# ACACIA: A Certificate-based Access-Controlled Internet Architecture

Tian Bu Li (Erran) Li Ramachandran Ramjee
Bell Labs, Lucent
tbu,erranlli,ramjee@bell-labs.com

## ABSTRACT

The current Internet architecture supports open connectivity, i.e., any host can send traffic to any other host. This has resulted in a number of security problems such as Distributed Denial-of-Service (DDoS) attacks, worms etc. In this paper, we propose ACACIA—A Certificate-based Access-Controlled Internet Architecture. In ACACIA, a source must first obtain an *access certificate* in order to send packets to a destination. This access request is routed through a separate DDoS-resilient access control infrastructure (ACI). The ACI is based on Distributed Hash Table-based servers that isolates attack requests using a load-adaptive replication strategy. The ACI only forwards legitimate requests to the destination node, which then issues an access certificate to the source. Packets with valid(invalid) access certificates are forwarded(dropped) and legacy packets with no certificates are forwarded in a low priority queue. ACACIA is designed to be mobility friendly with the permission dependent on source and destination identities and not their locations. Using analysis and trace-driven simulation, we show that ACACIA is highly DDoS resilient.

## 1. INTRODUCTION

The current open connectivity nature of the Internet has been plagued by security problems such as host compromise, Distributed Denial-of-Service (DDoS) attacks, worms, etc. In order to tackle these vulnerabilities, there has been many research proposals recently such as [7, 19, 28, 29]. These efforts propose architectural modifications that enable the network to filter and drop un-wanted packets before they reach their destination.

In this paper, we present ACACIA[1], a certificate-based access-controlled internet architecture. In ACACIA, each endpoint specifies an access control list that identifies who can and cannot communicate with it. This access control list is stored in Distributed Hash Table (DHT)-based access control servers. A source that wants to communicate with a destination would first request for "permission to send" from the access control servers. If the destination access control policy allows access from this particular source, the access control servers would contact the destination that would then issue a certificate to the source. The source can now send data packets with this certificate as part of the packet header. Routers or special filtering nodes in the data path in any of the administrative domains between the source and the destination would verify the authenticity of this certificate: packets with valid(invalid) certificates are forwarded(dropped) and legacy packets with no certificates are forwarded in a low priority queue.

One way of looking at ACACIA is that it is the next step in a series of refinements from [1, 18, 26, 28, 29] for a DDoS resilient Internet architecture. Resilience to DDoS attacks must be a first-class service at the IP layer and cannot be performed purely at an overlay layer. This is because, in any pure overlay-based solution (e.g. I3), if an attacker somehow discovers the IP addresses of the destination, the destination nodes are vulnerable to DDoS. Proposals such as [28, 29] build DDoS resiliency into the IP layer. The most recent DDoS resilient proposal called Traffic Validation Architecture (TVA) [29] narrows the attack vulnerability window to a small fraction of the forwarding bandwidth. However, as argued in [3] and as we show in Section 5, even narrowing the attack window to a small fraction of the bandwidth still allows a DDoS attacker to arbitrarily delay access to legitimate sources. In ACACIA, we move the attack vulnerability window to a separate DHT-based access control infrastructure. Given its inherent flat architecture with no single points of failure, DHT is a good match for dealing with DDoS. We design a load-adaptive replication strategy that isolates DDoS close to the source of attack. Furthermore, the pooling of DHT server resources of a third party DHT service provider to fend attacks to many destination sites results in a cost-effective and scalable way to handle DDoS attacks. The need for a shared or aggregated solution for handling denial of service attacks, similar to the Akamai service for handling flash crowds, has been argued by Dave Clark in [12].

Another way of looking at ACACIA is that it is a generalization of the open connectivity semantics of the current Internet. The use of an explicit access control list for each host enables the network to control reachability according to the needs of each host. In this light, ACACIA is similar to the "off-by-default" proposal of [7] where most of the nodes are unreachable unless traffic is initiated by them. While the authors in [7] propose to overload routers and routing tables with access control policies, in ACACIA we separate the access control infrastructure from the routing data plane for scalability since access control policies are inherently non-

[1] Acacia also refers to a family of desert trees whose wood is dense and resilient to penetration by water, insects and other decay agents.

groupable using IP prefixes. ACACIA allows destinations to pro-actively identify legitimate sources via a white-list, thereby ensuring that a valid source can obtain access to the destination with high probability. ACACIA also allows destinations to re-actively pushback attack source identities via a black-list: this, in effect, allows the implementation of push-back based filtering schemes such as [19] without the scalability concerns.

We believe that network access by mobile nodes will dominate the future Internet and thus, mobility friendliness is critical in any security architecture. By friendliness, we mean that the access privilege to a destination should not be invalidated due to mobility. In ACACIA, we adopt the well-known principle of location and identity separation [5, 20]. Thus, each end point or host has both an IP address as well as an opaque end point identifier (EID). When a legitimate source requests permission to send to a destination, the destination issues an *access certificate* that consists of two parts, *consent* and *binding*. The consent allows the source EID to send packets to the destination EID, and the binding binds destination EID to a destination IP address. Thus, the consent or permission to send is independent of the location of the source and destination.

To summarize, we make the following contributions in this paper. We present ACACIA, a network architecture that allows nodes to control their reachability by explicitly specifying their access control policies. By separating access control infrastructure from the routing data plane and using a common DHT-based infrastructure that stores and replicates access control policies based on a load-adaptive algorithm, ACACIA is DDoS resilient in a cost-effective and scalable manner. Furthermore, ACACIA is designed to be mobility friendly due to the dependence of consent to send on source and destination identities and not their locations. Using analysis and trace-driven simulation, we show that ACACIA is highly DDoS resilient. Finally, we hope that this paper reveals important design principles and architecture entities that can help shape the future Internet architecture.

The rest of the paper is organized as follows. We first present the design principles behind ACACIA in Section 2. In Section 3, we present the design of ACACIA and in Section 4, we discuss the protocols and algorithms in ACACIA. We then present the performance evaluation of DDoS resiliency of ACACIA in Section 5. In Section 6, we present a security analysis of ACACIA. We discuss other issues in Section 7 followed by related work in Section 8 and conclusion.

## 2. DESIGN PRINCIPLES

In this section, we describe the design principles behind ACACIA.

**Network-based access control:** *The network must have the ability to store and enforce access control policies that are specified both pro-actively and reactively.* Access control policy can be in the form of white-lists and/or black-lists. A pro-active policy with white-lists allows a set of destinations to communicate freely without being bothered by anybody else. A reactive policy with black-lists can enable the support of functionality of filtering-based such as [19]. The network must be able to enforce access control since performing this function at the destination cannot prevent DDoS attacks, given the link to the destination will become the bottleneck.

**Pooling of resources:** *A shared solution is necessary for cost-effective scaling of defense against DDoS attacks.* Consider a set of destinations that needs to deploy resources to defend against DDoS attacks. If each of those destinations allocate resources to handle worst-case DDoS attacks that could be launched from upwards of 10,000 to even a million nodes, it would be prohibitively expensive and inefficient. A cost-effective solution would be for a third-party provider to allocate enough resources to handle these attacks and offer this as a service to the destination. The need for such a solution based on pooling of resources has been argued by Dave Clark in [12].

**Attack isolation:** *An attack on the resources deployed for DDoS defense must be isolated as close to the source of attack as possible.* Isolating an attack close to the source limits the attacks damage and also helps the pooled resources to be used effectively. For example, the pooled resources can now be scaled to handle the largest attack rather than the sum total of all simultaneous attacks.

**Mobility friendly:** *Nodes should not lose their access privilege to a destination due to mobility.* This principle implies that once a destination has granted access privilege to a source, the mobility of the source or the destination should not result in the need for obtaining a new privilege by the source from the destination (similarly, in the reverse direction for a bi-directional flow). Note that this principle is one choice in the design space of trade-offs between security and mobility. In fact, this principle sacrifices some security for mobility friendliness since access privileges can no longer be tied to a specific source location (replay attacks that launch duplicate packets from many different locations towards the destination are thus possible: see Section 6 for more discussion).

**Scalability:** *The architecture must be scalable to billions of hosts with millions of access control policy specifications.* This principle is self-evident given the size and growth rates of the Internet. Note that this principle argues for separation of access control policy from routing since access control policies are non-groupable by IP addresses due to their very nature while routing tables perforce group and aggregate IP addresses for scalability.

Consider the two well-known types of DoS prevention approaches, filtering and capabilities: a filtering-based approach such as [19] supports network-based access control, pooling of resources and limited attack isolation (pushing filters to source domain has security and policy implications), but is not mobility friendly and has scalability concerns (fil-

ters are not aggregatable like routing tables). A capability-based approach such as TVA [29] provides attack isolation, scalability and limited network-based access control (data packets are protected while request packets are not) but is not mobility friendly and does not support pooling of resources.

## 3. ARCHITECTURE

We first present an overview of ACACIA, followed by the challenges in its design, and their solutions. The details of the algorithms and the protocols used in ACACIA are described in Section 4.

### 3.1 Overview

As mentioned earlier, in our architecture, a host or endpoint is identified by its endpoint identifier (EID) rather than its IP address. An EID is just an opaque bit string. To facilitate human readability, an endpoint also has a human readable identifier (HRI). We use a one-way hash function to map the HRI to EID. For example, the HRI of CNN using the ACACIA service could be www.acacia.cnn.com.

In ACACIA, only packets containing a valid permission, called the *access certificate* or certificate, are forwarded as high priority. Packets containing invalid certificates are dropped while legacy packets with no certificate are forwarded as low priority. Thus, when a source wants to establish a connection with the destination, it needs to obtain an access certificate from the destination.

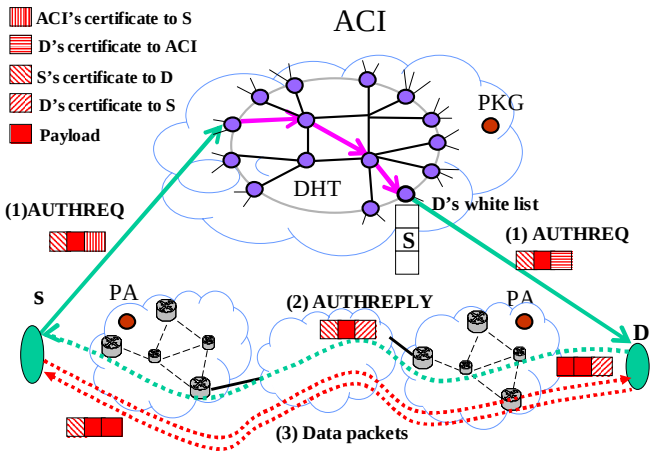


**Figure 1: ACACIA Architecture**

A sample signaling flow for a source and a destination to obtain access certificates to each other for bi-directional communication in ACACIA is shown in Figure 1. A source generates an authorization request (AUTHREQ) message[2] to the ACI. Our design envisions an ACI service that is offered by third parties, e.g. Akamai. The AUTHREQ message includes in its payload, a source access certificate that authorizes the destination to send traffic to the source. The AUTHREQ gets routed to an ACI node that handles the destination's access control policy. If the source is in the black-list(white-list) of the destination, the AUTHREQ is dropped(forwarded as high priority). If the source is in neither the white-list or black-list, the AUTHREQ is forwarded as low priority. Off-by-default is simply implemented by a black-list that includes a regular expression denoting all nodes. In the given example, when the destination receives the AUTHREQ, it constructs a destination access certificate for the source. It will then generate a AUTHREPLY message which includes this certificate. The AUTHREPLY message is sent directly to the source, bypassing the ACI, since the destination already has obtained a valid access certificate to communicate to the source from the AUTHREQ message. At this point, both the source and destination have access certificates for each other and can communicate freely. Routers or special purpose filtering boxes along the path between the source and destination verify the authenticity of the certificate, preventing unauthorized sources from transmitting packets to the hosts.

[2]signaling traffic also needs access certificates; this is discussed later as part of the bootstrapping mechanism.

### 3.2 Challenges

In order to support ACACIA, we need to address the following challenges:

**DDoS resilient ACI:** How do we design the ACI to be DDoS resilient?

**Access certificate:** How do we design the access certificate to be secure while also flexible enough to be mobility friendly?

**Bootstrapping:** How do we provide access certificates to the signaling messages such as AUTHREQ that are needed to establish access certificates?

### 3.3 Entities and Solutions

As shown in Figure 1, ACACIA mainly consists of a control plane called the Access Control Infrastructure (ACI) and a data plane where routers verify certificates. A key challenge in ACACIA is to design a certificate that can be securely authenticated by routers in any administrative domain while also being mobility friendly. Adding a scalability constraint to this results in the need for asymmetric cryptography based on public and private keys.

In ACACIA, we employ an identity-based signature (IBS) scheme since it lowers the cost and complexity of a public key infrastructure significantly (see [4] for background on this topic). In IBS schemes, a node's identity (or its hash) serves as its public key while its private key is obtained from a trusted third party called the Private Key Generator (PKG). A message is signed by an originator using its private key and the message is verified using the public keys of both the originator (i.e. its identity) and the trusted third party. One drawback of IBS-based schemes is that they have high computational overhead for verification - we discuss verification performance in Section 7. If mobility friendliness is not

critical, we can trade-off mobility for better verification performance by adopting a path-centric capability-based ACACIA that uses the ACI for transporting capability requests and replies. Section 7 also describes the design of such a scheme. For now, we present solutions to address each of the challenges described earlier that satisfies all our design principles.

### *3.3.1 Access Control Infrastructure (ACI)*

The Access Control Infrastructure has the following two main functions. First, ACI has to provide each source with a fair chance of ensuring delivery of its AUTHREQ to the destination, even in the presence of large number of attackers. Second, the ACI should control the delivery of the request based on the policy specified by the destination; the destination can specify its access control policy both pro-actively and reactively using white-lists and black-lists.

The ACI maintains a record for each host. The record contains its HRI, EID, and access control policy. ACI must be scalable and DDoS resilient since we have shifted the vulnerability to DDoS attack from end hosts to the ACI. Thus, we use use a distributed hash table (DHT) to implement the ACI. The record is stored in one or more ACI nodes (using multiple hash functions). In order to provide a source with fair access, ACI nodes maintain two queues: a per-source queue for AUTHREQs that originate locally and a per-neighbor queue for AUTHREQs that are forwarded by neighbors of the ACI. Since the number of sources that can directly access this node is limited by configuration[3], implementation of fair-sharing algorithm over these queues is scalable.

Since there is no hierarchical structure imposed, the DHT-based implementation of ACI removes any single point of failure. Attacks on the ACI can be targeted to specific nodes or specific objects. In the case of a node-based attack, we isolate the attack request processing on that node itself, thereby minimizing impact on rest of the ACI. This isolation is also important to protect legitimate sources that use this node since these sources will be queued in a per-source queue and get their fair share; otherwise, these sources have an even lower chance of being served (see Section 5 for analysis of isolation and fair queueing). In the case of an object-based attack from diverse points into the ACI (indistinguishable from flash crowds), we design a replication algorithm that enables the pooling of entire ACI resources to service the request and thus, protects the destination. Thus, the combination of fair queueing, and the adaptive isolation and replication algorithm makes the ACI DDoS resilient. We describe the details of the replication algorithm in Section 4 and evaluate its DDoS resiliency in Section 5.

Note that if the ACI also stores the IP address of the destination EID, the ACI supersedes the functionality of DNS. This has two main advantages: 1) the IP address of the destination is hidden from un-authorized nodes; and 2) vulnerabilities of the DNS to DDoS attacks no longer affect ACACIA. However, for initial deployment, it is also reasonable to keep DNS separate from the ACI. In this case, the ACI uses the DNS to resolve the destination endpoints HRI to an IP address before forwarding the AUTHREQ to the destination. Note that, the reverse possibility, i.e. extending the current DNS system to incorporate the role of ACI, is not attack resilient. This is because the hierarchical nature of the current DNS has many vulnerabilities. For example, any root of a subtree can be a single point of failure for all the descendant servers.

[3]Each source is provisioned with access certificates to a few nodes in the ACI infrastructure - see bootstrapping.

### *3.3.2 Access Certificate*

In IBS, each host requires a PKG to provide it with private keys. In ACACIA, the ACI issues HRI, EID and the corresponding private key for each host. The specific entity within ACI that issues private keys for hosts is the aci-PKG. Note that the aci-PKG only needs to issue keys during host sign-up and when keys need to be periodically changed. Thus, the aci-PKG is mostly off-line.

The access certificate contains two IBSs, a *consent* and a *binding*. The consent verifies that the source EID has permission to send to the destination EID while the binding binds the destination EID to the destination IP address.

The consent is generated by the destination for each AUTHREQ. The destination has a private key issued by the ACI PKG (aci-PKG) while the destination EID is its public key. The consent signature includes the source EID, destination EID, and the public key (index) of the well-known of ACI PKG. The router verifies this signature using the two public keys, the destination EID and the public-key of the aci-PKG.

The destination host by itself cannot generate the binding. Otherwise, any attacker can impersonate the destination by generating a valid consent and a valid binding claiming that they are located at a victim's IP address. Thus, the binding must be signed by a trusted entity. In ACACIA, the binding is generated by the entity that owns the destination address. This entity is referred to as the Prefix Authority (PA). When the PA obtains its prefix address space, it is also issued a private key by the prefix Private Key Generator (pref-PKG) of the root authority of IP addresses, ICANN. The PA's public key is the prefix itself (its identity). This trust relationship is similar to the address PKI proposed in [15, 16]. Thus, the binding signature includes the destination prefix, the public key (index) of the well-known public key of pref-PKG, destination EID and its IP address. The router verifies this signature using the two public keys, the destination prefix and the public key of pref-PKG. Note that the binding does not have to be generated for each AUTHREQ and can be refreshed periodically.

Thus, the routers only need the well-known and trusted public keys of the ACI provider and ICANN in order to verify the authenticity of the access certificate. If verification is successful, the router will put the packet in a per-destination

queue with high priority. If verification fails, the router will drop the packet. The details of the access certificate and router processing are described in Section 4.

**Mobility Friendliness:** Our design is mobility friendly. In a capability-based system [28, 29], the capability is tied to the path. If path changes, the capability is not valid and must be renegotiated. There is no way to pre-negotiate a new capability for an expected path change. In the mean time, the traffic has to be in a low priority queue contending with malicious traffic. Furthermore, consider the case of a public server that serves millions of mobile hosts. The server will be required to issue new capabilities every time one of these hosts move, resulting in an unfair and unpredictable burden on the server.

In ACACIA, the certificate is neither tied to the path nor to the source's IP address. The consent depends only on source/destination identities and thus, remains valid even if source/destination IP addresses change. An endpoint simply obtains a new binding whenever its IP address changes and requests its correspondents to use this new binding for subsequent communication to this IP address. In the case of a server serving millions of mobile hosts, the server only needs to use an updated access certificate (and IP address) provided by the source to deliver packets to the source at the new location. If both nodes move simultaneously, communication is re-established through the ACI. Thus, ACACIA is fully compatible with an end-to-end mobility solution as proposed in [25]. If Mobile IP is used as the mobility protocol, ACACIA can support route optimization without requiring a new consent. Thus, ACACIA is mobility friendly.

#### *3.3.3 Bootstrapping*

In ACACIA, we require packets to have valid access certificates in order to use the high priority channel. Nodes obtain access certificates by sending signaling messages, such as AUTHREQ. The question remains: how do these signaling messages obtain access certificates?

First consider a single-provider ACI service. Source and destination nodes that are customers of this service are authenticated and configured with access certificates to communicate using the high priority with a pre-specified number of nodes in the ACI. Now, we only need to ensure that nodes of the ACI provider can obtain access certificates to each other. However, this must be achieved dynamically since the ACI nodes run a distributed protocol implementing the DHT.

In order to prevent malicious nodes joining the ACI, we require that nodes in the ACI can verify the authenticity of other nodes in ACI. We use an IBS scheme inside the ACI for this purpose. Each ACI has an aci-control-PKG that issues private keys to each of the ACI nodes which enable these nodes to sign messages and be verified by other ACI nodes (this is similar to the certifying authority used in [11] to certify DHT nodes).

Consider the case where the ACI is already operational (nodes have valid certificates to their neighbors) and one of the nodes wants to add a new neighbor. The node simply routes a ACIAUTHREQ message over the already established DHT overlay using the high priority channels. The ACIAUTHREQ includes the nodes identity, a certificate for the new neighbor to access this node and a signature. The destination verifies the signature using the public key of the aci-control-PKG and the public key of the node (i.e., it identity). Once verified, the destination replies with a ACIAUTHREPLY containing a certificate for the source. This is sent directly to the source in a high priority channel using the access certificate in the ACIAUTHREQ. Thus, normal DHT operations can continue using high priority without being subjected to attacks. When a new ACI node is introduced, we can discover neighbors and obtain access certificates using low priority channels as the time to obtain certificates is no longer critical.

In a multi-provider environment, each of the provider ACIs maintain their own key space. They have arrangements with each other to store pointers to objects hosted by the other providers. Certificates for inter-provider communication are pre-configured, similar to how BGP peering is established between ISPs today. We can also adopt the techniques proposed in [6] for supporting multi-provider ACIs.

## 4. ALGORITHM AND PROTOCOLS

In this section, we present a detailed description of the algorithm used for load-adaptive replication in the ACI and the data plane protocol that enforces access control policies.

### 4.1 Algorithm

The ACI infrastructure must (1) pool resources to fend off DDoS attacks and (2) isolate the impact of an attack from influencing other entities, while providing legitimate sources fair share of being serviced by the ACI. Note that, load balancing mechanisms for DHT in the literature [13, 22] support pooling of resources with the objective of serving maximal number of requests. However, these do not have the attack isolation property. The attack isolation property is very important as the ACI provides a service, and normal behaving request patterns should not be severely impacted by attacks. There is a tradeoff between load balancing and attack isolation and we use attack isolation to protect legitimate sources and load balancing to protect destinations by pooling the ACIs resources.

Requests to an ACI node from locally associated endpoints are referred to as local requests while other requests routed to this ACI node are referred to as remote requests. For scalability reasons, we perform per-source queuing for local requests while we only perform per-neighbor queuing for remote requests. In addition, an ACI node gives local requests higher priority so that local requests can be served even during excessive remote requests.

We now describe our replication algorithm. We assume DHT routing (computing a hash function and forwarding) involves minimal overhead and thus, define load in terms of

the number of AUTHREQs that needs to be processed. For ease of description, let all objects have identical load and all nodes have identical capacity. At each ACI node, let us define node and object thresholds, $h_a$ and $h_O$, in terms of number of requests to that node for all objects and a given object, respectively.

The replication algorithm is triggered if the locally originated requests exceeds $h_O$ for any object $O$ or if the total requests of a node exceeds its node threshold. In the former case, the attack is isolated and in the latter case, load balancing is performed. Attack isolation is achieved as follows. When the object threshold is exceeded (i.e. potential local attack), then object $O$ is replicated to this ACI node. Further, the node does not forward any of its local requests for this object (it continues to forward remote requests). Since a legitimate local source will be fair-queued with the attack traffic, it gets its fair share of being served (if local requests are forwarded, then the legitimate source gets an even lower chance of being served - see the queueing analysis in the next section).

When the node threshold is exceeded, the node starts with the object with the maximal request rate to replicate. *Our replication algorithm minimizes the number of replicas needed while guaranteeing no node thresholds are exceeded (if it is possible to do so).* Unlike [22], we do not replicate to nodes which do not have any object requests passing through it. This is important for attack isolation.

Our algorithm is a dynamic programming based recursive algorithm. Note that, the request path for an object forms a tree rooted at the original node with the object. Lets define $A(T_i, R_i)$ to be the minimal number of replicas determined by the optimal algorithm in subtree $T_i$ given that the subtree needs to process $R_i$ requests. Denote $n_i$ as the request rate for object $O$ originated from node $i$, $N_i$ the total request rates under the subtree $T_i$ (including $n_i$), and $L_i$ the load in terms of request rate for all objects processed before replication starts. Let $d_i$ be the depth of subtree $T_i$. A leaf has a depth of zero. For ease of description, we describe the algorithm assuming the tree is a binary tree. Our algorithms work for general trees. Lets denote $T_i^l$ the left subtree of $T_i$, and $T_i^r$ the right subtree of $T_i$. We have the following recursive relationship $A(T_i, R_i) = \min(A_1(T_i, R_i), A_2(T_i, R_i))$ where $A_1(T_i, R_i)$ denotes the optimal when the root $i$ does not need a replica and $A_2(T_i, R_i)$ denotes the optimal when the root $i$ needs a replica.

$$A(T_i, R_i) = \begin{cases} 1, & d_i = 0 \,\&\, n_i > h_O; \\ 0, & d_i = 0 \,\&\, n_i \le h_O \,\&\, R_i = 0; \\ 1, & d_i = 0 \,\&\, n_i \le h_O \,\&\, R_i + L_i \le h_a \,\&\, \\ & \quad R_i > 0; \\ \infty, & d_i = 0 \,\&\, n_i \le h_O \,\&\, R_i + L_i > h_a; \\ \min( & \\ \quad A_1(T_i, R_i), & \\ \quad A_2(T_i, R_i)) & \text{Otherwise.} \end{cases} \tag{1}$$

Lets look at the base case of the recursion (Eqn. 1). For a leaf node, if object threshold is exceeded, an object will be replicated (first line); if no request needs to be handled (all done by upstream nodes towards the root), then no replica is needed (second line); if the node needs to handle requests and no thresholds are exceeded, then a replica is needed (third line); if handling the request exceeds node threshold and the algorithm insists the node to handle it, then the algorithm returns infinity which means this recursion is infeasible.

The relation when root $i$ does not need a replica is

$$A_1(T_i, R_i) = \begin{cases} \infty, & R_i > N_i - n_i \| n_i > h_O; \\ \min_{g=\max(0, R_i - N_i^r)}^{\min(N_i^l, R_i)} (A(T_i^l, g) + & \\ \quad A(T_i^r, R_i - g)) & \text{Otherwise.} \end{cases} \tag{2}$$

Node $i$'s subtree has a total request rate of $N_i$, of which $N_i - n_i$ are from its children ($N_i^r$ from right, $N_i^l$ from left). It is infeasible to ask its children to process $R_i > N_i - n_i$ as they can not process request not going through them. Node $i$ should request a copy if $n_i > h_O$. This is the base case (line 1 of Eqn. 2). The recursion finds the best split of load among its children. To speed up, one can define a minimal split threshold.

Let $q = n_i$ if $n_i > h_O$, and $q = \min(h_a - L_i, h_O, R_i)$ otherwise; that is, if root $i$ of subtree $T_i$ is placed a replica, $q$ is the maximal request rate it can process. Note that, if $n_i$ is greater than the object threshold, the local request will be handled locally; that is, no local requests will be forwarded, and remote requests for object $O$ will be forwarded. The recursive relation when root $i$ needs a replica is given in Eqn. 3.

$$A_2(T_i, R_i) = \begin{cases} \infty, & R_i > N_i; \\ \min_{g=\max(0, R_i - q - N_i^r)}^{\min(N_i^l, R_i - q)} (A(T_i^l, g) + & \\ \quad A(T_i^r, R_i - g) + 1) & \text{Otherwise.} \end{cases} \tag{3}$$

If the algorithm returns infinity, then there is no feasible solution; this means that, even if every node receives a replica, the request for object $O$ still can not be processed without overloading all nodes.

Note that, since each triggered replication does not increase the maximal load among nodes in ACI, if request rates do not change and replication is done one at a time, eventually our algorithm will be quiescent, i.e. no more replication will be triggered. If multiple replications are conducted at the same time, then we could use atomic commits to avoid unnecessary replication. As long as a node continues to serve a threshold number of requests $h'_O < h_O$ for the object, the time-to-live field of the object is refreshed.

We now discuss the state that needs to be maintained for implementing our algorithm. Our algorithm can be implemented in a distributed or centralized manner. In the former case, the only state needed are the number of local requests and total requests per object. Communication only occurs between neighboring nodes. To speed up the process, the root can limit the depth of the tree that participates. If all nodes' load are balanced, then a node keeps $O(Mlog(n)/n)$ state where $n$ is the number of ACI nodes and $M$ is the total

number of objects. In the centralized implementation, replication decision can be computed quickly at the root node of each object. However, this requires each ACI node periodically piggyback its load information (local and forwarded load per object) when forwarding AUTHREQs. This information is propagated to within a few hops of each node to minimize monitoring and state overhead. In our simulation, we implement the centralized version.

## 4.2 Protocols

ACACIA consists of two protocols, a control protocol described in the previous section that uses the AUTHREQ and AUTHREPLY messages to obtain access certificates and a data plane protocol that is represented by the access certificate carried in IP packets. The control protocol is straightforward and can be implemented at the application layer using UDP just like the DNS system. In this section, we first describe the data plane protocol in detail. We then discuss router functions for processing the data packets.

### *4.2.1 IP Packets with access certificate*

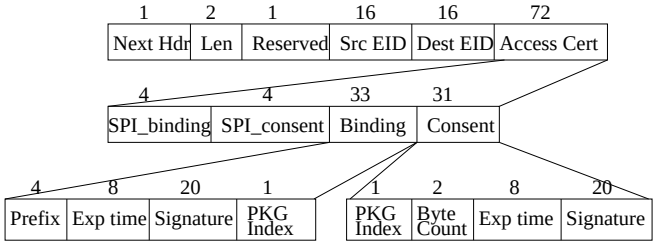


**Figure 2: ACACIA shim header**

We denote the packet sender as $u$, destination as $v$, destination ACI provider as $i$, prefix owner as $a$. They each has a public/private key pair $(PK, SK)$. The $PK$ is the same as the EID. Other than legacy traffic, all packets carry an access control header that extends the behavior of IP. The protocol id field in IP is set to ACACIA. A router detecting an ACACIA protocol type can begin to process the ACACIA header shown in Figure 2. The beginning of the header identifies the next header field (corresponding to the transport protocol such as TCP), and a length field. This is then followed by thee EID of the source and destination and the access certificate. The certificate includes two security parameter index (SPI) values denoting the crypto algorithm and parameters used in the binding and the consent portions respectively. Binding $B_v$ is made up of destination prefix, destination prefix-PKG's index, expiration time (the time that the binding stops being valid) and a signature $S_a$. The signature certifies the destination EID: $EID_v$, destination IP address: $IP_v$, and expiration time: $t_a$, using the prefix private key $SK_a$.

$$S_a = Sign_{SK_a}(EID_v, IP_v, t_a)$$

Note that, the IP address is not duplicated in the binding or the shim header. The consent $C_v$ consists of the aci-PKG index, the expiration time : $t_v$, bytes allocated to sender: $N$, and a signature $S_v$ that certifies source EID: $EID_u$ and destination EID: $EID_u$, expiration time (the time that the consent stops being valid): $t_v$ using the destination's private key $SK_v$.

$$S_v = Sign_{SK_v}(EID_u, EID_v, t_v, N)$$

Borrowing the host identifier tag from the Host Identity Protocol, EIDs are 16 bytes long. The expiration time is encoded using the Network Time Protocol (NTP)'s timestamp format which is 8 bytes long. The signatures are 20 bytes long. Thus the binding and consent together takes 68 bytes and the full ACACIA header is 104 bytes. This compares reasonably with a 8-byte capability per router-hop used in TVA [29]. Also, we can adopt the header overhead reduction techniques described in [29] for long flows. If ACACIA did not include the identity and address separation (i.e., we use IP addresses also as identities), the header size can be reduced to 40 bytes as the binding and EIDs are no longer necessary. This overhead is comparable to IPSEC Authentication Header size of 24 bytes that includes a 96-bit message authenticate code.

### *4.2.2 Router processing*

Each router maintains two queues: a high priority queue and low priority queue. Routers at the edge maintain a per-destination high priority queue and a flow state cache as well; this is needed for fine-grained fair queuing which prevents a source with an access certificate from overwhelming a destination in a short period of time. The flow state cache is also used to cache verification results. The flow state cache design is similar to the one in TVA (see [29] for details on a scalable way to implement this).

When receiving a packet, if the packets protocol id is not ACACIA, the packet is treated as a legacy packet, and the packet is processed in the low priority queue. Otherwise, the router first looks up its flow state cache. If an entry exists, the router then checks whether the byte count has not exceeded and the certificate time has not expired. If the checks pass, it then checks whether the access certificate is the same as the cached one. If yes and the access certificate was verified to be correct before, then the router will per-destination fair queue the packet in the high priority queue. If the flow state cache does not have an entry, then it will create an entry and initialize the byte count allowed and expiration time field. The router then finds the public key $PK_{prg}$ of the pref-PKG using the PKG index in the binding. It then uses $PK_{prg}$ and the prefix to verify the binding signature. The router then uses the aci-PKG of the destination (found using the PKG index) and the destination EID to verify the consent signature. If the binding and consent are both valid, the packet will be processed in a per-destination high priority queue. If any check fails, then the packet will be dropped.

Note that we assume that the router verifying the access certificate has global weak clock synchronization on the or-

der of few seconds. This is easily achievable if the routers run a protocol such as network time protocol (NTP).

## 5. EVALUATION

In this section, we evaluate the DDoS resiliency of the ACI in ACACIA. We start with an analytical comparison between the ACACIA and the TVA [29] where we demonstrate that ACACIA can survive significantly more intense attacks. Using simulations driven by the traces collected from the Internet, we then examine the DDoS resiliency of ACACIA as a number of factors such as utilization, the origin of attacks, the distribution of victim, and the intensity of attack vary. Lastly we investigate the overhead due to replication and policy updates.

### 5.1 Comparison with TVA

In both TVA and ACACIA, a source has to obtain an authorization in order to send data packets to a destination. The DDoS resiliency thus is determined by the available bandwidth for a legitimate source sending authorization requests[4]. We now compute the available request bandwidth for both schemes.

Each router in TVA reserves a fixed fraction of capacity, e.g., $5\%$ for request traffic that is shared among all capability requests through fair queuing. The packets of the same path identifiers are placed in the same queue where a path identifier is a 16-bit tag that is derived from the incoming interfaces of the current domain using a hash function. Note that the number of path identifiers may be greater than the total number of interfaces in the current domain since ingress routers only re-tag the incoming packets when the upstream domains are trusted; otherwise the old tag remains.

Consider Figure 3(a) where there are $k$ domains along the path between a source and a destination. The number of queues and capacity of outgoing link in egress router of Domain $i$ are $Q_i$ and $C_i$ respectively. The $n$ attackers are assumed to be uniformly distributed in the network such that there are $n/k$ attackers joining the legitimate source at each domain along the path towards the destination. The attackers are evenly distributed in the other $Q_i - 1$ queues with at least one attacker per queue if $n/k >> Q_i, \forall i$. We further assume that attackers always generate enough requests to keep their queues from being idle.

Let $B_i$ denote the available bandwidth for the source after traversing through domains 1 to $i$. We assume that the legitimate source is fairly queued with other $n/k$ attackers in the originating domain. Therefore we have $B_1 = C_1 1/(1 + n/k) = C_1 k/(n + k)$. From here, $B_i/C_i$ is the fraction of source traffic among all traffic domain $i$ forwards to domain $i + 1$. At most $C_{i+1}/Q_{i+1}$ of this can be forwarded to the next domain since the egress link bandwidth is equally shared among all the $Q_{i+1}$ queues. If $C_{i+1}/Q_{i+1} > C_i$, then $B_{i+1} = B_i$ since domain $i$ is the bottleneck and can not even output enough traffic to obtain its fair share of bandwidth in domain $i + 1$. Otherwise, we have $B_{i+1} = (B_i/C_i) \times (C_{i+1}/Q_{i+1})$. In the best case where $\forall i$, $C_i > C_{i+1}/Q_{i+1}$, we have the available bandwidth along the entire path to be

$$B_k = \frac{k}{n+k} \times \frac{C_k}{\prod_{i=2}^{k} Q_i} \tag{4}$$

where $C_k$ is also the server capacity for processing capability requests.

In the case of ACACIA, an attacker can choose to attack an ACI node (referred to as local attack) or an ACI object (referred to as infrastructure attack). Let $L$ be the maximum number of sources that are provisioned with access certificate to reach any ACI node. The attacker must first compromise $n < L$ of these sources to launch a node attack. Since our replication algorithm isolates the attack source, the $n$ attackers will simply share the capacity of the ACI node, $C_i$. Thus, available bandwidth is simply $C_i/(n + 1)$ for local attacks.

In the case of infrastructure attacks, the available bandwidth for ACACIA is also straightforward since a policy object will eventually be replicated to all ACI nodes by the replication algorithm as $n$ becomes large. If we assume all $n$ attackers are uniformly distributed and each originating ACI node performs per-source fair queuing, the available bandwidth for a legitimate source sending its request to ACI node $i$ is

$$\frac{D}{n+1} C_i \tag{5}$$

where $D$ is the total number of ACI nodes and $C_i$ is the capacity of ACI node $i$. We thus avoid the product term in the denominator of Equation 4 in ACACIA by actively replicating the access control policy at the originating ACI node and isolating the attack right there.

Figure 4 plots the fraction of successful requests versus the number of attackers in a log-log scale when the legitimate requests are sent once every second. We assume a TVA server and one ACI node have the same capacity of $1,000$ requests/sec. In addition, we assume $k = 5$, $Q_i = 25, \forall i$, and $D = 1024$. We observe that the ACACIA is significantly more resilient to DDoS attacks. The success rate for TVA request starts to drop steeply before the number of attackers reach 100 (note that we assume attackers are uniformly distributed with respect to path identifiers which is the worst case for TVA). In the case of ACACIA, the success rate is maintained at $100\%$ until the number of attackers reach a thousand (million) for local (infrastructure) attacks and then decrease linearly as more attackers come in.

Figure 5 shows that the success rate drops sharply as $k$, the number of domains between source and destination, and $Q$, the number of unique interfaces, increases This can be explained by Equation 4 where the success rate is inversely proportional the product of $k$ $Q$ values.

[4]Unlike ACACIA, in TVA, the destination also has to send capability response over a low priority channel but we ignore this for now.

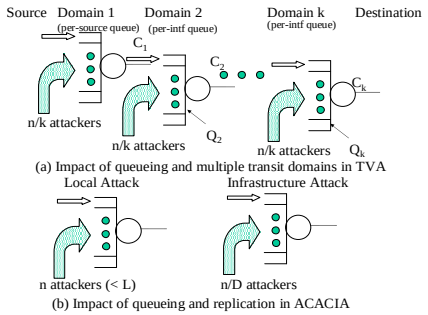


**Figure 3: Impact of queueing**

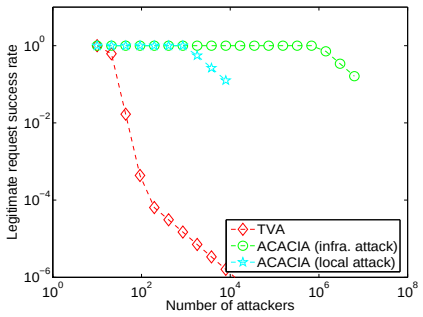


**Figure 4: Request success rate**

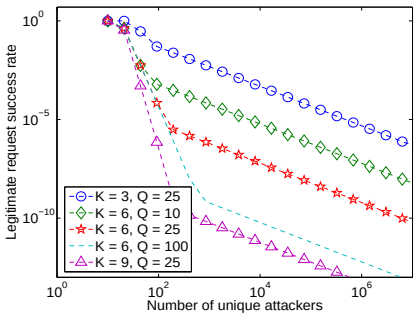


**Figure 5: Impact of Q,K on TVA**

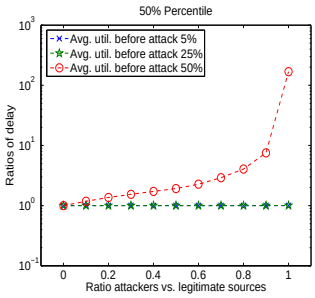


**Figure 6: 50% delay ratio**

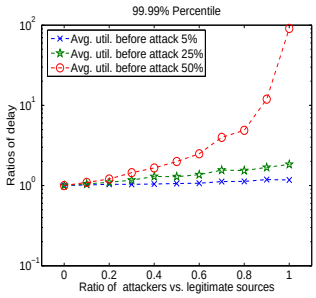


**Figure 7: 99.99% delay ratio**

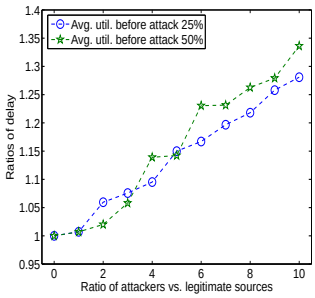


**Figure 8: Object attack**

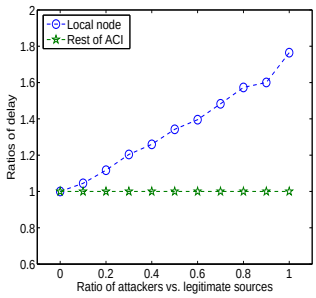


**Figure 9: Node attack**

## 5.2 Simulation evaluation

We simulate a Pastry network of 1024 with our replication algorithm as described in Section 4. We set the base to be 16 and the length of identifiers to be 128 as recommended in [23].

In today's Internet, before a source initiates a connection to a destination, it needs to perform a DNS lookup for the destination IP address. Therefore, the arrival pattern of DNS domain name lookups should be very close to that of the AUTHREQ in ACACIA. We use the DNS traces collected from a campus network between April 26 and May 9, 2004 for a total of 291 hours. The trace includes 3,693,728 domain name lookups with a total of 283,474 unique names. The lookup frequency of the trace follows an approximate Zipf-like distribution with gradient of $-1.078$ on a loglog plot.

**Metric:** When an attack occurs, it would cause delay in the processing of legitimate AUTHREQs. We focus on the service delay (including queueing) of the AUTHREQ at the destination ACI node, since forwarding and propagation delays are small and relatively insignificant. We can measure the resiliency of an ACI configuration by evaluating how much extra delay is incurred due to an attack. The smaller the extra delay, the more resilient a configuration. We define the *ratio between the delays with and without attack* as our primary metric of interest. A lower (higher) delay ratio denotes a more(less) attack resilient configuration. Unless otherwise specified, we will use a 99.99% quantile of delay values for comparison as it is very sensitive to higher load.

### 5.2.1 *Infrastructure, Object and Node attacks*

Let us first examine how our replication scheme reacts to an infrastructure attack where attackers are uniformly distributed in the network and the targeted objects are also randomly picked. We simulate the attacks for three network configuration with the average utilization before attack at $5\%$, $25\%$, and $50\%$ respectively. The average utilization is defined as the ratio between the total request arrival rate during normal operation and the total capacity of all nodes; thus, this utilization represents the average load at each of the ACI nodes during normal operation[5].

Figures 6 and 7 plot the delay ratios using delays at $50\%$ and $99.99\%$ quantiles respectively. Let us focus on the $99.99\%$ quantile figure since at $50\%$ quantile, the nature of the curve remains the same but with decreased steepness. When the attacking requests increase from zero to one times legitimate traffic load, the delay ratios increase for all three curves. However, the rates of increase are different; the higher the normal utilization level, the faster the increase as the number of attackers increase. Note that these three curves demonstrate that ACACIA can support multiple ACI providers that provide different degrees of attack resilience - a government or enterprise customer may prefer an ACI that provides high degree of resilience while a consumer may be fine with an ACI that is provisioned at 50% utilization.

Infrastructure attacks may also target a particular object (e.g. webserver). The worst case is when the attack is targeted to the most popular object in the ACI. We simulate this by increasing the number of requests from attackers for

[5]Since the trace size is fixed, we change server capacity to vary utilization levels.

the most popular object from zero to ten times the normal amount. We simulate the attacks for average utilization of $25\%$ and $50\%$ and plot the delay ratios in Figure 8. In both cases, the increase in the delay ratio is not at all significant as the number of attackers grow. This is because the victim object is replicated to more and more nodes as its popularity (attack intensity) grows and thus, the *resources of the entire ACI can be pooled to handle large attacks on a single website*.

Attacks may also break out from a local area where all attacking requests are initiated to the same ACI node. The worst case for a legitimate local source is when the attack requests target the same object (otherwise, DHT routing would distribute most of the attack traffic along different paths away from the source after the original node). We simulate this local attack and plot its impact on both the local node and the rest of the network. Figure 9 shows the delay ratios. Thanks to the *replication scheme that replicates the object to this node and then isolates the processing here, legitimate sources still have reasonable chance of getting served.* Further, the attack isolation results in no impact to the rest of the ACI.

### *5.2.2 Impact of Replication*

In this section, we evaluate the replication algorithm, its benefits and overheads. We also evaluate the impact of updates of policy objects on the replication algorithm. From now on, we assume average utilization of 25% and delay values are 99.99% quantiles. Consider a generic infrastructure attack that issues AUTHREQ to random objects stored in the ACI. As the number of attackers increase, more replications are done to compensate the load increase due to the attacks. Assuming a utilization of $25\%$, we plot the replication ratios versus attack intensity in Figure 10. The number of replicas per object increase from $1.05$ to $1.5$ as the attack load grow from zero to one times of legitimate load, illustrating *that not much replication is needed in the case of a generic infrastructure attack.*

Our replication scheme not only mitigates the impact of DDoS attack by distributing load more evenly among all ACI nodes, it also helps to load balance the request even when no attack exists. Due to the heavy-tailed nature of the request distribution over objects, the load of nodes in the ACI are not uniform. Our scheme replicates an object when its request rate exceeds a threshold, thus offloading the hot-spots. Figure 11 demonstrate the benefit of our replication scheme. We plot the ratio of the delay without and with replication at different loads. At higher loads, the benefit of replication increases. Note that while a solution such as Beehive [22] can address this aspect, our replication algorithm handles both popular objects/flash crowds and DDoS attacks in a unified manner.

Replications put load on the ACI and thus impact the serving delays. We evaluate the slow down of service due to replication in Figure 12. We plot the delays with different replication overheads (relative to the cost to serve a request), normalized to the delays assuming that replication overhead is zero. When the overhead is 2 and 4 times the service cost, the increased service delay due to replication is negligible. On the other hand, service delay almost doubles for the overhead value of 8, during high intensity attacks.

We finally evaluate the impact of updates to the access control policy. For example, destinations can dynamically update their access control policy to block requests from malicious sources. The ACI may update it either in real time or in batches. Real time update minimizes the impact of malicious sources whereas the batched approach reduces the update overhead in the ACI. The larger the interval between updates, the lower the update overhead. The hosted ACI node first updates its local policy database and then pushes the updates to the replicas, if any. Since popular objects are likely to have more updates than others, we simulate update object distribution by assuming updates are proportional to object requests in the ratio of 1:1000. Note that, due to replication, the update node distribution is very unbalanced since updates to all nodes may be needed for popular objects whereas one node update may be sufficient for unpopular objects. Figure13 plots the service delay versus the batch interval. The delays are all relative to the one assuming zero-overhead updates. As the batching interval approaches 7000 units of service time, the update overhead approaches zero (ratio close to one). This implies that *batching is effective in controlling update overhead*. For an ACI node that can process 1000 requests per second, an update delay of 7 seconds has very little impact on the service delay and after this delay, AUTHREQs from those malicious nodes no longer reach the destination.

## 6. SECURITY ANALYSIS

**Trust model:** In ACACIA, each endpoint can authenticate to the prefix authority, and trust the PA to issue correct bindings for it. The endpoint also trusts the ACI to faithfully execute its access control policy. The endpoint trusts its aci-PKG will not reveal its private key. Routers trust the public key of aci-PKGs and the pref-PKG. These public keys are distributed to routers by some network management entity.

The security of ACACIA is based on the inability of an attacker to get access certificate from a destination that does not want traffic from the attacker. Here we briefly analyze how ACACIA counters various threats.

**Forging or stealing certificate attack:** An attacker may know the EID and IP address of the destination. However, an attacker can not forge the access certificate because an attacker has neither the prefix private key nor the private key of the destination. An attacker may steal a legitimate access certificate belonging to a sender. If the attacker can spoof the sender EID, then the attacker can use the certificate to send traffic to the destination (changing the sender EID will cause the consent verification to fail). All packets with the same certificate will be queued in the same destination queue

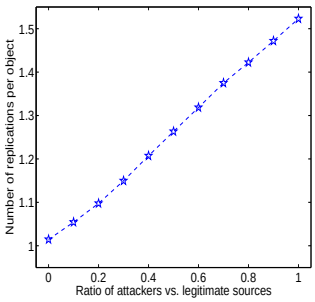


**Figure 10: Replication ratio**

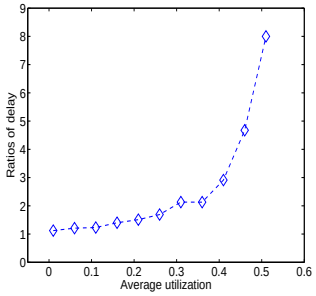


**Figure 11: Benefit**

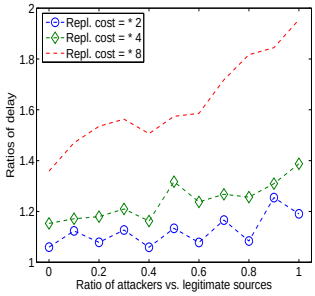


**Figure 12: Overhead**

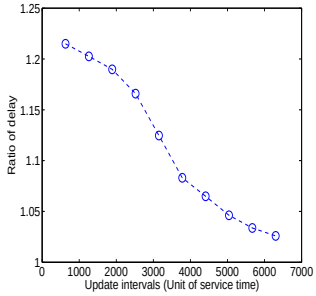


**Figure 13: Update overhead**

with byte count limits enforced for the given source EID. Thus, stealing certificate attacks only impact the legitimate source, not the destination. If the attacker domain implements source EID spoofing prevention mechanism, then the attack is limited to the same domain. In addition, certificate replay is ineffective if the certificate expires. To prevent certificate replay attacks entirely, each ACACIA packet can be optionally signed by the sender, and verified by routers. However, this crypto operation can be very expensive for endpoints if done on a per-packet basis.

**Stale information attacks:** An attacker may gain old private keys of a legitimate endpoint. So it can forge the consent. However, since it does not have the corresponding binding, it can not launch attacks to that endpoint or any other endpoint since the binding and consent will not be consistent. If an endpoint moves, its old binding to the old IP address may not have expired. So it can issue a valid consent. This will direct any sender's traffic to the old IP address. However, we require the prefix authority to refrain from reallocating old IP address if the old binding has not expired. In this way, such an attack can be thwarted.

**Various attacks from compromising entities in ACACIA:** If a legitimate host gets compromised, the host can only send as much traffic as it has been authorized to send but it can launch attacks on the ACI. A compromised router can mount replay attacks or drop legitimate packets. Because routers do not have the private key of the sender, receiver, or prefix authority, attacks from compromised routers are limited.

If an ACI node gets compromised, a number of attacks are possible. Note that ACI node compromise should be extremely rare since these are provider-hosted nodes and not generic hosts. First, the ACI node can attack the underlying DHT routing protocol. Security mechanisms to handle misbehaving DHT nodes are discussed in [11]. The ACI node can obviously impact the hosts whose records are stored there by not sending legitimate AUTHREQ to its endpoints. However, since the record is maintained in multiple ACI nodes and system randomly chooses on of those, its impact on endpoint will be limited. Sending unwanted AUTHREQ or abusing its access certificate privilege by directly attacking destination will still not result in DDoS since the ACI node has consent to send only at a limited rate but this will result in the detection of the compromised node. If the malicious ACI node generates unwanted AUTHREQ to DDoS a victim ACI node, we can isolate this attack if we use a per-originating ACI node queue. The node which first receives an AUTHREQ from a host should append its signature before forwarding and the final ACI node verifies this signature and places this request in the queue for the original ACI node, thereby isolating the impact of a compromised node attack. If the ACI node generates AUTHREQ with itself as the source to DDoS a destination host, it does not have any more power than a compromised regular host.

If a prefix authority gets compromised, the attacker can issue valid bindings to access hosts whose IP address share the prefix. This will draw traffic to the domain of the prefix authority. However, PA compromise should be very rare and its impact is localized to its own domain.

# 7. DISCUSSION

In this section, we discuss verification performance, alternative designs and other relevant issues.

## 7.1 Verification Performance

Ideally each router should be able to verify the authenticity of the access certificate at line speed. This requires fast signature verification algorithms. IBE-based cryptographic algorithms has been a very active area of research in the past few years. Consequently the performance of IBE-based crypto has been improving quite dramatically. For example, signature verification time in software has been reduced from seconds [10] to milliseconds [9] in the last few years. Signing requires point multiplication in elliptic field, while verification also requires a pairing operation that is a few times more expensive. Pairing is very amenable to parallelization in hardware. Recent hardware implementations [17] can perform pairing operation in 0.7 milliseconds at a clock speed of 20MHZ. With an ASIC operating at 1GHz processor speed, verification using the algorithm in [9] can be done in about 30 microseconds [8]. With a minimal ACACIA packet size of 168 bytes, this will give us a verification speed of about 40Mbps. While 40Mbps verification per chip is reasonable for initial deployment, it is definitely far short of the needs of the future Internet. On the other hand, the rate of progress in IBE-based research in recent years holds

promise. In order to achieve more speedup today, the verification has to be done in parallel, albeit at increased cost (e.g. if there are 10 security processors, then verification can be 10 times faster as packets can be queued independently). Further, we can use pipelining by letting the routers cooperate in the verification where each router stamps intermediate results and the full verification is done at the egress of the domain. All these techniques can drive per-signature verification into sub micro seconds, driving verification speed up to 10Gbps per line-card.

## 7.2 ACACIA with Capabilities

While we described ACACIA with an access certificate-based solution in this paper, we can also design a capability-based system that uses only the ACI of ACACIA. Such as system will have the superior line-rate verification performance of a capability-based system and inherit the following features of the ACI: 1) network-based access control 2) pooling of resources and 3) higher DDoS resiliency on the request channel. However, this system will no longer be mobility friendly as the capability is tied to the path.

The design of such a system can be achieved based on the proposal in [2]. The ACI provider now also deploys verification points (VPs) in the domains[6] of its customers. A source issues a capability request through the ACI. The ACI filters the request based on the destinations access control list. When/if the destination receives the request, the capability reply is sent back to the source through the ACI. The reply contains a token that is the last hash value of a one-way hash chain and a random sequence number. This token is installed into the source and destination domain VPs by the ACI. Thus, when the source includes this token in its packets, the VPs can verify them. The one-way hash chain enables the destination to renew the capabilities and the VPs to automatically verify the new capabilities without any signaling.

## 7.3 Incremental deployment and incentives

In order to support ACACIA, we need to deploy an ACI and upgrade hosts and routers. While this may appear daunting, fortunately, even a limited deployment of ACACIA provides some benefits and this can help bootstrap a full migration to an access-controlled Internet. Initially, an ISP may be willing to deploy an ACI with upgraded routers in order to offer a premium access-controlled network service (similar to the VPN service today) to some of its customers. For example, a customer (e.g. enterprise, bank) may provide an access control white-list (e.g. EID of employees, account holders). As long as the destination ISP is well-connected to the Internet, this would allow these customers to communicate freely without receiving unwanted traffic or being subjected to DDoS. Once this service gets deployed, other customers such as e-commerce sites or public servers with registered users may be interested since they can provide a white-list of prior customers, ensuring priority access for them. An ISP that does not want to offer the premium service may still upgrade in order to drop packets early and cut costs since ISP billing is often traffic volume or percentile-based. Finally, hosts have an incentive to upgrade as otherwise, they are delegated to the use of low priority channels.

[6]This has policy implications; strictly speaking, signaling to VPs deployed in these domains is sufficient

## 7.4 Key management

**Key escrow:** For identity-based crypto, the PKG knows the private key issued to each entity. However, this is not an issue in our architecture. In the case of hosts, since the provider also issues the EID for the subscriber, and the private key of an EID is only used to attest the EID, whether the provider has the private key or not is not an issue. For pref-PKG, the PKG is ICANN, the root authority that allocates prefixes. It is trusted by all PA. The prefix private key of each PA is only used to attest that the PA owns the prefix and not for any other purpose.

**Key revocation:** Since the binding expires periodically, if an endpoints key (or EID) expires, the endpoint will notify the PA and the PA will stop issuing bindings for the old EID. Revoking the EID is not a problem since EID is a hash of its HRI. The endpoint can just update its record in the ACI with a new EID. The default EID mapped from HRI using a well-known hash function can point to the updated EID. By using hashes of HRI rather than HRI itself, we avoid the problem of revoking the identity in IBE based crypto.

# 8. RELATED WORK

We have earlier discussed the most closely related work [7, 19, 28, 29]. Here we briefly describe other related work. The related work is loosely divided into two categories; one is on DDoS prevention techniques and the other is on architectural design.

Source address filtering [14] deployed at network ingress can prevent attackers from putting arbitrary source address in the header. It has been generalized to filtering any packet that can not have legitimately arrived even at the middle of the network [21]. However, this does not prevent spoofing within the same prefix of the network. Because it is easy to gain access to a large number of compromised hosts (Zombies), source address filtering does not work for attacks from legitimate sources.

To defend against DDoS attacks, one can trace the attack sources and punish the perpetrators [24]. Due to the large number of Zombies used in the attack, finding the attack origin can be very difficult. In addition, it is too late to prevent the attack.

SOS [18] and Mayday [1] have focused on designing overlay network architectures to protect destination from unwanted traffic. Incoming packets must be authenticated through overlay nodes. Then a secret is added to each packet header before forwarding to the destination. This secret is shared among all traffic through the overlay to the same destination.

Downstream routers are configured to discard all packets do not contain the secret. However, this approach is vulnerable to an attack discovering the shared secret.

The separation of endpoint identifier from address is borrowed from [5, 20]. This separation seamlessly accommodate mobility and multihoming. This separation naturally results in flat names for endpoint identifiers. As the current DNS relies on the use of hierarchical names, an infrastructure unlike the current DNS is needed. The use of DHT to replace the current DNS has been proposed in [5, 27]. Our use of DHT to provide DDoS resilient signaling builds on them.

## 9. CONCLUSION

The open nature of the current Internet architecture has given rise to many security problems and patched solutions such as firewalls and middleboxes that are alien to the original architecture. In this paper, we explore an alternative design of the Internet architecture that is based on a fundamentally different philosophy: hosts should be able to specify their access control policy to the network, and the network should enforce it. Entities in the architecture should have incentives to participate, and the impact of any compromise of any entity should be limited and isolated. The architecture should also be friendly to the ever increasing number of mobile hosts. A scalable design satisfying all these properties requires the use of asymmetric keys because shared keys are either not scalable in terms of router state or suffers from the peril of transitive trust. Although signature verification algorithms based on elliptic curve cryptography are improving rapidly, we recognize verification can be a potential performance issue. By sacrificing some desired architectural property such as mobility friendliness and the ability of every router enforcing the policy, we point out an alternative design without using asymmetric keys. We reckon that, by no means this paper represents a complete and comprehensive design in the space of new Internet architecture. It does reveal important design principles, architectural entities needed, and potential mechanisms required. These findings hopefully can help shape the design of the next generation Internet architecture.

## 10. REFERENCES


[1] D. G. Andersen. Mayday: Distributed filtering for Internet services. In *Proc. of USITS*, 2003.

[2] T. Anderson, T. Roscoe, and D. Wetherall. Preventing internet denial-of-service with capabilities. In *Proc. of HotNets*, November 2003.

[3] K. Argyraki and D. R. Cheriton. Network capabilities: The good, the bad and the ugly. In *Proc. of HotNets*, November 2005.

[4] J. Baek, J. Newmarch, R. S-Naini, and W. Susilo. A survey of identity-based cryptography. In *AUUG*, 2004.

[5] H. Balakrishnan, K. Lakshminarayanan, S. Ratnasamy, S. Shenker, I. Stoica, and M. Walfish. A layered naming architecture for the internet. In *Proc. of ACM SIGCOMM*, pages 343–352, 2004.

[6] H. Balakrishnan, S. Shenker, and W. Walfish. Peering peer-to-peer providers. In *Proc. of IPTPS*, 2005.

[7] H. Ballani, Y. Chawathe, S. Ratnasamy, T. Roscoe, and S. Shenker. Off by default! In *Proc. of HotNets*, College Park, MD, November 2005.

[8] P. S. L. M. Barreto and T. Kerins. Private communication. 2005.

[9] P. S. L. M. Barreto, B. Libert, N. McCullagh, and J.-J. Quisquater. Efficient and provably-secure identity-based signatures and signcryption from bilinear maps. In *Proc. of ASIACRYPT*, pages 515–532, London, UK, 2005. Springer-Verlag.

[10] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the weil pairing. In *Proc. of ASIACRYPT*, pages 514–532, London, UK, 2001. Springer-Verlag.

[11] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure routing for structured peer-to-peer overlay networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):299–314, 2002.

[12] D. Clark. Requirements for a future internet security as a case study. http://find.isi.edu/ presentation_files/ David_Clark-Security-Requirements-2.pdf, Dec. 2005.

[13] A. R. et al. Load balancing in structured p2p systems. In *Proc. of IPTPS*, pages 68–79, 2003.

[14] P. Ferguson. *Network Ingress Filtering: Defeating Denial of Service Attacks Which Employ IP Source Address Spoofing*. RFC 2267, Jan. 1998.

[15] Y.-C. Hu, A. Perrig, and M. Sirbu. Spv: secure path vector routing for securing bgp. In *Proc. of ACM SIGCOMM*, pages 179–192, 2004.

[16] S. Kent, C. Lynn, and K. Seo. Secure border gateway protocol (secure-bgp). *IEEE Journal on Selected Areas in Communications*, 18(4):582–592, 2000.

[17] T. Kerins, W. P. Marnane, E. M. Popovici, and P. Barreto. Hardware accelerators for pairing based. cryptosystems. In *Proc. of IEE Information security*, pages 47–56, 2005.

[18] A. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure Overlay Services. In *Proc. of ACM SIGCOMM*, pages 61–72, August 2002.

[19] R. Mahajan, S. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. *ACM Computer Communication Review*, 32(3):62–73, July 2002.

[20] R. Moskowitz, P. Nikander, P. Jokela, and T. Henderson. Host identity protocol. Technical report, Internet draft, 2005.

[21] K. Park and H. Lee. On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law Internets. In *Proc. ACM*

*SIGCOMM*, pages 15–26, Aug. 2001.

[22] V. Ramasubramanian and E. Sirer. O(1) lookup performance for power-law query distributions in peer-to-peer overlays. In *Proc. of NSDI*, 2004.

[23] A. Rowstorn and P. Druschel. Patry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM Middleware 2001*, Nov 2001.

[24] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical network support for ip traceback. In *Proc. ACM SIGCOMM 2000*, pages 295–306, Aug. 2000.

[25] A. C. Snoeren and H. Balakrishnan. An end-to-end approach to host mobility. In *Proc. of ACM MobiCom*, pages 155–166, 2000.

[26] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. In *Proceedings of ACM SIGCOMM*, pages 73–86, 2002.

[27] M. Walfish, H. Balakrishnan, and S. Shenker. Untangling the web from DNS. In *Proc. of NSDI*, 2004.

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

[29] X. Yang, D. Wetherall, and T. Anderson. A DoS-limiting network architecture. In *Proc. of ACM SIGCOMM*, pages 241–252, 2005.