporal secrets.<sup>1</sup> In this way, temporal secrets are guaranteed to be unique despite key ID wraparound.

To ensure that both stamping agents and routers agree on the current key ID, capabilities are associated with a key expiration interval upon issuance. The length of the expiration interval presents a natural tradeoff between control and overhead—short expiration intervals provide fine-grained control over secrets, but require more frequent key lookup. Expiration intervals must be chosen based upon operational experience with Platypus to suit the needs of the issuing AS and its resource principals. Our only synchronization requirement is that stampers have clocks that do not drift on the order of the expiration interval. In addition, to allow for transitions between secrets, we consider three secrets to be valid at any time: those for the current, previous, and next key IDs. To combat clock drift between Platypus routers, we expect that the routers are loosely time synchronized using a standard service such as NTP [30]; synchronization without NTP is an open problem.

### C. Security

Security in Platypus is provided by the fact that not all parties have the information needed to bind known capabilities to new packets or create new, usable capabilities. Table I shows the types of information known to various parties. To generate a temporal secret key, a party must have the waypoint key,  $k$ , which is known only to the router and the router's key server. Binding a capability to a packet requires only the temporal secret key,  $s$ , which is generated based upon  $k$  and the current time. Knowledge of one capability's temporal secret key, however, does not allow a party to generate temporal secrets for others. Resource principals wishing to transfer their full rights for a particular waypoint to a trusted third party can pass both the capability and corresponding temporal secret key.

While the capability can be passed in the clear, the temporal secret key must be communicated privately, ensuring that only the chosen third parties are able to receive it. These third parties can then use  $s$  to generate bindings to stamp their own packets. Others, including those sniffing packets on the network, can see capabilities and their bindings, but lack the secret  $s$  required

<sup>1</sup>Specifically, for a given time  $t$ , where  $t$  is the seconds part of a 32-bit UNIX timestamp in UTC, and an expiration interval of  $2^n$ , the corresponding key ID  $i = (t \gg n) \& 0xF$ . That is, the key ID is the last 4 bits of  $t$  after removal of the lower  $n$  bits; the key ID changes every  $2^n$  seconds. To compute a temporal secret  $s$  as in Fig. 3, a call to  $\text{GETTIME}(i) = ((t \gg n) \& 0xFFFFFFFF) | i$ , which returns the time value that corresponds to the given key ID.

to generate valid bindings. Periodic key expiration ensures that third parties cannot use temporal secrets indefinitely. In addition, any temporal secret key may be revoked by the resource principal through communication with the key server as will be described in Section IV-B.4.

Unfortunately, since bindings include almost all the invariant contents of a packet, intermediate nodes are restricted in power. For example, since the binding covers the payload (including TCP port numbers) Platypus packets are not compatible with port-altering network address translators (NATs), nor can they be fragmented. We do not consider the inability to fragment a significant limitation, as hosts typically perform path MTU discovery for all destinations. The NAT restriction, however, may be more significant. Any port-altering NATs traversed by Platypus packets on their way to a waypoint must be Platypus-aware. Once a packet has passed through its final Platypus waypoint, however, it may pass through NATs without ill effect. Similarly, packets may traverse any number of NATs before being stamped. Since most NATs are deployed at the edges of networks, the above suffices when packets are stamped inside the network. End hosts wishing to stamp their own packets, however, cannot be behind a port-altering NAT.

## IV. CAPABILITY MANAGEMENT

Platypus gains significant flexibility from the ability to transfer capabilities. Entities can collect capabilities from multiple resource principals and construct source routes to which no single entity would otherwise have rights. We describe capability management in several steps: First, we detail how capabilities are generated both in general and in special cases. Second, we describe how resource principals obtain temporal secrets for their own capabilities and capabilities delegated to them by others. Third, we present a policy framework for applying capabilities to IP packets.

### A. Capability Generation

While capabilities are generally minted by an ISP, there are two important cases when individuals may wish to create new capabilities based on those provided to them by their ISPs.

1) *Reply Capabilities*: Protocols such as TCP have been shown to work best when forward and reverse path characteristics are similar [6]. In order to use Platypus source routes, however, both ends of a flow must have their own capabilities and perform their own routing. Fortunately, it may often be the case that one of the communicating parties may wish to be responsible for both directions of the flow. For example, a client may wish to provide a server with a capability to enable the server to provide it with better performance. Platypus allows for resource principals to include a *reply capability* and its

corresponding temporal secret as part of a packet stream for the recipient to use in response.

For concreteness, we describe reply capabilities in the context of an HTTP flow. Suppose the client possesses a capability to route through some Platypus router to reach a Web server. The client wishes to provide a capability to the server for reply packets back to the client. (Obviously, the server or some router near the server must support Platypus stamping to make this possible.)

There must be some degree of trust in this relationship: the client must expect that the server is going to send it useful data if it is willing to provide a capability for the traffic. However, the client may not wish to divulge its capability and temporal secret key entirely. In particular, the client may want to transfer the appropriate capabilities with a restriction that they be used only to send packets to its address. Thus, the server would only be able to use the restricted capability to route to the client, who would be able to detect any abuse. Such a restricted delegation mechanism is of use in a more general setting; we turn to this problem next, and use the fully restricted variant for reply capabilities.

2) *General Delegation*: In general, a resource principal may want to specify a particular IP address prefix to which a third party may send packets using the principal's capability. Furthermore, the third party should be able to sub-delegate (specify a subnet of the previously delegated prefix) the capability without needing to contact the resource principal or key server. For example, as part of its services, a data center may offer (delegated) capabilities to owners of servers hosted at the data center that the server owners can then pass on to their clients (sub-delegation). To prevent abuse, the data center would like these delegated capabilities to be used only to send traffic to the data center. In turn, each server owner would want its customers to use its capabilities only to send traffic to its servers (and not to other servers hosted at the same data center). Platypus therefore allows the minting of *delegated capabilities*, which are derived from normal (or previously delegated) capabilities, but limited in their scope.

To facilitate the use of delegated capabilities, we extend the capability format as follows. First, when a packet is stamped with a delegated capability, a bit is set in the flags field of the capability specifying that the capability is a delegated capability. Immediately before the associated binding, the stamper places the constraining prefix (a 32-bit value), the prefix length (an 8-bit value), and a delegation ID (a 24-bit value). These values allow a Platypus router to verify both that the binding is valid and that the "next hop" of the packet (waypoint or final destination) is within the restricted prefix. Note that it is essential to check that the next waypoint in the capability list is within the restricted prefix in order to prevent hosts from colluding with fake waypoints to misuse delegated capabilities. Finally, the resource principal and ISP can use the delegation ID to track the use of delegated capabilities.

Here we present one protocol for building delegated capabilities—chaining delegation—which is simply a multi-round variant of the double-MAC, once again under the assumption

that the MAC is a secure pseudorandom function.<sup>2</sup> Similar to the manner in which routers generate secrets to give to resource principals, each party that wishes to delegate computes a delegation key  $d = \text{MAC}_s(\text{id})$  where  $s$  is an ordinary capability secret and  $\text{id}$  is the delegation ID. Next, for each bit  $p_i$  of the constraining prefix  $p$ , the party further restricts the delegation key by iteratively computing  $d = \text{MAC}_d(p_i)$ . Other parties can be given the key  $d$  to compute bindings for packets; they can also subdelegate through the same delegation key generation approach by iteratively constraining prefix bits.

## B. Capability Distribution

There are three main aspects to wide-area capability distribution: bootstrapping, lookup, and revocation. We describe our approaches to each in turn.

1) *Bootstrapping*: To bootstrap the capability distribution process, we expect that each AS provides an interface (likely a Web server) through which resource principals establish their accounts. This can occur in many ways. For example, the server and resource principal can set up a secure channel (using SSL, for example), and, after negotiating payment, the server sends a resource principal ID, randomly generated capability master key  $c_k$ , and the capability information to the resource principal.

2) *Ordinary Capability Lookup*: To look up the current temporal secret  $s$  associated with a capability, a resource principal generates a request by encoding the capability and a special request opcode as a string and prepends it to the key-lookup subdomain (specified during the bootstrap process) in a DNS TXT lookup request, which is routed by DNS to an appropriate key server. For example, a request for a capability issued by `ucsd.edu` with key-lookup subdomain `platypus.ucsd.edu` would be `<request>.platypus.ucsd.edu`. The DNS response is a similarly encoded DNS TXT record containing the temporal secret for the requested key ID encrypted under the capability master key. The resource principal decrypts and verifies the response, yielding the current temporal secret  $s$  for the specified capability.

The use of DNS for key lookup may seem clumsy; a more natural approach might be to contact the key server directly. To contact the server, however, a resource principal would have to first perform a DNS lookup for the key server and then transmit its lookup request, requiring multiple round trips. Instead, Platypus piggybacks the request for a key, shortening the lookup latency to about one RTT, allowing for extremely short expiration intervals. By using DNS to distribute keys, Platypus realizes caching, distributed authority, and failure resistance without having to build a separate key distribution infrastructure. In particular, Platypus key lookups are cacheable since requests are plain text and replies are encrypted under the capability master key for the requested capability. If multiple requests are made for the same shared capability, DNS caching will automatically decrease the load on the key server.

3) *Delegated Capability Lookup*: Lookup of delegated capabilities is fundamentally different from ordinary capability

<sup>2</sup>We can alternatively use a block cipher instead of a MAC for generating delegated keys.

lookup: parties must receive capabilities from a resource principal rather than from a capability server. We have devised a DNS-based mechanism that allows a server to distribute delegated capabilities to clients, leveraging the DNS lookup that typically precedes client-server exchanges on the Internet. If both the client and the server are Platypus-aware, the server can delegate a capability to the client as follows. Suppose a client wishes to contact a server `server.ucsd.edu`. Normally, a DNS resolver near the client issues a DNS query asking for the A record (IP address record) for `server.ucsd.edu`, which eventually is answered by the name server authoritative for `ucsd.edu`. Instead, we have the resolver issue a query for a TXT record for `deleg.server.ucsd.edu` (that is, it prepends `deleg` to the DNS name). The DNS server recognises this as a request for (a) the IP address of `server.ucsd.edu` and (b) a delegated capability for sending traffic to `server.ucsd.edu`; it returns a TXT response to the resolver containing these two items. The DNS resolver installs the received delegated capability in a client-side stamper and returns the address to the client; the stamper can subsequently stamp traffic from the client to the server. The server remains compatible with non-Platypus-aware clients by answering A queries in addition to TXT queries. Similarly, the client's resolver remains compatible with non-Platypus-aware servers by requesting an A record if no TXT record is returned.

An attacker may attempt to intercept a DNS response containing a delegated capability and use the delegated capability to flood the server, overwhelm the waypoint, or incur cost for the server's resource principal. Such an attack, however, is not fundamentally different from an ordinary flood attack against the server's access link. Alternatively, an attacker may tamper with the TXT record in the DNS response and insert a different delegated capability to divert traffic intended for the server to another location. Once again, this attack can be accomplished today by modifying a DNS A record reply.

4) *Revocation*: While expiration provides for coarse-grained control of temporal secrets, a resource principal may want to immediately revoke the current temporal secret when it suspects compromise. Platypus enables such revocation: to revoke a particular temporal secret, the resource principal computes the MAC of the capability and the current time under the capability master key and sends the {capability, time} pair, MAC, and the revocation opcode encoded as a DNS request. Platypus routers periodically receive updated revocation lists from their associated key servers which they consult whenever validating packets. The revocation list for the current key ID is flushed upon key ID rotation.

### C. Policy

So far we have discussed the mechanisms for stamping and delegation, deferring questions such as (a) how a stamper decides to stamp a particular packet and with which capabilities, (b) how a resource principal decides to delegate a capability to a peer, and (c) how its peer decides to accept a delegated capability. We now present a per-AS policy framework designed to address these questions.

Fig. 4 shows an instance of our policy framework in an in-network stamping scenario that uses DNS-based delegation. Cen-

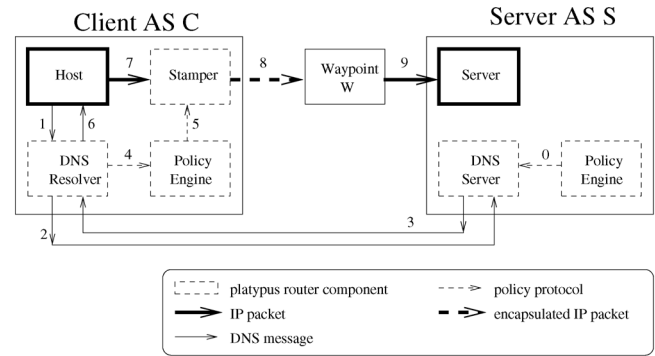


Fig. 4. Delegation and stamping in our policy framework.

tral to the framework is the *Policy Engine*, which implements the AS's policy. The policy engine instructs the stamper and DNS components based on policy, and obtains delegated capabilities through them. The stamper and policy engine are implemented inside a *Platypus router*, which is located on the path of inbound and outbound traffic. (Multihomed ASes would use several Platypus routers.) In Fig. 4 we assume AS *S* hosts a server and owns a capability *c* for waypoint *W*. AS *S* wishes clients in AS *C* to use this capability when they communicate with the server, and has therefore installed *c* and the policy in the policy engine.

We illustrate the interaction between these components by walking through the steps shown in the figure. First, *S*'s policy engine derives a delegated capability *d* and sends it to the DNS server, instructing it to "receive traffic from *C* through waypoint *W*" using *d* (Step 0). Next, a host in AS *C* wishes to contact the server and issues a DNS A query to *C*'s DNS resolver, which transforms it into a TXT query and forwards it to *S*'s DNS server (Steps 1-2). At this point the DNS server includes *d* in the DNS TXT response to the DNS resolver (Step 3), complying with the directive it received from the policy engine in Step 0. The DNS resolver passes any received delegated capabilities to *C*'s policy engine (Step 4), whose responsibility it is to select delegated capabilities for use. In this example we assume that AS *C*'s policy is to accept and use any delegated capabilities sent by peers. Therefore, the policy engine passes *d* to the stamper and tells it to "send traffic to *S* through *W* using *d*" (Step 5). Meanwhile, the DNS resolver also sends a DNS A response to the host (Step 6), which can now start sending traffic to the server (Step 7). Traffic to the server is stamped by *C*'s stamper and passes through *W* (Steps 8-9), in accordance with both ASes' policy.

## V. IMPLEMENTATION

We have built prototype software components for UNIX that provide Platypus stamping, key distribution, distribution of delegated capabilities, policy specification, and forwarding services. Fig. 5 depicts some of the key components in our prototype. Each is described in turn below.

### A. Forwarding and Stamping

We have implemented Platypus forwarding and stamping functionality as user-space daemons, (`prd` and `psd`), which runs in Linux and on Planetlab, and as Linux kernel modules,

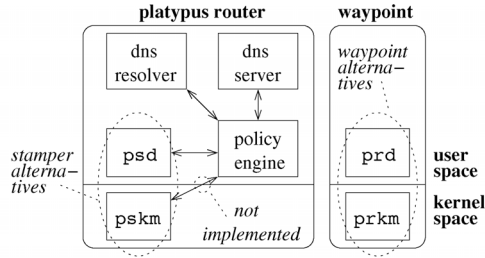


Fig. 5. Overview of implemented components.

(`prkm` and `pskm`). While `prd` implements our full policy framework, user-level packet capture and forwarding requires multiple user/kernel context switches, resulting in poor forwarding performance. Thus, we use `prkm` to better the potential forwarding performance of an in-kernel implementation. `prkm` processes Platypus packets entirely inside the kernel. Upon a packet arrival, in the kernel soft-IRQ context, `prkm` verifies the packet; if the binding is valid, the packet is updated and forwarded. By binding interrupt handling for different network interfaces to different CPUs on a machine, `prkm` can provide good scaling across multiple processors.

### B. Distribution of Delegated Capabilities

We implemented DNS-based distribution of delegated capabilities (Section IV-B.3) using the Poslib DNS library [33]. We leverage deployed DNS infrastructure by deferring DNS lookup work to existing DNS resolvers and servers, only performing transformations on DNS messages. For example, when a Platypus DNS server receives a TXT query, it transforms the query into the corresponding A query and lets a local conventional DNS server handle the query. On receiving the A response from the DNS server, the Platypus DNS server transforms the response into a TXT response, includes delegated capabilities as needed, and replies to the query.

### C. Policy

The policy engine is implemented as a user-level process that communicates with other components in the Platypus router using an XDR-based [41] *policy protocol*. The policy protocol allows the engine to give instructions to the stamper and DNS server (Steps 0 and 5 in Fig. 4) and receive delegated capabilities from a DNS resolver (Step 4). The engine’s instructions currently resemble the rule set of a firewall. (In reality these would be derived from high-level objectives created by an AS administrator.)

The policy specification supports *prefix-based* matching of traffic, allowing traffic to be sent to and received from specific remote ASes through specified waypoints, and *load-sharing*, allowing some proportion of traffic to be forwarded through specified waypoints. This proportion is specified as a percentage of packets or flows (for the stamper) or as a percentage of DNS queries (for the DNS server). The policy protocol consists of messages that update a database of rules and (delegated) capabilities stored by the stamper. Rules and capabilities in the data-

TABLE II  
EXAMPLE POLICY DATABASE

Rule	Source Prefix	Dest Prefix	Prob	Action
1	0.0.0.0/0	132.239.50.184/32	1.0	apply 1
2	0.0.0.0/0	132.239.50.184/25	1.0	apply 0
3	130.37.30.7/32	132.239.50.0/24	1.0	apply 2

TABLE III  
DATABASE USING THE ‘PROBABILITY’ FIELD AND ‘GOTO’ ACTION

Rule	Source Prefix	Dest Prefix	Prob	Action
1	none	not 132.239.50.184/32	1.0	goto 4
2	none	none	0.4	apply 1
3	none	none	1.0	apply 0
4	...			

base are identified using numeric identifiers that are assigned by the policy engine.

Next we highlight the key features of the policy database using the example databases in Tables II and III. Table II defines three rules. When given a packet to forward, the stamper searches the rules in order of rule ID until it finds a match. Rule 1 specifies that traffic from any address (a zero-length IP prefix) to 132.239.50.184 must be stamped with capability 1 (*apply 1*). Rule 2 specifies that any other traffic to an address in 132.239.50.184/25 should not be stamped: capability ID 0 denotes *do not stamp*. Rule 3 specifies to stamp packets to addresses in 132.239.50.0/24 using capability 2, but only if sent by host 130.37.30.7. Finally, if none of the rules match, the packet is not stamped.

Table III highlights the *probability* field—a value in the range [0,1]—that performs an action with the given probability. If the action is not taken, searching continues at the next rule. Using the ruleset in Table III, the stamper applies capability 1 to 40% of packets destined for 132.239.50.184 (rule 2). The other 60% of traffic for 132.239.50.184 is not stamped (rule 3). Remaining traffic is processed beginning from rule 4 (*goto 4* in rule 1), which is not shown.

The DNS server uses the database in a similar manner but *infers* a source and destination IP address from a DNS query that it receives; it expects that upon completion of the query, the host making the request will use the inferred addresses when sending traffic. The DNS server determines the destination address as the IP address that it returns in the response to a query. Similarly, we would like the DNS server to determine the IP address of the querying host and use that as the source address, but, in general, the querying DNS resolver may not be the originating host. However, we expect resolvers and DNS servers that perform recursive queries to be relatively close to the querying hosts, and that therefore the IP address of the querying host and that of the resolver are related (such as part of the same, small IP prefix) [22]. The DNS server thus uses the address of the querier as the traffic source IP address when searching its rules. This behavior also makes it safe for DNS to cache TXT records.

The policy database is structured and searched in a similar manner to a firewall’s rule database and can be similarly optimised using techniques such as ternary CAMs.

#### D. Protocol Interactions

We have attempted to design around possible negative interactions between Platypus and existing protocols. In particular, proper ICMP delivery is complicated by source routing. Since ICMP responses can occur for many reasons, the appropriate recipient of such messages can be ambiguous. For example, should an ICMP time expired message be sent to the last Platypus waypoint in the source route, the stamper, or the original source? The cause of such expiration may be due to in-network stamping or other problems such as routing loops. Further complicating the matter, non-Platypus routers may generate ICMP responses for source-routed packets and send them to the last waypoint in the source route. In both of the two primary cases—end-host stamping and in-network stamping—the end-host perceives its Platypus-enabled connectivity to be the same as ordinary network connectivity, thus we send all ICMP packets back to the original source address. The first 64 bits of the Platypus header contain the original source address, enabling RFC-compliant routers to include the original source address in ICMP error response packets; Platypus routers forward such ICMP packets along to the source, subject to standard ICMP rate limiting.

### VI. EVALUATION

In this section we consider the standalone performance of our prototype router and stamper, how many waypoints are needed in a real-world deployment, and how Platypus stamping can be used to improve the performance of wide-area file transfers.

#### A. Forwarding and Stamping Performance

Our experimental testbed consists of a central Linux-based router that performs both forwarding and stamping and several load generators connected through a gigabit Ethernet switch. The server is configured with two 64-bit, 2.2-GHz AMD Opteron 248 processors, two GB of PC2700 DDR memory, and three Intel Pro/1000 XT gigabit NICs; our tests used two of the NICs installed on a 100-MHz, 64-bit PCI-X bus. The load generating machines have 1.1-GHz Pentium III processors and Intel Pro/1000 XT gigabit NICs.

First we consider the absolute performance of forwarding and stamping. Fig. 6 compares the performance of Linux's in-kernel IP forwarding to `prkm`'s forwarding performance and `pskm`'s stamping performance for worst-case (minimum-size) packets. For forwarding tests, the load generators each direct identical 68-byte (20-byte IP header + 28-byte Platypus header + 20-byte TCP header, excluding the Ethernet header) Platypus packets at the router which validates the bindings and forwards the packets to the indicated waypoint. For stamping, the load generators send 40-byte packets which are stamped and forwarded by the router (by insertion of the 28-byte Platypus header with a capability and binding). To increase the offered load in a controlled fashion, we first saturate one router interface and then load the two interfaces at equal levels.

As seen in the figure, `prkm` is capable of forwarding packets with full UMAC authentication [8] at a maximum loss-free forwarding rate of approximately 767 Kpps (using a warm UMAC context cache; initializing the context takes 41.3  $\mu$ s),

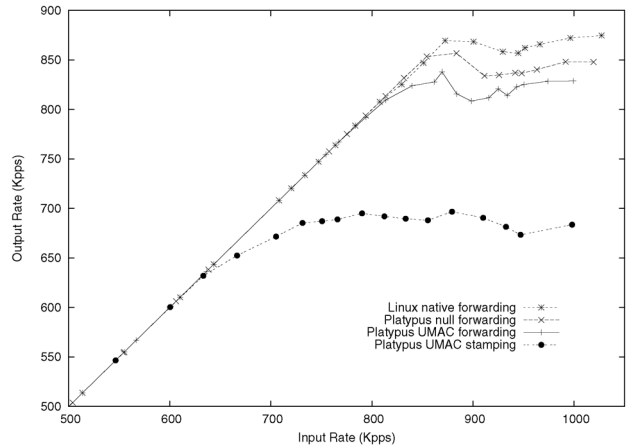


Fig. 6. 68-byte packet forwarding/stamping performance.

which is only slightly less than the performance of native Linux. To help calibrate for the fact that the kernel's forwarding code is more streamlined than that of `prkm`, we plot the performance of the `prkm` forwarding path without verification (labeled *null forwarding* in the figure). The results indicate that a significant portion of the performance degradation is due to factors other than capability verification. When forwarding MTU (1500-byte) packets, `prkm` is able to fully validate at approximately 2.5 Gbps without loss. Stamping performance is slightly worse: `pskm` is capable of loss-free stamping at approximately 633 Kpps. These results indicate that Platypus software routers and in-network stampers can yield good performance on modern hardware, enabling low-cost deployment of Platypus.

In addition to absolute forwarding numbers, we measured the amount of time actually spent validating bindings, as this latency may be observed by end hosts (as opposed to forwarding performance, which is largely a concern of the ISP). Table IV shows micro-benchmarks of `prkm` in its several stages. We performed these measurements by averaging three runs of `prkm` forwarding 1 000 000 packets spaced 16  $\mu$ s apart. For tests in which packet size affected performance, we benchmarked forwarding of 68-byte, 348-byte, and 1500-byte packets (including IP and Platypus headers, but excluding Ethernet headers), which correspond to minimum, moderate, and maximum packet sizes. Packet processing includes time to parse the packet headers, verify the binding, and update Platypus headers. Destination cache lookup includes retrieval of a `dst_entry` structure, and IP header building and verification includes time to place a new IP header on the packet. Finally, packet transmission time includes time until the packet is queued for transmission by the device.

#### B. Waypoint Deployment

We now consider the impact of waypoint deployment on the effectiveness of Platypus-like source routing. Clearly, the more numerous the waypoints, the more control Platypus can assert over a packet's path. By clustering the routers into groups which could be represented by a single Platypus waypoint, we attempt to determine the number of Platypus waypoints an ISP must deploy to provide a useful service to customers.

TABLE IV  
MICRO-BENCHMARKS FOR PRKM. ALL TIMES ARE AS MEASURED  
BY THE CPU CYCLE COUNTER

Packet size	68 byte	348 byte	1500 byte
Packet processing			
Null	172 ns	173 ns	181 ns
UMAC	695 ns	998 ns	1908 ns
Destination cache lookup	289 ns		
IP hdr build and verify	145 ns		
Packet transmission	1480 ns	1482 ns	1493 ns

In particular, we study the impact on end-to-end one-way path latency of routing indirectly through a set of waypoints; we vary the number of waypoints available. Previous research indicates that it is often possible to achieve significant performance improvements by inserting one level of indirection in a packet's route [4], [37], [21]. We consider how the best achievable path latency increases as more waypoint choices are available, as this indicates how *well chosen* waypoints must be. Intuitively, since POPs represent a collection of routers in a region, and networks are dense near large cities and sparse elsewhere, routers that have similar latencies to a given set of observation points can be naturally clustered together. It may be sufficient to place Platypus routers in only a few locations, as speed of light delays comprise most of the delay seen by packets in uncongested wide-area backbones. Thus, multiple, local waypoints would not significantly affect latency (but, conversely, might be useful for load balancing, for example).

We consider as potential waypoints router IP addresses reported by Skitter [11] for four major ISPs: MCI, Sprint, Qwest, and Global Crossing. We select five geographically diverse monitoring locations in the RON testbed [4], UC San Diego, Nortel (Nepean, Ontario, Canada), Coloco (Laurel, Maryland), Lulea (Sweden), and KAIST (Korea). From each monitoring location, we use ICMP timestamp probes to measure both the forward and reverse path latencies for each known router interface of the ISP in question [28]. This set of measurements was collected over a period of six days between January 22–27, 2004. We obtain approximately 240 measurements for each location/router pair and use the mean value. With this data, we compute a one-way, indirect end-to-end path delay between any two monitoring locations through each router interface.

As our goal is to understand the performance of limited numbers of waypoints, for each potential value, we cluster routers using  $k$ -means [27], designate their centers as waypoints, and compute the best end-to-end path latency between two locations via the optimal center router. Each  $k$ -means run is given an argument  $n$  representing the desired number of clusters, and thus the number of deployed waypoints; we have no control over how clusters form, and thus, the resulting performance is an upper bound on what can be achieved. Finally we compare the performance of the optimal cluster with the performance of the optimal router interface (which may or may not be a member of the optimal cluster).

Fig. 7 shows the results for MCI, the largest ISP we studied (8591 router interfaces); results for the other three ISP are similar [35]. As expected, the more waypoints, the closer the performance of the optimal cluster comes to performance of the op-

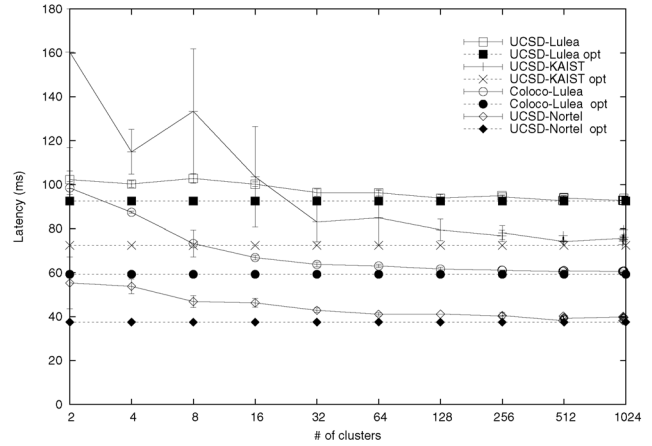


Fig. 7. We consider the impact of cluster size on indirection effectiveness for MCI (UUNet). We vary the number of clusters generated based upon observed latencies between the two specified source/destination points and every known router interface. For each indicated source/destination pair, we plot the measured one-way path latency using the ISP's optimal indirection router (opt) against the calculated path latency through the center of the optimal cluster. Data points represent averages; ten different clusterings were generated for each  $k$ -means input size. Error bars show the standard deviation.

timal router. Somewhat surprisingly, however, the best cluster centers approach the optimal at a relatively small number of clusters, suggesting that a small number of indirection points are likely sufficient for substantial benefit; this applies equally to Platypus and any overlay or source-routing system; this is likely due to geographic or POP locality among potential waypoints.

### C. Waypoint Overlay Routing Performance

Next, to demonstrate the applicability of Platypus to common wide-area network problems such as violations of delay-based triangle inequalities, we consider improving the performance of wide-area file transfers over PlanetLab using Platypus. As has been shown in the past [4], [37], we can improve the end-to-end latency of some Internet paths via overlay routing; we implement this indirection via a Platypus waypoint.

To evaluate the potential performance gains, we consider 1 MB file transfers between two PlanetLab nodes: a client at Cornell and a server at Laboratoire d'informatique de Paris 6; the RTT between these nodes on Oct. 31, 2006 was 305 ms. We select a PlanetLab node at TU Ilmenau in Germany as a potential Platypus waypoint; the indirect RTT between Cornell and Paris through this waypoint was 140 ms. Upon each file transfer initiation, the client resolves a DNS name that queries the DNS name of the server. The DNS request is handled by a Platypus DNS server and subsequently cached in DNS. Using our policy framework (Section V-C) and DNS-based distribution of delegated capabilities (Section IV-B.3) we configure our DNS server with two distinct policies: *direct*, in which a DNS response to the client does not include a delegated capability, and *indirect*, in which a DNS response to the client contains a delegated capability specifying the waypoint.

Table V shows the transfer times of 50 file transfers between Cornell and Paris with and without indirection. The transfer time for non-waypoint transfers includes at most a 3.7-second DNS

TABLE V  
TRANSFER TIMES (IN SECONDS) OF 1 MB FILE TRANSFERS BETWEEN  
PLANETLAB NODES AT CORNELL AND PARIS WITH OR WITHOUT  
INDIRECTION THROUGH A PLATYPUS WAYPOINT IN GERMANY

	5%	median	95%
Direct	13.9	14.1	16.9
Indirect	5.06	5.22	6.35

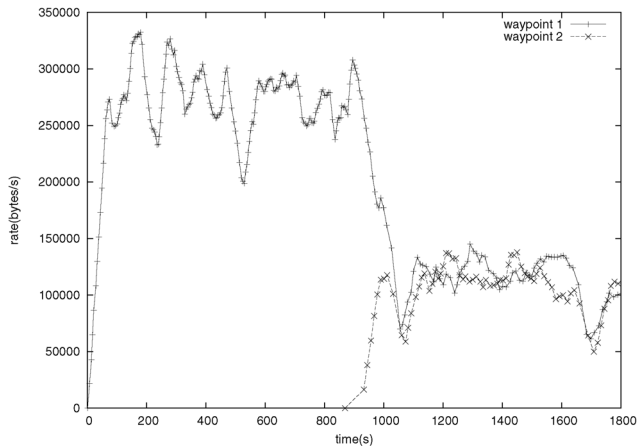


Fig. 8. Load-balancing across two waypoints. During the first half of the experiment the policy is to send traffic through waypoint 1. During the second half the policy load-balances across both waypoints at a per-flow granularity.

lookup overhead (due to a DNS failover after it fails to find a TXT record for the delegated capability). Even if we account for this inefficiency we find over 50% improvement in transfer time when using the waypoint.

#### D. Waypoint Load Balancing

In all our application scenarios (from Section II-A), Platypus users forward their traffic through selected waypoints. Consider, for example, a Web site that purchases Platypus service from an ISP; traffic that the server sends to specific clients uses Platypus to selectively improve performance. However, given the popularity of the website, it may overload a single waypoint at certain times of the day. To remedy this issue, we consider a policy in which the server selects a set of waypoints to forward traffic through and load balances across them. This functionality is important in many applications, since it is unlikely that a single waypoint can suffice for an arbitrarily large traffic volume.

Using the Platypus policy framework described in Section V-C, we evaluate a Web server application scenario with probabilistic load balancing across two waypoints. Each client makes ordinary HTTP requests to the server. The server's replies are stamped according to a policy that begins by sending all response traffic through a single waypoint. Halfway through the experiment we change the policy such that the response traffic is load balanced at the granularity of a TCP flow. Fig. 8 shows the effect of the policy change in the traffic demands at each waypoint.

## VII. DISCUSSION

During the design of Platypus, we have considered issues of performance, security, accounting, the effect of source-routed

traffic upon the network, and alternative means of capability delegation. In this section we discuss these considerations.

#### A. Distributed Accounting

Fine-grained flow accounting is an established problem in other contexts, but Platypus complicates the use of several common approaches. For example, many end hosts receive flat-rate pricing for their Internet service. ISPs can provide this service with bounded risk because the rate at which an end host can inject packets into the network is limited by the capacity of its access link. More sophisticated pricing plans may depend on the actual utilization, which requires the ISP to meter a customer's traffic, but such metering can be done at the customer's access link.

In Platypus, however, a customer may authorize third parties to inject packets into its ISP as part of a source route. Any accounting scheme that only charges customers for packets that traverse their access link clearly will not properly account for the customer's additional use. A straightforward approach would maintain counters for each resource principal at all Platypus routers within an AS, and bill for the total consumption. While auditing challenges may dissuade many ISPs from per-packet accounting, aggregate rate limiting is likely to be needed to support those customers that wish to pay a flat rate for a fixed amount of bandwidth. In order to implement such a pricing model in Platypus, an AS must have some way to restrict bandwidth consumption for a particular resource principal at one or more routers. We have developed mechanisms for such distributed rate limiting in separate work [36].

#### B. Replay Attacks

For rate-based accounting, we can constrain resource principals to a fixed, aggregate bandwidth. However, while packet bindings cannot be forged (modulo standard cryptographic hardness assumptions), they may be replayed by an adversary, who may wish to waste a resource principal's limited bandwidth for a given capability. Since capabilities expire periodically, a natural countermeasure to replay attacks is to track packets that traverse a router within some time window and only count each distinct packet once. A Bloom filter allows for tracking of packets in such a way, but may fill up over time, resulting in false positives. This issue can be addressed by maintaining a small circular array of Bloom filters which are cleared as they fill up [2], [38]. While an adversary may be able to log all packets and replay them after the corresponding Bloom filter is emptied, if the filters are emptied only at key expiration intervals, stored packets cannot be replayed.

#### C. Scalability

Previous work [3] has explored the use of access control lists for network resources. In contrast, a Platypus router does not need to keep track of permissions for end hosts, potentially providing for greater scalability. In particular, by using capabilities, Platypus is able to implement capability delegation without involving Platypus routers or key servers. The downside, of course, of capabilities is communication overhead (28 bytes per-packet in our prototype).

Careful selection of MAC algorithms is crucial for peak verification performance. We use UMAC in our software implementation, but expect PMAC [9] would be selected for hardware implementations. Since MAC computations are done with local information only, capability issuers can choose a MAC algorithm appropriate to their forwarding hardware or software. In addition, Platypus’s double-MAC design requires constant state for capability verification, regardless of the number of resource principals. ISPs may wish to keep additional accounting state for billing purposes, however. In the extreme case of per-packet billing, an ISP would need to keep a packet counter corresponding to each resource principal. While deployments of Platypus in the core may only need to handle a few thousand resource principals, deployments for a broadband ISP may have several million. Fortunately, Kumar *et al.* have shown that approximate counters with bounded error can be maintained per flow at very high speeds (OC-768) [25].

We contend that Platypus key management can also be scaled to support large numbers of resource principals. For key distribution, it is unlikely that all requests will arrive exactly at key ID-change boundaries, since Platypus does not require tight time synchronization between resource principals and routers. Even in such an unlikely event, Platypus key servers need only perform two MAC and one block-cipher calls for each request; servicing ten million such requests in one second is well within the limits of approximately 20 well-provisioned key servers. Furthermore, since key lookup requests and responses are small, each lookup requires only one packet receipt and transmission on the part of a key server.

Key servers must periodically distribute revocation lists to Platypus routers; while distribution can occur off the critical path, lookup cannot, so revocation lists must be stored in high-speed memory. In our current design, each revocation entry is twelve bytes, so a 16-MB SRAM chip could store about 1.4 million revoked capabilities.

A final aspect of scalability is the administrative overhead imposed on a source routing network that keeps track of available waypoints and their performance metrics. While it might appear that a network must keep in touch with the thousands of ASes that exist in the Internet, we argue that the ability to delegate capabilities allows capabilities to be resold by third parties that can take on the burden of waypoint discovery and performance measurement.

#### D. Traffic Engineering

Conventional wisdom holds that widespread source routing deployment would complicate traffic-engineering efforts. While there admittedly is cause for concern, we have reasons for optimism. Recent simulations by Qiu *et al.* show that while source-routed traffic can have deleterious interactions with intra-AS traffic engineering mechanisms in extreme cases, certain techniques may be able to mitigate these effects [34]. In their studies, however, source-routed traffic was capable of completely specifying intra-AS paths. Our design for Platypus is meant to allow ISPs to specify any globally routable IP address within their IP space as a Platypus waypoint and dynamically adjust the actual (set of) internal router(s) to which the IP corresponds in response to traffic load. By dilating waypoints in this way, an

ISP can meet its traffic engineering goals while delivering improved service to end hosts; we discuss this in greater detail in an earlier version of this work [35]. In addition, an ISP has the option of pricing capabilities in a way that attracts traffic to lightly loaded links or that compensates for the use of links that have little spare capacity.

Independent of its interaction with traditional traffic engineering, Platypus opens up a new dimension for traffic provisioning: time. Routing in today’s Internet has no temporal dimension—the advertisement of a route makes it immediately available. With Platypus, however, routes may have time-limited availability; that is, a route is available only when users possess the correct temporal secrets. By appropriately choosing expiration intervals and expressing route selection policy upon key lookup, ISPs can control the temporal aspects of traffic flow; in this way, Platypus may even serve to help achieve traffic engineering goals.

#### E. Alternatives for Capability Distribution

The use of DNS for distribution of delegated capabilities (Section IV-B.3) is suited to a usage model in which a server is interested in delegating a capability to a large number of clients. While this design choice entails modifications to DNS servers and resolvers, we have found that the required changes can be made in a modular fashion, i.e., without making DNS implementations more complex. By using interposition as described in Section V-B, we maintain the separation of concerns between domain administration and capability management. Indeed, while testing capability distribution for the UCSD website, we leveraged both UCSD DNS servers and our department’s DNS resolvers without having (or needing) to modify either.

Alternatively, the server can opt to use *in-band distribution*, which is designed for transmitting delegated capabilities from receivers to senders of particular flows and does not distinguish between client and server roles. Consider a Platypus-aware traffic receiver  $R$  with IP address  $dst$ —we show how  $R$  transmits a delegated capability to a Platypus-aware traffic sender  $S$  with IP address  $src$ . The mechanism is based on inserting “Platypus signaling packets” within the flow. A Platypus signaling packet is an IP packet that has the same source and destination addresses as the flow but uses a Platypus transport protocol. Thus, the signaling packet follows the same forwarding path as the flow.  $S$  periodically inserts a *delegation listen packet*, which contains a randomly generated key  $k_S$ , into the flow, advertising that it is capable of receiving delegated capabilities.  $S$  also stores the time at which it generated  $k_S$ . In response,  $R$  inserts a *delegation packet* containing the delegated capability  $c, t, dst$  and  $H_R = \text{MAC}_{k_S}(c)$ . Upon receiving the delegation packet,  $S$  verifies  $H_R$  and checks that the corresponding key  $k_S$  is recent. This process ensures that only parties on a recent default forwarding path from the  $S$  to  $R$  can have created the delegated capability, and thus prevents unauthorised diversion of packets.

## VIII. RELATED WORK

Source routing has been included as a feature in many Internet architectures over the years. For example, Nimrod [13] defined mechanisms for packets to be forwarded in both

flow-based and source-routed, per-packet fashions. Similarly, IPv6 provides support for the source demand routing protocol, SDRP [17]. SDRP allows for hosts to specify a strict or loose source route of ASes or IP addresses through which to route a packet. More recently, Yang described a new addressing architecture called NIRA [45] with the explicit goal of providing AS-level source routing. NIRA path selection consists of two stages: an initial discovery phase followed by an availability phase in which a host determines the quality of a particular route. A contemporary proposal, BANANAS, allows for explicit path selection in a multi-path environment, but does not allow for the insertion of arbitrary intermediate hops [44]. None of these proposals, however, have addressed the need to verify policy compliance of the specified route *on the forwarding plane*. To the best of our knowledge, we are the first to present a fully decentralized, authenticated source-routing architecture.

Frustrated with the lack of control provided by current wide-area Internet routing, researchers have proposed circumventing it entirely by forwarding packets between end hosts in an effort to construct routes with more desirable path characteristics [4], [37]. Unfortunately, the effectiveness of any overlay-based approach is fundamentally limited by both the number and the locations of the hosts involved in the overlay. We believe Platypus addresses both of these issues: overlay networks can view far away Platypus routers as additional members of the overlay and use nearby Platypus routers to increase the efficiency of their forwarding mechanisms.

Stoica *et al.* suggest that indirection be explicitly supported as an overlay network primitive; in the Internet Indirection Infrastructure (*i3*) packets may include a set of indirection points through which they wish to be forwarded [42]. Unlike Platypus waypoints, however, *i3* IDs specify logical entities, not necessarily network routing hops. Each ID is associated with one or more application-installed triggers that can involve arbitrary packet processing; there are no guarantees about the topological location of the overlay node(s) responsible for a particular ID.

Packet-level authentication credentials have been suggested in a number of other contexts. IPsec-enabled packets may contain an authentication header with information similar to a network capability [5], except without a routing request. In order to verify authentication headers, however, IPsec routers must hold one key for each source, far more than with Platypus. Per-packet authenticators have also been proposed to prevent DoS attacks [3], [12], [46]; it would be straightforward to implement a similar scheme using Platypus.

Perhaps the most closely related use is due to Estrin *et al.*, who introduced the notion of visas that confer rights of exit from one organization and entry into another [18]. Stateless visas provide a mechanism for per-packet authentication between two independent organizations, but not for expressing routing requests. Visas are the result of a bilateral agreement between a packet's source and destination; each packet contains exactly two visas—one for the source organization and one for the destination. In contrast, network capabilities are concerned with authentication and routing through intermediate ASes. In a subsequent paper [19], the authors also considered implementing preventative security measures within Clark's policy routing framework [14].

## IX. CONCLUSIONS AND FUTURE WORK

We argue that capabilities are uniquely well-suited for use in wide-area Internet routing. The Internet serves an extremely large number of users with an even larger number of motivations, all attempting to simultaneously share widely distributed resources. Most importantly, there exists no single arbiter (for example, a system administrator or user logged in at the console) who can make informed access decisions. Moreover, we believe that much of the complexity of Internet routing policy stems from inflexibility of existing routing protocols. We aim to study how one might implement inter-AS traffic engineering policies through capability pricing strategies. For example, an AS with multiple peering routers that wishes to encourage load balancing may be able to do so through variable pricing of capabilities for the corresponding Platypus waypoints. While properly modeling the self-interested behavior of external entities may be difficult, we are hopeful that this challenge is simplified by the direct mapping between Platypus waypoints and path selection (as compared, for example, to the intricate interactions of various BGP parameters).

## ACKNOWLEDGMENT

The authors thank A. AuYoung, M. Bellare, N. Feamster, R. Mahajan, D. Micciancio, T. Newhouse, S. Panjwani, S. Ramabhadran, J. Rexford, C. Tuttle, A. Vahdat, K. van der Merwe, and D. Wetherall for helpful discussions and feedback. The authors are indebted to N. Alldrin for his help with *k*-means clustering and to D. Andersen for the use of the RON testbed. Finally, the authors would like to thank K. Calvert and the anonymous reviewers for their comments.

## REFERENCES

- [1] S. Agarwal, C.-N. Chuah, and R. H. Katz, "OPCA: Robust interdomain policy routing and traffic control," in *Proc. IEEE OPENARCH*, Apr. 2003, pp. 55–64.
- [2] M. K. Aguilera, M. Ji, M. Lillibridge, J. MacCormick, E. Oertli, D. G. Andersen, M. Burrows, T. Mann, and C. A. Thekkath, "Block-level security for network-attached disks," in *Proc. USENIX FAST*, Apr. 2003.
- [3] D. G. Andersen, "Mayday: Distributed filtering for Internet services," in *Proc. USITS*, Mar. 2003.
- [4] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. T. Morris, "Resilient overlay networks," in *Proc. ACM SOSP*, Oct. 2001.
- [5] R. Atkinson, "Security architecture for the Internet protocol," in *IETF, RFC 1825*, Aug. 1995.
- [6] H. Balakrishnan, V. N. Padmanabhan, and R. H. Katz, "The effects of asymmetry on TCP performance," in *Proc. ACM Mobicom*, Sep. 1997.
- [7] M. Bellare, R. Canetti, and H. Krawczyk, "Pseudorandom functions revisited: the cascade construction and its concrete security," in *Proc. IEEE FOCS*, 1996, pp. 514–523.
- [8] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway, "UMAC: Fast and secure message authentication," in *Advances in Cryptology (CRYPTO'99)*, 1999, vol. LNCS 1666.
- [9] J. Black and P. Rogaway, "A block-cipher mode of operation for parallelizable message authentication," in *Advances in Cryptology (EUROCRYPT'02)*, 2002, vol. LNCS 2332.
- [10] M. Caesar and J. Rexford, "BGP policies in ISP networks," *IEEE Network*, vol. 19, no. 6, pp. 5–11, Nov. 2005.
- [11] CAIDA Skitter Project. [Online]. Available: <http://www.caida.org/tools/measurement/skitter/>
- [12] M. Casado, T. Garfinkel, A. Akella, D. Boneh, N. McKeown, and S. Shenker, "SANE: A protection architecture for enterprise networks," in *Proc. ACM/USENIX NSDI*, May 2006.
- [13] I. Castañeyra, N. Chiappa, and M. Steenstrup, "The Nimrod routing architecture," in *IETF, RFC 1992*, Aug. 1996.
- [14] D. D. Clark, "Policy routing in Internet protocols," in *IETF, RFC 1102*, May 1989.

- [15] D. D. Clark, J. Wroclawski, K. R. Sollins, and R. Braden, "Tussle in cyberspace: Defining tomorrow's Internet," in *Proc. ACM SIGCOMM*, Aug. 2002.
- [16] N. G. Duffield and M. Grossglauser, "Trajectory sampling for direct traffic observation," in *Proc. ACM SIGCOMM*, Aug. 2000.
- [17] D. Estrin, T. Li, Y. Rekhter, K. Varadhan, and D. Zappala, "Source demand routing: Packet format and forwarding specification," in *IETF, RFC 1940*, May 1996.
- [18] D. Estrin, J. C. Mogul, and G. Tsudik, "Visa protocols for controlling interorganizational datagram flow," *IEEE J. Sel. Areas Commun.*, vol. 7, no. 4, pp. 486–498, May 1989.
- [19] D. Estrin and G. Tsudik, "Security issues in policy routing," in *Proc. IEEE Symp. Security and Privacy*, May 1989, pp. 183–193.
- [20] N. Feamster, J. Borkenhagen, and J. Rexford, "Guidelines for interdomain traffic engineering," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 5, pp. 19–30, 2003.
- [21] 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 *Proc. USENIX OSDI*, 2004.
- [22] K. P. Gummadi, S. Saroiu, and S. D. Gribble, "King: Estimating latency between arbitrary Internet end hosts," in *ACM SIGCOMM Internet Measurement Workshop 2002*, Nov. 2002.
- [23] G. Huston, "Commentary on inter-domain routing in the Internet," in *IETF, RFC 3221*, Dec. 2001.
- [24] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-hashing for message authentication," in *IETF, RFC 2104*, Feb. 1997.
- [25] A. Kumar, J. Xu, L. Li, J. Wang, and O. Spatschek, "Space-code Bloom filter for efficient per-flow traffic measurement," in *Proc. IEEE INFOCOM 2004*, Mar. 2004, vol. 3, pp. 1762–1773.
- [26] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian, "Delayed Internet routing convergence," *IEEE/ACM Trans. Networking*, vol. 9, no. 3, pp. 293–306, Jun. 2001.
- [27] J. B. MacQueen, "On convergence of  $k$ -means and partitions with minimum average variance," *Ann. Math. Stat.*, vol. 36, 1965.
- [28] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson, "User-level Internet path diagnosis," in *Proc. ACM SOSP*, Oct. 2003.
- [29] R. Mahajan, D. Wetherall, and T. Anderson, "Understanding BGP misconfiguration," in *Proc. ACM SIGCOMM*, Aug. 2002.
- [30] D. L. Mills, "A brief history of NTP time: Memoirs of an Internet timekeeper," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 2, pp. 9–21, 2003.
- [31] A. Nakao, L. L. Peterson, and A. Bavier, "A routing underlay for overlay networks," in *Proc. ACM SIGCOMM*, Aug. 2003.
- [32] W. B. Norton, "Internet service providers and peering," in *Proc. NANOG*, Jun. 2000.
- [33] Poslib DNS Library. [Online]. Available: <http://www.posadis.org/poslib/>
- [34] L. Qiu, Y. R. Yang, Y. Zhang, and S. Shenker, "On selfish routing in Internet-like environments," in *Proc. ACM SIGCOMM*, Aug. 2003.
- [35] B. Raghavan and A. C. Snoeren, "A system for authenticated policy-compliant routing," in *Proc. ACM SIGCOMM*, Aug. 2004.
- [36] B. Raghavan, K. Vishwanath, S. Ramabhadran, K. Yocum, and A. C. Snoeren, "Cloud control with distributed rate limiting," in *Proc. ACM SIGCOMM*, Aug. 2007.
- [37] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson, "The end-to-end effects of Internet path selection," in *Proc. ACM SIGCOMM*, Sep. 1999.
- [38] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountios, B. Schwartz, S. T. Kent, and W. T. Strayer, "Single-packet IP traceback," *IEEE/ACM Trans. Networking*, vol. 10, no. 6, pp. 721–734, Dec. 2002.
- [39] A. C. Snoeren and B. Raghavan, "Decoupling policy from mechanism in Internet routing," in *Proc. HotNets*, Nov. 2003.
- [40] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with Rocketfuel," in *Proc. ACM SIGCOMM*, Aug. 2002.
- [41] R. Srinivasan, "XDR: External data representation standard," in *IETF, RFC 1812*, Aug. 1995.
- [42] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, "Internet indirection infrastructure," in *Proc. ACM SIGCOMM*, Aug. 2002.
- [43] I. Stoica and H. Zhang, "LIRA: An approach for service differentiation in the Internet," in *Proc. NOSSDAV*, Jun. 1998.
- [44] H. Tahilramani Kaur, S. Kalyanaraman, A. Weiss, S. Kanwar, and A. Gandhi, "BANANAS: An evolutionary framework for explicit and multipath routing in the Internet," in *Proc. ACM SIGCOMM FDNA*, Aug. 2003.
- [45] X. Yang, "NIRA: A new Internet routing architecture," in *Proc. ACM SIGCOMM FDNA*, Aug. 2003.
- [46] X. Yang, D. Wetherall, and T. Anderson, "A DoS-limiting network architecture," in *Proc. ACM SIGCOMM*, Aug. 2005.
- [47] D. Zhu, M. Gritter, and D. R. Cheriton, "Feedback based routing," in *Proc. HotNets*, Oct. 2002.



**Barath Raghavan** is a graduate student in the Computer Science and Engineering Department at the University of California at San Diego. His research interests include network protocol design, applied cryptography, network security, Internet architecture, game theory, and distributed systems.



**Patrick Verkaik** received the M.S. and B.S. degrees in computer science from Vrije Universiteit Amsterdam. He is a Ph.D. student in the Computer Science and Engineering Department at the University of California at San Diego.

He has worked as a researcher at CAIDA, AT&T Research, and Microsoft Research. His research interests include wireless networking, inter-domain routing and machine learning.



**Alex C. Snoeren** (S'00–M'03) received the Ph.D. degree in computer science from the Massachusetts Institute of Technology (2003) and the M.S. degree in computer science (1997) and B.S. degree in computer science (1996) and applied mathematics (1997) from the Georgia Institute of Technology, Atlanta.

He is an Associate Professor in the Computer Science and Engineering Department at the University of California at San Diego. His research interests include operating systems, distributed computing, and mobile and wide-area networking. Prof. Snoeren has

been a member of the Association for Computing Machinery (ACM) since 1999.