ABSTRACT
The ability to map packets in a network back to the principals responsible for their transmission, coupled with the capacity to block those principals, could prevent misbehavior. Blocking malicious traffic in routers requires filters to distinguish malicious from valid traffic far from the victim. This architecture conflates the roles of ISPs as providers of connectivity (the pipe) and of accountability (stopping abuse). We believe that ISPs need not bear the responsibility for deciding the policy (whom to block) and implementing the mechanism (filters) of network layer accountability.

We assume the existence of trusted identity authorities, the ability to compute HMACs at intermediate routers and verify signatures at customer-edge routers, and the availability of servers to act as partially-trusted storage devices and representatives of an accountability provider.

With these features, we present: (1) a secure distributed hash table that holds blocking filters and helps to ensure that neighbors in the DHT obey the protocol, and (2) protocol mechanisms that allow high performance verification of accountable traffic, traceback of unaccountable traffic, and the revocation of permission to connect.

1. INTRODUCTION

Misbehavior in the Internet can be attributed to two major factors: first, that attacks can be launched with relative anonymity through source spoofing or zombie machines, and second, that it is not in a provider’s interest to deny connectivity to a paying customer simply because some attack traffic leaves the network. Packet traceback [12, 14, 17] and reverse-path filtering to prevent spoofing [13, 9] attack the first problem, limiting the anonymity of attackers and allowing an abused destination to reliably accuse a sender of misbehavior. In each approach, however, the network providing connectivity to the source machine is expected to disconnect it (perhaps completely) to protect a distant destination.

It has been widely, but anecdotally, reported that the cost of a single technical-support call can negate a provider’s profit from a consumer for that year. That is, Internet service providers are not impartial: each has a financial interest in turning a blind eye to misbehavior by their own customers.\(^1\) As such, we believe that an essential principle for any protocol designed to find, block, and prevent network abuse is to not rely on any ISP. Restated, providing connectivity must not imply accepting responsibility.

We take this principle one step further; that a sender’s ability to inject traffic into the network should be decoupled from his point of attachment. Blocking a misbehaving source should block a principal who cannot reconnect with a new address in a new network (e.g., at a coffee shop) or spoof a different address. This leads us to incorporate identities issued as public keys from a globally-known, trusted authority. This trusted accountability provider (TAP) prevents arbitrary users from joining and, more importantly, bars blocked subscribers from re-joining under a pseudonym. We generalize this design to support many TAPs, each with its own set of policies and underlying implementations, which agree on a protocol for validating packets in transit. That is, we attempt to design for the tussle [4], not for an outcome.

In this paper we present NA, a complete network accountability system in which:

1. An accountability provider is the sole trusted entity; all other participants (routers, accountability provider representatives, etc.) can be implicated if they misbehave. We ensure that its services need not be implemented centrally, allowing it to be made scalable and resilient to attack.

2. Blocking filter state is not stored at any router; a combination of Passport [9]-inspired tokens and DHT-based filter storage compels the source network to check for blocks. We show how to make filter storage reliable on untrusted DHT nodes.

3. Blinding keys, rather than long-lived identifiers, are exposed to destinations, preserving (some) privacy. Machines near senders guard the sender’s long-lived identity, though we do not explicitly anonymize clients or servers as in SOS [6].

4. What constitutes abuse is determined by a destination alone, it need not be proven to an authority: if traffic is undesired, future traffic can be blocked. The sender can continue to send to other destinations.

5. Cryptographic operations can be largely avoided along the fast path through “waivers,” explicit statements

\(^1\)Protecting their customers from outside attack may be a separate service.
that a server will not be implicated by a client, and “tokens” that enable traceback.

To implement this network accountability system, we designed, implemented, and evaluated:

1. An accountable distributed hash table that, with the supervision of the trusted accountability provider and an assumption that no more than \( k \) consecutive nodes have been compromised, provides guaranteed service. Once published, blocking filters cannot hidden by DHT nodes or ignored by clients.

2. Protocol mechanisms (a) for high-performance verification of accountable traffic without all-pairs shared keys, (b) for traceback of unaccountable traffic (improperly signed, but carried anyway) to a misbehaving inter-domain link, and (c) for revoking permission to connect without a separate, protected control channel.

The rest of the paper is structured as follows: we summarize related work in Section 2. In Section 3 we present the design of NA, and in Section 4 we describe the NA secure DHT. We discuss attacks on NA in Section 5 and NA partial deployment approaches in Section 6. We evaluate NA overhead in Section 7, and conclude in Section 8.

2. RELATED WORK

NA allows receivers to block specific senders. Several recent proposals achieve similar goals using different mechanisms.

In TVA [18], senders initiate a connection by sending a capability request on a dedicated channel; this request is marked by each router along the way. If the traffic is desired by the receiver, the marked request is returned to the sender. The sender then includes this returned capability in subsequent packets. During an attack, packets without capabilities are preferentially dropped. Portcullis [11], an extension to TVA, uses sender-solved puzzles to protect the capability request channel. Unlike NA, TVA and Portcullis both pin the network path between the sender and the receiver (and require a symmetric path for optimal performance). NA relies on the TAP to issue global identities that enable receivers to block senders regardless of their network point of attachment; such a facility does not exist in TVA or Portcullis. Finally, NA requires only the TAP nodes be correct and trusted; all other entities may be corrupt. TVA and Portcullis require a trusted and correct transit infrastructure.

AIP [1] binds self-certifying addresses with network interfaces and assume that all interfaces have trusted hardware that obeys a “shut-off” packet sent by a receiver. Using these primitives, AIP builds an accountability infrastructure that receivers can use to stop abusive senders. Like NA, AIP provides receivers fine-grained control over which senders are blocked; unlike NA, the blocks in AIP are bound to network interfaces and not to packet sending principals. NA relies on trusted authorities to certify network-layer-independent identities. In comparison, AIP relies on a pervasive trusted hardware deployment at end hosts. Packet passports [9] is a system for unambiguously identifying a packet’s source AS. This information can be used to track abusive senders. The packet passport system relies on a shared key between every pair of participating ASes (whether they are neighboring ASes or not). These keys are used to create HMACs (like tokens in NA) that can eventually be used to identify the source AS of any packet. Passports require the predetermination of AS paths for every end-to-end route. Bender et al. [3] assume a trusted identity authority (like NA), but require inter-AS shared secrets (they build on packet passports). Their system attempts to provide the same functionality as NA (decouple connectivity from responsibility) but requires additional assumptions. The requirement of shared keys between every AS pair (packet passports) vs. pervasive trusted hardware (AIP) vs. trusted identity authorities in NA represent explorations in three starkly different points in the design space for network-layer accountability. NA minimizes the number of trusted entities in the system (only the TAP nodes are trusted) and is the only system that can withstand malicious transit routers and ASes.

Simon et al. [13] present a system where receivers can identify the source of traffic and block traffic from any source. Their scheme consists of per-customer ingress filtering at ISPs, and a trusted Filter Request Server (FRS) located in each ISP. To block a source, a receiver contacts the local FRS, which in turn contacts the FRS in the source’s ISP. The source’s FRS is responsible for filtering traffic from that source to the receiver. AITF [2] uses a mechanism similar to IP Record Route to mark the provenance of packets. When a receiver wants to block traffic that follows a certain path, the receiver asks its gateway to contact the sender’s gateway, similar to how FRSEs contact each other. In comparison to these systems, NA’s notion of blocks transcend the sending principal’s network point of attachment. Perhaps more significantly, these schemes hold the sender’s AS responsible for traffic from the AS and for blocking abusive senders. In fact, AITF includes a provision for disconnecting an entire AS if blocks are not properly administered. In NA, destinations block individual sending principals; once blocked, these senders can no longer obtain necessary tokens to send packets to the blocked destination. The source AS is unaware of specific blocks, and instead simply verifies that only accountable packets depart its network.

While not expressly providing accountability, NUTSS [5] is an Internet architecture that ensures that connection requests follow user-defined policies, including those that
prohibit a sender from sending to a receiver. These policies are stored and applied by “P-box” nodes near the destination. In NA, such policy prohibits senders from obtaining tokens to send to the destination.

Finally, IP traceback [12, 14, 17] is a mechanism that victims invoke to (partially) trace back the path an attack packet traversed. Traceback mechanisms require minimal changes to hosts and can potentially be used as a building block for an end-to-end accountability solution. The “accusation" protocol in NA uses tokens to traceback unaccountable packets. The NA traceback mechanism always isolates the malicious entity that sourced the attack packets.

3. **NA: DESIGN**

The primary goal of NA is to ensure that only accountable packets may transit networks to destinations that have not blocked the sender. Accountable packets are those signed by a principal authorized by a trusted authority. Waivered packets require no signatures, but carry proof that the destination expressly consented to receiving such packets. We term “unaccountable" those packets with an invalid signature, and “legacy" those packets having no accountability material at all.

In NA, principals send packets to hosts. Each packet contains sufficient information for a destination host to block the sending principal (for any reason). NA provides the following property:

Packets are forwarded to a destination only if the destination has not blocked the sending principal, regardless of the sender’s network point of attachment.

Lesser goals that shape our design are as follows. We wish to distribute verifiable functions away from the trusted authority, to protect it from attack or overload with mundane requests; this leads to complexity but should provide resilience. We limit computation cost in transit networks by avoiding asymmetric cryptography in transit; this leads to larger packets. We avoid revealing user identifiers: even though IP addresses could be used to correlate accesses by the same user, we prefer that network-layer accountability not require revealing identity to destinations. We wish to allow return traffic to be sent cheaply; this allows servers and high-volume connections to avoid much of the NA overhead. Finally, we wish to support extremely many fine-grained blocks in the system, leading to a need to distribute the blocks across many servers.

We divide time into epochs on the order of minutes. All participants are synchronized to within an epoch; all certifications of public keys have an expiry time expressed as the expiration epoch. In our implementation, we use seconds since midnight UTC, January 1, 1970, as returned by `gettimeofday()`, divided by 512 to provide an easily computed, uniform epoch in 32 bits.

A sender may be able to continue sending packets to a destination even after the destination has blocked the sender. However, the “period of vulnerability” is bounded by the epoch. A blocked sender may also be able to send packets if it can find aid from a corrupt component; however, NA will expose the corrupt entity to keep principals blocked.

### 3.1 Components Overview

The NA architecture consists of five components which we describe in turn.

**The TAP** The TAP is the trusted accountability provider. Its role in the architecture is to certify and renew the keys of well-behaved, but untrusted, participants (each of which is described below). It must be able to accept proofs that a component misbehaved (signed statements that catch the component in a lie), deny certificate renewal to those participants, and not reissue new certificates to the same. All entities know the TAP’s public key and can verify TAP signatures.

The TAP is the only trusted entity in the system. We assume that the TAP never divulges its private key and always discharges its protocol obligations correctly. The TAP may be implemented in a distributed manner with arbitrarily many replicas. These replicas need not synchronize any state, except for the current TAP public/private key pair and a list of principals whose keys are not to be renewed in the current epoch.

**Packet-sending Principals (Users)** A principal (or a user) in NA is identified by a public key, \( P_{\text{pk}} \), which the TAP certifies (by signing) through an out-of-band mechanism. In Section 6, we define principals that forward legacy traffic (from existing machines, bearing no signatures) into the accountable network, likely with limitations. These legacy gateways appear to the architecture as ordinary users that can be blocked.

**Autonomous Systems (AS)** If the entire network were under the control of the same administrator, protecting against misbehavior would be simpler. Although no part of NA requires that the trust domains be split along autonomous system boundaries, we use the term AS because the idea connotes responsibility for routers and ownership of IP address ranges for hosts. The terms of pairwise connections between ASes are encoded in service level agreements, similarly, these pairwise connections may represent different levels of trust or different ways to address misbehavior.

Administrators of ASes have speaks-for keys, signed by the TAP, to express that the administrator is permitted to place blocks on behalf of specific destination addresses or to grant waivers that allow return traffic to be unsigned. These keys may be delegated, restricting sub-keys to more-specific address prefixes. The TAP
bonds a speaks-for key to a prefix by including the prefix its the signature of the key.

Neighboring ASes periodically exchange a symmetric key; border routers can create and verify message authentication codes (as realized in tokens, described below in Section 3.5) using this key.

An AS may permit hosts to exchange packets within the same AS without verification; one might allow traffic to APRs, to diagnostic services, or to bootstrapping (DHCP or DNS) servers, for example, without signatures. However, we expect that such legacy traffic would not be accepted by neighboring ASes. An AS may choose to transit, limit, or (likely with increased deployment) outright reject legacy traffic, but must not transit unaccountable traffic.

Routers in an AS may be malicious or be compromised; we describe attacks by routers in Section 5.3.

APRs APRs are servers, distributed in the network, that certify short-lived blinding keys for legitimate principals and generate initial tokens after checking for a block or validating a waiver. Blinding keys hide principal keys to provide a measure of unlinkability across connections. Tokens are an optimization that allow ASes to forgo asymmetric cryptographic operations.

Each APR has a key signed by the TAP. Each AS contains at least one APR; like neighboring ASes, APRs periodically exchange a symmetric key with its host AS.

APRs may be malicious or compromised.

Secure DHT nodes The list of blocks—statements of the form “10.0.16 rejects packets from Principal A”—is maintained in a DHT, secured using protocols described in Section 4. The DHT nodes may be co-located with APRs or be independently provisioned.

3.2 An Accountable Packet

Accountable packets have a valid signature chain from the TAP to the packet: the TAP’s signature of the APR’s public key, the APR’s signature of the blinding key, and the blinding key’s signature of the packet contents. This chain provides a means to identify and block the sender of the packet: the sender is responsible for the packet. The APR is responsible for blocking the sender. The TAP is responsible for ensuring correct behavior of the APR. If the sender or APR is not behaving according to protocol, the destination can appeal to the next-highest authority present in the signature chain.

Verifying the entire signature chain of a packet in transit would be computationally infeasible. Thus, certain routers in every AS place tokens in the packet to signify that they have checked the signatures, or trust someone that has. An initial token, provided by the APR, indicates that the destination has not blocked the sender. Subsequent tokens, added by each AS, indicate that the signatures verify—or the packet was received from another AS that claimed the signatures verify.

To further reduce the computational overhead of sending traffic, waived traffic is exempted from signatures. If a sender S trusts return traffic from an intended destination D, S may give D permission to send return traffic without inserting any signatures into the packet. D’s APR verifies that S has permitted unsigned return traffic before issuing an initial token to D.

3.3 Process Overview

Figure 1: Life cycle of accountable packets.
We next outline how NA operates in normal operation, with misbehavior only by the source principal. We will then describe the steps in more detail. Assume that Alice is a principal, certified by the TAP, who wishes to talk to address dstIP; in turn, dstIP will wish to block Alice from sending further messages. Figure 1 illustrates this process and participants at a high level.

### Registering a blinding key

1. Alice commits to a blinding key that she will later use to obtain tokens from a nearby “source” APR (sAPR) (Section 3.4).

### Obtaining an initial token

1. Alice contacts sAPR with a registered blinding key and an intended destination (dstIP) (Section 3.5).
2. sAPR uses the secure DHT lookup protocol to find any block that dstIP’s AS’s administrator may have placed on Alice (Section 4).
3. If there is no block, sAPR provides a token that enables Alice to send to dstIP (Section 3.5).

### Constructing a waiver (optional)

1. Alice signs a message for the destination’s APR that will instruct it to provide tokens to the destination without a DHT lookup (Section 3.7).

### Packet Transfer

1a. Alice sends her outgoing packets with her blinding key and includes an initial token; a router in her AS checks signatures and the token. Or,
1b. For waivered packets, Alice hashes the waiver token with the message.
2. Each egress router places a token in the packet. Upstream ingress routers discard packets with invalid tokens. The process repeats on every peering link (Section 3.6).

### Placing a block

1. The administrator of dstIP decides to block Alice. He places the block by sending the blinding key to sAPR, and
2. sAPR uses the secure placement protocol to place a block using $P_{pk}$ (not the blinding key) into the DHT (Section 4), and returns a receipt to the administrator (Section 3.8).

Alice must receive new tokens before an epoch expires. Once sAPR receives a request to block Alice from sending to dstIP, sAPR no longer gives new tokens to Alice (for sending messages to dstIP). Alice cannot change her network point-of-attachment and send messages to dstIP since her local APR will find the block in the DHT and not issue her a token.

In the rest of this section, we describe the messages exchanged for each of these sub-protocols. We denote a principal $A$’s public and private key pair by $A_{pk}$ and $A_{sk}$ respectively, a message $m$ signed using key $k$ by $[m]_{k}$ and a message $m$ encrypted using key $k$ by $\{m\}_{k}$.

### 3.4 Registering a blinding key

Assume Alice that has generated a key pair $(P_{pk}, P_{sk})$, that the TAP has certified $P_{pk}$, and that Alice wants to send messages to dstIP. Assume that sAPR is within Alice’s AS (AS1, say) and that Alice’s host is pre-configured with sAPR’s address. Alice generates a blinding key pair $(B_{pk}, B_{sk})$ and commits to it by signing $B_{pk}$ with $P_{sk}$. The format of Alice’s commitment (CommitAlice) is CommitAlice $= [B_{pk}, \nu_{B}]_{P_{sk}}$ where $\nu_{B}$ is the epoch until when Alice proposes to use this blinding key. The key registration process is as follows:

- $Alice \rightarrow sAPR: \text{CommitAlice}, [P_{pk}]_{TAP_{pk}}$; registration request
- $sAPR$ checks that the TAP signed $P_{pk}$
- $sAPR$ stores $B_{pk} \rightarrow P_{pk}$

$sAPR \rightarrow Alice: [B_{pk}, \nu_{B}]_{\text{sAPR}_{pk}}$; signed blinding key

Alice may register as many blinding keys as sAPR will allow.

### 3.5 Obtaining an initial token

Initial tokens are bound to a blinding key, $B_{pk}$, a destination, dstIP, an epoch, c, and an issuing APR. Initial tokens make the statement “a valid $P_{pk}$ registered $B_{pk}$ and dstIP’s administrator has not blocked $P_{pk}$.” An initial token is required to send packets to a different domain and is checked by routers in Alice’s domain.

When Alice wishes to communicate with dstIP, she obtains an initial token as follows:

- $Alice \rightarrow sAPR: [B_{pk}, \nu_{B}]_{\text{sAPR}_{pk}}, \text{dstIP}$; token request
- $sAPR$ verifies its signature on $B_{pk}$
- $sAPR$ verifies that $c \leq \nu_{B}$
- $sAPR$ retrieves $P_{pk}$ and checks the DHT
- to ensure Alice has not been blocked by dstIP

$sAPR \rightarrow Alice: T_0$; initial token

The format of the initial token (T0) is:

$H(K_{\text{sAPR} \rightarrow \text{AS}1, B_{pk}, \text{dstIP}, c, sAPR_{pk}, sAPR_{TAP}})$

where $K_{A \rightarrow B}$ is the shared symmetric key between $A$ and $B$ ($A, B$ are neighboring ASes or an APR and the AS it is in). $H(\cdot)$ is a secure MAC, such as HMAC.

### 3.6 Packet Transfer

With an initial token, Alice can send a packet to dstIP. She signs her message with her blinding key and includes her initial token in the packet. Figure 2 shows the fields Alice embeds.

Forwarding an accountable packet consists of first, verifying the material within the packet, and second, appending the next token that the subsequent AS will verify. These two steps may be separated, so that packets are verified when they enter a network and given tokens just before they exit.
Figure 2: Accountable packet format. The rightmost column represents the number of bytes required for each field, when using cryptographic primitives described in Section 7.

<table>
<thead>
<tr>
<th>Field</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP Header</td>
<td>20</td>
</tr>
<tr>
<td>Version + Flags</td>
<td>2</td>
</tr>
<tr>
<td>Transport protocol</td>
<td>1</td>
</tr>
<tr>
<td>TAP index</td>
<td>2</td>
</tr>
<tr>
<td>sAPRpk</td>
<td>4</td>
</tr>
<tr>
<td>sAPRpk</td>
<td>14</td>
</tr>
<tr>
<td>Bpk</td>
<td>14</td>
</tr>
<tr>
<td>[sAPRpk, sAPRIP, νsAPR]</td>
<td>40</td>
</tr>
<tr>
<td>[Bpk, νBpk, sAPRpv]</td>
<td>28</td>
</tr>
<tr>
<td>( m ) _alk allegiance</td>
<td>28</td>
</tr>
<tr>
<td>νsAPR</td>
<td>4</td>
</tr>
<tr>
<td>νBpk</td>
<td>4</td>
</tr>
<tr>
<td>c</td>
<td>4</td>
</tr>
<tr>
<td>( T_0 = H(K_{sAPR\rightarrow AS_1}, B_{pk}, dstIP, c, sAPRpk, sAPRIP) )</td>
<td>4</td>
</tr>
<tr>
<td>( T_1 = H(K_{AS_1\rightarrow AS_2}, B_{pk}, dstIP, c, H(m)) )</td>
<td>4</td>
</tr>
<tr>
<td>( T_2 = H(K_{AS_2\rightarrow AS_3}, B_{pk}, dstIP, c, H(m)) )</td>
<td>4</td>
</tr>
<tr>
<td>( T_{n-1} = H(K_{AS_{n-1}\rightarrow AS_n}, B_{pk}, dstIP, c, H(m)) )</td>
<td>4</td>
</tr>
<tr>
<td>Transport (message m)</td>
<td>( var )</td>
</tr>
</tbody>
</table>

Figure 3: Waivered return packet format.

The first gateway to see the packet must verify the embedded signature chain from the TAP to the message: \([sAPR, sAPRpk, νsAPR, sAPRpv] \_alk allegiance, [Bpk, νBpk, sAPRpk], and [m] \_alk allegiance\). It then checks that the APR information present in the initial token \( T_0 \) (\( K_{sAPR\rightarrow AS_1}, sAPRpk, and sAPRIP \)) belong to the same sAPR and that the token was created for Bpk. If any check fails the packet is dropped. These steps are unique to the first gateway because it is responsible for validating the entirety of the packet; later gateways check only that the packet was validated by a (transitively) trusted neighbor.

Before the packet leaves the network, a router stamps the next token on the packet. This token conveys that the packet is valid: that the signatures and previous token have been checked and that the next domain should transit it. The format of the next (and any subsequent) token is: \( T_1 = H(K_{AS_1\rightarrow AS_2}, B_{pk}, dstIP, c, H(m)) \). The packet arrives at the destination with a chain of tokens that can be used to diagnose in-network misbehavior (Section 5). Each token along the path conveys the same statement (that the token’s creator believes the packet to be accountable), even when it has not directly verified the signatures in the packet.

The use of tokens does not require shared keys between every AS pair or the predetermination of AS paths, as in Passports [9].

3.7 Waivers

High-volume servers and power-constrained devices may not be able to sign each packet they transmit. NA allows hosts to state with a “waiver” that unsigned return traffic will not be blocked. Hosts must possess or have access to a valid speaks-for key \( K_{sf} \) having control over the IP address; we expect the key to be delegated specifically for one address for a short duration, but a waiver may be generated by a service, such as the APR.

We assume that Alice can transmit a connection nonce n to Bob in secret; perhaps by encrypting it using Bob’s well-known or stored public key, in-band in an established secure transport protocol atop accountable packets, or by other means. Alice then sends a waiver of the form:

\[ K_{sf}, [K_{sf}] \_alk allegiance, AliceIP, c, H(n), AliceIP, c \_alk allegiance \]

where c is the epoch until which any node possessing n may send waivered packets to AliceIP. The waiver is only valid if \( K_{sf} \) is valid; if \( K_{sf} \) is delegated, the certificate chain from the TAP is included in the waiver.

Bob verifies the signature of the hash of the nonce, and forwards Alice’s waiver to its APR. The APR verifies that AliceIP created a waiver allowing Bob to send to Alice and grants a secret waiver token for AliceIP without querying the DHT. The format of a waiver token is \( H(K_{sAPR\rightarrow AS_1}, AliceIP, c) \). However, because this token is not bound to a principal, only to the holder of a nonce, it could be abused if eavesdropped. To keep this waiver token secret and prevent replay, APR encrypts it before transferring it to Bob. The waiver token is not included directly in the packet, but hashed with the message.

Bob does not include any signatures in packets for Alice; instead, Bob includes a HMAC of the packet computed using Alice’s nonce as the key. Bob’s AS allows waivered packets from Bob to depart without alarm. The format of the waivered packet is shown in Figure 3.

Figure 4 presents a state transition diagram showing how a sender transitions to and from being able to send waivered traffic. The sender traverses the path through obtaining an initial token to being able to send accountable packets. It may receive a waiver to send outgoing packets, \( \omega \); this causes it to obtain a waiver token from the APR and then begin sending waivered packets. From either sending state, the sender may add
3.8 Placing a Block

Suppose the administrator of dstIP wishes to block Alice. She forwards one of Alice’s packets to her AS administrator (say dAdm). Given a packet from Alice, dAdm can extract s APR’s address and Alice’s blinding key. Assume that dAdm possesses the speaks-for key corresponding to address block 1.1/16.

dAdm places the block:

$$\text{BlockReq}_{1.1/16} = \text{"1.1/16",}\ d\text{Adm}_{pk},\ B_{pk},\ dstIP,\ type,$$

$$\text{"1.1/16",}\ d\text{Adm}_{pk}]_{\text{TAP}_{pk}},\ [B_{pk},\ dstIP,\ type]_{\text{dAdm}_{sk}}$$

where type ∈ {“new”, “republish”}, as follows:

$d\text{Adm} \to \text{s APR} : \text{BlockReq}_{1.1/16} ; \text{block request}$

$s\text{APR} \to d\text{Adm} : r_1, ..., r_k, \{\text{Commit}_{Allice}\}_{\text{dAdm}_{sk}} ; \text{block receipt}$

The APR in the destination network, d APR, may grant tokens without consulting the DHT for the purpose of sending block messages to other APRs. This permits sending block requests while overloaded. d APR must inspect the combination of the block request and the packet headers to ensure that the request is not spurious: for example, that $B_{pk}$ followed a chain from the TAP and that the destination for the token is s APRIP.

Because blocks are stored in the DHT, they may, over time, be discarded; as the DHT changes, it is increasingly difficult to ensure that the DHT nodes that committed to storing the block still exist. A dAdm that insists upon permanent blocks may republish them before they expire. To facilitate republishing, the s APR returns an encryption of Alice’s public key. A republished block request includes this encryption to s APR, who decrypts it to find Alice’s commitment. After Alice’s original commitment expires, s APR may expunge it; it is the responsibility of dAdm to keep this state.

To block further waivered traffic, the victim must both block the principal used to solicit a waiver, if applicable, and refuse to renew the waiver.

3.9 Multiple TAPs

Our description focused on a single TAP. We expect many TAPs to be simultaneously deployed and for destinations to accept packets signed with identities issued by TAPs of their choosing. We assume that each TAP has a globally-unique index to be carried in all accountable packets and used to identify TAPpk.

Senders must know which TAPs a destination honors, and APRs issue tokens to a sender only if the sender has a key signed by a TAP that the destination accepts.

Administrators use $K_{sf}$ to sign a list of mappings from address prefix to TAPs they accept. This mapping is public and can be disseminated through DNS.

4. NA SECURE DHT

The NA DHT relies on a assumption that at least one of $k$ nodes having consecutive IDs is alive and not corrupt. With this property, the DHT guarantees both safety and liveness. The basic principles behind our secure DHT can be applied to any DHT; we use the terminology from Chord [15]. As in Chord, we assume that each data item is replicated on the $k$ nodes whose IDs succeed the item’s ID. DHT nodes implement the original Chord protocol for forwarding queries.

For each node, the TAP generates a certificate of the node’s neighborhood (the node, its predecessor, and $k$ successors). Time is divided into join and renew eras—each on the order of hours, compared to the epoch length of minutes. Join and renew eras alternate, so that an given epoch is in either a join or renew era. Nodes must renew their certificates at least once every renew era to remain in the DHT for subsequent eras. Nodes can join only during a join era. The TAP issues certificates, valid for the subsequent renew era, to joining nodes, as well as the nodes whose neighborhoods are affected by the join. Nodes with unchanged neighborhoods renew their certificates during the re-certify era.

DHT nodes may join the system by obtaining a certificate from any TAP replica; similarly, DHT nodes may renew their certificates from any TAP replica.

The format of the node certificate ($\text{nodeCert}_n$) for node $n$ is:

$$\text{nodeCert}_n = [n,\ pred(n),\ succ(n), ... ,\ succ^k(n),\ \eta]_{\text{TAP}_{pk}}$$

where each node $n'$ is represented as:

$$n' = [\text{IP},\ \text{port},\ [n'_{pk}]_{\text{TAP}_{pk}}]$$
and \( \eta \) is the era when the certificate was issued. The certificate expires in the renew era following the era in which it was issued. The ID of node \( n \) is \( H(n_{pk}) \) for some hash function \( H(\cdot) \).

The NA DHT defines the following operations: (1) Join, (2) Lookup, (3) Block, and (4) Renew.

**DHT Join** Joining node \( n \) queries the DHT to find the node \( r \) that is responsible for \( n \)'s ID. \( n \) collects the certificates of \( r \)'s \( k \) predecessor nodes, \( l_1, \ldots, l_k \). Then \( n \) finds a TAP replica and presents the certificates to the TAP. The TAP verifies all certificates as well as the signature of \( n \)'s public key.

The TAP node generates new node certificates (for the next era, \( \eta \)) and distributes them as follows:

\[
\begin{align*}
\text{TAP} & \rightarrow n : [n, l_k, r, \ldots, r_{k-1}, \eta]_{\text{TAP}, k} \\
\text{TAP} & \rightarrow r : [r, n, r_1, \ldots, r_k, \eta]_{\text{TAP}, k} \\
\text{TAP} & \rightarrow l_1 : [l_1, l_0, l_2, \ldots, n, \eta]_{\text{TAP}, k} \\
\vdots \quad & \quad \quad \\
\text{TAP} & \rightarrow l_k : [l_k, l_{k-1}, n, \ldots, r_1, \eta]_{\text{TAP}, k} \\
\end{align*}
\]

\( k \) preds. updated

**DHT Lookup** Nodes maintain and update their finger tables as in Chord. For each neighbor, \( k \) successors, and finger table entry, nodes store the full node certificate (instead of only the node ID and address as in Chord), which is returned during a route lookup. Progress along the ring is assured since at least one in every \( k \) nodes is known, correct, and alive.

NA APRs incur the full logarithmic overhead of DHT lookups only for new senders. An sAPR caches the address of the DHTs node responsible for a new sender, and directly tries the cached addresses (followed by its neighborhood if necessary) for subsequent DHT lookups or block placements.

When queried on key \( P_{pk} \), DHT nodes return its entire record for \( P_{pk} \). This limits the number of queries an APR makes to one per source per epoch.

**DHT Publish** Suppose sAPR has received a block request from dAdm. sAPR uses the DHT lookup procedure to find \( k \) DHT nodes \( (d_1, \ldots, d_k) \) that Alice’s public key maps to. sAPR forwards the block request BlockReq\(_{1/16}\) and Alice’s commitment to each of these DHT nodes.

\[
\begin{align*}
\text{sAPR} & \rightarrow d_1 : \{\text{BlockReq}_{1/16}, \text{Commit}_\text{Alice}\}_{\text{sAPR}} \\
& \quad : \text{Node } d_1 \text{ verifies block request and commitment} \\
d_1 & \rightarrow \text{sAPR} : d_{pk}, \{d_{pk}, \text{BlockReq}_{1/16}\}_{\text{d}_i}, \\
& \quad \ldots \text{and returns a DHT receipt.}
\end{align*}
\]

**DHT Renew** Before the expiry of a renew era, each DHT node \( d \) must get a new node certificate from a TAP node for the subsequent renew era. The renew protocol proceeds as follows: \( d \) presents its current certificate to a TAP node, which queries \( d \)'s \( k \) predecessors, obtains their latest node certificates (including any for the next renew era that may contain a new successor for \( d \)), and issues a new node certificate for \( d \) if and only if all of the information is consistent. If any of the predecessors can produce a certificate, newer than \( d \)'s, with a different view of \( d \)'s neighborhood, then \( d \) is not given a new node certificate. Such a (conflicting) certificate may exist if a new node had joined in the prior join era, and \( d \) had decided to suppress the certificate it had received during that join.

### 4.1 Security Properties

NA DHT nodes store full node certificates in their finger tables; possession of a certificate for any node in a given span of the DHT implies knowledge of a correct node nearby. A (maliciously) unresponsive node could drop any forward, lookup, or publish request sent to it. In that case, the sender of the original request would simply contact the other nodes in the unresponsive node’s certificate. This (coupled with item replication) guarantees that all forward, lookup and publish operations succeed.

An honest node \( h \) that has recently inherited a new span (say \( (l, R) \)) of the ID space may inadvertently return an incorrect negative response, for instance, if not all items have been transferred or republished. Node \( h \) can detect this condition since the republish interval for a block is globally known. Suppose \( h \) receives a query from \( s \) in the range \( (l, R) \) for which it has no item. \( h \) returns its node certificate to \( s \) along with a statement specifying that its negative response is not authoritative. \( s \) can find a node in \( h \)'s node certificate which will be able to provide an authoritative answer.

A malicious node \( m \) cannot provide a false answer to a lookup. Such a response, which must be signed by \( m \), can be used to provably implicate (and expel) \( m \), since \( m \) must have generated a conflicting DHT receipt when the item was originally published.

Malicious nodes may try to prevent other nodes from joining, so that, for instance, failed nodes cannot be replaced, or certain neighborhoods are preserved. In Chord, two consecutive nodes can forever prevent a new node from joining in between them by never updating their successor and predecessor pointers. The NA join and renew protocols expose such behavior. Nodes affected by a join receive two certificates valid for a given renew era \( \eta \): the renewed certificate given in the previous renew era (era \( \eta - 2 \)) and the updated certificate (reflecting the join) in era \( \eta - 1 \). If a node \( n \) presented the first certificate for renewal (to get a certificate for \( \eta + 2 \)), one of \( n \)'s \( k \) predecessors is guaranteed to present a (conflicting) certificate, signed in era \( \eta - 1 \) by the TAP during the interactive phase of the renew protocol.

## 5. ATTACKS

In the previous sections, we have described the default operation of NA, without failures or node corrup-
In this section, we show how NA detects misbehavior and eventually evicts malicious nodes.

We classify attacks into four categories based on how attackers are identified:

**NA:** The protocol messages in NA include sufficient information to unambiguously implicate corrupt nodes.

**Local:** The NA protocol can resolve the misbehavior to two neighboring entities, either of which may be corrupt. However, the “good” entity knows the corrupt neighbor and modify its trust policy accordingly.

**DHT:** Corrupt DHT nodes are identified and removed using the secure DHT protocol from Section 4.

**TAP:** Some attacks require the assistance of a TAP node for resolution. Victims provide the messages that indicate a corrupt entity to the TAP, which can verify the deceit, decide to not recertify the node for future epochs, and communicate the decision to replica TAPs.

Table 1 matches the most prominent attacks NA components may attempt with the mechanism that would isolate the attacker. In the rest of this section, we describe selected attack resolution mechanisms in detail.

### 5.1 Source-Generated Attacks

**Source floods** NA enables destinations to explicitly block source principals. If a source floods a destination domain with accountable traffic (i.e., packets with valid blinding and APR keys), the victim network can block the source. We ensure that our block request message (from the dAdm) fits into a single Internet MTU IP datagram and does not require an acknowledgment for the block to be activated. Thus, the block protocol will be able to block the source in the next epoch, even if the source can saturate the victim’s access link.

If the source sends unaccountable traffic (i.e. traffic with an invalid signature), then the source’s domain—which is responsible for verifying each signature—will discard the packet.

If the source sends legacy traffic (without an accountability header), then the traffic may be forwarded or discarded as described in Section 6.

**DHT storage exhaustion** A malicious sender may try to overload DHT nodes by attacking many destinations, causing DHT nodes to hold increasing block information and generate block receipts and causing its APR to verify many DHT receipts. We expect TAPs to restrict the number of distinct simultaneous blocks on a principal, after which the principal will not be issued.

Table 1: NA attacks and resolution. Items with a * are explained in Section 5, † in Section 4.1
new tokens, and may be excluded from further participation in the system. Corrupt colluding ASes could try to get an honest principal (h) blocked by issuing blocks on h. However, they could only do so if h had sent packets to each of these ASes. Any reasonable limit on the number of blocks (say 20-50) would ensure that honest principals are not susceptible to this attack.

5.2 Corrupt APRs

Flooding Like any sender, a corrupt APR may try to flood a destination, a DHT node, or a TAP node. If the APR node sends packets using a blinding key registered to APRpk, the blinding key would be blocked as if it were a normal sender. If the misbehaving APR does not honor the block, the administrator who blocked the APR does not receive receipts and forwards the request to the TAP, who eventually evicts the APR. If the APR does return receipts, but continues to send traffic, the administrator presents the TAP with the returned receipts and an example of the traffic that violates the contract. The TAP evicts the APR.

An APR is unable to send packets using a registered blinding key (without the corresponding Bsk, it is unable to sign packets). A corrupt APR can never implicate an honest user of misbehavior.

However, a corrupt APR may generate valid tokens for a blinding key that does not map to any registered user and use these tokens to attack a destination. All signatures in the packet verify. The victim administrator tries to block the source principal, but the block request does not succeed since the corrupt source APR is not able to produce a valid commitment (which the DHT nodes require in order to produce a DHT receipt). The destination administrator fails to receive receipts from the source APR and complains to a TAP node. The TAP node challenges the source APR to produce a valid blinding key commitment, at which point the source APR will be exposed.

Ignoring block requests If an APR ignores block requests (or is unable to return the necessary number of receipts), the victim administrator must appeal to the TAP. The TAP will challenge the source APR, and eventually ensures that the block is placed (or evicts the APR).

Forgoing lookups or issuing tokens to a blocked sender A source and an APR may collude. The APR could issue a token allowing a sender (using a different blinding key) to send packets to a destination that has previously blocked the sender. APRs that forgo DHT lookups may do the same. In this case, the destination will presumably block the sender a second time. When the second block is published to the DHT, the storing node will observe that the destination has blocked Ppk twice—an indication that the APR misbehaved. The DHT node will not return a receipt, and may alert the TAP.

5.3 Corrupt Routers

Corrupt routers may fabricate packets with (bogus) accountability headers, resulting in packets with an invalid signature chain—preventing the destination from blocking the sending principal—to arrive at a destination. Similarly, routers may not check packet signatures, allowing unaccountable traffic to enter the network. Additionally, a corrupt router may simply replay valid packets to flood the victim.

Generating or forwarding invalid packets (Accusation Protocol) If a victim receives an unaccountable packet, it must rely on the tokens to isolate the faulty node. The victim’s administrator inspects the last token in the packet and determines which of its upstream ASes the packet came from (by determining which shared key was used to check the token). The victim administrator then “accuses” the upstream AS of sending an invalid packet. The upstream AS (say AS1) verifies the last token, and tries to map the second-to-last token to one of its upstream neighbors. If this process is successful (and the second-to-last token maps to AS2), AS1 “accuses” AS2. This process recurses until the packet traverses the last “good” AS (say ASg) into the AS that generated it (say ASb).

Note that all that is required to verify a token is the blinding key, destination IP, epoch the packet was created, the hash of the packet contents, and the token itself. Thus accusation messages (containing the full token chain), like block requests, fit into a single packet.

When the original packet was generated, the corrupt router in ASb may have fabricated all upstream tokens, in which case the ASb administrator will have no other upstream AS to blame. In the worst case, the ASs router may have been able to add one “valid” upstream token, since it may know the secret shared with the upstream AS.

In the first case, either the ASb administrator will locate and expunge the corrupt router (and stop the attack), or the attacks will continue. If the attacks continue, each AS will continue to receive accusations from downstream routers and ASg must apply some local policy. This policy could include technical solutions (e.g., check signatures on incoming packets) or monetary remedies (e.g., the AS peering SLA could state that the offending AS will pay a small sum for each unaccountable packet that it transits).

Suppose the AS chooses to validate signatures if the accusations continue. This will enable ASg to immediately isolate ASb (since ASg will immediately know that ASb is transiting unaccountable packets). ASg will not forward these packets, stopping the flow of accusations. If ASb continues to source unaccountable packets, ASg
may take further action.

If the SLA between AS_{g} and AS_{b} includes the monetary provision, then AS_{b} will have to pay for each unaccountable packet. AS_{g} may have also had to pay its downstream AS (after all, AS_{g} did transit an unaccountable packet); however, in this case, AS_{g} recoups its cost because of the payment from AS_{b}.

If the router in AS_{b} added a “valid” token for a previous hop AS (since the neighboring ASes share a secret key), AS_{b} will be able to “blame” a good upstream AS (say AS_{up}). However, AS_{up} will not be able to map the packet to any of its upstream neighbors. If the AS_{up} administrator can immediately assert that its routers have not been corrupted, it can isolate AS_{b}.

However, it is often difficult for a network administrator to certify that none of the hosts or routers in its AS has been compromised. In this case, AS_{up}’s administrator can locally audit the router between itself and AS_{b}; ensure it is not compromised, and verify the signatures of all packets traversing the peering link. If the accusations continue, then AS_{b} is unambiguously implicated, and AS_{up} may choose to renegotiate the peering.

Verifying signatures on every packet may be prohibitively expensive on high speed links. AS_{up} can implicate AS_{b} without verifying packet signatures. Once the AS_{up} administrator verifies that its peering router with AS_{b} (say R) is not compromised, it generates a temporary symmetric key (k), and instructs R to add two tokens on each outgoing packet. The first outgoing token is computed using k (known only to R) and the second token is the regular token computed using the key shared between AS_{up} and AS_{b}. If subsequent accusations from AS_{b} includes a token signed with k, then some host or router within AS_{up} is compromised, and the AS_{up} administrator has to audit its internal routers. However, if AS_{up} is not compromised, then the accusation packets from AS_{b} will not contain a token signed with k (since AS_{b} does not know k). However, all packets transited from AS_{up} included a token signed with k; this enables AS_{up} to unambiguously implicate AS_{b}.

Replaying valid packets A router can replay a fixed packet only until the next epoch, when it will be dropped by subsequent domains. However, this does not solve router replay in general. If routers persist in replaying accountable packets, Bloom filters can be installed in upstream domains, as in Passports [9]. When the Bloom filter detects a replay, the duplicate packet is stored. Further duplicates are compared against stored packets to ensure that the match was not a false positive. The filters can be refreshed every epoch, limiting the storage needed.

5.4 Attacks using waivered packets

Assume Alice sent a waiver allowing Bob to send unsigned packets to Alice. When Alice receives an unsigned packet, she can use the HMAC to assert that she had explicitly permitted the sender to send unsigned packets. If the HMAC in the packet is corrupt (or not present), Alice may use the usual “accusation” protocol to stop/isolate the sender.

If Alice were malicious, she may try to evict Bob (who is good) by initiating the accusation protocol on packets that include a proper HMAC. The tokens in the packet would eventually terminate at Bob, who can produce Alice’s waiver explicitly allowing Bob to send unsigned packets. Bob does not have to reveal his private key to decrypt the nonce in order to prove that Alice sent a waiver; he only has to produce the nonce and show that Alice signed a hash of the nonce. Bob now accuses Alice (tracing the packet tokens forward) of misbehavior. Bob’s accusation includes Alice’s waiver, her accusation message, and the nonce in cleartext:

\[ K_{sf}, [K_{sf}]_{\text{TAP}_{pk}}, \text{"accuse"}, \text{nonce} \]

Each AS on the forward path can now independently check that (1) Alice did allow Bob to send unsigned messages, (2) check that Alice signed the nonce and (3) that Bob had included a proper HMAC on the return packet. This evidence is sufficient for Alice’s AS to isolate Alice’s host (or for Alice’s AS’s upstream to isolate Alice’s AS).

Bob’s AS does not check signatures on waivered packets. However, this does not allow any new attacks. If Bob sends malicious traffic to a destination that has given a waiver to Bob, the victim domain will not re-new Bob’s waiver, and the APR in Bob’s domain will stop issuing tokens to Bob when the waiver expires.

6. PARTIAL DEPLOYMENT

NA does not require global deployment. Partial deployment solutions depend on whether the original sender is NA-aware and whether there are legacy transit domains between accountable domains. We assume that all senders and destinations in an accountable domain are NA-aware; however, NA-aware principals may send packets from legacy domains.

<table>
<thead>
<tr>
<th>Sender</th>
<th>Transit</th>
<th>Dest.</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS</td>
<td>ASes</td>
<td>AS</td>
<td></td>
</tr>
<tr>
<td><strong>NA-aware sender</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Legacy</td>
<td>*</td>
<td>*</td>
<td>Sender tunnels via Acct. gateways; sends legacy traffic if acc. gateway not found</td>
</tr>
<tr>
<td>Acct. Some legacy</td>
<td>Acct.</td>
<td></td>
<td>Acct. ASes tunnel to each other</td>
</tr>
<tr>
<td>Acct. Some legacy</td>
<td>Legacy</td>
<td></td>
<td>Acct. header stripped at last accountable AS</td>
</tr>
<tr>
<td><strong>NA-oblivious sender</strong></td>
<td></td>
<td></td>
<td>Legacy gateways take responsibility for legacy traffic</td>
</tr>
</tbody>
</table>

Table 2: Partial deployment scenarios.
Two new entities interact with legacy ASes. An accountability gateway enables NA-aware senders in legacy domains to send traffic to accountable domains. A legacy gateway forwards sanitized legacy traffic onto accountable domains.

Sending accountable traffic from legacy domains

Accountable senders in legacy domains may wish to send traffic to domains that only accept accountable traffic. The sender’s traffic is routed through an accountability gateway, APRgw.

Accountability gateways have a TAP-signed APR key that they use to issue tokens to senders that have committed to a blinding key and have not been blocked, much like a normal APR. Unlike a normal APR, accountability gateways issue tokens to senders in legacy ASes; these senders must tunnel accountable packets to the accountability gateway’s AS to use the token.

The blocking procedure is the same: Should a receiver decide to block Alice, it follows the standard blocking protocol by sending a block request to APRgw, who publishes the block in the DHT.

Transiting through legacy domains

Accountable ASes can tunnel over legacy ASes to extend the reach of deployed, accountable “islands”, much like the MBone and 6Bone. The tunnel endpoints exchange symmetric keys and tokens created with these keys span the tunneled “link”.

It may be the case that an egress router, r, in an accountable domain cannot forward an accountable packet to an accountable domain, for instance if it cannot find a tunnel that reduces the distance to the packet’s destination or if r determines that the destination is in an adjacent legacy domain. In this case, r removes the accountability header from outgoing packets, changes the IP protocol field to the protocol field in the accountability header, recomputes the IP header checksum, and forwards the resultant legacy packet.

Legacy senders

Legacy gateways allow legacy senders to send packets to accountable ASes. A legacy gateway (L) possesses a Prpk from the TAP. L accepts legacy traffic, creates and commits to a blinding key corresponding to the source IP address, and forwards the packets using the usual NA protocol. In effect, L takes responsibility for the legacy traffic by signing it. Destinations that block L no longer receive any legacy traffic through this gateway. Like any user Prpk, if sufficient destinations block L, its key may no longer be renewed by the TAP.

7. EVALUATION

We have implemented the secure DHT with a TAP and a NA router. We present microbenchmarks for each type of overhead from our implementation. The TAP and DHT are implemented in Ruby, using SWIG wrappers to OpenSSL [10] and the i3 [16] Chord implementation. Our DHT implementation uses the Chord implementation to route messages. We have augmented the Click router [7] to perform NA token manipulations and signature verification.

We use the OpenSSL implementation of ECDSA [8] for signatures and SHA-1 HMAC for tokens. ECDSA affords short public keys and signatures without expensive bi-linear pairing operations. We use 160-bit keys for TAPpk, Ppk, and speaks-for keys, and 112-bit keys for APRpk, Bpk and DHT node keys. (The latter three types are refreshed relatively often; their keylength can be shorter.)

We present an evaluation of the overhead for different NA sub-protocols. NA overheads can be partitioned into six broad categories:

**Accountable traffic:** Accountable traffic incurs processing overheads for signatures and verifications, latency overheads for block lookups, storage overheads for storing commitments, and packet overheads for storing an accountability header.

**Traffic with waivers:** Once waivers are issued, NA operations require no asymmetric cryptography. The primary overhead is in constructing and checking HMACs.

**APR and gateway:** APRs and legacy gateways must store blinding key mappings, publish blocks to the DHT, and verify DHT receipts.

**Administrator:** Administrators sign block requests, create waivers, and store and verify DHT receipts.

**DHT node:** DHT nodes must store blocks and return signed publish receipts and lookup responses.

**TAP node:** TAP nodes sign keys for principals, APRs, and DHT nodes. They resolve disputes and evict misbehaving APRs and DHT nodes.

The primary computational overhead in NA comes from signature creation, signature verification and HMAC operations. The time required for the signature computations on a typical workstation is shown in Table 4. The HMAC operations require less than 0.02 ms for 1500 byte packets.

Table 3 shows the number of different cryptographic operations required for NA component protocols, the number of local and remote round trips each protocol incurs, and the sizes of the protocol messages. We observe the following from Table 3:

**Line 2** One router in the source AS must verify two short signatures and one long signature for accountable packets. These verifications can be parallelized and two of them may be cached.

**Line 3** Principals can pre-commit to blinding keys and use them as necessary. There is no need to commit to a blinding key for each new connection.

**Line 7** The primary latency overhead for connection
setup is the DHT block lookup, which must occur once per new connection. This lookup usually incurs only a single remote hop (to a DHT node responsible for holding the principal’s blocks) since the sAPR can cache the address of the DHT node after the first lookup (which requires log(N) remote hops).

**Line 9** Creating a waiver requires two signature operations, only one of which must be performed at connection setup time. The nonces can be generated and signed offline.

**Line 13** Once a waiver is verified and a token obtained, no more asymmetric operations are required.

**Line 14** If a packet can be verified (three verification operations) and the destination wishes to block the sender, it needs to create one signature with its speaks-for key. Block requests fit in a single datagram.

**Line 17-18** If a destination receives a packet with an invalid signature or an invalid nonce, it invokes the accusation protocol. The accusation packets also fit into a single datagram, and require no signatures beyond the accountability header. Each successive upstream AS must check their token (one HMAC) and then compare the previous token against all keys shared with their other AS neighbors (one HMAC operation per neighbor).

**Line 19** Publishing a block into the DHT requires k remote hops (no logarithmic lookup overhead) since the sAPR has already cached the address of the DHT node responsible for the blocked principal.

**Line 23,25** TAP nodes have to verify k + 2 certificates per join or renew operation. The interactive component of the renew protocol requires k + 1 RTTs (k RTTs to gather the k predecessor node certificates, and one RTT to distribute the new node certificate to the renewing node).

The compilation of overheads in Table 3 suggests that NA can be deployed in the wide-area using current hardware. In the remainder of this section, we use the analysis in Table 3, connection-level trace data from access links at different institutions, and the microbenchmarks from our implementation to extrapolate the performance of a hypothetical NA deployment.

**TAP overhead for DHT renewals** Assume a single TAP that operates a 4.096 node DHT and suppose the average TAP replica to DHT node latency is 100ms. and that a TAP node can send 200 (969-byte) packets per second (1.55 Mbps). If k is set to 8, four TAP nodes (each with the CPU capacity of a regular workstation as in Figure 4) can renew 4,096 DHT nodes in less than one minute (23 seconds of CPU time each and around five seconds of transmission time).

**Overhead at sAPRs due to token renewals** We

<table>
<thead>
<tr>
<th>Operation</th>
<th>Actor</th>
<th>Frequency</th>
<th>Signatures</th>
<th>Verifications</th>
<th>Local RTT</th>
<th>Remote RTT</th>
<th>Header size</th>
<th>Message size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sender</td>
<td>As needed</td>
<td>112-bit</td>
<td>112-bit</td>
<td>Local operation</td>
<td>185 var.</td>
<td>185 var.</td>
<td>185 var.</td>
</tr>
<tr>
<td>2</td>
<td>Router</td>
<td>As needed</td>
<td>112-bit</td>
<td>112-bit</td>
<td>Local operation</td>
<td>185 var.</td>
<td>185 var.</td>
<td>185 var.</td>
</tr>
<tr>
<td>3</td>
<td>Sender</td>
<td>As needed</td>
<td>112-bit</td>
<td>112-bit</td>
<td>Local operation</td>
<td>185 var.</td>
<td>185 var.</td>
<td>185 var.</td>
</tr>
<tr>
<td>4</td>
<td>sAPR</td>
<td>As needed</td>
<td>112-bit</td>
<td>112-bit</td>
<td>Local operation</td>
<td>185 var.</td>
<td>185 var.</td>
<td>185 var.</td>
</tr>
<tr>
<td>5</td>
<td>Sender</td>
<td>As needed</td>
<td>112-bit</td>
<td>112-bit</td>
<td>Local operation</td>
<td>185 var.</td>
<td>185 var.</td>
<td>185 var.</td>
</tr>
<tr>
<td>6</td>
<td>sAPR</td>
<td>As needed</td>
<td>112-bit</td>
<td>112-bit</td>
<td>Local operation</td>
<td>185 var.</td>
<td>185 var.</td>
<td>185 var.</td>
</tr>
<tr>
<td>7</td>
<td>sAPR</td>
<td>As needed</td>
<td>112-bit</td>
<td>112-bit</td>
<td>Local operation</td>
<td>185 var.</td>
<td>185 var.</td>
<td>185 var.</td>
</tr>
<tr>
<td>8</td>
<td>sAPR</td>
<td>As needed</td>
<td>112-bit</td>
<td>112-bit</td>
<td>Local operation</td>
<td>185 var.</td>
<td>185 var.</td>
<td>185 var.</td>
</tr>
</tbody>
</table>

Table 3: Overhead of NA sub-protocols. Header size reflects packets with five tokens. The DHT has N nodes and a replication factor of k. The size of a node certificate C is 74(k + 2) + 44. \[^{x..y}\] means x in expectation, but y in the worst case. All signatures and verifications for the same operation can be performed in parallel.
analyzed connection-level trace data from two institutions. We assume that each unique source address corresponds to a different principal.

Our first trace contained 20 hours of data and contained 4,238,872 connections from 2,096 unique sources—one of which was responsible for 990,396 connections alone. The sAPR conducts a regular DHT lookup for each new principal (host) and also contacts a known node (using a cached certificate) once per epoch per source if the source starts a connection in that epoch. Extrapolating using our implementation, the sAPR requires 322 seconds (0.4% average and 5.8% maximum CPU utilization) for all its cryptographic operations, requiring an average of 1.4 messages per second (2.27 Kbps) and a maximum of 58 messages per second (94 kbps) to all DHT nodes combined. We found that our experience with this dataset was not unique: we inferred similar resource requirements on a second trace captured from the access link of a major institution with two allocated /16 prefixes. The trace contained 2,416 unique IP addresses that made 16M outgoing connections over a 19-hour period. The computation time for tokens at both APRs never exceeds 0.16 seconds in either datasets.

**Under attack** The local network administration at the authors’ institution manually blacklists IP addresses by installing a firewall rule for persistent attackers. Between August 2007 and December 2007, 2,736 hosts were blacklisted (approximately 15 per day). The overhead of this level of blacklisting is negligible.

Next, we consider an extreme case where a single victim AS is synchronously attacked by a 10,000 node botnet, where each bot is from a different AS. This is the worst case attack since the victim AS cannot simply block corrupt sAPRs, it must install a new block per attacking principal.

Under attack, the administrator verifies three signatures and signs one message (plus the packet). The total CPU time per block is less than 7 ms; 10,000 block requests can be generated in 70 seconds. Each block request is 309 bytes in size, so an administrator with a 1 Mbps connection would require 25 seconds to contact all of the attacker’s sAPRs. Since block generation and deliver can happen in parallel, administrators require approximately 70 seconds to block 10,000 attackers in different domains. This is less than an epoch length; by the next epoch, the flood will be blocked.

**DHT capacity** Without a global deployment of NA, it is not clear how to evaluate the number of blocks that a DHT node is expected to store. In this regard, we examine how many blocks a typical workstation could manage. Incoming blocks can be processed at a rate of 314 per second; this would saturate the processor and a 1 Mbps link. Collectively, the 4,096-node DHT can complete 1.28 million operations per second.

<table>
<thead>
<tr>
<th>Key Size</th>
<th>Sign</th>
<th>Verify</th>
</tr>
</thead>
<tbody>
<tr>
<td>112 bits</td>
<td>(\mu = 0.91, \sigma = 0.03)</td>
<td>(\mu = 1.11, \sigma = 0.01)</td>
</tr>
<tr>
<td>160 bits</td>
<td>(\mu = 1.77, \sigma = 0.01)</td>
<td>(\mu = 2.07, \sigma = 0.02)</td>
</tr>
</tbody>
</table>

**Table 4:** Times in milliseconds to perform each operation on a 3.4 GHz Intel Pentium 4 CPU.

8. **DISCUSSION**

NA is a network-layer accountability architecture that explicitly decouples connectivity from responsibility. NA’s security is grounded in trusted identity certification authorities (TAPs). NA allows destinations to specify per-source principal blocks, which apply regardless of the principal’s network attachment point. NA does not mandate local policy, e.g., each AS is free to choose how it classifies misbehavior, and can withstand the corruption of all non-TAP components.

Extrapolated numbers from our implementation of NA show that computationally, it is feasible to implement NA with current hardware. We have strived to make the trade-offs in NA explicit, and we expect the point in design space that NA explores to be both viable and important.

9. **REFERENCES**


