Towards a Secure Internet Architecture Through Signaling

Saikat Guha, Paul Francis
Cornell University
{saikat, francis}@cs.cornell.edu

Abstract

The current model for flow establishment in the Internet: DNS Names, IP addresses, and transport ports, is inadequate. Not all of the problem is due to the small IPv4 address space and resulting NAT boxes. Even where global addresses exist, firewalls cannot glean enough information about a flow from packet headers, and so often err, typically though not always by being over-conservative: disallowing flows that might otherwise be allowed. This paper presents a novel architecture, protocol design, and implementation, for secure flow establishment in the Internet. The architecture, called NUTSS, takes into account the security policies of both endpoints and network providers. NUTSS uses URIs for naming machines, applications, users, and flows, and uses signaling protocols to authenticate and authorize the high-level named flows, and subsequently bind them to ephemeral IPv4 (or IPv6) 5-tuple data flows. Combined with recent NAT traversal techniques, NUTSS works with IPv4 and reduces the need for IPv6. This paper analyzes modern security requirements, describes NUTSS and how it leads to a secure Internet, describes our prototype implementation, and outlines other benefits of NUTSS, including mobility, indirection, anycast and multicast, and billing.

1 Introduction

The POSIX sockets interface defines the core networking services offered by a wide range of OS's — in other words, the OS network services that an application developer can depend on being available. The sockets interface originally offered a small set of critical services:

- Provide long-term stable user-friendly names and (non-user-friendly) addresses to identify specific applications running on specific machines (DNS names, IP addresses, and ports).
- Provide the ability to transmit and receive a stream of bytes (TCP) or datagrams (UDP) to and from the identified applications.

Over the years, this minimal set of services has deteriorated. Because there aren't enough IPv4 addresses, not all machines can be identified, or if they can the identities are often ephemeral. IPv6 aims to solve this problem by providing more addresses. Even with IPv6, however, because of firewalls it may still not be possible to transmit or receive packets to/from a given machine and application. This is of course a feature, not a bug, at least in the mind of the firewall operator, and suggests that the original small set of services provided by the network, and made ubiquitously available through the sockets interface, is inadequate in today's hostile network environment.

In this paper, we argue that the network and OS interface should be extended to include a network ACL (Access Control List) service. Towards this end, this paper contributes an architecture, protocol, and initial implementation for such a service. Creating this service, however, is not a matter of simply standardizing an API that makes current personal firewall technology [1, 2, 3] available at the sockets interface. There are broadly two reasons for this:

First, it isn't enough to filter packets inside the endpoint's OS. The endpoint (application or user) may wish that certain packets (for instance a DoS attack) not reach its network interface or even the ISP access link. In addition, the endpoint is not the only party that has a stake in how packets are filtered. The network provider may also have an ACL policy for the network. Furthermore, the network provider's policy may differ from the endpoint's policy. Thus, endpoints and elements deep in the network must be able to work together to provide a safer, more controllable network service, and the network ACL architecture must allow for this.

Second, current (IPv4) or planned (IPv6) network protocols don't provide adequate semantics or mechanisms for a network ACL service. Among other things, IPv4 addresses are not globally unique, IPv6 addresses can be but are likely to be ephemeral in practice, neither of them are user-friendly, the port number does not adequately identify the application at network firewalls, and the DNS service does not understand enough about the querying user or impending application to know whether or how to answer a query.

At the center of our architecture is the use of URIs for naming machines, applications, users, and flows, and the use of signaling protocols to authenticate and authorize the high-level named flows, and subsequently bind them to ephemeral IPv4 (or IPv6) 5-tuple data flows. In doing this, we eliminate the need for permanent IP or port assignments for *non-public server* communications. Significantly, in light of recent advances in NAT traver-

sal [4, 5], we eliminate the need for globally unique IP addresses. In other words, not only is IPv6 not adequate for access-controlled global communications, it is not even necessary.

Note that the idea of using URIs and signaling to richly name a flow and subsequently bind it to a 5-tuple is not new: the Session Initiation Protocol (SIP) does this today for VoIP flows. Our architecture heavily exploits SIP, enhancing it slightly to work for all types of applications, not just media. We combine this with a new simple on-path signaling mechanism used to provide the credentials needed by firewalls to allow flows to pass.

This architecture is called NUTSS, for its constituent components: NATs, URIs, Tunnels, SIP, and STUNT. In addition to the above security-oriented contributions, this paper outlines how NUTSS may be used to provide additional features, such as user and device mobility, site multi-homing, asynchronous message passing, and gateway service to other types of networks such as IPv6 or specialized sensor networks.

Altogether, this paper makes four contributions. (1) We broadly analyze Internet security requirements, taking both endpoint and network policies into consideration in §2. Based on this, we are, to the best of our knowledge, the first to propose for this purpose a combined off-path and on-path signaling approach — NUTSS. (2) We design an architecture for NUTSS which exploits and extends SIP, and show how it works to provide Internet security in §3 and §4 respectively. (3) We describe a proof-of-concept implementation of NUTSS in §5. (4) We outline how NUTSS helps solve a broad range of network problems, including mobility, multi-homing, bridging disparate networking technologies, indirection, and billing in §6.

2 Security in the Internet

The end-to-end argument states that the higher layers (end) cannot fully depend on the lower layers (middle) to provide functions such as error recovery and security, and therefore must itself provide these functions [6]. As a result, it may be redundant and inefficient to provide these functions in the middle, and therefore doing so is justified only as performance enhancements. The original security architecture of the Internet was strongly guided by this argument. It prescribes that *almost all* security is best provided by the ends, and that very little "performance enhancement" can be had by doing security in the middle. In other words, the function of the network is to deliver packets regardless of whether or not they are desired by the end. It is up to the end (the receiving host's OS and application) to determine whether or not to drop a given received packet. The E2E argument led to a protocol design for connection establishment, namely the address/port model, that intentionally provides very little information to the middle about what is happening at the ends.

We believe that this argument has not fared well with respect to Internet security, and that the proliferation of firewalls and other security-oriented middleboxes is symptomatic of this. To be clear, we certainly believe that the end is the appropriate place for most security. The middle must, however, play a larger role than originally anticipated, and the Internet protocol design has made allowing this difficult.

The problems with the pure E2E model of security are as follows. First, it does not protect against DoS attacks. This problem alone dictates that there must be some way for the middle to filter packets, ideally as close as possible to the attacker. In some cases, control over this filtering might be at or near the receiving end. In other cases, such as where an ISP wishes to protect itself or customers that are affected by the attack but not directly under attack, the middle may control this filtering [7].

The second problem with the pure E2E model of security is that it is difficult to fully secure all end hosts. Take, for instance, a typical enterprise network. The end-users themselves are at best incapable of securing their end-hosts, and at worst are hostile to the policies of the enterprise's IT department. It is up to IT, then, to secure the end systems. IT can attempt to do so with an arsenal of tools, such as software installation packages [8], and end-host firewalls and virus checkers [3]. Any error in this process, however, may leave one or more end-hosts vulnerable to attack. A firewall at the enterprise's gateway serves as a first line of defense to guard against any weaknesses in end-host security.

Related to the second problem is the case where the middle has a legitimate stake in controlling packet flows, but has little or no direct control over end-hosts. For instance, in some environments, such as a university campus, IT cannot directly control what OS or software is installed in end-hosts (though they may be able to enforce usage of a virus scanner). Another example is where an ISP's customers' end-hosts are infected with software that delivers SPAM, thus loading the ISP's network. The ISP has a legitimate reason for wanting to prevent those packet flows.

Security in the middle has evolved in ad hoc ways. A class of boxes have been developed that allow the middle to disallow specific flows, steer flows to alternate endpoints, and route flows over encrypted tunnels. We generally refer to these boxes as *middleboxes*.

The problems with the existing model of deploying middleboxes are as follows. First, middleboxes have very little information about a flow when deciding on its outcome. Consider, for instance, how a firewall works. It must decide whether or not to block a flow based

largely on the first packet. To do otherwise would not protect against DoS attacks or single packet vulnerabilities [9]. It is up to the firewall, then, to infer whether a flow is malicious based largely on the IP addresses, port numbers, and content of the packet. Firewalls provide security to the extent that this information is sufficient for the firewall to identify the intent of the sender application, but that is not always the case. Users can have dynamic or non-unique IP addresses (DHCP, NAT), applications can run on non-standard ports, and flow contents can be encrypted. Consequently, middleboxes today err on the side of caution and often block unmalicious flows.

The second problem with existing middleboxes is that they are often forced to act unilaterally without consulting the end. IP/TCP/UDP alone does not allow middleboxes to negotiate with the endpoint how a flow should be handled. Firewalls, for instance, cannot ask the intended destination whether or not it is prepared to handle a particular flow without first allowing the (potentially malicious) flow to succeed.

One might ask – what would it take to do security better in the Internet? This leads to some basic questions. *Who* needs to be involved in providing security: the end, the middle, both? *What* information needs to be known about flows, *when* and by *whom*? And, *how* is this information disseminated? The remainder of this section conducts a thought-experiment to answer these questions, and evaluates how existing and proposed Internet architectures stack up in this context.

2.1 Who needs to be involved?

Broadly speaking, any endpoint or network that carries a flow may potentially have a say in whether the flow is allowed. Endpoints typically allow flows from certain remote endpoints and disallow flows from others. In an enterprise, endpoints may delegate some policy to the enterprise's IT department. The IT department may, in addition, have mandatory policy that protects the enterprise network and endpoints from other compromised or hostile endpoints on the (public) Internet. IT may further need to delegate DoS-related policy to the ISP upstream of the enterprise's bottleneck link to the Internet.

In general, both endpoints and any network on the data-path may have policy governing a flow. In some cases, the policy at the end may reinforce the policy in the middle, while in other cases, it may conflict. In practice, however, the farther one moves away from the end, the less one has to say about the flow.

2.2 What needs to be known?

Policy is typically defined in terms of users and applications, and aims to restrict network use to authorized endpoints or place special requirements on flows. For instance, physical access or Virtual Private Networks (VPN) are used to authorize users today, firewalls perform port-based filtering to restrict applications, and services are configured to respond only over Transport Layer Security (TLS). We formalize this de facto expressiveness of policy today by defining its components.

Policy consists of the following two elements. The first matches specific flows that the policy refers to. The second element of policy determines how matched flows are treated. As an example of matching policy to flow, policy may be defined for flows that are unencrypted and cross the public Internet. Another example may be policy for flows from a particular user and application. The less a middlebox has to guess about a flow, the more precisely it can target just the right flows and the less it needs to err on the side of caution by restricting other flows. To that end, the following information is useful to the middleboxes.

- **Endpoint Identity -** includes not only the applications at which a flow originates or terminates, but also the users and the boxes running the applications. Endpoint identity can be used in policy to target flows to or from specific individuals, applications, hosts, or a combination of the three.
- **Transport Attributes -** are protocol-level properties of a flow such as mode and strength of encryption, choice of access link, bandwidth requirements etc. Policy can be written to target a flow based on the requirements the flow places on the network, and the level of security and confidentiality that it provides.
- **Data Attributes -** are labels and taints applied to the endpoints establishing a flow. Operating Systems, such as Asbestos [10], assign taints to applications based on data accessed by the application. Declaring this taint allows policy to differentiate between flows from an application that may have accessed confidential data from applications that have not.

Regarding the second element of policy, determining how matched flows are treated, middleboxes today have the capability to block flows, steer flows to alternate destinations, apply bandwidth caps, encapsulate a flow inside secure tunnels, log and audit flows, and even scrub flows to remove viruses. These actions can be separated into two classes.

- **Flow Steering -** where the policy redirects the flow to an alternate middlebox for processing. This includes a firewall that may block the flow, a caching proxy that may respond to the flow from an internal cache, or a virus checker that may transparently scrub the flow free of viruses and allow it to continue.

- **Transport Modification -** where the policy modifies the transport attributes of the flow. This includes policy routing over a VPN tunnel, applying traffic shaping measures to protect against DDoS, and using NATs to hide the private transport address.

Both endpoints and the middle networks that have policy regarding a flow must be able to precisely match flows that the policy targets, and take appropriate action. The next-generation secure Internet should, in our opinion, support high-level policy definitions (users and applications, not IP addresses and ports) and explicitly support the rich set of actions that middleboxes can take today.

2.3 When and how are flow details known?

There is tension between providing all the information needed to match policies with flows on one hand, and preserving privacy on the other. For instance, to help prevent DoS attacks, an endpoint may wish to withhold its IP address until it has determined that it trusts the remote endpoint with that information. As another example, a flow initiator may wish to remain anonymous by not providing a name and by steering its packets through an anonymizing middle box. The flow recipient, on the other hand, may refuse to accept anonymous flows.

2.4 Related Work and Internet Security Today

> "In the Internet, you cannot lock your doors. So you do the next best thing — you sit behind the door with a shotgun." [11]

As already mentioned, one weakness of the E2E security architecture is that IP addresses are publicly routable and anyone can launch a DDoS attack knowing the target's IP address. Furthermore, anyone can learn the target's IP address over DNS; recursion and caching prevent the DNS server from controlling who learns the target's IP. Several DDoS solutions have been proposed and implemented. Akamai keeps the target's IP secret by using DNS to redirect users to a distributed cache [12]. Akamai's solution amounts to security by obscurity where if the target's true IP address is ever revealed, it can be attacked directly. Riverhead allows enterprises and ISPs to deploy on-path firewalls that use heuristics to mitigate a DDoS attack [7]. Many proposed solutions to the DDoS problem, including Pushback [13], AITF [14], Off-By-Default [15], SIFF [16], Capabilities [17], SOS [18], and Mayday [19] configure upstream routers to drop attack traffic from unwanted sources. These approaches rely on the expressiveness of the flow 5-tuple to distinguish good traffic from bad. In the presence of multiple users on the same host, dynamic IP allocation, NATs, and applications running on non-standard ports the 5-tuple is ephemeral [20, 21], and policy defined on 5-tuples serves little purpose when the same tuple can be malicious or unmalicious at different times. Consequently these approaches sometimes unnecessarily block unmalicious traffic that is misclassified (or generalized) as malicious traffic, or do not block all malicious traffic.

The other weakness in the E2E security model mentioned earlier is that not everyone who has a stake in security is empowered to provide that security. In this model, the middle has little say in what flows are allowed and must rely completely on the endpoints for the protection of the endpoint and the network itself. In select scenarios, in an enterprise for example, the IT department can enforce this control over the ends through software update and configuration management tools like Marimba [8]. In other cases, such as with DoA [22], endpoints can explicitly invoke security services provided by the middle. Such solutions, however, do not protect against malicious or compromised endpoints that may preempt the IT department's control and abuse the network. An alternate solution is where the middle exerts direct control on flows with the help of an on-path middlebox. While middleboxes protect the network against uncooperative endpoints, they face the aforementioned problems which we repeat here: firewalls must infer malice based largely on the 5-tuple, D-WARD [23] infers malice based on deviations from a "normal traffic model", NATs protect only against drive-by intrusions from the outside, and VPNs cannot authenticate remote users that are not VPN members. Ultimately, such middle-only approaches that cannot explicitly negotiate the intent of the flow with the endpoint rely on heuristics and potentially block unmalicious flows.

Protocols such as UPnP [24] and Midcom [25] where the endpoint can explicitly request a middlebox to perform some action, are designed to provide some coordination between middleboxes and endpoints. Such requests, however, cannot be made by the middlebox or the remote endpoint. Middleboxes, for instance, cannot challenge the endpoints for authorization for arbitrary flows, nor force the endpoints to use encryption.

Several other Internet architectures have been proposed that wean away from 5-tuple addressing and end-only control. TRIAD [26], IPNL [27], and the i3 service [28] route based on URLs, FQDNs, and flat identifiers respectively. Selnet [29], Plutarch [30], AVES [31], and Metanet [32] view the Internet as a collection of realms where inter-realm elements (the middle) help in endpoint resolution. In order to provide secure connectivity in the Internet, we believe the middle must play a yet larger role in helping negotiate flows through networks between endpoints.

Altogether, we believe that involving the middle in providing security is imperative, and to that end, endpoints and middleboxes must be able to coordinate to set up flows. In the following section we describe an Internet signaling primitive that allows the end and the middle to coordinate such, and in §4 we describe how a secure Internet architecture can be constructed using this signaling primitive.

3 NUTSS Signaling Primitive

The discussion in the previous section suggests that an "exchange of information" (i.e. signaling) between endpoints and middleboxes must take place before application data packets can flow between endpoints. This signaling allows the middle and the ends to allow or disallow flows as well as negotiate the characteristics of the flows. Noting that any signaling primitive is heavyweight, at least compared to no signaling, we should re-state that the signaling described here is not meant to apply to public servers. DoS attacks aside, access to public servers is supported well by the lighter-weight address/port/DNS model. Rather, our signaling primitive is meant for situations where the endpoint and its associated networks have some policy or another that restricts access to the endpoint.

The NUTSS architecture supports not one but two signaling modes: an off-path mode and on-path mode. By off-path, we mean that the signaling message take a different path through the network than the path taken by the (application) data packets. The primary benefit of off-path signaling is the flexibility it offers in where to place functions. For instance, this allows a network administrator to centralize its firewall and other security policies, and to place them far away from the actual end-networks and end-hosts to which those policies apply. This is important, for instance, in a default-off communications setting, because it prevents potentially hostile flow initiators from getting their packets anywhere close to the target (victim) host.

On-path signaling is required, on the other hand, because ultimately it is necessary for elements in the data-path (middleboxes) to know about an impending data flow. Firewalls need to pass the approved flows, routers may need to reserve resources for the flows or steer them in certain ways (towards a virus checker, for instance), and so on. An off-path protocol alone cannot do this, so our design also explores how to couple the off-path and on-path portions of the flow establishment problems.

Fortunately, there already exists a mature off-path signaling protocol in the form of the Session Initiation Protocol (SIP) [33]. SIP is a signaling protocol for initiating interactive multimedia sessions such as video and voice over the Internet. SIP enables a powerful set of features: mobility, rich naming of users and endpoints, discovery, the ability to negotiate different protocols, independence from underlying transport, and the creation of infrastructure to support it all. We have found that, with minor enhancements, SIP does everything we need for establishing data flows. This is not surprising because SIP is designed not to offer a specific service, but rather as a set of primitives over which services can be built. Add to this the fact that there is an extensive and growing SIP infrastructure and expertise, and we find that SIP is an ideal primitive upon which to build a secure Internet architecture.

Attribute	Example Values
(user component)	user@domain.org
app	ftp|mail|web|...
type	client|server|peer
software	vsftp|sendmail|firefox|...
version	2.14.3
location	home|office|laptop|...

Table 1: Core fields in endpoint descriptors

In this section we describe the design of the NUTSS signaling primitive. Though NUTSS has a number of benefits (we discuss some of them in §6), we use the problem of designing a secure Internet architecture as our running example. We discuss our design of a secure Internet architecture in Section 4 to concretely illustrate how NUTSS meets the security requirements identified in the previous section.

3.1 Names and Descriptors

Endpoints in NUTSS are identified by rich descriptors that encode high-level information including the user, application, and host. Flows in NUTSS are also identified by rich descriptors that include certainly the source and destination endpoints, but also transport and data attributes such as encryption and taints. The use of high-level descriptors simplifies policy definition by removing ephemeral elements from the name such as IP addresses and port numbers. It also makes naming independent of the underlying network, thus allowing an internet that supports multiple network types (IPv4, IPv6, sensor nets, etc.)

The actual encoding of an endpoint in NUTSS is through SIP URIs. URIs in SIP contain a required user component that looks like an email address, followed by a number of key=value attributes (e.g. bob@acme.org;app=ftp;type=client). The URI, in this case, identifies the user (bob), the domain (acme.org), the application (app=ftp), and the type of endpoint (type=client). While any number of key types can be defined, Table 1 lists the core endpoint-attributes understood by NUTSS .

Flow descriptors in NUTSS are encoded as SIP

Attribute	Example Values
To	alice@rr.net;app=ftp;type=server
From	bob@acme.org;app=ftp;type=client
Protocol	m=data ftp tcp
Address and Port	c=IN IP4 128.84.223.109:32524
Encryption	a=enc:ssl|ipsec|...
Taint	a=taint:bob.confidential

Table 2: Core fields in flow descriptors

headers and SDP [34] extents carried inside SIP messages. SIP header fields such as `To:` and `From:` contain the SIP URIs encoding the endpoint descriptors for the destination and source respectively. The body of the SIP message contains a Session Description Protocol (SDP) element that encodes flow specific parameters. The SDP format is used to encode the IP addresses and ports, mode and strength of encryption, taints carried by the flow etc. While any number of flow-attributes can be defined, Table 2 lists the core flow-attributes understood by NUTSS . As an example of a flow descriptor, an unencrypted flow from Bob's FTP client, which has accessed data confidential to Bob, to Alice's FTP server is encoded as follows.

```
To: alice@rr.net;app=ftp;type=server
From: bob@acme.org;app=ftp;type=client;
 software=ftp;version=0.17

m=data ftp tcp
c=IN IP4 128.84.223.109:32524
a=enc:none
a=taint:bob.confidential
```

> **Property 3.1:** Descriptors in NUTSS are long-term stable, network-independent, user-friendly, rich descriptions of the endpoint or flow.

By design, endpoint descriptors are independent of the network location of the endpoint, and thus serve as long-term stable identifiers. Flow descriptors are constructed mostly by the source application with some help from the user. For instance, when Bob instructs his FTP client to contact Alice's server, he simply provides alice@rr.net to the application. The FTP application then appends its own app,type,software,version attributes to Bob's identity (learned from an environment variable), appends the app,type of the remote endpoint it wishes to contact to Alice's identity (given by Bob), and appends the encryption settings (based perhaps on a drop-down option selectable by Bob). Consequently, the destination "name" supplied by the user is user-friendly and stable (just like an email address). Overall, descriptors in NUTSS are flexible representations of endpoints and flows.

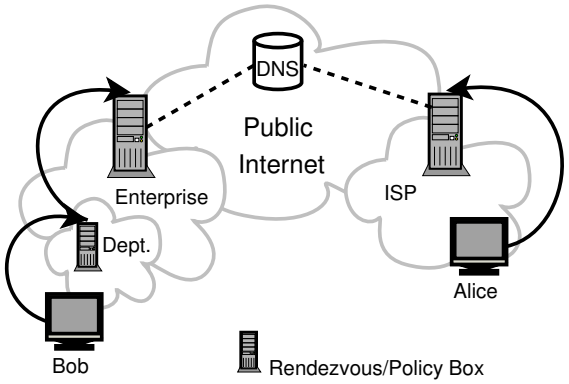

Figure 1: Endpoint registration and policy boxes

3.2 Routing Signaling Messages

Signaling messages are addressed to endpoint descriptors and are routed by the signaling infrastructure. Added is the constraint that signaling messages must visit a box designated by intermediate networks that has security policy that may effect the message. These boxes are called policy boxes, and are distinct from middleboxes in that data packets do not have to flow through them. A middlebox may of course also be a policy box.

For instance, consider Figure 1 where Bob connects to the Internet through a departmental network of an enterprise, while Alice connects through an edge-ISP. Bob's request to initiate a flow (the flow descriptor mentioned previously) is addressed simply to alice@rr.net;app=ftp;type=server. It is up to the signaling infrastructure, then, to discover where in the network Alice is, and route the message to her after the message has been vetted by boxes designated by Bob's department and enterprise, and Alice's edge-ISP.

Rendezvous in the NUTSS signaling design extends the rendezvous mechanism provided by SIP. Each *domain*, defined as an independent network (e.g. department network, enterprise, ISP), maintains one or more policy boxes that keep track of the ephemeral IP address and port through which an endpoint in that domain can be reached; this ephemeral address may be that of the endpoint itself, or of another policy box that may have more specific knowledge. For each domain connected to the "public Internet", the address of a policy box overseeing that domain is resolved over DNS. Given such an infrastructure, rendezvous becomes a combination of the endpoint registering its network location with one of its local policy boxes that recursively creates a chain of registrations to an outermost policy box, and a resolver that walks down this chain of registrations.

Endpoint registration in NUTSS involves the following steps. First, the endpoint registers a mapping between its descriptor and its ephemeral address and port

with a policy box for the local domain; the policy box may update a backend database that is shared with other policy boxes. Next, the policy box registers a mapping between the endpoint's descriptor and the policy box's *own* address (or addresses of all the policy boxes for that domain), with a parent policy box, if any. This process continues until the endpoint descriptor is registered at an outermost policy box connected to the public Internet. For the settings in which it is appropriate, the endpoint can discover a local policy box, and a policy box can discover its parent through manual configuration or over DHCP. Later we describe a more automatic discovery mechanism that works in all settings where the endpoint (or policy box) sends a packet that is captured at a firewall for that (or parent) domain, which in turn responds with the necessary configuration. The whole registration process creates a small chain of mappings (typically of length one or two) from the public Internet to the endpoint threaded through the policy boxes of the intermediate domains. Each policy box maintains state proportional to the number of active endpoints behind it, which is far more scalable than per-flow state that firewalls already maintain today. Messages to an endpoint are routed through this chain subject to domain policy, which is enforced by the policy boxes.

> **Property 3.2:** The signaling infrastructure routes messages to endpoints through policy boxes designated by domains on-path between the two endpoints.

In order to route a signaling message from a source to the destination, the signaling infrastructure routes the message up the chain of registrations created by the source, potentially across the public Internet to the destination's domain, and down the chain of registrations created by the destination. For example, Bob's FTP client constructs the flow descriptor addressed to Alice (alice@rr.net;app=ftp;type=server) as illustrated earlier. This request is sent to Bob's local policy box. Since Alice's endpoint is not in the local domain, the box forwards the request to its parent, the enterprise policy box. The enterprise box queries DNS for the policy box for Alice's domain and forwards the message to it. The message is received by Alice's ISP's policy box which finds Alice's local address in its mapping table and routes the message to the endpoint. Any policy box en route can refuse to forward the signaling message if local policy forbids Bob from initiating an unencrypted FTP connection with Alice.

Mobility: Mobility of endpoints is handled by the signaling infrastructure, which creates the necessary glue between the endpoint's location and its "home domain". Figure 2 illustrates how the signaling path is constructed for a mobile user. In the figure, Alice is a

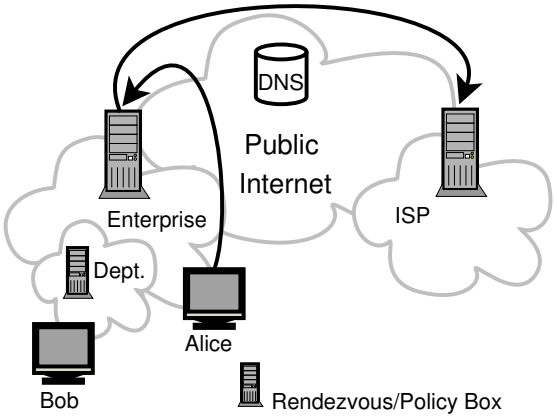

Figure 2: Registration of mobile endpoints

mobile user who joins the network while visiting Bob's enterprise. Alice registers her endpoint descriptor with the local domain's policy box. The enterprise box notes that Alice (alice@rr.net) is not a part of its domain (acme.org) and checks its policy whether or not to allow Alice to access the network. If Alice is granted access, the enterprise box registers a mapping between Alice's endpoint descriptor and the enterprise box's address with Alice's ISP's policy box. When Bob contacts Alice, the signaling message may be routed directly to Alice by the enterprise's policy box (if policy permits). Messages from Alice that originate outside the enterprise are routed to her ISP's policy box as usual, which then redirects the message to the enterprise's policy box as per the mapping created during registration. In all cases, notably, the 5-tuple for the data flow is negotiated dynamically (as we explain later), and can therefore take advantage of the direct path between the source and mobile host.

> **Property 3.3:** The signaling infrastructure uses partial attribute matching for routing messages.

Endpoints may register only a part of their descriptor for privacy, flexibility, and scalability reasons. While Alice's FTP server can register its full endpoint descriptor (alice@rr.net;app=ftp;type=server;software=vsftpd;version=2.0.4) at a policy box, it may choose to hide certain details, such as the software,version or even app,type, in order to protect its privacy. Policy boxes may wish to drop certain attributes to better aggregate multiple endpoint descriptors from the same user (such as aggregating the endpoint descriptors for Alice's FTP server and webserver with a single descriptor alice@rr.net). The NUTSS signaling infrastructure routes messages to endpoints as long as the user component is present in each registration; the tradeoff for hiding attributes,

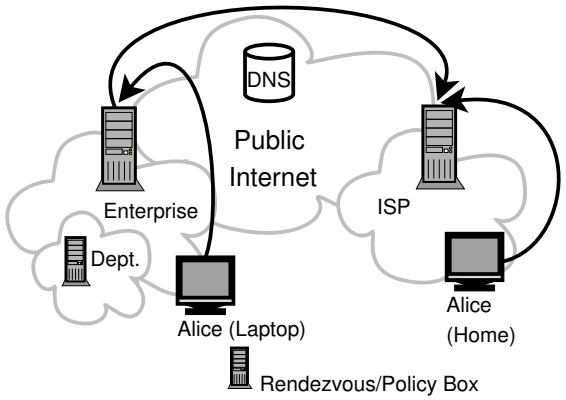

Figure 3: Registration of multiple endpoints

however, is that the discovery process may take longer since multiple candidate endpoints may need to be contacted before the right one is identified.

Policy boxes route messages to endpoints whose descriptors match all or some of the attributes listed in the destination address for the message. If an attribute is present in both the destination address and the endpoint descriptor registered at the policy box then their values must match for that endpoint to be considered a potential destination. Attributes mentioned in only one of the destination address or registered descriptor and not the other do not remove that endpoint from consideration, however, endpoints with a larger number of matching attributes are preferred. For instance, a message addressed to alice@rr.net;app=ftp;type=server may be routed to the endpoints registering the descriptors alice@rr.net;app=ftp;type=server;software=vsftpd (match on 2 attributes), or alice@rr.net (0 attributes) in that order of preference, but not to alice@rr.net;app=web;type=server (conflict on 1 attribute). Candidate endpoints are contacted in the order of preference (most matching attributes first). If the message is accepted by the destination endpoint, an acknowledgment (ack) is sent back along the signaling path to indicate success, otherwise a negative-acknowledgment (nack) is sent. If a nack is receive, the policy box tries the next candidate and so on.

Multi-Homing: Routing based on partial attributes adds flexibility in selecting signaling paths. In Figure 3, Alice runs an FTP server on her laptop connected through the enterprise network, and a second FTP server on her home machine connected to the ISP network. Her home machine does not wish to disclose the application running on it and therefore registers the endpoint descriptor: alice@rr.net;location=home. Alice's laptop registers alice@rr.net;app=ftp;type=server;location=laptop through the enterprise network. Incoming messages for any one of Alice's FTP servers (with attribute app=ftp) matches both entries. Alice's ISP prefers to redirect the message through the enterprise network to Alice's laptop. If the laptop does not respond (nack is sent by the enterprise policy box), Alice's ISP routes the message to the next candidate: the home machine. Messages addressed specifically to her home FTP server (with attributes app=ftp;location=home) match only the first entry and is routed only to the home machine.

3.3 Flow Negotiation and Setup

Once the off-path signaling path above is created, the endpoints and policy boxes negotiate flow parameters for the actual data flow. This serves three purposes. First, endpoints and policy boxes can authenticate each other before any on-path resources need to be reserved for the flow. Second, the policy boxes can participate in the negotiation, for example, by requesting endpoints to use encryption for certain flows. Third, the policy boxes can issue to the endpoints, *capability tokens* that configure data-path elements, such as firewalls and NATs, to route the flow.

> **Property 3.4:** Off-path signaling is used to discover endpoints and negotiate flow parameters. Capability tokens issued by policy boxes are used to couple off-path signaling with on-path signaling. On-path signaling configures middleboxes through which the flow passes.

As mentioned earlier, on-path signaling is necessary to configure middleboxes to allow the impending flow. Different middleboxes may require different levels of on-path signaling. Firewalls, for instance, may require per-packet proof that the endpoint has the required capability to send that packet. NATs, on the other hand, require only per-flow configuration/capability tokens to create the necessary mappings when an external endpoint initiates the data flow. In either case, the policy box generates a capability token that it signals to both the endpoints along with instructions on when the token is used; signaling the token to both ends allows either end to initiate the flow. The endpoints present the tokens to on-path middleboxes by embedding them inside data packets. Consequently, middleboxes in our architecture can potentially be stateless. Middleboxes interpret the tokens in data packets and take appropriate actions to enable data-flow between the endpoints. As an example of this coupling, the policy box may generate an encrypted signed certificate (the capability token) once off-path signaling has been used to authenticate the endpoints and authorize the flow. This token is signaled to the endpoints, which embed a proof of this capability in each data packet. Firewalls in a domain drop pack-

ets that do not contain this proof of authorization from a policy box in that domain.

4 A Secure Internet Architecture

The principle of least privilege states that in order to enhance protection against malicious behavior, information and resources available to a process must be restricted to the minimum necessary to complete the job [35]. In the contrapositive, a (malicious) process that operates with more information or resources than necessary can potentially abuse those resources to attack a victim. The original Internet architecture was not guided strongly by this argument. It, instead, prescribes that any endpoint be able to send packets to any other endpoint regardless of whether or not it *needs to* in order to perform its job.

We believe that the principle of least privilege has much to offer with respect to Internet security. To clarify, we certainly believe that applications should be free to communicate over the Internet. The Internet must, however, protect endpoints against flows that are not necessary for their functioning. To that end, the NUTSS security architecture disallows, by default, all flows initiated unilaterally. As mentioned earlier, NUTSS doesn't apply to public servers where policy allows anyone to unilaterally initiate a flow. Flows to private servers, however, must be authorized by all policy boxes, and endpoints concerned before they are allowed to succeed.

Given the NUTSS signaling primitive from the previous section, we now define concretely the NUTSS secure Internet architecture as one use-case of the signaling infrastructure. NUTSS relies on policy boxes to authorize flows, and firewalls to enforce the policy. A domain (or end-host) may run its own policy boxes or delegate to a third-party service provider; the policy boxes themselves may be located anywhere – either inside the domain (or on the end-host) they oversee, or outside. Firewalls are deployed by domains at the edges of their network (and by end-hosts in their OSs), but may also be deployed on intermediate routers within the domain.

All flows are turned "off by default" [15], except for flows to and from policy boxes (i.e the flows necessary for the signaling infrastructure). These "on by default" flows can be selectively turned off when necessary. Each new flow is negotiated over the signaling infrastructure; a successful negotiation results in capability tokens from policy boxes that, when present in a data packet, allows the firewall to determine whether the packet should be allowed to pass through. Altogether, the NUTSS architecture operates by largely turning off connectivity in the private-server Internet and then selectively turning it back on when necessary.

Security policy is defined in terms of the users, applications, and hosts between which flows need to be established. In cases where the OS can manage taints [10], taints applied to applications based on data handled can be conveyed through signaling. In addition, policy can restrict flows based on transport attributes such as mode and strength of encryption.

4.1 Network Elements

As mentioned earlier, the NUTSS security architecture relies on policy boxes (P-Box) and firewalls (M-Box) to provide security. P-Boxes form the off-path signaling infrastructure explained in the previous section, and may be thought of as SIP proxies with minor modifications. M-Boxes are (very simple) firewalls that can check the validity of a capability token in the data packet, and drop the packet if validity cannot be ascertained. NUTSS assumes the IP routing core and DNS in their present form. It also assumes the existence of a PKI infrastructure such as IPsec and SSL certificates [36], or PGP [37] that are available today. Finally, NUTSS assumes a deployment model where a domain is connected to the Internet through one or more ISPs.

> **Property 4.1:** NUTSS can be deployed with services and functionality already present in the Internet and end-hosts today.

End-hosts may have a P-Box application for handling policy decisions for that host, and a personal firewall (available in most OSs today) that serves as the M-Box. End-hosts may or may not have hardware and software support for trusted computing [38], or secure operating systems [10]. To the extent that such support is available, NUTSS can leverage it to provide greater security, but it does not require this support.

P-Boxes and M-Boxes for a domain share a common secret. The secret is used by the P-Box to create unforgeable capability tokens that the M-Box can verify. Endpoints (or more specifically the end-user) have certificates that can be used to authenticate them, verify them as a source of a signed message, and encrypt flows to them. Each domain also has a similar certificate that can be used to identify and securely communicate with any P-Box for that domain. The endpoint, however, may not necessarily be configured a priori with the certificates of the domains via which it connects.

4.2 Deployment

> **Property 4.2:** M-Boxes block all packets that do not contain the necessary authorization. Packets to or from P-Boxes in the same domain as the M-Box are always authorized. Error responses to packets blocked by a M-Box contain information necessary to discover and contact the P-Box for that domain.

M-Boxes are configured to drop all packets by default except those to and from P-Boxes for that domain. Absent DHCP or manual configuration, an endpoint may automatically discover the local P-Box by sending a data packet to any public address outside the domain, which is blocked by the M-Box that in turn responds with an error message that contains configuration information necessary to securely contact the domain's P-Box. A P-Box likewise can automatically discover its parent P-Box (if any) by sending a data packet, which is allowed to pass through that domain's M-Box but is blocked by the parent domain's M-Box (if any). A similar error message allows the P-Box to securely contact the parent domain's P-Box. Endpoints register their descriptor with the local P-Box, which in turn registers with the parent P-Box to construct the signaling glue as explained in the previous section.

Endpoints and P-Boxes communicate over mutually-authenticated TLS channels established for each hop of the signaling path. Signaling messages are visible to P-Boxes along the signaling path (for routing and policy purposes). However, some portions of the message may be encrypted end-to-end (e.g. session keys for the impending flow).

P-Boxes contain policy information for their domain. P-Boxes may optionally allow endpoints and other P-Boxes to upload a copy of their policy information to it so that policy may be consulted closer to the source of a message. In keeping with the principle of least privilege, the default policy is to deny all flows. The policy information explicitly allows very specific flows to be established, and potentially forces flows to be routed through various middleboxes such as virus checkers etc. Endpoints initiate flows by sending flow descriptors over the signaling channel to the remote endpoints. P-Boxes en route consult local policy to authorize or deny the flow, and if authorized, convey policy requirements such as mandatory encryption or use of a middlebox. The endpoints and P-Box, finally, negotiate the ephemeral 5-tuple and capabilities that can be used for the flow. Policy descriptions are described in detail later on in this section.

P-Boxes generate capability tokens for each authorized flow that allows an M-Box to decide whether to let packets through. The capability token is non-forgeable and issued for a specific ephemeral flow 5-tuple, and can potentially encode an expiration time or the number of packets allowed. One example of a simple token for a flow is a one-way hash of the 5-tuple salted with the secret shared between the P-Boxes and M-Boxes for the domain. The token is conveyed to the endpoints over a secure channel, which then sign each packet with it. The signature, embedded in the packet itself, can be verified by the M-Box. One lightweight signature is a MAC (such as HMAC-MD5 [39]) computed over the packet payload, sequence number and the capability token. This signature (not the capability token itself) is embedded in each data packet or just the first packet depending on the level of security required. The M-Box, upon receiving the packet, independently reconstructs the capability token by computing the salted hash over the 5-tuple gleaned from the packet and the shared secret, and recomputes the signature from the packet contents. If the signature thusly computed matches that embedded in the packet, the packet is allowed to pass; otherwise, it is blocked and an error message is sent.

```
1:if (from.type IS client
2:      AND to.type IS server)
3:    accept()
4:end
```

Figure 4: Policy definition to allow only client-to-server flows.

Action	Meaning
accept()	Flow authorized
deny()	Flow rejected
redirect(host:port, params)	Contact given middlebox
require(enc)	Require encryption

Table 3: Core actions supported in NUTSS policy definitions

4.3 Policy and Authentication

> **Property 4.3:** Policy in NUTSS is compact and expressive. It matches high-level flow attributes and allows negotiation of flow characteristics.

Policy definitions in NUTSS target specific flows based on high-level attributes (user, application, encryption etc.) and allow flow properties to be changed before a flow is authorized. Policy definitions, inspired by the Call Processing Language (CPL) [40], are described as simple programs. Consider the policy in Figure 4 that only allows a "client" to initiate a flow with a "server" and no other flows — a policy proposed by Handley et al. in [41].

P-Boxes apply policy by interpreting the policy definition on the flow descriptor. The policy, as illustrated above, consists of two elements: "match operations" (e.g. `IS`) that match specific attributes in the endpoint or flow descriptors, and "actions" (e.g. `accept()`) that define how the flow must be handled. While NUTSS can support any number of attributes to match and actions to perform, the core attributes are listed in Tables 1 and 2, and the core actions are listed in Table 3.

> **Property 4.4:** Flow attributes are securely authenticated before policy is applied. Various trust models are supported.

The P-Box must authenticate the endpoint before it can apply match operations in policy descriptions. As defined previously, endpoint descriptors consist of a user component (alice@rr.net) and a set of attributes (app=ftp;type=server; software=vsftpd;version=0.2.4.13). The P-Box can authenticate the user by requesting the endpoint to present its certificate, or respond to a RADIUS or Kerberos challenge over the signaling channel. Authenticating the application depends on the trust model and the hardware and software support available at the endpoint. If the endpoint is equipped with trusted hardware, the P-Box can request the endpoint to provide an attestation chain rooted in trusted hardware [42] that identifies the endpoint software stack including the application. If trusted hardware is not available but the end-host operating system and the end-user can be trusted, then the P-Box can request the user to assert that the attributes in the descriptor are accurate. It does so by requesting the end-host operating system to prompt the user for confirmation independent from the application initiating the flow.

As an example of the expressiveness of policy in NUTSS, Figure 5 illustrates a policy for securely connecting to an enterprise FTP server. The policy allows only Acme Inc. users (line 4–5) to access the FTP server being run by Acme Inc. (lines 1–3). The policy further requires that only FTP clients be able to access the FTP server (lines 6–8). Additionally, if the user is accessing the server from outside the enterprise, then he must use encryption (lines 9–A). Finally, in all cases, files should be accessed through a virus checker (line B). Policy definitions such as this may be written by administrators, or to avoid bugs, be generated from high-level declarative definitions using tools similar to those existing today [43].

The policy in Figure 5 is run when an endpoint requests to contact the FTP server. Upon encountering line A, the P-Box appends the a=enc:ssl tag to the flow descriptor being processed as defined in Table 2. Upon encountering line B, the P-Box replaces the c=... parameter in the flow descriptor with c=IN IP4 clean.acme.org:8110. The P-Box also appends the necessary capability tokens to the flow descriptor that additionally encodes the ⟨uid⟩ parameter for the redirection (line B). The new flow descriptor is signaled to both endpoints, both of which subsequently establish (potentially encrypted) flows to the virus checker at the provided address. The two flows contain the same ⟨uid⟩ encoded in the capability token; this allows the virus checker to determine which two flows must be connected. The checker then proceeds with its application-level task whereby it receives FTP file data (upload or download) over one of the flows, checks the received file for the presence of any viruses, and if none are found, sends it over the other flow.

```
1:if (to.user IS root@acme.org
2:      AND to.app IS ftp
3:      AND to.type IS server)
4:  if (from.user NOT LIKE *@acme.org)
5:    deny()
6:  if (from.app NOT ftp
7:        OR from.type NOT client)
8:    deny()
9:  if (from.location NOT internal)
A:    require(ssl)
B:  redirect(clean.acme.org:8110, uid())
C:end
```

Figure 5: Policy descriptor enabling secure access to an enterprise FTP server

4.4 Resistance to Attacks

The use of certificates, attestation chains, and encryption for the signaling and (potentially) the data path thwarts many standard attacks including impersonation attacks, trojan attacks, and certain types of man-in-the-middle attacks. This section discusses how the architecture defends against new attacks introduced by the NUTSS architecture itself. Due to space constraints, we provide only a high-level description of the attacks and solutions.

Attack Model: The attacker is able to observe, intercept, arbitrarily modify, and insert any packet into the network. As a result, the attacker can send/receive packets from/to any IP address, and prevent packets from being sent/received by legitimate end-hosts. However, the attacker cannot circumvent an M-Box by creating new network links. Additionally, the attacker is not located on the M-Box or P-Box.

DoS Attacks on endpoint or P-Box: The attacker consumes network resources near the victim, or network and computational resources at the victim's P-Box. *Solution:* The victim never reveals its IP address until it trusts the remote endpoint with it; consequently, not knowing the IP address, it is hard for a casual attacker to route packets to the victim's bottleneck link. If the victim's IP address is compromised, NUTSS allows the victim to discard the ephemeral IP and pick a new one (and temporarily revoke routing advertisements for the previous one). Alternatively, the ISP upstream of the bottleneck link can deploy NUTSS to block the DoS attacker closer to the source. If the P-Box itself is under attack, then it can be replicated and distributed to multiple ISPs far away from the victim and close to the attacker, and load balanced over DNS [44] or IP anycast [45]. This allows the P-Box(es) to absorb a DoS on the signaling channel itself, while protecting the signaling channel to the victim. If the attack on the P-Box is not bandwidth related, but rather computation-load related, then the P-

Function	SIP Message
bind/listen	REGISTER
connect	INVITE
accept	200 OK Response
{get,set}sockopt	MESSAGE (sometimes)
close	BYE
send/recv/select/...	-

Table 4: Library functions in NUTSS

Box can challenge the attacker with crypto-puzzles [46] over the signaling channel. This mitigates the amplification of processing at the P-Box required to verify endpoint certificates.

Man-In-The-Middle Attacks: The attacker attempts to impersonate a P-Box to the endpoint and an endpoint to the real P-Box, thus enabling it to decrypt and observe signaling messages. *Solution:* The endpoint can differentiate between the real P-Box and fake P-Box for a domain because only the real P-Box has the secret necessary to cross the M-Box for the domain. The endpoint can exploit this asymmetry by challenging the P-Box (real or fake) to prove that it can access the external network, by retrieving a challenge response from a global public service. Care must be taken, however, that the endpoint's challenge cannot be replayed by the fake P-Box to the real P-Box.

5 Implementation

We have implemented a proof-of-concept version of the NUTSS signaling primitive and NUTSS security architecture using off-the-shelf components, and demonstrated its feasibility by running unmodified applications over it. Our implementation allows endpoints to bind to long-term stable identifiers, allows endpoints to rendezvous with each other by routing signaling messages between them irrespective of their network location, authenticate each other and request flows, allow such flow-requests to be authorized by endpoints and policy boxes designated by the intermediate domains, negotiate encryption for the flow, all while supporting mobile endpoints (where the endpoint may disconnect from the network mid-flow and subsequently reconnect at another location) without requiring any modifications to applications.

We use SIP [33] for off-path signaling, SER [47] with CPL [40] support as the policy box for domains, and iptables [1] as the firewall. We use CPLEd [43], a GUI tool, to create domain policies in CPL and upload them to the policy box. Endpoint support is implemented as a library for Linux applications. The library consists of 9K lines of C code and relies on a number of external libraries including eXosip2 [48] and OpenSSL [49]. Our library has two interfaces. The first interface offers NUTSS equivalents of the socket API including an AF_NUTSS address family, a sockaddr_ns socket-address structure that encodes the user's name, domain and application's name, socket functions listed in Table 4 that accept remote addresses in the NUTSS format, and socket options (in level *SOL_NUTSS*) for setting encryption parameters. In order to use this interface, applications must be modified accordingly.

The second interface to our library accommodates unmodified applications. The library is pre-loaded into memory (before the application is loaded) through the LD_PRELOAD functionality of the Linux dynamic loader, that allows our library to transparently hijack the libc socket calls (Table 4) in the application and redirect them to NUTSS functions. Endpoint details (user, domain, app, encryption etc.) are configured into environment variables by the end-user. The end-user also enters specially encoded hostnames into the client application (such as ftp.server.alice.domain.org.encoded.nutss), which are intercepted by the library when the application calls gethostbyname. The library decodes the name to an endpoint descriptor (alice@domain.org;app=ftp;type=server), maps it to a unique key encoded as a private IP address that it returns to the application. Subsequent calls to connect, for example, with that IP address are translated to NUTSS calls to the mapped descriptor.

Internally, both interfaces work as follows. At startup, applications bind to their descriptors, which triggers a SIP REGISTER message to the policy server; the registration process may involve authentication. When a connect is called, a SIP INVITE is created for the appropriate destination. The message is routed through the policy box that subjects it to policy. If accepted, the destination responds with a 200 OK that, in its body, includes a SDP with the address and port of a newly created TCP socket, and whether or not the flow should be encrypted. Upon receiving the response, the source initiates a TCP connection to the address/port in the SDP, and if required, secures the flow with SSL.

While SER does not by itself insert capability tokens, our SER module inserts a token as the SIP message is routed. Endpoints use external libraries *libiptc* and *libipq* [1], a firewall-interface into the kernel, to intercept the TCP packets generated by the kernel for that flow and stamp them with the token in the TCP header option field. iptables running on intermediate boxes check for this value before allowing the packet.

Aside from mobility already supported in SIP, our library allows mobility in the middle of a data-flow. If a flow breaks due to network mobility, the library can be configured to notice this and block the application. Upon rejoining the network, the library re-REGISTERs

with SIP and sends SIP re-INVITEs for the original flow. Once the SIP dialog is re-established, the endpoints exchange over it the application-level counts of the number of bytes sent/received (that the library keeps track of), re-negotiate transport addresses and resume the TCP flow. Once the new flow is established, the library re-transmits the bytes undelivered in the previous flow from a local buffer that it maintains. Configured such, the library provides transparent mid-flow mobility for streams.

We have successfully run a number of applications using our library using both the interfaces. Our library works with client-server applications written to our interface, as well as with unmodified legacy applications including ttcp, telnet, ftp, postfix and mail with varying degrees of success depending on the breath of the socket API used by them. We are currently in the process of implementing more socket API functions to fully support these and other legacy applications.

Since ours is only a proof-of-concept implementation of the NUTSS architecture, performance numbers are irrelevant as they relate only to our local environment and implementation. Nevertheless, some brief comments on performance are worth making. We found that there is little added latency in establishing connections (less than 100ms) and no change in end-to-end bandwidth (when buffering for mobility is disabled). This is because signaling in our particular setting only added one additional round-trip exchange at flow initiation (INVITE and 200 OK) and otherwise negligible additional processing on the endpoints and policy boxes during data transfer. In a real-world setting, however, complex policy, negotiations, mid-flow re-configurations etc. may make signaling more heavy-weight. Notwithstanding, we believe we have demonstrated that NUTSS is feasible today and provides a wide range of enhanced functionality to the private-server Internet.

6 Discussion

This paper represents a first stab at a comprehensive signaling-based Internet security architecture. Given the breadth of the problem, and the newness of the work, there are likely to be many specific design decisions that are modified or overturned as we expand and gain experience with our implementation. Nevertheless, we believe that the basic approach of combining off-path and on-path signaling, and of basing the off-path part on SIP, is compelling.

Although this paper focuses on security, which we believe is the single most important problem in the Internet today, the basic idea of treating IP addresses and ports as ephemeral and non-unique, and using end-to-end signaling with URIs as the means of managing 5-tuple flows has applicability to a number of Internet problems. These are briefly outlined below.

NAT Traversal: SIP is already used to traverse UDP over legacy NATs for VoIP [50, 4], and similarly could be used to traverse TCP over legacy NATs [51]. Some NAT boxes modify passing SIP messages to signal NAT address and port assignments [52]. Finally, the NUTSS P-Box could communicate with the NAT box (an M-box) to request a NAT assignment, and communicate this in SIP.

Application Anycast, Multicast, Failover: SIP can be used to give endpoints control over routing. In cases where multiple endpoints register the same endpoint descriptor, the endpoints can configure the P-box to forward the signaling message to any one endpoint (anycast), all endpoints (multicast), or select them in a primary-backup order (failover).

Auditing, Billing: Where endpoint software can be trusted (TPM [38]) endpoints can report usage through SIP. Where endpoints are less trusted, the P-Box can negotiate with the endpoints to securely use an auditing or billing M-Box for instance where each end establishes secure flows to the M-Box. Such middleboxes can then operate at the application layer, and audit/bill application functions such as audit IM logs, or bill video differently than voice with the endpoints explicit consent.

Presence, Publish Subscribe: SIP is already used to convey user presence (signing on, signing off) in SIMPLE [53], and can similarly be used for application presence. Endpoints can subscribe at a P-Box to be notified of other endpoints joining or leaving the network, and provide, for instance an application-level "answering-machine" service.

Store and Forward: Signaling can be used to provide an email-like store and forward service for applications. While SIP does not do this currently, it can be modified to allow the P-Box to accept a signaling message on behalf of an endpoint in its absence. The P-Box can later forward the message to the endpoint when it joins the network without any intervention from the source of the message.

7 Summary

In this paper, we propose NUTSS, an off-path and on-path signaling approach that takes into account both endpoint and network policy. We design an architecture for NUTSS that exploits and extends SIP, and show how it works to provide Internet security. We have implemented NUTSS and demonstrated it to be feasible and far more functional than the existing IP address/DNS/port architecture. We also explore other network functionality that our architecture makes possible including mobility, multi-homing, bridging disparate networking technologies, indirection, and billing. There is a tremendous amount of work still to be done. Fore-

most among these is to write a complete specification, implement that spec, and start gaining experience with real applications over NUTSS. Much of the process for this must include a wider community of contributors, perhaps through an IETF or IRTF working group. As always, the devil is in the details, and our partial proof-of-concept implementation notwithstanding, there are many details to be worked out. Nevertheless, we believe that the basic approach of combining off-path and on-path signaling, and of basing the off-path part on SIP, is compelling.

References

[1] Harald Welte, "The netfilter/iptables Project." [Online]. Available: http://www.netfilter.org/

[2] Microsoft, "Internet Connection Firewall." [Online]. Available: http://www.microsoft.com/windowsxp/using/networking/learnmore/icf.mspx

[3] Symantec Corp., "Norton Internet Security." [Online]. Available: http://www.symantec.com/

[4] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy, "RFC 3489: STUN – Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)," Mar. 2003.

[5] S. Guha and P. Francis, "Characterization and Measurement of TCP Traversal through NATs and Firewalls," in *Proceedings of the 2005 Internet Measurement Conference*, New Orleans, LA, Oct. 2005.

[6] J. H. Saltzer, D. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM Transactions on Computer Systems*, vol. 2, no. 4, pp. 277–288, 1984.

[7] Riverhead Networks, Inc., "DDoS Mitigation: Maintaining Business Continuity in the Face of Malicious Attacks," [Cached copy] http://nutss.net/cite/riverhead.pdf. [Online]. Available: http://www.riverhead.com

[8] BMC Software, "Marimba Product Line." [Online]. Available: http://www.marimba.com/

[9] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "Inside the slammer worm," *IEEE Security and Privacy*, vol. 1, no. 4, pp. 33–39, 2003.

[10] P. Efstathopoulos, M. Krohn, S. VanDeBogart, C. Frey, D. Ziegler, E. Kohler, D. Maziéres, F. Kaashoek, and R. Morris, "Labels and event processes in the asbestos operating system," in *Proceedings of the 20th ACM Symposium on Operating Systems Principles*, Brighton, UK, Oct. 2005.

[11] Larry Peterson, 2005, Private communications.

[12] Akamai Technologies, Inc., "Akamai: How it works." [Online]. Available: http://www.akamai.com/

[13] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker, "Controlling high bandwidth aggregates in the network," *ACM Computer Communications Review*, vol. 32, no. 3, pp. 62–73, July 2002.

[14] K. Argyraki and D. R. Cheriton, "Active internet traffic filtering: Real-time response to denial-of-service attacks," in *Proceedings of the 2005 USENIX Annual Technical Conference*, Anaheim, CA, Apr. 2005.

[15] H. Ballani, Y. Chawathe, S. Ratnasamy, T. Roscoe, and S. Shenker, "Off by Default!" in *Proceedings of the HotNets'05*, College Park, MD, Nov. 2005.

[16] A. Yaar, A. Perrig, and D. Song, "Siff: A stateless internet flow filter to mitigate ddos flooding attacks," in *IEEE Symposium on Security and Privacy*, Pittsburgh, PA, May 2004, pp. 130– 143.

[17] X. Yang, D. Wetherall, and T. Anderson, "A dos-limiting network architecture," in *Proceedings of the SIGCOMM '05*, Philadelphia, PA, Aug. 2005.

[18] A. D. Keromytis, V. Misra, and D. Rubenstein, "Sos: secure overlay services," *SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 4, pp. 61–72, 2002.

[19] D. Andersen, "Mayday: Distributed filtering for internet services," in *Proceedings of the USITS '03*, Seattle, WA, Mar. 2003.

[20] S. M. Bellovin, "A Look Back at 'Security Problems in the TCP/IP Protocol Suite'," in *Proceedings of the ACSAC'04*, Tucson, AZ, Dec. 2004.

[21] T. Aura, M. Roe, and A. Mohammed, "Experiences with host-to-host IPsec," in *Security Protocols, 13th International Workshop*, Cambridge, UK, Apr. 2005.

[22] M. Walfish, J. Stribling, M. Krohn, H. Balakrishnan, R. Morris, , and S. Shenker, "Middleboxes no longer considered harmful," in *Proceedings of the OSDI '04*, San Francisco, CA, Dec. 2004.

[23] J. Mirković, G. Prier, and P. Reiher, "Attacking ddos at the source," in *Proceedings of ICNP'02*, Paris, France, Nov. 2002.

[24] Microsoft Corporation, "UPnP – Universal Plug and Play Internet Gateway Device v1.01," Nov. 2001. [Online]. Available: http://www.upnp.org/standardizeddcps/documents/UPnP_IGD_1.0.zip

[25] M. Stiemerling, J. Quittek, and T. Taylor, "MIDCOM Protocol Semantics," June 2004, Work in progress.

[26] M. Gritter and D. R. Cheriton, "An architecture for content routing support in the internet," in *Proceedings of the USITS '01*, San Francisco, CA, Mar. 2001.

[27] P. Francis and R. Gummadi, "Ipnl: A nat-extended internet architecture," in *Proceedings of the SIGCOMM '01*, San Diego, CA, Aug. 2001.

[28] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, "Internet indirection infrastructure," in *Proceedings of the SIGCOMM '02*, Pittsburgh, PA, Aug. 2002.

[29] C. Tschudin and R. Gold, "SelNet: A Translating Underlay Network," Uppsala University, Uppsala, Sweden, Tech. Rep. 2003-020, Nov. 2001.

[30] J. Crowcroft, S. Hand, R. Mortier, T. Roscoe, and A. Warfield, "Plutarch: An argument for network pluralism," in *Proceedings of the SIGCOMM '03 Workshops*, Karlsruhe, Germany, Aug. 2003.

[31] T. S. E. Ng, I. Stoica, and H. Zhang, "A waypoint service approach to connect heterogeneous internet address spaces," in *Proceedings of USENIX Annual Technical Conference*, Monterey, CA, June 2002.

[32] J. Wroclawski, "The metanet: White paper," in *Proceedings of Workshop on Research Directions for the Next Generation Internet*, Vienna, VA, May 1997. [Online]. Available: http://www.cra.org/Policy/NGI/papers/wroklawWP

[33] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "RFC 3261: SIP Session Initiation Protocol," June 2002.

[34] M. Handley and V. Jacobson, "RFC 2327: SDP: Session Description Protocol," Apr. 2004.

[35] J. Saltzer and M. Schroeder, "The protection of information in computer systems," *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278–1308, Sept. 1975.

[36] VeriSign Inc., "Security (SSL Certificates), Communications, and Information Services." [Online]. Available: http://www.verisign.com/

[37] P. R. Zimmermann, *The official PGP user's guide*. Cambridge, MA: MIT Press, 1995.

[38] Trusted Computing Group, "TPM Specification Version 1.2." [Online]. Available: http://www.trustedcomputinggroup.org/

[39] H. Krawczyk, M. Bellare, and R. Canetti, "RFC 2104: HMAC: Keyed-Hashing for Message Authentication," Feb. 1997.

[40] J. Lennox, X. Wu, and H. Schulzrinne, "RFC 3880: Call Processing Language (CPL): A Language for User Control of Internet Telephony Services," Oct. 2004.

[41] M. Handley and A. Greenhalgh, "Steps towards a DoS-resistant internet architecture," in *Proceedings of SIGCOMM'04 Workshops*, Portland, OR, Aug. 2004.

[42] M. Gasser, A. Goldstein, C. Kaufman, and B. Lampson, "The digital distributed system security architecture," in *Proceedings of 12th NIST-NCSC National Computer Security Conference*, 1989, pp. 305–319.

[43] Fraunhofer Fokus, "CPLEd - A CPL Editor." [Online]. Available: http://www.iptel.org/products/cpled/

[44] M. J. Freedman, K. Lakshminarayanan, and D. Mazières, "OASIS: Anycast for Any Service," in *Proceedings of NSDI'06*, San Jose, CA, May 2006.

[45] H. Ballani and P. Francis, "Towards a Global IP Anycast Service," in *Proceedings of SIGCOMM'05*, Philadelphia, PA, Aug. 2005.

[46] X. Wang and M. K. Reiter, "Defending against denial-of-service attacks with puzzle auctions," in *SP '03: Proceedings of the 2003 IEEE Symposium on Security and Privacy*. Washington, DC, USA: IEEE Computer Society, 2003, p. 78.

[47] Fraunhofer Fokus, "SIP Express Router." [Online]. Available: http://www.iptel.org/ser/

[48] Antisip SARL, "The eXtended osip library." [Online]. Available: http://www.antisip.com/

[49] OpenSSL Team, "The Open Source toolkit for SSL/TLS." [Online]. Available: http://www.openssl.org/

[50] J. Rosenberg, "Internet draft: ICE – Interactive Connectivity Establishment," Feb. 2004, Work in progress.

[51] S. Guha, Y. Takeda, and P. Francis, "NUTSS: A SIP-based Approach to UDP and TCP Network Connectivity," in *Proceedings of SIGCOMM'04 Workshops*, Portland, OR, Aug. 2004, pp. 43–48.

[52] Y. Kawarasaki, T. Shibata, and T. Takahashi, "IPv4/IPv6 SIP interworking methods in dual-stack network," in *Proceedings of APCC'03*, Penang, Malaysia, Sept. 2003.

[53] J. Rosenberg, "RFC 3856: A Presence Event Package for the Session Initiation Protocol (SIP)," Aug. 2004.