Balancing Accountability and Privacy in the Network

David Naylor
Carnegie Mellon University
dnaylor@cs.cmu.edu

Matthew K. Mukerjee
Carnegie Mellon University
mukerjee@cs.cmu.edu

Peter Steenkiste
Carnegie Mellon University
prs@cs.cmu.edu

ABSTRACT

Though most would agree that accountability and privacy are both valuable, today's Internet provides little support for either. Previous efforts have explored ways to offer stronger guarantees for one of the two, typically at the expense of the other; indeed, at first glance accountability and privacy appear mutually exclusive. At the center of the tussle is the source address: in an accountable Internet, source addresses undeniably link packets and senders so hosts can be punished for bad behavior. In a privacy-preserving Internet, source addresses are hidden as much as possible.

In this paper, we argue that a balance *is* possible. We introduce the Accountable and Private Internet Protocol (APIP), which splits source addresses into two separate fields — an *accountability address* and a *return address* — and introduces independent mechanisms for managing each. Accountability addresses, rather than pointing to hosts, point to *accountability delegates*, which agree to vouch for packets on their clients' behalves, taking appropriate action when misbehavior is reported. With accountability handled by delegates, senders are now free to mask their return addresses; we discuss a few techniques for doing so.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

Keywords

accountability; privacy; source address

1. INTRODUCTION

Today's Internet is caught in a tussle [13] between service providers, who want accountability, and users, who want privacy. Each side has legitimate arguments: if senders cannot be held accountable for their traffic (e.g., source addresses are spoofable), stopping in-progress attacks and preventing future ones becomes next to impossible. On the other hand, there are legitimate anonymous uses of the Internet, such as accessing medical web sites without revealing personal medical conditions, posting to whistleblowing web sites, or speaking out against an oppressive political regime.

At the network layer, mechanisms for providing one or the other often boil down to either strengthening or weakening *source addresses*. In an accountable Internet, source addresses undeniably link packets and senders so miscreants can be punished for bad behavior, so techniques like egress filtering and unicast reverse path forwarding (uRPF) checks aim to prevent spoofing. In a private Internet, senders hide source addresses as much as possible, so services like Tor work by masking the sender's true source address.

We argue that striking a balance between accountability and privacy is fundamentally difficult because the IP source address is used both to to identify the sender (accountability) *and* as a return address (privacy). In fact, the function of the source address has evolved to be even more complex, serving a total of five distinct roles: packet sender, return address, error reporting (e.g., for ICMP), accountability (e.g., uRPF), and to calculate a flow ID (e.g., as part of the standard 5-tuple).

This paper asks the question, "What could we do if the accountability and return address roles were separated?" Our answer, the Accountable and Private Internet Protocol (APIP), does just that, creating an opportunity for a more flexible approach to balancing accountability and privacy in the network. APIP utilizes the accountability address in a privacy-preserving way by introducing the notion of *delegated accountability*, in which a trusted third party vouches for packets and fields complaints. With accountability handled by delegates, senders have more freedom to hide return addresses. We make the following contributions:

- An analysis of the roles of the source address in today's Internet.
- The definition of design options for an accountability address and the accompanying mechanisms for holding hosts accountable in a privacy-preserving way.
- An analysis of the impact of these design options on the privacy-accountability tradeoff.
- The definition and evaluation of two end-to-end instantiations of APIP.

The remainder of the paper is organized as follows. After teasing apart the various roles of the source address (§2), §3 discusses challenges in balancing accountability and privacy. §4 gives a high-level overview of APIP. §5 describes possible designs for delegated accountability while §6 analyzes their implications for privacy. §7 discusses real-world deployment

SIGCOMM'14, August 17–22, 2014, Chicago, IL, USA.

http://dx.doi.org/10.1145/2619239.2626306.

issues and presents two example end-to-end instantiations of APIP. We evaluate the feasibility of APIP in §8 and finish with a discussion of related work (§9) and conclusion (§10).

2. SOURCE ADDRESS OVERLOAD

We now investigate the roles of source addresses, since they play a key role in the seemingly fundamental conflict between accountability and privacy in the network. Source addresses today attempt to fulfill at least five distinct roles:

1) **Return Address** — This is a source address's most obvious role: the receiving application uses the source address as the destination for responses. (This is, for example, built into TCP connection establishment.)
2) **Sender Identity** — Historically, source addresses have been used as a crude (and ineffective) way of authenticating a sender or to link multiple sessions to a single "user".
3) **Error Reporting** — If a packet encounters a problem, the ICMP error message is directed to the source address.
4) **Flow ID** — Source addresses are one component of the 5-tuple used to classify packets into flows, both in the network (monitoring, traffic engineering) and at endpoints (demultiplexing).
5) **Accountability** — Techniques such as uRPF checks and egress filtering can be viewed as weak accountability mechanisms protecting against certain types of address spoofing. Recent work offers stronger protection than that offered by IP. For example, AIP [4] uses cryptographic identifiers as source addresses that can be used to verify that the host identified really did send the packet.

Somewhat to our surprise, many proposed architectures use source addresses for the same purposes. This includes proposals that are very different from IP, such as architectures that use paths or capabilities, rather than addresses, to identify a destination. For instance, SCION [37] headers include AS-level paths selected jointly by the ISPs and source and destination networks to specify how to reach the destination. However, each packet still has an AIP-style source identifier that fulfills the above roles. Also, in ICING [28] and capability-based architectures such as TVA [36], packets carry pre-approved router-level paths, but they also carry traditional source and destination addresses.

To understand the impact of repurposing the source address as an accountability address in APIP, we ask two questions about each role:

(1) Is it needed by the network? If not, it can be moved deeper in the packet, opening up more design options and simplifying the network header.

(2) Is it needed in every packet? If not, it could be stored elsewhere, e.g. on the routers or end-hosts that will use it. This simplifies the packet header, but may add complexity to protocols that have to maintain the state.

Table 1 summarizes the answers to these questions. Two high-level takeaways emerge: (1) not all roles involve the network, and (2) some information is not needed in every packet. The following observations are particularly relevant to the accountability versus privacy tussle:

1) **The accountability role is the network's primary use of source addresses.** Error reporting benefits the host, not the network. Hosts could choose to forgo error reports for the sake of privacy or have them sent to the accountability address. Flow ID calculation could use the accountability address.
2) **The return address role is not used by the network at all.** It could be moved deeper in the packet, encrypted end-to-end, and/or omitted after the first packet of a flow.

3. ACCOUNTABILITY VERSUS PRIVACY

A number of research efforts focus on improving either accountability or sender privacy in the network, but unfortunately this often comes at the price of weakening the other. To illustrate this point, we summarize one well-known technique for each and then elaborate on the goals of this paper.

3.1 Previous Work

Accountability and Nothing But The Accountable Internet Protocol (AIP) [4] is a network architecture whose primary objective is accountability. Each host's *endpoint identifier* (EID) is the cryptographic hash of its public key, and AIP introduces two mechanisms that use these "self-certifying" EIDs to hold hosts accountable.

First, first-hop routers in AIP prevent spoofing by periodically "challenging" a host by returning the hash of a packet it purportedly sent. Hosts maintain a cache of hashes of recently sent packets and respond affirmatively if they find the specified packet hash; the response is signed with the private key corresponding to the source EID. If a challenge elicits no response or the response has an invalid signature, the router drops the original packet. Second, AIP proposes a *shutoff* protocol: a victim under attack sends the attacking host a *shutoff packet*, prompting the attacker's NIC to install a filter blocking the offending flow. Shutoff packets contain the hash of an attack packet (to prove the host really sent something to the victim) and are signed with the victim's private key (to prove the shutoff came from the victim).

AIP suffers from three important limitations: first, cryptographically linking senders to their packets' source addresses precludes any possibility of privacy. Second, though bad behavior is always linkable to the misbehaving host, AIP does not facilitate a long-term fix—the shutoff protocol is only a stop-gap measure. Finally, AIP requires that well-intentioned owners install "smart NICs" that implement the challenge and shutoff protocols, since a compromised OS could ignore shutoffs. We draw heavily on ideas from AIP while addressing these limitations.

Privacy and Nothing But The best available solution for hiding a return address is using a mix net or onion routing service like Tor [12, 31, 32]. Observers in the network only see the identity of the two onion routers on that link in the Tor path. Of course, accountability is much more difficult to achieve since the identity of the sender is hidden inside the packet, behind one or more layers of encryption. Liu et al. propose an architecture that offers a high degree of privacy by baking Tor into the network itself [25]. However, in addition to the lack of accountability, the increased header overhead and latency make Tor unsuitable as a default, "always-on" solution.

Role	Where Used	Layer	Comments
Return Address	Destination	Transport	Routers forward purely based on the destination address; the return address is used only by the destination.
Sender Identity	Destination	Application	No longer used to authenticate users, but may be used to, e.g., track "users" across sessions in web access logs.
Error Reporting	Routers	Network	Destination for error messages.
	Destination	Network	
Flow ID	Destination	Transport	End-hosts need a way to demultiplex flows.
	Routers	Network	Routers distinguish flows for traffic monitoring/engineering.
Accountability	Routers	Network	In designs like AIP, routers may require a valid (challengeable) source address.
	Destination	Network	It must be possible to identify and shut down a malicious flow.

Table 1: The roles a source address plays and where each is used.

3.2 Goals

The examples show that the source address represents a control point in the tussle between privacy and accountability. Unfortunately, it is a very crude one since there seem to be only two settings: privacy (x)or accountability. The high level goal of this paper is to redefine the source address so it can properly balance the accountability and privacy concerns of providers and users.

Accountability At the network layer, by accountability we mean that *hosts cannot send traffic with impunity: malicious behavior can be stopped and perpetrators can be punished*. Specifically, we would like our design to have the following three properties:

1) Anyone can verify that a packet is "vouched for" — someone is willing to take responsibility if the packet is malicious.
2) Malicious flows can be stopped quickly.
3) Future misbehavior from malicious hosts can be prevented (i.e., by administrative or legal action).

Privacy Our focus is on providing *the ability for a sender to hide its network address* so it can hide its identity from third-party observers in the source domain, from transit ISPs, and (optionally—see §6) from the destination. We assume these adversaries can observe all packets. Note that while our goal is to make it possible to hide the sender's address, APIP does not specify any one particular address hiding mechanism. We do not consider anonymity from the operator of the source domain itself (since it can identify the sender based on the physical "port" through which the packet entered the network).

Application-layer privacy concerns are outside the scope of this paper, nor are we concerned about hiding a packet's destination; senders wishing to make their packets unlinkable to the destination should use solutions such as Tor. Finally, though we do not introduce new techniques for flow anonymity, i.e., the inability of observers to link packets belonging to the same flow, we discuss how our solutions affect the linkability of packets in a flow.

4. BASIC DESIGN

The Accountable and Private Internet Protocol (APIP) ***separates accountability and return addresses***. A dedicated accountability address allows us to address the limitations of an accountability-above-all-else approach like AIP by introducing ***delegated accountability***. Rather than identifying the sender, a packet's accountability address identifies an *accountability delegate*, a party who is willing to vouch for the packet. With accountability handled by delegates, senders are free to ***mask return addresses*** (e.g., by encrypting them end-to-end or using network address translation) without weakening accountability.

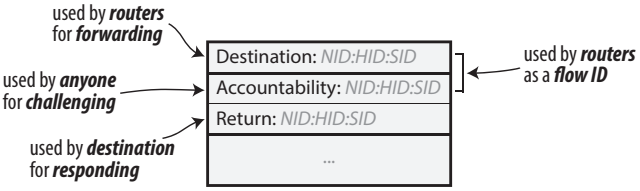


Figure 1: Packet carry a destination address (used by routers for forwarding), an accountability address (used to report malicious packets), and an optional return address (used by the receiving endpoint for responding).

Addressing We think APIP is applicable to many different network architectures, so as much as possible we avoid making protocol-specific assumptions. To discuss source addresses generally, we adopt three conventions.

First, each packet carries at least two addresses (Figure 1): (1) a *destination address* (used to forward the packet) and (2) an *accountability address* (identifying a party—not necessarily the sender—agreeing to take responsibility for the packet). It may also carry a *return address* (denoting where response traffic should be sent) as a separate field in the packet. Return addresses may not be present in all packets, e.g., they may be stored with connection state on the receiver. Also, as we discuss later, the return address may not always be part of the *network* header.

Second, an address consists of three logical pieces: (1) a *network ID* (NID), used to forward packets to the destination domain, (2) a *host ID* (HID), used within the destination domain to forward packets to the destination host, and (3) a *socket ID* (SID), used at the destination host to demultiplex packets to sockets. We write a complete address as *NID:HID:SID*. These logical IDs may be separate header

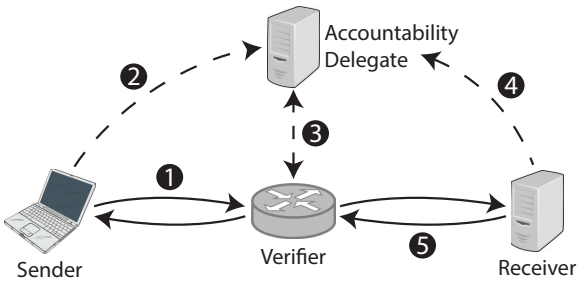
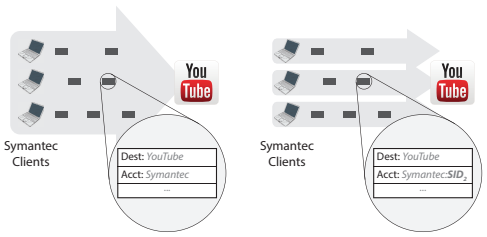


Figure 2: High-level overview of APIP.



(a) Without *SID*s. (b) Per-host *SID*s.

Figure 3: Adding *SID*s to accountability addresses for flow differentiation.

fields or could be combined (e.g., an IP address encodes both an NID and an HID; the port number serves as an SID).

Finally, to simplify our description of APIP, we initially assume that HIDs are *self-certifying*, as defined by AIP, to bootstrap trust in interactions with accountability delegates. We relax this assumption in §7.3

Life of a Packet Figure 2 traces the life of a packet through APIP.

1. The **sender** sends a packet with an *accountability address* identifying its **accountability delegate**. If a *return address* is needed, it can be encrypted or otherwise masked.
2. The **sender** "briefs" its **accountability delegate** about the packet it just sent.
3. A **verifier** (any on-path router or the receiver) can confirm with the **accountability delegate** that the packet is a valid packet from one of the delegate's clients. Packets that are not vouched for are dropped.
4. If the **receiver** determines that packets are part of a malicious flow, it uses the *accountability address* to report the flow to the **accountability delegate**, which stops verifying (effectively blocking) the flow and can pursue a longer term administrative or legal solution.
5. The **receiver** uses the *return address* in the request as the *destination address* in the response.

It is useful to identify the key differences between APIP and the AIP and Tor protocols discussed in Section 3. Delegated accountability offers two key benefits over AIP. First, it dramatically improves sender privacy: only the accountability delegate, not the whole world, knows who sent a packet. Second, it offers a more reliable way of dealing with malicious flows compared to a smart NIC. Third, it offers a clearer path to long-term resolution to bad behavior. For example, the delegate can contact the well-intentioned owner of a misbehaving host out-of-band (e.g., requiring them to run anti-virus tools). While Tor provides stronger privacy properties than APIP, by simply changing how source addresses are treated, APIP can provide sender privacy with much lower overhead since the return address can be hidden from the network. Techniques for doing so (§6) are lightweight enough to be viable options for "default on" use.

5. DELEGATING ACCOUNTABILITY

This section describes how accountability can be delegated. We will assume delegates can be trusted, e.g., their role is played by a reputable commercial company or source domain. We discuss the problem of rogue delegates in §7.1. APIP defines four aspects of delegate operation: the form of the address used to reach a delegate plus the three operations all delegates must support — the delegate "interface," so to speak. Delegates expose one operation to their clients:

> **brief(packet, clientID)**: Whenever a host sends a packet, it must "brief" its delegate — if the delegate is to vouch for the packet on behalf of the sender, it needs to know which packets its clients actually sent.

To the outside world, accountability delegates offer two operations, borrowed largely from AIP:

> **verify(packet)**: Anyone in the network can challenge a packet; its accountability delegate responds affirmatively if the packet was sent by a valid client and the flow has not been reported as malicious.
>
> **shutoff(packet)**: Given an attack packet, the victim can report the packet to the accountability delegate; in response, the delegate stops verifying (blocks) the flow in question and pursues a long term solution with the sender.

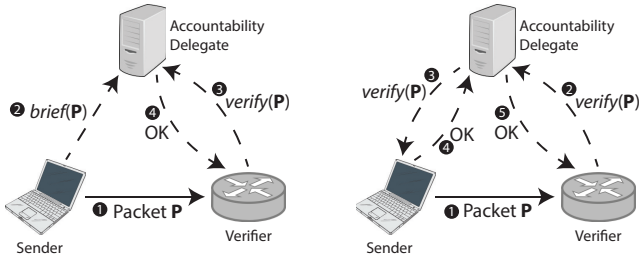
We now discuss options for constructing the accountability address and for implementing the delegate interface.

5.1 Accountability Addresses

Accountability addresses serve two related functions. First, the address is used to send verification requests and shutoffs to an accountability delegate. The NID:HID portion of the address is used to direct messages to the delegate server. Second, routers often need to identify flows, e.g., for traffic engineering (TE) or monitoring purposes, and today source addresses are often part of the flow ID. The granularity of this ID is even more important in APIP since traffic is verified (and blocked) per flow. In this section, we discuss the implications of replacing source addresses with accountability addresses for flow identification.

Creating Flow IDs Routers construct flow IDs using information available in the network and transport headers. However, in APIP, if an accountability address merely points to a delegate, packets from all clients of a particular delegate will be indistinguishable, robbing routers of the ability to distinguish flows at a finer granularity than *delegate↔destination* (Figure 3a). This may be too coarse-grained, especially since the flow ID is used for dropping packets from malicious flows. In effect, every flow that shares a delegate with a malicious flow will share its fate. (TE tends to work with coarser-grained flows, so destination addresses alone may be sufficiently granular.)

The simplest way to support finer-grain flow IDs is to include the delegate's SID in the calculation, similar to the



(a) Fingerprint Collection (b) Recursive Verification

Figure 4: Briefing Techniques

way port numbers are used today. For example, delegates could assign a group of SIDs to each client source domain, which it can use to define network-level flows as it sees fit (see below). Accountability delegates with many clients would require a large pool of SIDs to achieve fine granularity (e.g., at the host or TCP flow level). If the SID is not sufficient, it is possible to add a separate flow ID field to the packet header to improve granularity. In our discussion, we will use the term flow ID to refer to both the SID only and SID plus dedicated field approaches.

Controlling Flow Granularity How the flow ID is assigned affects both privacy and the amount of collateral damage caused when an aggregate flow containing a malicious sender is blocked. At one extreme, a delegate could use a single flow ID for all its customers, which provides the biggest possible anonymity set, but may result in a lot of legitimate traffic being dropped if any client sends malicious traffic. At the other extreme, assigning senders unique flow IDs (Figure 3b), or a separate flow ID per TCP flow, allows fine grain filtering, but allows sender/TCP flow linkability. The solution we propose is for delegates to assign each client a pool of flow IDs which it can assign to packets based on internal policies. Delegates check that clients are using flow IDs assigned to them as part of the verification process (§5.3).

Source Domain Accountability Address Management An interesting alternative to senders picking a flow ID for each packet (within boundaries set by their delegate) is to have flow IDs assigned at the level of the source domain. For example, individual hosts could send packets with a traditional source address. If a packet leaves the source domain, the gateway routers replace it with an accountability address and hide the return address (like a NAT; see § 6). This approach is especially attractive for source domains that act as the accountability delegate for their hosts (§7.4). Centralized management simplifies managing the pool of flow IDs, enforcing policies, and incremental deployment. The drawback is that individual users lose control over sender privacy.

5.2 Brief()

Accountability delegates need to know which packets their clients have sent if they are to vouch for them when challenged with a **verify()**. We consider two approaches in this section. Accountability delegates can choose any method, possibly on a per-client basis.

Fingerprint Collection The simplest solution is for senders to proactively send their delegates fingerprints of the packets they send (Figure 4a). The delegate stores the fingerprints (e.g., for 30 seconds), and when it receives a **verify()**, it searches for the fingerprint and returns `VERIFIED` if it finds it. For reasons explained in §5.3, a packet's fingerprint is actually more than just a simple hash:

$$F(P) = H\left(K_{SD_S} \mid\mid P_{header} \mid\mid H(P_{body})\right)$$

Here H is a cryptographically secure hash function and K_{SD_S} is a symmetric key established when sender S signed up for service with delegate D_S. It is included in the fingerprint to prevent observers from linking P to $F(P)$. Each brief includes a client ID, a fingerprint, and a message authentication code (MAC):

Sender transmits packet and brief:

$$\begin{aligned} S \rightarrow R: &\quad P \\ S \rightarrow D_S: &\quad \textbf{brief}(P) = \text{clientID} \mid\mid F(P) \\ &\qquad \mid\mid \text{MAC}_{K_{SD_S}}(\text{clientID} \mid\mid F(P)) \end{aligned}$$

To reduce delegate storage requirements and network overhead, rather than sending full-sized fingerprints, hosts can instead periodically provide their delegate with a bloom filter of the fingerprints of all packets sent since the last brief. Accountability delegates keep the filters received in the last thirty seconds.

Note that in either case (fingerprints or bloom filters), the delegate can vouch for its clients' packets without knowing anything about their contents. Finally, if gateway routers assign accountability addresses, they can also take responsibility for briefing the delegate.

Bootstrapping Who vouches for briefs? That is, how do senders get briefs to their delegates if the packets carrying them cannot be verified? Clients include a special "token" in brief packet headers (e.g., as the SID in the destination address) proving to the delegate that the brief is from a valid client. Since verification requests include a copy of the unverified packet's header (see §5.3), the delegate can see that both the accountability *and destination* addresses point at the delegate, indicating the packet is a brief, cueing the delegate to check for the token. Delegates can use any scheme to select tokens. One possibility is using a hash chain based on a shared secret. Each brief uses the next hash in the chain, preventing replays. (This ensures the brief is from a valid client—we discuss "brief-flood" DoS attacks from malicious clients in §7.2.)

Recursive Verification The alternative to fingerprint collection is to have hosts store the fingerprints of recently sent packets. When a delegate receives a **verify()**, the delegate forwards the verification packet to the host that sent it. The host responds "yes" or "no" to the delegate, which passes the response on to the original challenger (Figure 4b). In this case, **brief()** is a NOP. Recursive verification reduces network and storage overhead, but the catch is that in order to work each verification request must carry enough information for the delegate to map the packet to a customer. This impacts the flow ID granularity (§5.1): when using recursive verification, delegate must ensure that no two clients share a flow ID (or it must be willing to forward a verification to multiple clients).

5.3 Verify()

Verify() is nearly identical to AIP's anti-spoofing challenge, the difference being that an AIP challenge asks, "Is this packet's source address spoofed?" whereas **verify()**

asks, "Do you vouch for this packet?" In AIP, first-hop routers periodically verify that packets purporting to be from a particular host are not spoofed. Likewise, in APIP routers periodically verify that flows are using valid accountability delegates and have not been reported for misbehavior. Verified flows are added to a whitelist whose entries expire at the end of each *verification interval* (e.g., 30 seconds); if a flow is still active, it is re-verified. Consider a sender S, a receiver R, and a router V ("verifier"). If V receives a packet P from S to R and the flow $S \rightarrow R$ is not in the whitelist, V sends a verification packet to S's accountability delegate, D_S (identified in the packet). To avoid buffering unverified packets, V can drop P and send an error message notifying S that P was dropped pending verification.

The verification packet includes P's fingerprint plus a MAC computed with a secret key known only to V. D_S now checks three things: (1) it has received a brief from S containing $F(P)$, (2) the accountability address in P is using an *SID* assigned to S, and (3) transmission from S to R has not been blocked via a shutoff (§5.4). If everything checks out, D_S returns a copy of the verification packet signed with its private key to V, which adds $S \rightarrow R$ to its whitelist. The protocol, below, shows both fingerprint collection ($\star$) and recursive verification ($\blacklozenge$), though only one or the other would be used in practice. (K_V is a secret known only to V; $K^+_{D_S}/K^-_{D_S}$ is the delegate's public/private keypair; K_{SD_S} is the symmetric key shared by S and D_S.)

Sender transmits packet and brief:

$S \rightarrow R$: P

$\star$ $S \rightarrow D_S$: **brief**(P)

Verifier sends error to sender and verification to delegate:

$V \rightarrow S$: `DROPPED (VERIFYING)` $||$ $F(P)$

$V \rightarrow D_S$: **verify**$(P) = P_{header} \,||\, H(P_{body}) \,||\, \text{MAC}_{K_V}(P_{header} \,||\, H(P_{body}))$

Delegate verifies packet and responds:

$\blacklozenge$ $D_S \rightarrow S$: $\{$**verify**$(P)\}_{K_{SD_S}}$

$\blacklozenge$ $S \rightarrow D_S$: $\{$`VERIFIED` $||$ **verify**$(P)\}_{K_{SD_S}}$

$D_S \rightarrow V$: $\{$`VERIFIED` $||$ **verify**$(P) \,||\, K^+_{D_S}\}_{K^-_{D_S}}$

V: add flow entry to whitelist

There are three points worth noting about this protocol. First, D returns the original verification packet so V does not have to keep state about pending verifications. V uses the MAC to ensure that it originated the verification request, preventing attackers from filling V's whitelist with bogus entries by sending it verifications it never asked for.

Second, the delegate needs to know the packet's destination address (R) so it can check if traffic $S \rightarrow R$ has been shut off. Since briefs only contain fingerprints, the delegate does not already have this information, so the verification request includes a copy of P's header. It also includes a hash of the body so the delegate can finish computing the fingerprint of packet being verified to check that it matches a brief in its cache.

Third, the last line in the protocol adds the flow to the white list, identified by its accountability address, destination address, and flow ID, as described in §5.1.

ISP Participation Though *anyone* can verify a packet, APIP is most effective when routers closest to the source perform verification. An ISP that suspects a customer/peer might not be properly verifying its traffic can apply business pressure or possibly dissolve peering relationships if it finds an inordinate amount of unverified traffic. Another concern is that domains might verify traffic with a long verification interval (that is, after verifying a packet from a flow, the same flow is not verified again for an extended period of time). This allows malicious flows to do damage even if the flow's delegate receives a **shutoff()** since the flow will not be blocked until the next verification. The impact of long verification intervals could be mitigated if transit networks also verify traffic (see §8.1 for expected time-to-shutoff); after a **shutoff()**, the filter moves toward the sender as closer routers re-verify the flow. Also, even if routers are slow to react, APIP still facilitates a long-term fix eventually.

5.4 Shutoff()

Today, when hosts or routers identify a malicious flow, they can locally filter packets and work with neighboring ISPs to stop traffic. In APIP, they can also send a **shutoff()** request to the attacker's delegate. This is particularly useful for receivers, who should have the final say as to whether a flow is wanted or not. The protocol differs from AIP's shutoff protocol in two important ways. First, shutoffs are directed to accountability delegates, not to senders. Second a delegate can not only block the offending flow, but it can also pursue a long-term fix. The **shutoff()** protocol is shown below (between receiver R and S's delegate D_S regarding packet P from sender S):

Sender transmits packet and brief:

$S \rightarrow R$: P

$S \rightarrow D_S$: **brief**(P)

Receiver sends shutoff:

$R \rightarrow D_S$: **shutoff**$(P) = \{P_{header} \,||\, H(P_{body}) \,||\, \textit{duration} \,||\, K^+_R\}_{K^-_R}$

Sender's delegate verifies shutoff and takes action:

D_S: check $H(K^+_R) == \textit{dest}(P_{header})$

D_S: block offending flow for *duration* sec

Receivers can *always* shut off traffic directed at them. When the delegate receives the **shutoff()**, it checks that the **shutoff()** was signed by the private key corresponding to the recipient of the packet in question (so the **shutoff()** contains both the victim's public key and the original packet's header; the delegate compares the hash of the public key to the packet's destination address). If the verifier is a router and the **shutoff()** is signed by an ISP's key, it might also be honored, but perhaps only with manual intervention—if a reputable ISP says one of your clients is attacking its network, chances are you should listen. After verifying a **shutoff()**, the attacker's delegate responds in two ways.

Short-term fix: To provide the victim immediate relief, the delegate blocks the offending flow by ceasing to verify packets from the attacker to the victim. Routers only save flow verifications in their whitelists temporarily; when a router on the path from S to R next tries to verify the attack flow, the delegate responds `DROP_FLOW`. This means the attack could last up to a router's verification interval—we discuss expected shutoff time in §8.1. If delegates work with ISPs, response time could be shortened by pushing verification revocations from delegates to routers. Alternatively, if we assume widespread shutoff support in NICs, delegates could send shutoffs to directly to attackers, as in AIP.

Long-term fix: Since clients sign contracts with their delegates, a delegate can contact the owner of misbehaving hosts out-of-band. Since most unwanted traffic comes from botnets of compromised hosts with well-intentioned owners [23], the owner will generally fix the problem, at which point the delegate can unblock flows from the host. If a client refuses to comply, the delegate can terminate service or report him to the authorities.

6. MASKING RETURN ADDRESSES

APIP separates accountability from other source address roles, allowing senders to hide the return address from observers in the network. APIP does not define any particular privacy mechanism, but rather enables various lightweight, "always-on" strategies for increasing the default level of privacy for all traffic without weakening accountability. We offer two examples: end-to-end return address encryption and network address translation. Since our focus is sender-flow unlinkability, our primary concern is the size of the sender anonymity set from the perspective of four possible adversaries: the source domain, observers in the source domain, transit networks, and the receiver (Table 2).

End-to-end Encryption Since the return address is used only by the receiver and not by routers, a simple idea is to encrypt it end-to-end (e.g., using IKE [18], à la IPsec); now only the destination and accountability addresses are visible in the network. We imagine two variants: one in which return address encryption is a network layer standard and can be performed end-to-end or gateway-to-gateway and one in which the return address is moved to a higher layer (e.g., transport or session layer).

Of course, though the return address is encrypted in the forward direction, it will plainly visible as the destination address in responses; determined attackers may be able to link the outbound and inbound traffic (e.g., with timing analysis). Still, even this simple strategy offers increased privacy against passive observers, e.g., reviewing logs from a core ISP.

Network Address Translation Encrypting the return address end-to-end hides it from the network, but not from the destination. For privacy from the network *and* the recipient, edge ISPs' border routers could perform address translation on outbound packets' return addresses by changing *NID:HID:SID* to *NID:HID':SID'*. (This can be done deterministically to avoid keeping large translation tables [30].) Note that in contrast to the encryption option, response packets sent by the destination will not reveal the identity of the original sender (in the destination address). The downside is that, in contrast to encrypted return addresses, the anonymity set shrinks closer to the source.

Today, increased use of NAT might be a controversial proposition, but cleaner thinking about source addresses mitigates some of the chief arguments against it. For example, in 2006 the entire nation of Qatar was banned from Wikipedia when one user vandalized an article because the country's sole ISP uses a NAT with one external IP address [3]; in APIP, all hosts could share one external return address while still being held individually accountable via the accountability address.

Second, NATs are traditionally deployed for address space separation—the privacy they provide is a side effect [35]. This is known to cause problems for servers or P2P applications. In contrast, we suggest NATing for privacy, which can be done selectively for outgoing connections. Incoming connections are not affected, so servers, for example, can publish their internal address to DNS and receive incoming connections without any kind of hole punching. Of course, NATing for incoming connections also has security benefits, but this is an orthogonal issue.

Reducing Overhead Not all packets need a return address. For connection oriented traffic, the return address needs to be sent to the destination only once during the establishment of the connection (and also when a mobile device switches networks). The destination can store it and reuse it for later packets. Doing so (1) ameliorates the header overhead introduced by splitting accountability and return addresses in the first place and (2) allows NATs to modify many fewer packets.

Beyond the First Hop No matter how far a packet travels, the sender anonymity set is still just the sender's source domain. Though this may be sufficient for most senders, the NAT approach can be extended by performing address translation at more border routers. Though core ISPs are unlikely to do this, if the first 2–3 domains in the path do, a packet's sender anonymity set grows significantly before reaching the core (and ultimately its destination).

7. IN THE REAL WORLD

7.1 Holding Delegates Accountable

Delegates have three responsibilities: protecting the privacy of their clients, verifying packets with fingerprints that match those sent by valid clients, and dropping invalid packets. We briefly discuss how malicious or compromised delegates can harm either their clients or allow their clients to harm others.

(1) Releasing private information about clients. Delegates can learn a lot about who their clients communicate with, information they could use for their own benefit or reveal to a third party. Upon discovering a leak, the client can terminate service, find a new delegate, and potentially pursue legal action for breach of contract. Note that delegates only see packet headers, *not* packet contents. (An interesting direction for future work is exploring anonymous briefing schemes, e.g., based on cryptography or the use of an intermediary.)

(2) Failing to verify clients' packets. Delegates can effectively perform a DoS attack on clients by failing to verify their packets. Senders can detect this due to the excessive number of `DROPPED (VERIFYING)` or `DROPPED (VERIFICATION FAILED)` error messages they will receive from verifiers. Again, the client can then terminate service.

(3) Verifying invalid packets. Delegates can support attacks by verifying packets for which they did not receive a brief from a client or which belong to flows that have been shut off. (Such a delegate may be compromised or misconfigured or may even be colluding with an attacker.) Victims can detect such malicious behavior (e.g., by observing that their shutoff requests have been ignored).

Who Can be a Delegate? The likelihood of any of the above problems occurring depends on *who can be a dele-*

Adversary	End-to-end Encryption	Address Translation
Source Domain	Source domain always knows a packet's **sender**.	Source domain always knows a packet's **sender**.
Observers in Source Domain	Other **source domain customers**.	The **sender**. The sender's address is observable until the packet reaches the border router where NAT is performed.
Transit Networks	Starts as **source domain's customers** and **grows** the farther the packet travels. By the time it reaches the core, it could have come from anywhere.	**Source domain's customers**.
Receiver	The **sender**. The receiver decrypts the return address, which is the sender's address. If the sender is concerned with anonymity from the receiver, end-to-end encryption is not a viable option.	**Source domain's customers**.

Table 2: Comparison of sender anonymity set, as seen by different adversaries, for end-to-end encryption and NAT.

gate. The issue of delegate oversight is complex; given space constraints, we can only hope to lay the groundwork for discussion and future work.

At one extreme, a single central authority could provide some form of oversight over delegates, similar to how ICANN accredits TLDs; verifiers would then only accept delegates on a whitelist published by this authority. This has the advantage that delegates can be monitored and misbehaving delegates can be immediately removed from the whitelist, creating an incentive for responsible delegate management. On the other hand, vetting all delegates is a huge burden for a single authority and the role (and power) of a single organization in charge of such a critical function is likely to raise political concerns.

The other extreme is a free-for-all: a host can pick any host to be its delegate. This flexibility opens the door for many deployment models. Besides commercial (third party) delegates, hosts can be their own delegates (similar to AIP), use their source domain as their delegate (§7.4), or form a peer-to-peer delegate network, in which hosts (or domains) vouch for one other. The critical drawback of this flexibility is weaker protection against attacks—bots in a botnet, for example, can vouch for one another's packets regardless how many **shutoff()**s they receive. It will clearly be harder to defend against such attacks compared to the case where there are only a limited number of vetted delegates.

Naturally, a pragmatic solution likely falls somewhere in between these extremes. For example, a set of well-known commercial "delegate authorities" could emerge, similar to today's certificate authority infrastructure, each publishing a delegate whitelist. Alternatively, various groups could maintain delegate blacklists based on historical incidents, similar to today's security companies' publishing malware signatures. Individual verifiers can then decide which delegates to accept, a decision that could depend on many factors, including their position in the network (tier 1 versus edge), local regulation, historical information, or the "trust domain" they belong to [37]. Many other forms of semi-structured self regulation are possible.

7.2 Attacking APIP

We need to ensure that hosts cannot use APIP mechanisms to undermine APIP itself; two potential such attacks are "verification-flooding" and "brief-flooding."

Verification-Flooding Attackers could attempt to overwhelm an accountability delegate with bogus verification requests—rendering it incapable of verifying honest hosts' packets—by sending a large number of dummy packets with accountability addresses pointing at the victim delegate. To these bogus verifications, the delegate could respond `DROP_HOST` (as opposed to `DROP_FLOW`). Source domains should track the number of `DROP_HOST`s their customers generate, taking action if it is too high.

Brief-Flooding Similar to verification flooding, malicious *clients* could target their own delegate by sending a flood of briefs. This attack is tricky, since it is hard to distinguish from an honest host that happens to send lots of packets. Delegates can enact their own policies (e.g.: will accept 1 brief per second; must use bloom filters), which should be agreed upon when the client initially signs up for service.

7.3 Bootstrapping Trust

We now relax our initial assumption that HIDs are self-certifying; doing so does not break APIP, but requires us to do a small amount of extra work in **brief()**, **verify()**, and **shutoff()**.

brief() Clients already encrypt **brief()**s using a symmetric key established when the client registered for service, so no change is required.

verify() Delegates use their private keys to sign verification responses. If keys are not bound to HIDs, a PKI can be used instead; verifiers now need to check a delegate's certificate before trusting a response.

shutoff() Victims sign **shutoff()** messages to convince the attacker's delegate that the shutoff truly came from the recipient of the offending packet. While we think it is reasonable to assume delegates register keys with a PKI for signing **verify()**s, there are many more hosts than delegates, so here we instead rely on verification. Upon receiving a **shutoff()**, the attacker's delegate sends a verification packet to the victim's delegate to check that the shutoff really came from the original packet's recipient:

$$\begin{aligned} R \rightarrow D_R: &\quad \textbf{brief}(\textbf{shutoff}(P)) \\ D_S \rightarrow D_R: &\quad X = \textbf{verify*}(\textit{shutoff}) \\ D_R \rightarrow D_S: &\quad \{\texttt{VERIFIED} \mid\mid X\}_{K^-_{D_S}} \end{aligned}$$

When verifying a **shutoff()**, the delegate needs to look inside the shutoff packet, at the header of the original packet that prompted the shutoff, and check that its destination

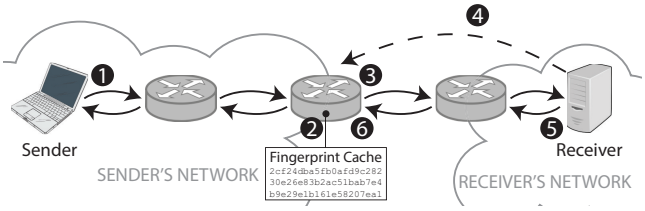


Figure 5: **Design 1:** ❶ Host sends packet using its own address as the source address. ❷ The ISP's border router saves a hash of the packet and ❸ performs address translation on the source address. ❹ If the packet is malicious, the receiver sends a **shutoff()** to the border router, otherwise ❺ it responds. ❻ The border router translates the response's destination address back to the original sender's address.

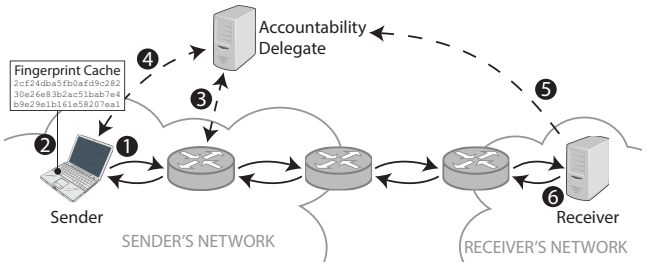


Figure 6: **Design 2:** ❶ Host sends packet and ❷ saves hash. ❸ First-hop router sends **verify()** to the packet's accountability delegate, which ❹ forwards the verification to the host. ❺ Using the accountability address, the receiver can send a **shutoff()**; otherwise ❻ it responds using the return address.

	Design 1	**Design 2**
Delegate	Source domain	Third party
Briefing	Fingerprint collect.	Recursive ver.
Source Addr.	Single field	Separate fields
Return Addr.	NAT	NAT or encrypt

Table 3: Two possible instantiations of APIP.

(the victim) sent the shutoff. (We denote **verify()** with this additional check **verify*()**.)

7.4 Concrete Designs

APIP is an architecture that allows routers and destinations to identify an entity that is willing to take responsibility for the packet, but properties of APIP depend on how it is deployed. For the sake of concreteness, we now sketch two end-to-end instantiations of APIP with very different properties. Table 3 summarizes the two designs.

Design 1 In the first design (Figure 5), the source domain acts as the accountability delegate for its hosts. Hosts are not aware of APIP and send packets with traditional source addresses. The gateway routers for the domain use address translation to mask the return address, turning the source address into a combined accountability address and masked return address. They also collect packet fingerprints and respond to **verify()** and **shutoff()** requests. Gateway routers could either collectively act as a distributed accountability delegate and keep briefs locally, or they could periodically send briefs to a shared accountability server. This first design could be viewed as a variant of AIP, but implemented at the source domain level instead of individual senders.

This design offers a number of advantages. First, it is very efficient: gateway routers already naturally see all packets, eliminating the overhead of briefing a third party. Second, the source domain can immediately block malicious flows at the source in response to a shutoff, whereas external delegates can typically only stop flows indirectly. Third, hosts do not need to be modified.

Finally, this first design allows for incremental deployment of APIP over IP. Domains could implement accountability and address translation, as described. Packets would need to be marked to indicate whether the source domain supports APIP (e.g., by repurposing an ECN bit). Since both return traffic and **verify()**s/**shutoff()**s would arrive at the domain's border routers, **verify()** and **shutoff()** would each be assigned a new IP protocol number so the border router knows what to do with the packet. Since IP addresses are not cryptographic, external keys would have to be used to ensure the integrity of the **verify()** and **shutoff()** operations, as described in §7.3.

Design 2 The second design (Figure 6) uses a commercial third party that offers accountability delegation as a service (perhaps as part of a bundle with antivirus or firewall software). In this design, senders insert both an accountability address and a return address in the packet; the return address can be masked either with encryption or a NAT. Since the delegate is off-site, recursive verification is attractive: rather than regularly sending briefs, hosts save packet hashes and the delegate challenges clients when it itself is challenged.

One advantage of this solution is that it allows companies, universities, or small domains to avoid the hassle of managing delegate servers themselves by outsourcing delegation. Another advantage is that it is harder for observers in the network to determine what source domain the sender belongs to. The drawback is that there is more overhead than in the first design.

8. EVALUATION

The primary question we consider in the section is: "is delegated accountability technically feasible?" Using a trace of NetFlow data from the border routers of a mid-sized university, we explore the costs of **brief()** and **verify()** and the efficacy of **shutoff()**. The five minute trace was taken on June 18, 2013 at noon and contains ten million flows. We then present a short privacy analysis.

8.1 Delegated Accountability

Brief() Briefing the delegate incurs computational overhead at the sender, storage overhead at the delegate, and bandwidth overhead in the network. In §5.2 we suggested that senders could report their traffic to their delegates by sending a list of packet fingerprints or by sending a bloom filter of recent fingerprints.

Computational Overhead Producing a packet fingerprint requires computing two hashes; we assume that computing the MAC of the fingerprint, in the worst case, costs one addi-

tional hash. Commodity CPUs can compute in the neighborhood of 5–20 MH/s [1], which translates to 0.9–3.4 Gbps (conservatively assuming 64B packets). This is more than reasonable for endhosts; for servers sending large volumes of traffic, we expect data centers could outsource briefing to an in-path appliance built with ASICs—current ASIC-based bitcoin miners perform 5–3,500 GH/s, translating to 0.9–600 Tbps.

Storage Overhead Next we consider the storage requirements at the delegate for saving briefs. Briefs are periodically purged from the cache; if a **verify()** arrives for a legitimate packet whose brief has been deleted, the sender must retransmit the packet. Assuming a single delegate serves all hosts in our trace, the first two series in Figure 7 show the size of the brief cache for different expiration intervals assuming the delegate stores individual fingerprints, each a 20 byte SHA-1 hash. The remaining series consider the space required if, instead of sending a fingerprint per packet, hosts send a bloom filter every second for each active flow[1]. We assume that hosts size each filter appropriately based on how many packets from the flow were sent during the past second. If briefs expire every n seconds, we report brief cache sizes based on both the average and maximum number of packets seen across n-second bins in our trace.

Bandwidth Overhead Figure 8 shows the bandwidth required for the same briefing schemes discussed above. Sending fingerprints consumes an additional 2.5% of the original traffic volume, which was just under 10 Gbps; the bloom filter schemes consume 0.25%–0.5%. The bloom filter results assume a simple update scheme: every second, each host sends a bloom filter for packets sent in each flow[1] during the last 30 seconds; when the delegate gets a new filter for an existing flow, it replaces the old filter (using a bloom filter alternative that supports resizing is interesting future work). Note briefing overhead can be avoided entirely if (1) the ISP is the accountability delegate, so border routers save hashes directly, or (2) delegates use recursive verification (§5.2).

Verify() The magnitude of verification overhead is determined by the verification interval and flow granularity (fewer flows means fewer verifications; in our analysis, each "flow" is a TCP flow, so our numbers are an upper bound). There is a tradeoff between long and short verification intervals. The longer the interval, the more whitelist space is required at routers to remember which flows have been verified and the longer a malicious sender could transmit before being shut off. On the other hand, shorter intervals mean more work (more **verify()**s) for the delegate.

Computational Overhead Figure 9 shows how many **verify()**s per second an accountability delegate serving all the hosts in our trace would see at various verification intervals. A key observation here is that after 10 seconds, increasing the verification interval does not significantly decrease verification rate at the delegate since most flows are short. This knee suggests that 10 seconds might make a good interval for use in practice.

In our trace, a verification interval of 10 seconds generates a *maximum* of 78,000 **verify()**s per second, each causing a lookup in a table with 1.5 million entries (assuming delegates save briefs for 30 seconds) and one signature generation at the delegate. We think these numbers are reasonable and leave enough headroom to comfortably handle larger networks—CuckooFilter [38] achieves more than 180,000 lookups per second in a table with *one billion* entries and the Ed25519 signature system [8] can perform 109,000 signatures per second on a 2010 quad-core 2.4GHz Westmere CPU.

Storage Overhead As routers verify flows, they keep a whitelist of verified flows so that every single packet need not be verified. Whitelist entries expire at the end of each verification interval, at which point the flow is re-verified. We use our trace to estimate the size of this whitelist.

To account for network architectures with addresses significantly larger than IP's, we assume each address is 60 bytes—20 bytes each for the *NID*, *HID*, and *SID*. (Many research efforts explore self-certifying IDs [6, 27, 4, 17] which are typically the hashes of a name or public key; we choose 20 bytes since SHA-1 produces 20 byte digests.) Entries identify flows at a host-host granularity, so each one is 120 bytes (two 60 byte addresses).

Figure 10 shows the size of the whitelist as we vary the verification interval. For a given verification interval, we group the flows in our trace into bins the size of that interval; flows belong to a bin if they were active during that time period (so a flow could belong to multiple bins). The figure reports the whitelist size based on both the average number of flows across bins as well as the maximum seen in any bin. A 10 second interval requires a maximum of 94 MB of whitelist space.

Shutoff() After receiving a **shutoff()**, a delegate blocks malicious flows by ceasing to verify them. The next time a router on the path from the attacker to the victim verifies the flow, the delegate returns `DROP_FLOW` and the router blocks the flow. How quickly this happens after a **shutoff()** depends on how many on-path routers perform verification and how often they verify each flow. Figure 11 shows the expected delay before a shutoff takes effect for different verification intervals as a function of the number of participating routers.

8.2 Privacy

How much privacy does APIP buy? If a sender uses its source domain as a delegate, this depends on the size of that domain. Raghavan et. al. find that, if ISPs were to aggregate prefixes geographically, over half of the prefixes advertised by many popular last-mile ISPs would include more than 10 million IPs [30].

If a sender uses a third party delegate, the anonymity set grows the farther a packet travels from the source domain. To see how much, we use Route Views [2] data from January 1, 2014 to roughly estimate AS "fanout." For each AS, we track how many customer networks sit beneath it in the AS hierarchy. (To be conservative, we only count an AS as a customer if it originates a prefix. Transit networks with no hosts do not contribute to an anonymity set.) For each BGP announcement, we add the origin AS to its first- and second-hop providers' customer sets. Figure 12 shows a CDF of first- and second-hop anonymity set sizes. Notably, 50% of ASes originating prefixes have at least 180 first-hop "siblings" and 90% have over 900 second-hop siblings. Though

[1]In practice, we think hosts should send one bloom filter for *all* traffic each second, not one per flow. Unfortunately, for privacy reasons, our trace did not include local addresses, so we could not merge flows originating from the same sender.

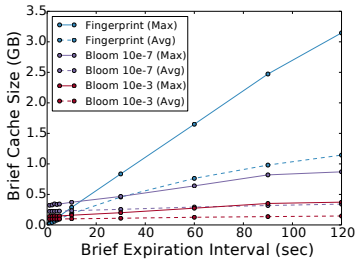


Figure 7: Brief cache size at delegate vs. fingerprint expiry interval.

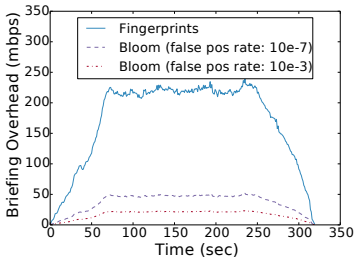


Figure 8: Briefing bandwidth overhead.

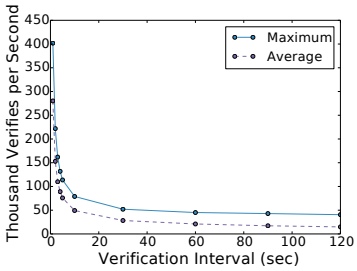


Figure 9: Verification rate at delegate vs. flow verification interval.

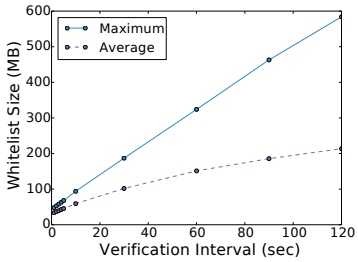


Figure 10: Size of whitelist of verified flows vs. flow verification interval.

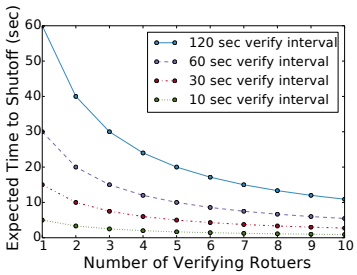


Figure 11: Expected time to shutoff vs. number of on-path verifiers.

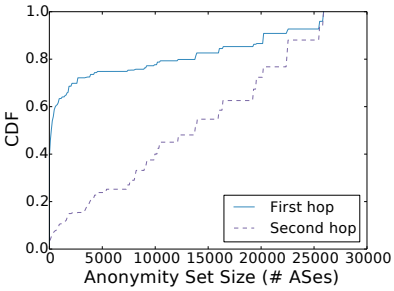


Figure 12: Anonymity Set Size

drawing conclusions about AS topology based on BGP announcements is imprecise, these ballpark figures give an idea of the anonymity benefits of delegated accountability.

9. RELATED WORK

Privacy Various techniques exist for hiding network source addresses, including crowds [32], mixes [12], and onion routing [31]. Real-world implementations based on these ideas include Anonymizer[2] and Tor[3]. Liu et al. consider building onion routing into the network architecture itself [25]. NDN [20] takes a more radical approach by eliminating source addresses altogether; data finds the sender by following "bread crumbs" left by the request. The drawback to all of these approaches is a complete lack of accountability; there is no easy way to link malicious traffic with senders.

Raghavan et. al. [30] describe ISPs offering NAT for privacy as a service but uses a single source address. LAP [19] is similar to (but more secure than) our "NAT-at-every-hop" approach but does not consider accountability.

Accountability Techniques like ingress/egress filtering [16, 24] aim to provide some degree of accountability by reducing the prevalence of source address spoofing; more sophisticated variants exist [29, 14, 21]. This class of approaches has limitations we address: (1) source addresses are only protected on a domain granularity, (2) filtering by itself provides no "shutoff" mechanism for misbehaving hosts who do not send spoofed packets, and (3) it is not compatible with schemes for hiding return addresses for the sake of anonymity.

As described in §3.1, our **verify()** and **shutoff()** mechanisms borrow heavily from AIP [4], which in turn based its mechanisms on ideas presented by Shaw [34] and in AITF [5]. By modifying these mechanisms to work with delegates, we make them privacy-preserving, enable long-term resolution, and avoid relying on self-certifying IDs.

Accountability delegates are described in [7], but the protocol is costly and is not evaluated; privacy receives only passing mention.

Balancing Accountability and Privacy The idea of identify escrow is not new (e.g., [10]). In particular, our notion of delegated accountability is similar in flavor to the *contractual anonymity* described in RECAP [33], in which a service provider (e.g., an online forum) offers its users anonymity which can be broken only if they violate a prearranged contract. The key difference is that RECAP provides contractual anonymity at the application layer while we balance anonymity and accountability at the network layer, which poses unique constraints (like requiring source addresses to be both routable and anonymizable).

Addressing The use of addresses that consist of separate network, host and socket IDs, creating separate identifiers and locators, has been widely proposed [15, 26, 22]. [11, 9] discuss the meaning of source addresses, though without our focus on privacy and accountability.

10. CONCLUSION

This paper attempts to show that a balance between accountability and privacy in the network *is* possible. By decoupling source addresses' roles as accountability addresses and return addresses, APIP strikes a balance between the two seemingly incompatible goals. Delegated accountability allows routers to verify that each packet they forward is vouched for and allows attack victims to report the abuse while at the same time permitting senders to hide their return addresses. Furthermore, the changes to traditional thinking about source addresses required to implement APIP

[2] `http://www.anonymizer.com/`
[3] `https://www.torproject.org/`

are not radical; though more exploration is clearly required, we think the ideas presented here could be applied to the current Internet.

Acknowledgments

Many thanks to the reviewers and to our shepherd, John Wroclawski, for their insightful suggestions. This research was funded in part by NSF under award number CNS-1040801 and by DoD, Air Force Office of Scientific Research, National Defense Science and Engineering Graduate (NDSEG) Fellowship, 32 CFR 168a.

11. REFERENCES

[1] Mining Hardware Comparison. https://en.bitcoin.it/wiki/Mining_hardware_comparison.

[2] University of Oregon Route Views Project. http://www.routeviews.org.

[3] Wikipedia qatar ban 'temporary'. `http://news.bbc.co.uk/2/hi/technology/6224677.stm`, Jan. 2007.

[4] D. G. Andersen, H. Balakrishnan, N. Feamster, et al. Accountable internet protocol (AIP). SIGCOMM '08, pages 339–350, New York, NY, USA, 2008. ACM.

[5] K. J. Argyraki and D. R. Cheriton. Active internet traffic filtering: Real-time response to denial-of-service attacks. In *USENIX Annual Technical Conference, General Track*, pages 135–148, 2005.

[6] T. Aura. Cryptographically Generated Addresses (CGA). RFC 3972 (Proposed Standard), Mar. 2005. Updated by RFCs 4581, 4982.

[7] A. Bender, N. Spring, D. Levin, and B. Bhattacharjee. Accountability as a service. *SRUTI*, 7:1–6, 2007.

[8] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, 2012.

[9] M. B. Braun and J. Crowcroft. SNA: Sourceless Network Architecture. Technical Report UCAM-CL-TR-849, University of Cambridge, Computer Laboratory, Mar. 2014.

[10] J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Advances in Cryptology-EUROCRYPT 2001*, pages 93–118. Springer, 2001.

[11] C. Candolin and P. Nikander. IPv6 source addresses considered harmful. In *NordSec '01*, pages 54–68, 2001.

[12] D. Chaum. Untraceable electronic mail, return address, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

[13] D. D. Clark, J. Wroclawski, K. R. Sollins, and R. Braden. Tussle in cyberspace: defining tomorrow's internet. SIGCOMM '02, pages 347–356, New York, NY, USA, 2002. ACM.

[14] Z. Duan, X. Yuan, and J. Chandrashekar. Constructing inter-domain packet filters to control ip spoofing based on bgp updates. In *INFOCOM*, 2006.

[15] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis. The Locator/ID Separation Protocol (LISP). RFC 6830 (Experimental), Jan. 2013.

[16] P. Ferguson and D. Senie. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2827 (Best Current Practice), May 2000. Updated by RFC 3704.

[17] D. Han, A. Anand, F. Dogar, et al. XIA: efficient support for evolvable internetworking. NSDI'12, pages 23–23, Berkeley, CA, USA, 2012. USENIX Association.

[18] D. Harkins and D. Carrel. The Internet Key Exchange (IKE). RFC 2409 (Proposed Standard), Nov. 1998. Obsoleted by RFC 4306, updated by RFC 4109.

[19] H.-C. Hsiao, T.-J. Kim, A. Perrig, et al. Lap: Lightweight anonymity and privacy. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 506–520. IEEE, 2012.

[20] V. Jacobson, D. K. Smetters, J. D. Thornton, et al. Networking named content. CoNEXT '09, pages 1–12, New York, NY, USA, 2009. ACM.

[21] C. Jin, H. Wang, and K. G. Shin. Hop-count filtering: an effective defense against spoofed ddos traffic. In *CCS '03*, pages 30–41. ACM, 2003.

[22] V. Kafle, K. Nakauchi, and M. Inoue. Generic identifiers for id/locator split internetworking. In *K-INGN 2008.*, pages 299–306, 2008.

[23] S. Kandula, D. Katabi, M. Jacob, and A. Berger. Botz-4-sale: Surviving organized ddos attacks that mimic flash crowds. In *NSDI '05*.

[24] T. Killalea. Recommended Internet Service Provider Security Services and Procedures. RFC 3013 (Best Current Practice), Nov. 2000.

[25] V. Liu, S. Han, A. Krishnamurthy, and T. Anderson. Tor instead of ip. In *HotNets '11*.

[26] D. Meyer, L. Zhang, and K. Fall. Report from the IAB Workshop on Routing and Addressing. RFC 4984 (Informational), Sept. 2007.

[27] R. Moskowitz and P. Nikander. Host Identity Protocol (HIP) Architecture. RFC 4423 (Informational), May 2006.

[28] J. Naous, M. Walfish, A. Nicolosi, et al. Verifying and enforcing network paths with icing. CoNEXT '11, pages 30:1–30:12, New York, NY, USA, 2011. ACM.

[29] K. Park and H. Lee. On the effectiveness of route-based packet filtering for distributed dos attack prevention in power-law internets. In *SIGCOMM CCR*, volume 31, pages 15–26. ACM, 2001.

[30] B. Raghavan, T. Kohno, A. C. Snoeren, and D. Wetherall. Enlisting ISPs to improve online privacy: IP address mixing by default. PETS '09, pages 143–163, 2009.

[31] M. G. Reed, P. F. Syverson, and D. M. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications*, 1998.

[32] M. K. Reiter and A. D. Rubin. Crowds: Anonymity for web transactions. *TISSEC*, 1(1):66–92, 1998.

[33] E. J. Schwartz, D. Brumley, and J. M. McCune. A contractual anonymity system. In *NDSS '10*, 2010.

[34] M. Shaw. Leveraging good intentions to reduce unwanted network traffic. In *Proc. USENIX Steps to Reduce Unwanted Traffic on the Internet workshop*, page 8, 2006.

[35] P. Srisuresh and K. Egevang. Traditional IP Network Address Translator (Traditional NAT). RFC 3022 (Informational), Jan. 2001.

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

[37] X. Zhang, H.-C. Hsiao, G. Hasker, et al. SCION: Scalability, control, and isolation on next-generation networks. In *Security and Privacy 2011(SP), 2011 IEEE Symposium on*, pages 212–227, 2011.

[38] D. Zhou, B. Fan, H. Lim, M. Kaminsky, and D. G. Andersen. Scalable, high performance ethernet forwarding with cuckooswitch. CoNEXT '13, pages 97–108, New York, NY, USA, 2013. ACM.