Future Architectures for Middlebox Processing Services on the Internet and in the Cloud

Justine Sherry

Future Architectures for Middlebox Processing Services on the Internet and in the Cloud

by Justine Sherry

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor S. Ratnasamy
Research Advisor

(Date)

* * * * * * *

Professor S. Shenker
Second Reader

(Date)

Abstract

Middleboxes, such as caches, firewalls, and intrusion detection systems, form a vital part of network infrastructure today. Administrators deploy middleboxes in diverse scenarios from enterprise networks, to datacenters, to access networks. However, middleboxes are universally deployed under what we call the 'unilateral model', where middleboxes are deployed and configured by administrators alone, for the benefit of hosts in a single domain alone.

In this thesis, we present two new deployment models for middleboxes which offer new capabilities for middlebox usage as well as new business models for middlebox deployment. Netcalls is an extension to the Internet architecture that allows end host applications to invoke and configure middleboxes in any network their traffic traverses; for example, we present a web server that invokes inter-domain DDoS defense when it detects that it is under attack. APLOMB is a system that allows enterprise networks (as well as individual end hosts) to tunnel their traffic to and from a cloud service that applies middlebox processing to their traffic, avoiding the costly and management-intensive burden of administering middleboxes in a local network. Netcalls and APLOMB allow ISPs and cloud providers (respectively) to monetize their deployment of middleboxes by offering them as a service to third-party clients; all the while presenting new capabilities, in the case of netcalls by enabling application interaction and in the case of APLOMB by providing better scalability and easier management.

We discuss both of these proposals and their benefits in detail; we then discuss challenges and opportunities towards their deployment and adoption.

Acknowledgements

Many thanks to my wonderful collaborators and undergraduate mentees, who not only made all this work happen but made it a joy to develop. Vyas Sekar, Colin Scott, Shaddi Hasan were the APLOMB team; Daniel Kim, Amy Tang, Seshadri Mahalingam, and Steve Wang were the undergraduate team on the netcalls project.

I am greatly indebted to Arvind Krishnamurthy, who tried to get rid of me once but still puts up with me nonetheless.

Finally, my advisor Sylvia Ratnasamy is not only my guide but my hero in research; I'm looking forward to another *n* years of trying to emulate her as best as I'm capable of.

Contents

1 Introduction

Through the widespread deployment of middleboxes, networks today implement a range of advanced traffic processing functions that go beyond simply forwarding packets to inspect, transform and even store the packets they carry. That middleboxes play a central role in modern networks is, by now, widely recognized: reports put the middlebox market at $6B [25]. The wide array of capabilities provided by these devices provide valuable benefits in security and performance: *e.g.*, enterprise network administrators place middleboxes to enforce exfiltration policies, compress traffic to save on bandwidth costs, or cache content to improve page load times; administrators in cell phone networks uses middleboxes to NAT traffic and rewrite HTTP content for optimized cell performance; and administrators in datacenters use load balancers to distribute tasks across multiple servers and SSL terminators to minimize their security vulnerability footprint.

Across these diverse deployment scenarios, the deployment model for middleboxes is nonetheless similar in that network administrators deploy middleboxes strictly to provide service to devices within their administrative domain, and in that network administrators exclusively configure the policies enforced by these devices. This 'unilateral' deployment model allows a network administrator to independently implement a new network service within their own domain, without requiring upgrades or changes to end hosts or other networks outside of the administrator's own domain. The successful proliferation of middleboxes today is arguably due in some part to the simplicity of this unilateral deployment model.

However, in this thesis we argue that the unilateral deployment model – middlebox services only for local traffic with a single policy specified by a local network administrator – only scratches at surface of potential benefits from middleboxes. We present two new deployment architectures for middlebox services, one called Netcalls and the other called APLOMB. Both "break" the unilateral deployment model by allowing ISP or cloud providers to expose their middlebox processing capabilities to third party customers who route their traffic through the provider network to receive processing. As we'll illustrate, these designs (1) present new business models for the providers who expose access to their middleboxes, and (2) enable new capabilities and use cases for middleboxes.

Netcalls, presented in Chapter 2, extends Internet Service Providers to expose programmatic interfaces to the middleboxes they support, allowing clients to invoke and configure middleboxes in *any* network their traffic traverses (so long as that network supports the Netcalls extensions). Netcalls takes a broad notion of client, and allows individual end host applications to customize the middlebox processing their traffic receives. As we discuss in §2.6.3, this programmability allows applications to make new and useful use of middlebox processing; we illustrate examples of these new uses with a web server that invokes DDoS defense in the network when it detects an attack, a web client that invokes traffic compression for large downloads, and an Android service to preferentially connect to WiFi networks that support an IDS for mobile phones.

APLOMB, presented in Chapter 3, focuses on enterprise middlebox deployments. Under APLOMB, middleboxes reside in the cloud and enterprises tunnel their traffic to and from the cloud to receive processing; policy configurations are still specified by administrators alone (rather than applications, as in Netcalls). As we'll show in §3.2, moving middleboxes to the cloud can alleviate major challenges in enterprise network administration including administrative complexity, overload failures, and high costs for provisioning.

As with all architectural proposals, a key challenge to adoption is "deployability" – if the burden of upgrade is too high, or a large number of users need to adopt the upgrade before it is usable, it is unlikely that widespread adoption will get off the ground. When considered for their deployability properties, APLOMB and Netcalls can be seen at opposite ends of a spectrum. Netcalls is an expansive design that allows end host *applications* to invoke and configure middlebox services in *any* ISP their traffic traverses. Although the netcalls architecture operates gracefully under partial deployment – the state in which only some clients and some networks deploy a particular middlebox or even the netcalls architecture itself – it nonetheless is a substantial leap from the state of middlebox deployments under the unilateral model. Why should networks service providers invest in programmable middleboxes when today there exist no applications that can invoke them? Why should application developers write code to invoke middlebox APIs that are unsupported by every major network? This 'chicken and egg' problem among other deployability challenges, hinders immediate adoption of a netcalls architecture.

APLOMB, at the opposite side of the spectrum, proposes a more limited – but more deployable architecture. Rather than ISPs exporting the services of their middleboxes, APLOMB clients tunnel their traffic to the cloud to receive middlebox processing. APLOMB's target client is not application developers, but enterprise administrators – a known, existing market for a cloud provider to offer their middlebox services to. Indeed, a handful of startups today offer APLOMB-like solutions for WAN optimization [6] and intrusion detection [26].

We discuss deployability challenges for APLOMB and Netcalls in Chapter 4, where we return to our comparison of the two architectures and how APLOMB can be used to partially implement Netcalls as a step towards full deployment of a netcalls-like service.

We now turn to the architectures themselves in detail. In Chapter 2, we describe netcalls and in Chapter 3 we discuss APLOMB. In Chapter 4 we return to our deployability discussion and then conclude.

2 Netcalls: Programmable APIs to Middlebox Processing Services

2.1 Introduction

In this chapter, we present netcalls, an API by which endpoints interact with middlebox services in the network. With netcalls, networks that deploy middleboxes expose an interface to other networks announcing what middlebox services they provide, and an interface to end host applications allowing applications to specify what middlebox services they would like performed on their traffic. With netcalls, end host applications see the abstraction of a single logical network with named services that can be turned on/off, or configured.

We face two challenges in designing netcalls. The first is that, ideally, we want an API that is *general* and yet middleboxes are inherently diverse, spanning a variety of goals (*e.g.*, security, performance, interoperability), configuration requirements (*e.g.*, configuring filter rules *vs.* caching policies *vs.* cryptographic keysets) and topological requirements (*e.g.*, WAN optimizers placed at both endpoints of communication *vs.* proxies placed once near web clients). The API we design must thus walk a fine line between generality and accommodating the specialized needs of different middlebox services.

The second challenge is that the "backend" network architecture required to support the netcalls API, must meet the high standards for *deployability* set by middleboxes; to do otherwise would be to lose the key to the success of middleboxes. Operators today can unilaterally deploy and use a middlebox whether or not another network or user does so.[1] In this spirit, we aim for a design that accommodates the partial adoption of *every* mechanism we introduce. Hence, our solutions must assume that some networks will deploy a particular middlebox service while others

[1]We note that, by definition, our solutions cannot attain the same ease of deployment as middleboxes – applications that choose to use our APIs will require modification and interdomain operation fundamentally requires that two networks cooperate.

may see no incentive to do so; that an application built using our netcalls API might communicate with legacy applications that do not; and that while some networks implement the backend support for netcalls, others will not.

Approach. Netcalls addresses the above challenges through a design that decomposes service use into three distinct interactions between the user and network: service initialization, configuration and invocation.

To *initialize* a service in a general manner, netcalls introduces the concept of a placement pattern which is a logical description of *where* in the network the user wants a service initialized. On receiving a client's initialization request, the network identifies network domains that offer the requested service and satisfy the placement pattern. The selected networks then configure their internal routers to direct the client's traffic to appropriate middleboxes. Once service has been initialized, the network returns a handle to the user and internally maintains state mapping the handle to the selected service-providing domains. When the user later tries to *configure* the service, it can do so using service-specific configuration requests and this handle (plus the network state) ensures that these requests can be directed to service-providing domains that know how to interpret them. Thus initialization is completely general while service-specific configuration requests are "steered" in a general manner.

To ensure deployability, networks calls for initialization and configuration occur on the control plane, decoupled from the actual service processing of data plane traffic. As legacy traffic flows through a service-providing network, if it matches the users' initialization request it is routed via a middlebox offering the service. Thus, the data plane requires no change to packet formats or interdomain protocols and is compatible with existing router and middlebox equipment.

Why Netcalls? The traditional viewpoint [33, 54, 67] is that the transparent nature of middleboxes leads to unexpected interactions (*e.g.* end-to-end violations) and management complexity (*e.g.*, hijacking). Exposing middleboxes to endpoints would bring previously hidden behaviors into the open and ease these problems. While there is certainly merit to this argument, we propose that the greater benefit may be one of opportunity rather than remedy: a general API to middleboxes will enable new application models (for end-users and online services) and new business models (for operators).

From the perspective of endpoints, there are many scenarios where applications would benefit from such APIs. For example, we used netcalls to build a webserver whose QoS/load monitoring module initiates in-network DDoS protection, and an Android WiFi interface that preferentially connects to networks that deploy intrusion detection tailored to mobile phones. Many more such examples are possible.

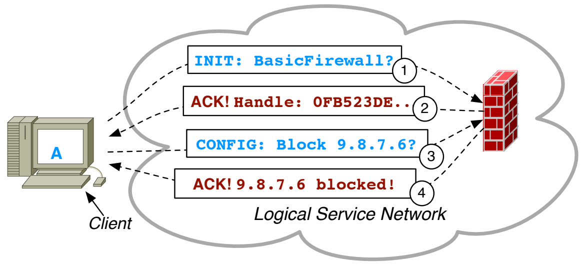


Fig. 2.1: A client initializes a firewall with netcalls.

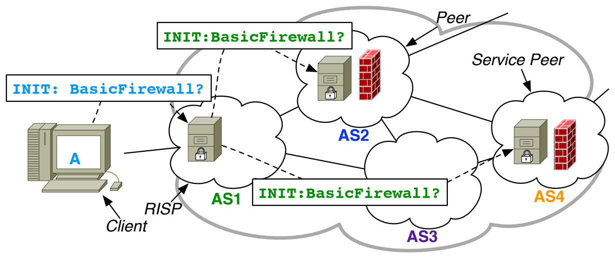


Fig. 2.2: A RISP resolves an interdomain service.

Moreover, looking forward, we anticipate a growing role for network services due to the trend towards lightweight client devices that will increasingly rely on cloud-based services. In such a world, the network's impact on user experience grows and hence so does the potential for network optimizations, *e.g.* through "opportunistic" WAN optimization between clients and servers, or network caching in cooperation with cloud services.

From the perspective of network providers, a standard and general API by which endpoints must explicitly request advanced services offers a hook around which to build new accounting and business models, potentially expanding opportunities for providers to monetize their deployment of new features.

In the rest of this paper, we describe the netcalls API and supporting architecture (§2.2-§2.5) and the implementation of netcall applications (§2.6). We then evaluate netcalls (§3.5), discuss related work (§2.8), and conclude (§2.9).

2.2 Overview

We now present an overview of the netcalls architecture and design rationale. Netcalls present application developers with a simplified abstraction of a single logical network with a capability that can be turned on, off, or configured. To application developers, the abstraction of a single logical network is appealing for its simplicity – the client only specifies what service it wants invoked. Nevertheless, implementing this abstraction is challenging because the logical network is in reality a large-scale federation of independent networks each with their own policy and deployment goals.

To start, we consider a client application requesting a firewalling service called `BasicFirewall`:

```
fw = BasicFirewall.init(...);
fw.block("9.8.7.6");
```

Figure 2.1 illustrates the interaction between client and network for the above request. First, the client requests (1,2) the network to initialize the firewall service on its incoming traffic, after which the network will direct the client's inbound traffic to traverse a firewall device. Then, in (3,4), the client requests configuration of its firewall service, adding a new rule to its access control list; the network updates the firewalling devices accordingly.

Before discussing any technical mechanisms behind this process, we assign terminology to Fig 2.1. We refer to named capabilities like BasicFirewall as *services*, identified by a *service name*. Clients use netcalls to *initialize* and *configure* services. Initialization (the first step in Fig. 2.1) refers to instructing the network to redirect the client's traffic to a device that performs the requested service. We refer to 'configuration' when clients specify service-specific customization, *e.g.* adding a rule to a firewall, specifying a cache timeout policy for a proxy, or requesting summary statistics from a traffic monitor. The configuration step is service-specific in that it would not make sense to specify a cache timeout policy to a firewall, or add a filtering rule to a traffic monitor. Hence, each service name is associated with a set of functions defined independently for each service. For BasicFirewall, `add_rule` is such a function.One final step not illustrated in Fig. 2.1 is *invocation*, when the client's packets traverse the network and trigger service processing.

Returning to our example, we now consider the network's perspective, shown in Fig. 2.2. When the client initializes BasicFirewall, it directs its request to a server hosted by the ISP (AS 1, in Fig. 2.2) that serves as its "service access provider". In our example, we assume this ISP does not implement the BasicFirewall service and hence it looks for one or more external ISP(s) that do support BasicFirewall. For this, the ISP consults some autonomous system (AS) level topology information that it has gathered, and selects two external ISPs (ASes 2 and 4) who together serve all of the client's inbound traffic. It forwards the client's initialization request to these two external ISPs, and after their acknowledgments, it stores a record of the client's request and the selected ASes and returns an acknowledgment and a *service handle* to the client. When the client requests to add a rule to the BasicFirewall, the handle allows the ISP to recall which external networks are providing service for the client, and then forward the client's request to them.

We refer to any network that deploys a service like BasicFirewall as a *service network*; in Fig. 2.2 ASes 2 and 4 are service networks. AS 1, on the other hand, serves as a *resolution ISP* or RISP; we refer to the server in the RISP's domain that the client directs its requests to as a *resolution server*. We assume that each client contracts with one or more networks to serve as its RISP. We say 'contracts with' because we expect that providers will charge for access to services. Correspondingly, when a RISP forwards a request to another service network, there must be a contractual relationship in place between the RISP and the other network. Service networks that a RISP settles with are called the RISP's *service peers.*

Design Rationale

A core goal in each of our design choices is to adapt to *partial deployment* of any mechanism we introduce.

RISPs. As described above, clients only convey netcalls to a resolution server hosted at a single network, the client's RISP. We chose this model for two reasons. First, it simplifies application logic. Consider a client invoking some service S that it requires once, on the forwarding path from itself to a destination D. Where are there devices that can perform S? How does D direct its traffic to these devices? By making these questions the responsibility of a RISP, application developers don't have to write logic for discovering where and how services are performed in the network. Second, contacting only a single RISP simplifies the process of payment/settlement. Client payments for interdomain services require a contract with a single ISP, rather than several (just as they pay for Internet service today). ISPs maintain a contract with clients (just as they do today) and *service peers*, networks who they have business relationships with for services (just as they have peering relationships for traffic exchange today).

API Design. Directing traffic through a RISP leads to a new challenge: how is the RISP able to resolve requests to services that it does not support itself? For example, in resolving a client's initialization request, how is the RISP to know that a firewall should be applied once on inbound traffic, but that a WAN optimizer should be deployed twice at the endpoints of communication? We resolve this by adding a set of parameters called a *placement pattern* to a clients initialization request: placement patterns are a general abstraction by which clients describe where in the network to place the service. We describe placement patterns in depth in §2.3.

Statefulness. When a netcalls client initializes a service, the network stores *state* related to the client's request. Keeping state in the network allows netcall clients to invoke services even on traffic from end hosts who have not adopted netcalls – this is important given the prevalence of client-server communication and that clients and servers may not share the incentive to adopt netcalls. Consider the alternative – common in most network designs [83, 69, 85] – in which packets must contain a new header describing their service demands. Embedding a new service header means the *sender* must have adopted the service; if the sender does not support the new service, it cannot be used. However, for services like BasicFirewall the receiver is the endpoint that wants service processing, not the sender. With netcalls either the sender or the receiver can request processing features; it is not necessary for both hosts to adopt netcalls for one of the hosts to initialize services independently.

Best Effort Availability. As a fundamental consequence of partial deployment, netcalls must accept a service model of *best effort service availability*. A client application may request a service that is not supported by any appropriately placed network, in which case the network will return an error informing the client that the service is unavailable. No architecture can compensate for a service the network simply doesn't offer. As a consequence, applications must be prepared to *adapt* to service unavailability. For example, in §2.6 we'll show a mobile phone that turns on extra local anti-virus when connecting to a network that *does not* support an intrusion detection system.

2.3 Client API

We now describe the netcalls API in detail. As mentioned previously, clients send their netcall requests to a *resolution server* in its RISP.While our discussion treats the resolution server as a single entity, it can be replicated using standard approaches.

The core function in the client-RISP protocol is INIT, which the client uses to initialize a service. The parameters for the INIT function are shown in Table 2.1.

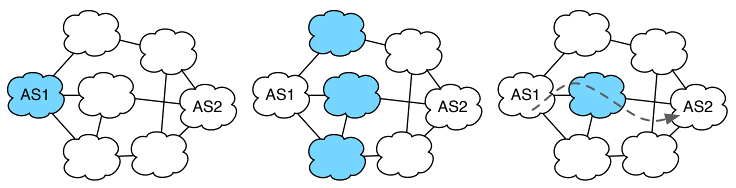
Parameter	**Parameter Options**
Service Name	Client's desired service.
Traffic Filter	Traditional 5-tuple (sender's IP or prefix, receiver's IP or prefix, sender's port, receiver's port, protocol) for which matching traffic should traverse the service. Entries may be wildcarded.
Incoming, Outgoing, or Bidirectional	Whether the service should be applied on traffic: (1) to, (2) from, (3) to-and-from the requester.
Frequency	How many times the service needs to be supported on the traffic's path. (1) once, (2) twice, (3) as many times as possible, (4) at every hop of the path.
Proximity	Whether the service is required (1) in the client's network, (2) in the remote endpoint's network, (3) near the clients network, (4) near the remote endpoint's network, or (5) in a specific ASN.
Coverage	In the case where the placement pattern cannot be fully satisfied, whether (1) partial coverage is acceptable, or (2) complete coverage is a must.

Table 2.1: Parameters for INIT requests.

The first three are the service name and two parameters expressing what subset of the client's traffic the service should apply to (*e.g.* "all outgoing traffic on Port 80 to 1.2.3.4").

The lower three parameters address a core challenge for the netcalls API: that the RISP may not know anything about the service's functionality. Without information beyond the service name, the RISP has no way to learn that a firewall should be performed once on inbound traffic, while a WAN optimizer should be applied on outbound traffic at both endpoints of communication. To express these topological requirements, the lower three parameters form a *placement pattern*, a general model that captures how and where the network should apply the service. The three properties described by these parameters are (1) *Frequency*, how many instances of the service should that traffic should traverse en route between source and destination; (2) *Proximity*, whether the service is required in the client's network, receiver's network, or somewhere in between; and (3) *Coverage*, is it acceptable if service is only available for a subset of the requested traffic (e.g., 'covering 4 of 5 incoming paths').

Using the coverage, frequency, and proximity parameters, the client can specify diverse placement patterns; Figure 2.3 illustrates three placement patterns that appeared repeatedly in the usage scenarios we considered. The 'Individual' pattern



(a) Individual (b) Perimeter (c) Path

Fig. 2.3: Common INIT request Placement Patterns.

(2.3a) allows a client to name a specific AS in which service should be implemented, defined by a frequency of once, and a proximity of 'in ASN X'. Another placement pattern, useful for many security services, is a 'Perimeter' placement (2.3b), specified by a wildcarded 'remote' network (thus requiring the service on paths to all networks), a frequency of once, a proximity of 'near the local network' and with 'partial coverage acceptable'. Firewalls, intrusion detection systems, and traffic monitoring services all might use this pattern. Finally, a 'Path' (2.3c) placement is specified with a traffic filter to a single IP address/prefix and variable frequency and proximity values – a WAN optimizer might specify a frequency of twice, at both the source and destination networks, while a bandwidth reservation service might specify a frequency of 'at every hop along the way', with no proximity parameter.

After initializing the service (as described in §2.4), the RISP returns an identifying service handle to the client. The RISP stores the service handle along with a 'service resolution record' containing data about the client's request including which external networks are performing the service; the client can later use the service handle to update or reconfigure the service it initialized using the four remaining API functions, shown in Table 2.2.

The resolution server sends an error back to the client if any request fails, either because the service is not supported by any available network or because the requested service violates the RISP's *policy*. For example, a RISP might want to restrict use of certain services – in an enterprise network which hosts a resolution server, the enterprise's server might drop all requests that attempt to manipulate firewall settings. More importantly, a RISP *must* reject all requests to manipulate traffic that the requesting client does not have authority over. We discuss the authentication process in §2.5.1.

Function	Description
INIT	Initializes a service. Described in §2.3.
CONFIG	Encapsulates a service-specific protocol (for example, `set_rule: block 9.8.7.0/24` for a firewall). On receiving a CONFIG message, the RISP maps the provided service handle to the corresponding service ASes and forwards the contents to these ASes.
KEEPALIVE	Extends the lifetime of the service, as the RISP periodically culls long-living INIT state past its expiration date.
TERM	Terminates the service.
GETCONFIG	Requests a list of all configurations (service handles, services, and parameters from INIT messages) which apply to the client's IP address or prefix.

Table 2.2: Functions from the netcalls API.

2.4 Service Networks

We now detail the requirements for networks which support the netcalls API, either as RISPs, service networks, or both. When a RISP receives a service initialization request, it performs *discovery and resolution* to decide which external ISPs can provide the requested service – we describe this in §2.4.1. We then describe the protocol by which the RISP communicates with these external ISPs (§2.4.2) and how networks implement services (§2.4.3). Finally, we discuss three extensions networks may support to improve service availability (§2.4.4).

2.4.1 Discovery and Resolution

When (the resolution server at) a RISP receives a client's INIT request, it must select which networks are *appropriate* to perform the service. Appropriate networks are those (a) with which the RISP has a contracted service peering relationship, and (b) suit the INIT placement pattern. Understanding whether a network suits a placement pattern requires that the RISP know the AS-level paths taken by traffic to and from the requesting client.

For INIT requests involving outbound traffic from the client, the RISP must know the AS-level paths from the client to remote destinations. This is trivial: the RISP's own BGP paths show which networks outbound traffic will traverse. INIT requests involving inbound traffic to the client are more complicated since the RISP must discover the *incoming* AS-level paths to the client. For this, our baseline solution is that the RISP queries its service peers to obtain the AS-level paths from the service peers to the client, as well as the addresses of sources whose traffic traverses these peer networks en route to the client (service peers can easily obtain the latter information through measurement). Given the contractual agreement already in place between the RISP and its service peers, it is reasonable to assume that service peers will share such information. We find that, for all practical placement patterns that we encountered, this is sufficient, *i.e.*, peer-provided paths together with knowledge of a RISP's physically-adjacent domains give the RISP sufficient path information to resolve requests. This is because service peers are the only (external) candidate networks at which service can be initialized and hence incoming paths not included in the set of peer-provided paths would not be useful in any case.

One could, however, devise placement patterns that could not be answered with peer-provided paths. For example, "place service at the AS two hops away from the source" – since peer-provided paths only reveal the AS-path from (potential) service networks to the client, but not from the source to these service networks, such a request could not be resolved. Ideally, to cover all potential scenarios, one might hope for a solution where all ASes share path information or a global topology information service. As a practical approximation of the latter, our implementation supplements (§2.6) peer-provided paths with path discovered by measurement [21, 19, 59]. Ultimately, if the available topology information is insufficient for the RISP to resolve the request it would respond to the user with an error; the client may try again with a different placement pattern or forego service (which all clients must be capable of given the expectation of only partial deployment).

2.4.2 Network-to-Network Protocol

Just as ISPs expose a resolution server with an API for clients, ISPs also expose a server and API to other networks. For simplicity, we refer to this inter-network server as a resolution server as well. We briefly summarize two functions from the network-to-network API here. The SVC_REQUEST function allows a requesting network to initialize, configure, and terminate services in an external network by encapsulating the client's original INIT, CONFIG, and TERM requests. The INFO_REQUEST function allows an ISP to inform other networks whether or not it supports a service, whether a service is still active, and share its AS-level paths.

2.4.3 Service Implementation

When a service-providing network accepts a new request, it arranges to support the service internally in a manner inspired by SDNs [66, 42, 61], and recent work on middebox management [74]. The resolution server pushes $\{$traffic-filter, mbox-address$\}$ mappings to the appropriate switches within the network, where 'traffic-filters' are defined in terms of the 5-tuple traffic pattern from the user's INIT request, and 'mbox-address' is the network address of the middlebox offering the service. Incoming traffic at a switch is matched against the traffic pattern entries, and traffic matching a filter is redirected to the corresponding middlebox.

2.4.4 Extensions for Improved Availability

Our basic design allows clients to invoke services in any network which their RISP peers with and which their traffic traverses. We now describe extensions which networks can optionally support to expand availability.

Multipath routing. This optimization aims to improve availability for 'Path' services in the event that the default AS path does not include an appropriate service-providing network. In such cases, the RISP may provide increased availability by considering alternate policy-compliant AS paths. We chose MIRO [84] as our multipath routing solution because it is simple, backwards-compatible with BGP and functions through bilateral agreements in a manner that does not require the participation of every AS on the path.

Remote RISPs. To expand its a customer base (and provide service access to clients whose ISPs are not netcalls-aware), an ISP may serve as a RISP for end hosts who receive connectivity service from another network. The client tunnels its outgoing traffic via the RISP, then the RISP can use the same strategy for topology and routing discovery as it does for its direct customers. This extension allows non-access networks, such as cloud providers, to serve as RISPs (as in [76]).

Service brokers. Our discussion so far has assumed a RISP only negotiates with service-providing ASes with which it has a direct agreement regarding settlements. While simple, it can be unrealistic for a RISP to maintain service peering relationships with a large number of other networks and this might lead to low availability of services. Brokers address this concern. If a RISP A wishes to invoke a service in an AS C with whom it does not (service) peer, but both A and C are peers of a third AS B, then AS B may serve as a 'broker' for the exchange between A and C. This extension is not part of the base design because to do so with proper authentication (discussed in §2.5) requires deployment of a PKI.

2.5 Securing Services

Securing the netcalls architecture requires both *control plane security* and *data plane security*.[2] Control plane security means that service initialization can only occur when requested by a party who has ownership over the impacted traffic and is able to pay for use of the service. Data plane security means that services only apply processing to non-spoofed traffic belonging to the real client. The key challenge in ensuring both of these is doing so while remaining within our stringent deployability requirements. In what follows, we present a qualitative description of our solutions to securing netcalls.

2.5.1 Control Plane Security

For an ISP to accept a request to initialize a service – whether from a client/end host or from a service peer – we must ensure that each request originates from an entity with ownership of the impacted IP address or prefix. In the absence of a public key infrastructure, netcalls rely only on the trust built in to commercial relationships between clients and their ISPs and between ISPs and their service peers.

First, we consider client to RISP authentication: an AS A accepting an INIT request from an end host C. If A is a managed domain (*e.g.* an enterprise), A may accept a normal TCP handshake with no additional credentials as sufficient authorization for C's request. However, this model of authorization assumes that man in the middle attacks are not a threat, and fails when a third party has access to C's network connection, *e.g.*, when a home user invites a neighbor over who connects their laptop to C's IP address and invokes services without the home owner's permission. To protect against these scenarios, A may also provide C with a credential allowing C to request services over TLS with both client and server authentication enabled.

Second, we consider network to network authentication: an AS A accepting a SVC_REQUEST from one of its service peers, AS B. As service peers, A and B exchange their public keys and allocated prefixes out of band. We assume that A and B are truthful in their prefix exchange because peering relationships reflect real-world trust; violation of that trust is cause for severance of the relationship and even

[2]Our aim is not to improve Internet security in general but merely to ensure that netcalls does not introduce new security problems for either legacy or netcall-adopting clients. For example, we do not resolve DDoS, although some of our envisioned services could be used to mitigate DDoS attacks (*i.e.*, firewalling services). DoS is a long-standing problem in the Internet architecture; the netcalls architecture neither worsens nor improves this state.

legal action. When B sends a SVC_REQUEST to A, both B and A authenticate each other using TLS with client-server authentication enabled. A then checks that B's request impacts traffic to or from a prefix owned by B – if not, A should reject the request, since B does not have authority over the impacted traffic. If B's request does pertain to a prefix owned by B, A accepts the request. Although A never validated the original INIT request, A trusts that B has done so, and knows that even if B failed to do so, *the only traffic that would be impacted by B's failure to perform proper authentication would belong to B's own customers*.

2.5.2 Data Plane Security

A 2009 study found that 34% [40] of networks do not prevent their users from spoofing their source address. Exploiting this, an attacker might attempt to spoof traffic in order to avoid service processing, or to receive service processing when it otherwise wouldn't.

This latter problem leads to attacks inflating service charges, analogous to attacks on cloud services. Cloud providers deal with this threat not only through technical means, but through practical pricing schemes; for example, by placing caps on volume processed before shutting down, or by providing logging mechanisms to allow customers to dispute false charges. We expect that middlebox service providers will similarly develop responsible pricing schemes, however, we present several mechanisms to address common attack scenarios below. Our goal is not to ensure that 'no packet generated by a spammer ever traverses a middlebox', but rather to restrict the magnitude of such traffic so that it cannot impact aggregate traffic (*e.g.*, 90th percentiles as commonly used for bandwidth pricing today).

Consider the following, in which an attacker A uses spoofed traffic to manipulate a client B's service:
(1) To avoid undesired processing, *e.g.* security services, A sends traffic to B appearing to originate from an address B considers benign and has no security rule for.
(2) To inflate B's volume-based service charges, A sends traffic appearing to originate from B or en route to B spoofed as someone B frequently communicates with.
(3) To steal service processing, A sends his own traffic spoofed as B to another endpoint he colludes with.

We present three techniques to address the above attacks, without requiring netcall adoption by the attacker:

Exhaustive Service. Inbound services which the sender might find undesirable (primarily security applications) should be applied to all inbound traffic with a perimeter request. Under this INIT pattern, there is no way to avoid processing. We find that this security policy parallels security policies enforced by typical enterprises: all traffic traverses firewalls, no matter the origin. This resolves attack (1).

Connection Termination. For processing services that 'terminate' a TCP session on bilateral traffic (*e.g.* WAN Optimizers or load balancers), spoofing attacks are not possible. The middlebox maintains connections to (and hence completes a handshake with) both endpoints, ensuring that neither endpoint is spoofed. This solution can resolve attacks (2) and (3), but not for processing services that service TCP bilateral sessions without terminating them, or observe only one direction of traffic.

Shared SYN Cookies. For general TCP security, we introduce Shared SYN Cookies, which leverage TCP SYN cookies to validate flows. Traditional SYN cookies resolve "SYN Flood" attacks by allowing a server to identify ACK packets for flows it previously SYN/ACKed, without having to keep state for the incomplete handshake. The server sends SYN/ACK replies to SYN requests with specially crafted sequence numbers; each sequence number includes a coarse timestamp, an MSS value, and a cryptographic hash of the connection's IP addresses, port numbers, and the timestamp. When it later receives an ACK in reply to its SYN/ACK, it inspects the (Seq. Number - 1) value in the ACK and uses the hash to validate that it represents a valid connection.

Shared SYN Cookies are generated using the same technique as normal TCP SYN Cookies, but the client shares its key for generating the SYN cookie hash with the middleboxes processing its traffic. Then, the client generates sequence numbers for all of its connections (both those it initiates and those initiated by another client) using SYN cookies. This allows the middlebox to validate even unilateral traffic in a TCP session. Consider a connection between our client B and another host, C. For traffic that appears to come from B, the middlebox inspects the SYN or SYN/ACK where B announces its initial sequence number and validates that the hash value originated with B. For traffic that appears to come from C, the middlebox inspects the SYN/ACK or ACK where C replies to B's initial sequence number, and similarly validates the hash. After this, it can establish an entry in its flow table for the session. Shared SYN Cookies safeguard against attacks (2) and (3).

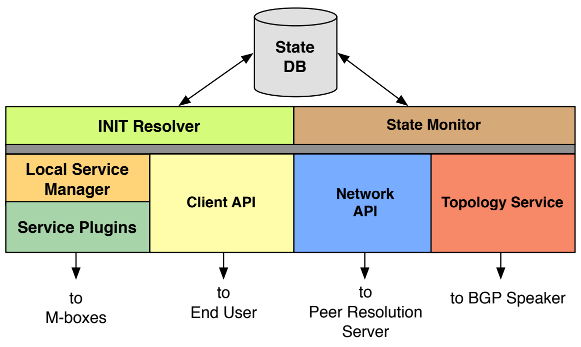


Fig. 2.4: Software Architecture of Resolution Server

2.6 Implementation

We prototyped the system components of the netcall architecture and three applications that use network services. We now describe: our prototype *resolution server* (§2.6.1), the client-side libraries that implement the netcall API (§2.6.2) and our three prototype applications (§2.6.3).

2.6.1 Resolution Server

Every network that exposes a netcall API deploys a resolution server. Our resolution server is a 4200-line Java web server exposing an RPC interface to end host clients and to other resolution servers.We illustrate the software architecture of the resolution server in Fig. 2.4.

The *State Database* stores: (i) a list of the clients for which the network serves as RISP, (ii) a list of the network's service peers, with the IP addresses of their resolution servers and the names of the services they offer, and (iii) service resolution records (SRRs) for each client request that the network is currently servicing. The *Topology Service* maintains a model of the AS graph and routes which it obtains by combining local BGP routes, the network's BGP peers, and INFO_REQUEST responses from other networks. The *Client API* and *Network API* modules implement the client-to-RISP and RISP-to-network protocols. The *State Monitoring* module monitors BGP updates for route changes that impact SRRs in the state database and updates these appropriately. If a BGP update disables a service by routing away from a service-performing AS, the state monitor sends an error to the client.

The *Local Service Manager* and *Service Plugins* are used to configure the local network to provide traffic-processing services. Our implementation currently assumes services are implemented by middleboxes and these middleboxes are deployed adjacent to a switch at a network choke point. For our prototype deployments, we use Click-based [60] software switches and software middleboxes that we run on general-purpose servers. The Local Service Manager maintains a Service Plugin for each service supported by the network. Each plugin implements a standard interface with functions to call for INIT, CONFIG, and KILL requests and communicates with the switches and middleboxes as appropriate. E.g., on receiving an INIT request, the plugin for the requested service updates the switch to divert traffic matching the INIT request to the middlebox in question. On receiving a CONFIG request, plugins either forward CONFIG requests to their associated middlebox, or they implement some functionality at the Resolution Server itself. We provide examples of different plugins for our prototype applications later in this section.

2.6.2 Netcall Clients

We implemented the netcall protocol over XMLRPC, since it is simple to program against in many languages. The raw RPC interface is not exposed directly to application programmers. Instead, application developers simply import a library and use service-specific functions like `initialize_firewall()` or `filter_ip_address(ip)`; the library then generates the appropriate INIT and CONFIG requests. In Figure 2.5 we show an example of the code implementing the library (top), and the code implemented by the application developer (bottom). As a consequence of these libraries, extending an existing codebase to leverage network services is relatively simple as we illustrate with the applications we describe later in this section.

Our implemented APIs allow any application to initialize any service for its host – even for ports bound other applications. While the Resolution Server stops malicious end hosts from interfering with the traffic of other end hosts, it cannot protect against malicious processes on the same end host. Longer term, we expect that OS support for the protocol will centralize netcall requests through a single, privileged process. Applications, rather than communicating directly with the resolution server, will instead submit their INIT request to the OS that will either reject the request (based on local policies) or forward it to the resolution server.

2.6.3 Applications and Services

We modified three applications to use network services: (i) an Apache webserver that initializes in-network filtering services for overload protection, (ii) a web client

```
import org.netcalls.API.*;
public class CompressSvc{
    ...
    InitPathRequest p = new InitPathRequest(
        localIP, dstIP, localPort, dstPort,
        COMPRESSION_SVC_ID,
        PathParams.BIDIRECTIONAL,
        PathParams.ENDTOEND, ...
    );
    ServiceToken t = ServiceManager.init(p);
    ... //configure service
    return t;
}
```

```
import org.netcalls.API.*;
import org.netcalls.CompressSvc.*;
...
ServiceToken t = CompressSvc.init(ip, port);
... //send data
sock.close();
ServiceManager.kill(t);
```

Fig. 2.5: Application developers program against standardized libraries to initialize services.

that initializes in-network compression services ("WAN optimizers") for reduced bandwidth consumption and latency and, (iii) an Android OS WiFi management interface that initializes in-network IDS for malware protection. In implementing these, we aimed to answer two main questions. First, do the netcall abstractions allow clients to express requests for a broad range of services? Second, are such services useful to end applications? We believe our experience with these services answers both in the affirmative.

DDoS Defense. We developed a webserver that initializes firewall services in remote networks when overloaded. We extended an existing firewall implementation to allow clients to add new rules with a CONFIG request. Our netcall client is an Apache webserver that we modified to use netcalls. For this, we extend the existing `mod_qos` [13] module in Apache, which detects overload and restricts access to the webserver based on usage patterns, redirecting troubling hosts/prefixes to a 'service overloaded' webitializes. We augmented `mod_qos` in 97 LOC to initialize firewalling close to the offending hosts' networks to prevent DDoS. Our integration with `mod_qos` demonstrates how application-layer context can beneficially inform network behavior.

Ideally, we would test our application by deploying resolution servers and services at every AS on the Internet; since we cannot do this in practice, we instead leverage EC2 as follows. We install our modified webserver along with its local firewall and resolution server in our local testbed. We then emulate the wide-area Internet topology over EC2 by having each EC2 node serve as a 'surrogate AS', installing software switches, firewalls and resolution servers at each. We then deployed web clients to act as attackers on 20 EC2 nodes. Figure 2.6 shows a time sequence of aggregate malicious traffic sent, malicious traffic at the switch in the web server's location, and malicious traffic reaching the webserver itself [3]. At 25 seconds, the now-overloaded web server initializes the firewall. At time 45 seconds, the web server initializes firewalls in remote networks. We see that neither the webserver nor its local firewall is ever overloaded, as firewalls deeper in the network drop malicious traffic before it reaches the web server's site.

Traffic Compression. Enterprise networks today deploy WAN optimization appliances that compress traffic to minimize bandwidth costs. WAN optimization requires that both the sending and receiving networks deploy appliances that compress/decompress traffic and hence, today, such compression is typically limited to

[3]Obviously the request rates shown are not enough to overload a web server: we artificially set the bandwidth cap in `mod_qos` to be very low to avoid flagging the attention of network administrators!

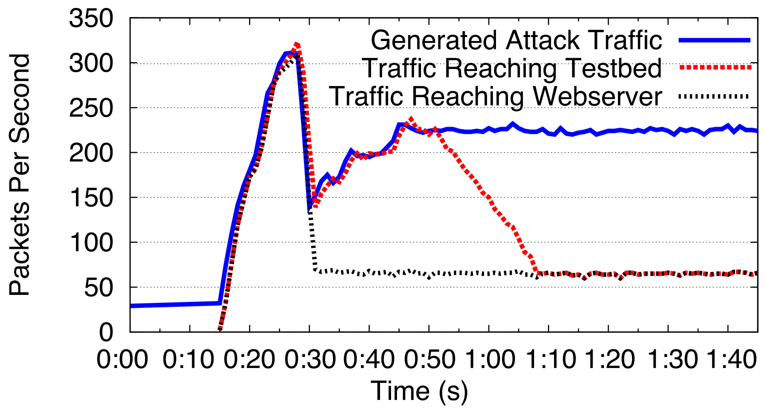
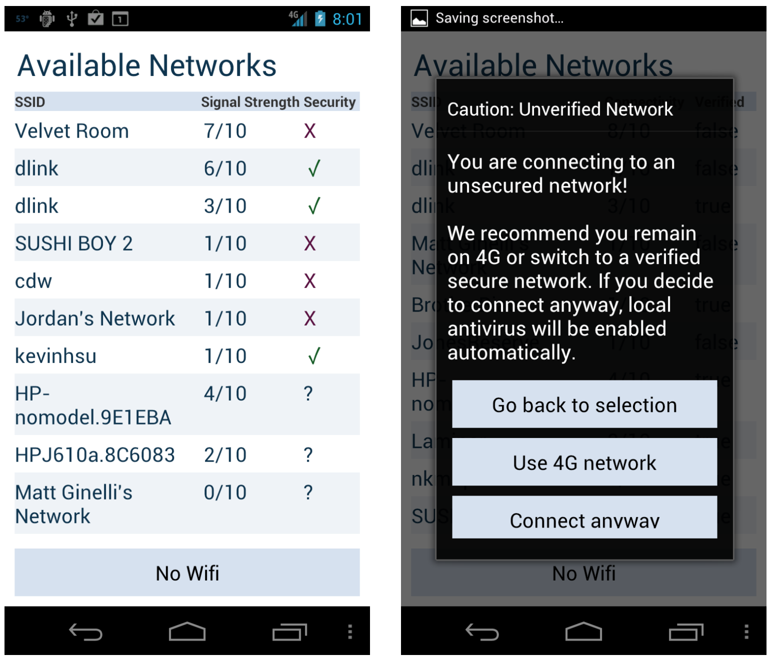


Fig. 2.6: The webserver reacts to a DDoS attack by invoking network firewalls.

communication between the different sites of a single enterprise. With netcalls' interdomain discovery capability, clients can instead initialize WAN optimization services to any destination. We modified an existing command-line web client to request compression whenever the user flags a particular request as a 'large file' download – *e.g.* downloading ISOs, video streams, *etc.* This change required only 25 LOC. We investigated the benefits of invoking compression persistently to commonly contacted destinations. Running a traffic trace we obtained from a large enterprise[76] through our testbed, we found that, for this enterprise, enabling WAN optimization to and from the ten most commonly contacted external ASes reduced the enterprise's *total* bandwidth utilization by 21%. Invoking compression to and from 100% of external networks would reduce bandwidth by 27% – hence, even partial deployment can provide substantial benefits for this service.

Android Security. Cell provider data networks protect smartphones by filtering malicious traffic, but typical WiFi networks offer no such protection. We designed a netcall-enabled IDS targeted towards Android smartphones, and modified the Android WiFi interface to preferentially connect to networks that deployed this service. Smartphones who invoke the service not only receive traditional firewalling and IDS, but a set of Android-specific filters for real malware [5].

Upon connecting to a new WiFi network the smartphone attempts to enable security features. During this process, the security application maintains network connectivity but bans all other applications from connecting to the network.To establish trust between the client and service, the service interface includes a `validate` function, under which the service instance must present a certificate signed by a


Available Networks

SSID	Signal Strength	Security
Velvet Room	7/10	X
dlink	6/10	√
dlink	3/10	√
SUSHI BOY 2	1/10	X
cdw	1/10	X
Jordan's Network	1/10	X
kevinhsu	1/10	√
HP-nomodel.9E1EBA	4/10	?
HPJ610a.8C6083	2/10	?
Matt Ginelli's Network	0/10	?

No Wifi

Caution: Unverified Network

You are connecting to an unsecured network!

We recommend you remain on 4G or switch to a verified secure network. If you decide to connect anyway, local antivirus will be enabled automatically.

Fig. 2.7: An end host security suite on the Android device warns users when a security service is unavailable.

trusted authority. We imagine trusted authorities to be either the client's service provider (*e.g.* T-Mobile, AT&T) or a dedicated security provider (*e.g.* Norton, McAffee). If the security service is unavailable, the WiFi interface (shown in Fig. 2.7) prompts the user asking whether to remain on 4G secure service from their provider, connect to the insecure network and launch a local anti-virus, or try to find another hotspot which does provide security features.

To test our security service with real malware, we create a sandboxed deployment with just our Android client, the Android IDS, a resolution server, and a sandbox server that spoofs traffic from the Internet to the Android client and deny malware any access to the public Internet. For our Mobile IDS, we created 170 new rules to detect 23 classes of malware; when running malware software within our testbed, our rules caught 11 out of 11 malware attacks we deployed in our sandbox. It is possible for the Android phone to perform filtering on its own, obviating the need for any in network functionality at all; however, offloading this filtering work to the network saves battery life. In our experiments with the Android device, continuous downloads while running a local antivirus drained the battery in

5 hours and 45 minutes; with the antivirus disabled and offloaded to the network, the battery life for the same workload lasted 8 hours 30 minutes – a 47% increase in battery lifespan. Our modifications to the Android WiFi service to check service availability, warn the user and startup antivirus included 124 LOC.

2.7 Evaluation

We now evaluate our netcalls design for request latency, control plane scalability, service stability, and service availability.

2.7.1 Latency

Setup latency impacts how service designers can make use of INIT Requests - if the setup latency is high relative to the duration of their intended use, they are unlikely to invoke service processing.

To measure expected latency values, we used a Resolution Server deployment on PlanetLab, having each of 138 PlanetLab nodes serve as surrogate servers for 33,508 ASNs. We assigned each AS a surrogate server with the following algorithm: (1) we assigned the networks that hosted a PlanetLab node the node that they hosted; (2) we assigned networks for which we had router IP addresses [9] the node with the lowest RTT to their address space; (3) we assigned each remaining network the node serving a majority of its peers. Complicating server assignment is the fact that a single resolution server may surrogate for multiple networks; to avoid co-located networks querying each other, we assigned each network a secondary server as well. We then had clients at each site query their local resolution server with INIT requests of each type of placement pattern, such that none of the requests resolved to the local network. We measured the end-to-end latency of each request from the client. Fig. 3.8 shows the observed latencies for each type of query.

How long does it take to perform an INIT request across the wide area? In the SINGLE case, the median query took 334 milliseconds, and the 90th percentile took 906 milliseconds. These values are closely followed by the PERIM and PATH cases, with median query times of 347 and 374 ms respectively. Overall, the wide area latency (from Resolution Server to Resolution Server) dominates the setup latency - end to end latency for a single request requires slightly more than 2 RTTs.

At what granularity can application developers invoke services without suffering a serious performance penalty? A service setup time of 2-3 round-trip times is

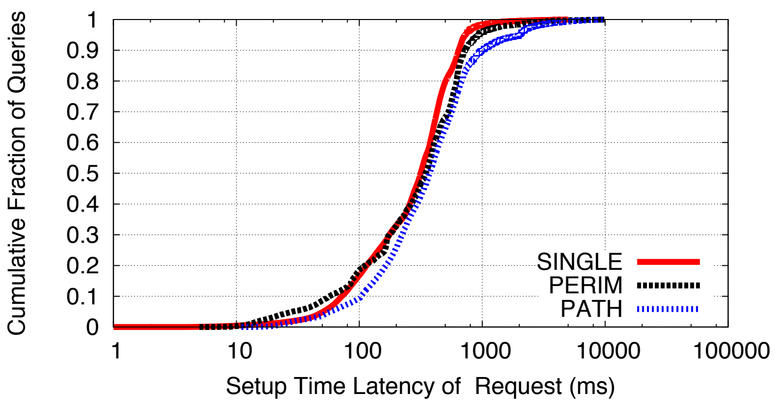


Fig. 2.8: Setup time (ms) to establish services.

negligible overhead for setting up persistent services (*e.g.* firewalls), services for long-lived connections (*e.g.* acceleration for a large file transfer), or frequently used services (*e.g.* invoking a proxy on web browser startup and then proxying all requests to Google). However, for short-lived flows of only a few round trip times, the setup penalty will noticeably impact performance; thus netcalls are not appropriate for this use case.

2.7.2 Scalability

In this section, we consider the scalability of the netcalls API and whether or not it is feasible for a network to handle INIT requests for a large number of clients. It is impossible to evaluate what we expect to be 'typical' usage patterns before netcalls are deployed; hence we focus here only on upper bounds for the number of INIT requests per second and storage requirements for SRRs, the state kept at the network mapping service service handle to the set of networks performing services for the client. We derive our bounds from a network dataset from a very large enterprise of over 100,000 hosts.

How large are the state requirements for a RISP's resolution server(s)? As an upper bound on state, we consider a model where every host initializes a service for every connection it participates in. As mentioned in the previous section, we expect typical service use to be much less frequent; none of the services we designed depend on per-connection service initialization. In a trace from a week's worth of connections in the large enterprise, at an average point in time there were 108,430 active connections; this would lead to 36.2 MB of SRR state. The peak number

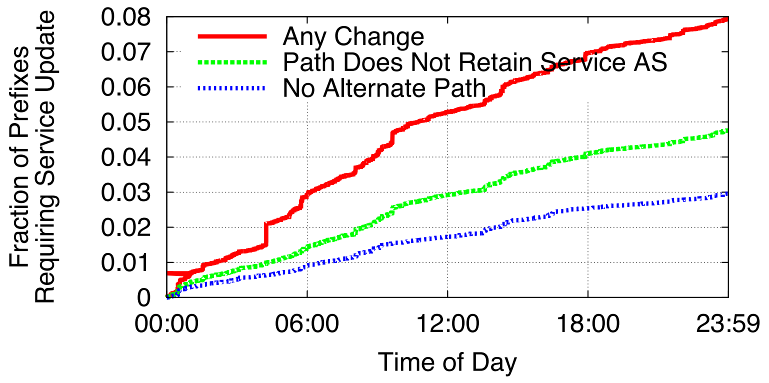


Fig. 2.9: Fraction of AS paths over time experiencing service failure.

of active connections over the course of the week was 240,836; this would lead to 80.4MB of SRR state: a trivial amount of data to store for 100,000 clients.

How many requests per second must a RISP's resolution server(s) be able to handle? Looking at connection load, we once again look for an upper bound, assuming that clients send an INIT request for every TCP session they start (even though INIT requests in practice are likely to be much less frequent). Given an average of 108,430 active connections and an average TCP session length of 60 sec, the average rate of requests would be around 1,800 connections/sec. At peak hours, with an average number of active connections at 270,836, the rate of requests would be about 4,000 connections/sec.

This request load would be easily accommodated by a small number of servers ($<$10), particularly in light of recent results on scaling connection processing [68, 53]. Nevertheless, ISPs can set policies for the number of requests they will accept per client or the granularity at which they will allow INIT requests, hence, ISPs can to some extent control the amount of state and requests they accept.

2.7.3 Stability

If traffic for a particular service is rerouted away from its service-performing AS due to BGP updates, the client must re-initialize the service. To investigate how often such a scenario occurs, we used routing update logs from the RouteViews [19] project. We created an imaginary AS as a customer of ASes 7018 (AT&T), 3356 (Level 3), and 31500 (a small ISP) and constructed its routing table using default shortest AS path preferences. Then, for each prefix, we selected an AS randomly and labeled it a ‘service-performing network’. Starting at Midnight, January 10, 2010, we monitored updates to the imaginary ASes routing table and monitored when updates routed away from service-performing ASes.

How often does a path change remove a service performing path from the route to a prefix? In Figure 2.9, we show the cumulative fraction of paths which have experienced change over 24 hours. While 8% of paths experience change within 24 hours, only 5% of paths experience change that removes its service-performing AS from the path. Within the first hour, less than half a percent of paths lose their service AS. If the imaginary AS prefers paths that route through the service AS over shortest paths only 3% of paths experience change that loses the service AS; even if one service provider withdraws a path through the service-providing AS, another provider’s path may still traverse the same AS.

What fraction of connections will experience service interruption mid-flow? Service interruption only impacts the client if it occurs during a flow on a path in use. Typical connections are short, and to popular prefixes (which studies have shown to have relatively stable paths [71]). To capture the impact of path instability on connections, we combined our BGP trace with a real enterprise trace, assuming that the enterprise was a customer of our imaginary AS and that the traces occurred over the same week long period. Over this period, the hosts in the enterprise contacted over 200 ASes, but we observed only one connection that would have been disrupted by a BGP update. This leads to to believe that from the clients perspective, BGP service interruptions will be negligible.

2.7.4 Availability

The netcalls architecture expands the reach of deployed services by allowing clients to invoke interdomain services. We evaluated this benefit in simulation, using an AS-level routing and topology simulator modeled after those used by other groups [55, 63, 84]. Our simulator modeled the AS graph from January 20, 2010, with 33,508 AS nodes and their peering relationships [8, 19]. For each simulation, we randomly annotated a fraction of the ASes as ‘service-adopting’ ASes, indicating that the AS supported the requested service; we biased this annotation such that large ASes were twice as likely as the average AS to support the service. We then randomly selected 10k random (Source, Destination) AS pairs and and checked whether the forwarding path(s)[4] between them contained ‘service’ ASes

[4]If the source AS was multihomed, we checked all its paths.

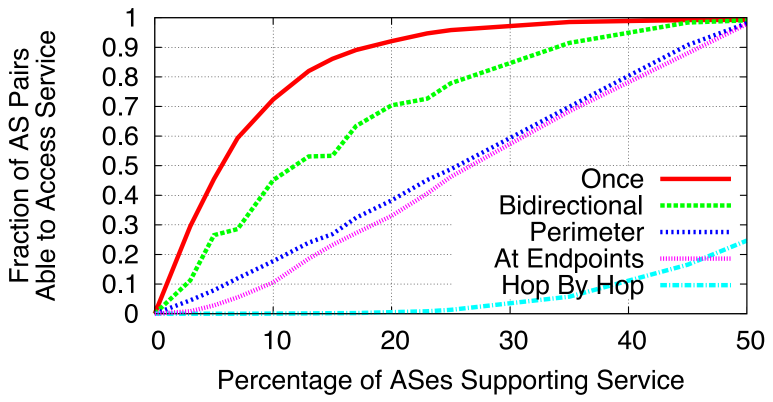


Fig. 2.10: Fraction of AS pairs able to access service on path between them for various INIT requests.

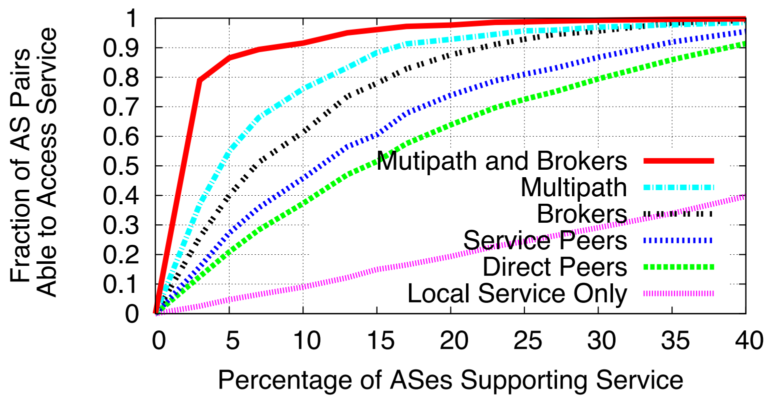


Fig. 2.11: Fraction of AS pairs able to access service on path between them with and without extensions.

appropriate to fulfill a service request under a number of constraints.

How often is a service available between a random pair of ASes? In Figure 2.10, we show the fraction of AS pairs where the networks on the default path between them support the service: the y-axis shows the fraction of AS pairs with a path that provides access to the service, and the x-axis shows the fraction of ASes deploying the desired service interface. About 70% of AS pairs have at least one service-performing AS on a path between them, even when only 10% of networks adopt the service: a $7\times$ improvement in service availability from extending service invocation across interdomain boundaries. As could be expected, service availability drops for more demanding service types. However, all service types except for ‘Hop by Hop’ services (at every AS along the path) are available for almost 100% of AS pairs once the service is deployed in only 50% of networks.

When the requesting AS can only invoke services with its service peers, how often is a service available? Because we expect networks to only invoke services with peers, just because a service is deployed doesn’t mean that a network will be able to invoke the service (and hence Fig. 2.10 is an overestimate). In Figure 2.11, we show how often a service is available once along the forwarding path between source and destination, at an AS which the source AS peers with. When the requesting AS can only invoke services with its physical (Direct) peers, service availability is roughly 35% when 10% of ASes deploy the service. When we assigned each requesting AS 5 additional ‘service peers’ (selected from Tier-1 ISPs), availability improved such that about 45% of pairs could invoke the service at 10% deployment. Hence, the ability to invoke interdomain services improves service availability even when the requesting network can only invoke services in a restricted number of external networks.

How do the multipath and brokering extensions proposed in §*2.4.4 improve availability?* Returning to Figure 2.11: our proposed extensions dramatically increase availability. With both extensions in use at 10% deployment, almost 90% of AS pairs had access to the service, providing almost universal availability with only very limited service deployment.

2.8 Related Work

Having sketched netcalls’ goals and approach, we briefly contrast our work with related efforts.

Network services. Research has pursued the general vision for in-network services for decades now with several pioneering proposals for specific new services [65, 44, 41]. We focus on the design of a general API rather than a specific service. In this sense, we view our efforts as complementary to this prior work.

Perhaps closer to our goals, is prior work on architectural support for network services more generally [83, 69, 33]. These prior efforts were rooted in the assumption that supporting rich in-network processing required a fundamentally different architecture and hence designed solutions to *replace* the IP service model. In contrast, our design efforts lead us to believe that our goals can be well achieved by *augmenting* the existing service model and see no need to replace IP. A further distinction relative to Active Networks is our more constrained model of network services wherein operators pre-install advanced functionality and expose to clients the ability to *invoke* (but not define!) these functions.

Middlebox services today. Some ISPs already expose services to immediate customers [38]; the IETF MidCom group [77] explores standards for communication with *local* middleboxes. Our proposal complements these, targeting clients at scale across network domains in a general manner. Traffic processing is also available through overlay or cloud services [1, 6].

Middlebox-centric network architectures. Seminal proposals on integrating middleboxes into the broader architecture focused on the naming implications of middleboxes, proposing that clients explicitly address the specific middleboxes for their traffic to traverse [82, 36, 78]. These proposals resolve the tension of applying unsolicited functionality to clients' traffic. But, they leave unresolved how clients discover these middleboxes, how clients reason about which middleboxes to select given routing and topology conditions and the role of network providers and their policies in offering and managing middlebox-based services. netcalls tackles the above unresolved questions and, in so doing, proposes a different approach. To reduce complexity for end clients and provide network operators with a stake in service selection, we argue that the appropriate abstraction is instead to have clients name the functionality itself and leave the network to resolve how and where it is performed.

Typed Networking [67], recognizing that hosts may want to avoid certain processing, envisions a 'negotiation' between middleboxes and hosts where boxes on the forwarding path signal the client, that can then opt out of processing. They do not consider an opt-in capability and hence issues of service discovery and availability.

Network evolution. Recent efforts [66] define open APIs between switches and operators within a single domain; they do not discuss the APIs a network exposes externally–to end clients and to other networks. Our work likewise complements recent research on programmable routers [46, 52, 64] by showing how the capabilities they enable can be exposed to clients.

Service Discovery is a common component of many systems. Most of these however operate in contexts, with goals or technology different from ours; *e.g.* targeting ISP-assisted application-layer service composition [70], wide-area discovery using IP multicast [30], using new naming infrastructures [81], *etc.*

2.9 Conclusion

We presented netcalls, an API by which client applications invoke advanced processing functions from the network. We presented three end host applications that invoke netcalls to defend against DDoS, compress high-bandwidth connections, and secure against malware.

We do not by any means expect that netcalls is the final say in discussion of how to best integrate advanced network processing into the network architecture. However, we believe our contribution - a vision for high-level, programmable interfaces that provide access to federated services across the entire Internet - moves the space forward towards a practical design that is easy to use from the perspective of application developers, while providing network providers a stake in selection, deployment, and profit from advanced services.

3 APLOMB: Enterprise Middlebox Services in the Cloud

3.1 Introduction

Today's enterprise networks rely on a wide spectrum of specialized appliances or *middleboxes*. Trends such as the proliferation of smartphones and wireless video are set to further expand the range of middlebox applications [25]. Middleboxes offer valuable benefits, such as improved security (*e.g.*, firewalls and intrusion detection systems), improved performance (*e.g.*, proxies) and reduced bandwidth costs (*e.g.*, WAN optimizers). However, as we show in §3.2, middleboxes come with high infrastructure and management costs, which result from their complex and specialized processing, variations in management tools across devices and vendors, and the need to consider policy interactions between these appliance and other network infrastructure.

The above shortcomings mirror the concerns that motivated enterprises to transition their in-house IT infrastructures to managed cloud services. Inspired by this trend, we ask whether the promised benefits of cloud computing—reduced expenditure for infrastructure, personnel and management, pay-by-use, the flexibility to try new services without sunk costs, *etc.*—can be brought to middlebox infrastructure. Beyond improving the status quo, cloud-based middlebox services would also make the security and performance benefits of middleboxes available to users such as small businesses and home and mobile users who cannot otherwise afford the associated costs and complexity.

In this chapter, we discuss APLOMB, an architecture that enables outsourcing the processing of their traffic to third-party *middlebox service providers* running in the cloud. APLOMB is a less drastic change to the middlebox status quo than netcalls, yet it still represents a significant change to enterprise networks. We validate that this exercise is worthwhile by examining what kind of a burden middleboxes impose on enterprises. The research literature, however, offers surprisingly few real-world studies; the closest study presents anecdotal evidence from a single large enterprise [75]. We thus start with a study of 57 enterprise networks, aimed at understanding (1) the nature of real-world middlebox deployments (*e.g.*, types and numbers of middleboxes), (2) "pain points" for network administrators, and (3) failure modes. Our study reveals that middleboxes do impose significant infrastructure and management overhead across a spectrum of enterprise networks and that the typical number of middleboxes in an enterprise is comparable to its traditional L2/L3 infrastructure!

Our study establishes the costs associated with middlebox deployments and the potential benefits of outsourcing them. We then examine different options for architecting cloud-based middlebox services. To be viable, such an architecture must meet three challenges:

(1) Functional equivalence. A cloud-based middlebox must offer functionality and semantics equivalent to that of an on-site middlebox – *i.e.*, a firewall must drop packets correctly, an intrusion detection system (IDS) must trigger identical alarms, *etc.* In contrast to traditional endpoint applications, this is challenging because middlebox functionality may be topology dependent. For example, traffic compression must be implemented *before* traffic leaves the enterprise access link, and an IDS that requires stateful processing must see *all* packets in *both* directions of a flow. Today, these requirements are met by deliberately placing middleboxes 'on path' at network choke points within the enterprise – options that are not readily available in a cloud-based architecture. As we shall see, these topological constraints complicate our ability to outsource middlebox processing.

(2) Low complexity at the enterprise. As we shall see, an outsourced middlebox architecture still requires some supporting functionality at the enterprise. We aim for a cloud-based middlebox architecture that minimizes the complexity of this enterprise-side functionality: failing to do so would detract from our motivation for outsourcing in the first place.

(3) Low performance overhead. Middleboxes today are located on the direct path between two communicating endpoints. Under our proposed architecture, traffic is instead sent on a detour through the cloud leading to a potential increase in packet latency and bandwidth consumption. We aim for system designs that minimize this performance penalty.

We explore points in a design space defined by three dimensions: the redirection options available to enterprises, the footprint of the cloud provider, and the complexity of the outsourcing mechanism. We find that all options have natural tradeoffs across the above requirements and settle on a design that we argue is the sweet spot in this design space, which we term APLOMB, the Appliance for Outsourcing Middleboxes. We implement APLOMB and evaluate our system on EC2 using real end-user traffic and an analysis of traffic traces from a large enterprise network. In our enterprise evaluation, APLOMB imposes an average latency increase of only 1 ms and a median bandwidth inflation of 3.8%.

To summarize, our key contributions are:

- A study of costs and concerns in 57 real-world middlebox deployments, across a range of enterprise scenarios.
- A systematic exploration of the requirements and design space for outsourcing middleboxes.
- The design, implementation, and evaluation of the APLOMB architecture.
- A case study of how our system would impact the middlebox deployment of a large enterprise.

A core question in network design is where network functionality should be embedded. A wealth of research has explored this question for various network functionality, such as endpoints *vs.* routers for congestion control [29, 58, 48] and on-path routers *vs.* off-path controllers for routing control plane functions [47, 66]. Our work follows in this vein: the functionality we focus on is advanced traffic processing (an increasingly important piece of the network *data* plane) and we weigh the relative benefits of embedding such processing in the cloud *vs.* on-path middleboxes, under the conjecture that the advent of cloud computing offers new, perhaps better, options for supporting middlebox functionality.

Roadmap: We present our study of enterprise middlebox deployments in §3.2. In §3.3 we explore the design space for outsourcing middleboxes; we present the design and evaluation of the APLOMB architecture in §3.4 and §3.5 respectively. We discuss outstanding issues in §3.6 and related work in §3.7 before concluding in §3.8.

3.2 Middleboxes Today

Before discussing outsourcing designs, we draw on two datasets to discuss typical middlebox deployments in enterprise networks and why their challenges might be solved by the cloud. We conducted a survey of 57 enterprise network administrators, including the number of middleboxes deployed, personnel dedicated to them, and challenges faced in administering them. To the best of our knowledge, this is the first large-scale survey of middlebox deployments in the research community. Our dataset includes 19 small (fewer than 1k hosts) networks, 18 medium (1k-10k

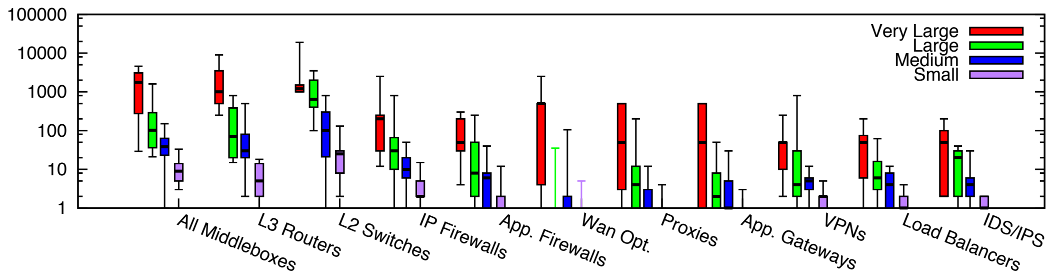


Fig. 3.1: Box plot of middlebox deployments for small (fewer than 1k hosts), medium (1k-10k hosts), large (10k-100k hosts), and very large (more than 100k hosts) enterprise networks. Y-axis is in log scale.

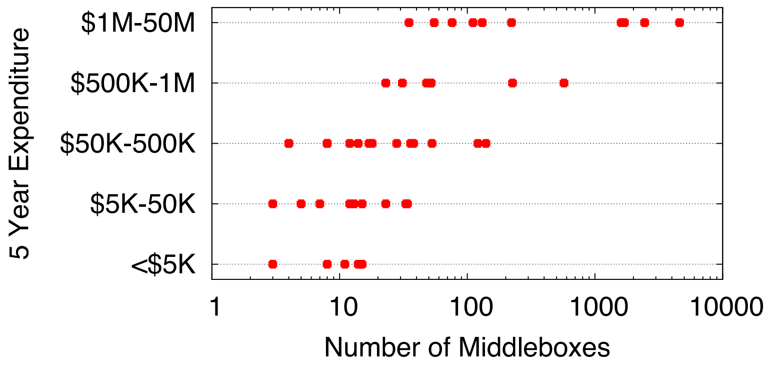


Fig. 3.2: Administrator-estimated spending on middlebox hardware per network.

hosts) networks, 11 large (10k-100k hosts) networks, and 7 very large (more than 100k hosts) networks.

We augment our analysis with measurements from a large enterprise with approximately 600 middleboxes and tens of international sites; we elaborate on this dataset in §3.5.3.

Our analysis highlights several key challenges that enterprise administrators face with middlebox deployments: large deployments with high capital expenses and operating costs (§3.2.1), complex management requirements (§3.2.2), and the need for overprovisioning to react to failure and overload scenarios (§3.2.3). We argue these factors parallel common arguments for cloud computation, and thus make middleboxes good candidates for the cloud.

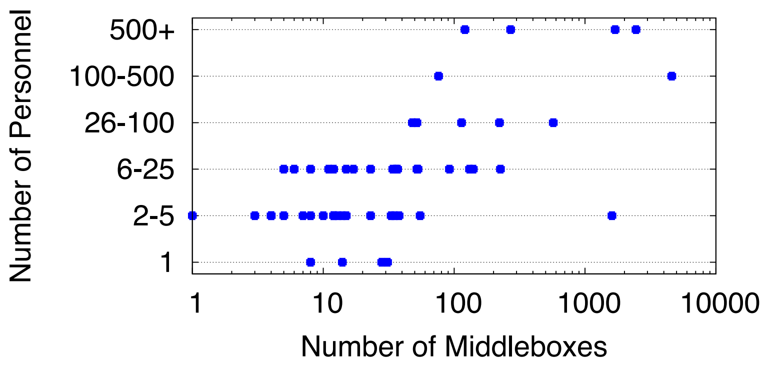


Fig. 3.3: Administrator-estimated number of personnel per network.

3.2.1 Middlebox Deployments

Our data illustrates that typical enterprise networks are a complex ecosystem of firewalls, IDSes, web proxies, and other devices. Figure 3.1 shows a box plot of the number of middleboxes deployed in networks of all sizes, as well as the number of routers and switches for comparison. Across all network sizes, the number of middleboxes is on par with the number of routers in a network! The average very large network in our data set hosts 2850 L3 routers, and 1946 total middleboxes; the average small network in our data set hosts 7.3 L3 routers and 10.2 total middleboxes.[1]

These deployments are not only large, but are also costly, requiring high up-front investment in hardware: thousands to millions of dollars in physical equipment. Figure 3.2 displays five year expenditures on middlebox hardware against the number of actively deployed middleboxes in the network. All of our surveyed very large networks had spent over a million dollars on middlebox hardware in the last five years; the median small network spent between $5,000-50,000 dollars, and the top third of the small networks spent over $50,000.

Paralleling arguments for cloud computing, outsourcing middlebox processing can reduce hardware costs: outsourcing eliminates most of the infrastructure at the enterprise, and a cloud provider can provide the same resources at lower cost due to economies of scale.

[1]Even 7.3 routers and 10.2 middleboxes represents a network of a substantial size. Our data was primarily surveyed from the NANOG network operators group, and thus does not include many of the very smallest networks (*e.g.* homes and very small businesses with only tens of hosts).

3.2.2 Complexity in Management

Figure 3.1 also shows that middleboxes deployments are diverse. Of the eight middlebox categories we present in Figure 3.1, the median very large network deployed seven categories of middleboxes, and the median small network deployed middleboxes from four. Our categories are coarse-grained (*e.g.* Application Gateways include smartphone proxies and VoIP gateways), so these figures represent a *lower bound* on the number of distinct device types in the network.

Managing many heterogeneous devices requires broad expertise and consequently a large management team. Figure 3.3 correlates the number of middleboxes against the number of networking personnel. Even small networks with only tens of middleboxes typically required a management team of 6-25 personnel. Thus, middlebox deployments incur substantial operational expenses in addition to hardware costs.

Understanding the administrative tasks involved further illuminates why large administrative staffs are needed. We break down the management tasks related to middleboxes below.

Upgrades and Vendor Interaction. Deploying new features in the network entails deploying new hardware infrastructure. From our survey, network operators upgrade in the median case every four years. Each time they negotiate a new deployment, they must select between several offerings, weighing the capabilities of devices offered by numerous vendors – an average network in our dataset contracted with 4.9 vendors. This four-year cycle is at the same time both too frequent and too infrequent. Upgrades are too frequent in that every four years, administrators must evaluate, select, purchase, install, and train to maintain new appliances. Upgrades are too infrequent in that administrators are 'locked in' to hardware upgrades to obtain new features. Quoting one administrator:

> Upgradability is very important to me. I do not like it when vendors force me to buy new equipment when a software upgrade could give me additional features.

Cloud computing eliminates the upgrade problem: enterprises sign up for a middlebox *service*; how the cloud provider chooses to upgrade hardware is orthogonal to the service offered.

Monitoring and Diagnostics. To make managing tens or hundreds of devices feasible, enterprises deploy network management tools (e.g., [22, 14]) to aggregate exported monitoring data, *e.g.* SNMP. However, with a cloud solution, the cloud provider monitors utilization and failures of specific devices, and only exposes a

	Misconfig.	**Overload**	**Physical/Electric**
Firewalls	67.3%	16.3%	16.3%
Proxies	63.2%	15.7%	21.1%
IDS	54.5%	11.4%	34%

Table 3.1: Fraction of network administrators who estimated misconfiguration, overload, or physical/electrical failure as the most common cause of middlebox failure.

middlebox *service* to the enterprise administrators, simplifying management at the enterprise.

Configuration. Configuring middleboxes requires two tasks. *Appliance configuration* includes, for example, allocating IP addresses, installing upgrades, and configuring caches. *Policy configuration* is customizing the device to enforce specific enterprise-wide policy goals (*e.g.* a HTTP application filter may block social network sites). Cloud-based deployments obviate the need for enterprise administrators to focus on the low-level mechanisms for appliance configuration and focus only on policy configuration.

Training. New appliances require new training for administrators to manage them. One administrator even stated that existing training and expertise was a key question in purchasing decisions:

> Do we have the expertise necessary to use the product, or would we have to invest significant resources to use it?

Another administrator reports that a lack of training limits the benefits from use of middleboxes:

> They [middleboxes] could provide more benefit if there was better management, and allocation of training and lab resources for network devices.

Outsourcing diminishes the training problem by offloading many administrative tasks to the cloud provider, reducing the set of tasks an administrator must be able perform. In summary, for each management task, outsourcing eliminates or greatly simplifies management complexity.

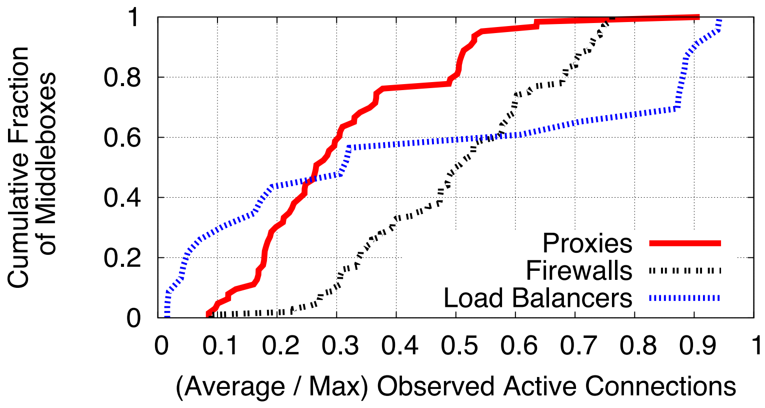


Fig. 3.4: Ratio of average to peak active connections for all proxies, firewalls, and load balancers in the very large enterprise dataset.

3.2.3 Overload and Failures

Most administrators who described their role as engineering estimated spending between one and five hours per week dealing with middlebox failures; 9% spent between six and ten hours per week. Table 3.1 shows the fraction of network administrators who labeled misconfiguration, overload, and physical/electrical failures as the most common cause of failures in their deployments of three types of middleboxes. Note that this table is *not* the fraction of failures caused by these issues; it is the fraction of administrators who estimate each issue to be the *most common* cause of failure. A majority of administrators stated misconfiguration as the most common cause of failure; in the previous subsection we highlight management complexity which likely contributes to this figure.

On the other hand, many administrators saw overload and physical/electrical problems as the most common causes of errors. For example, roughly 16% of administrators said that overload was the most common cause of IDS and proxy failure, and 20% said that physical failures were the most common cause for proxies. A cloud-based capability to elastically provision resources avoids overload by enabling on-demand scaling and resolves failure with standby devices – without the need for expensive overprovisioning.

3.2.4 Discussion

To recap, our survey across 57 enterprises illuminates several middlebox-specific challenges that cloud outsourcing can solve: large deployments with high capital and operating expenses, complex management requirements inflating operation expenses, and failures from physical infrastructure and overload. Cloud outsourcing can cut costs by leveraging economies of scale, simplify management for enterprise administrators, and can provide elastic scaling to limit failures.

Outsourcing to the cloud not only solves challenges in existing deployments, but also presents new opportunities. For example, resource elasticity not only allows usage to scale *up*, but also to scale *down*. Figure 3.4 shows the distribution of average-to-max utilization (in terms of active connections) for three devices across one large enterprise. We see that most devices operate at moderate to low utilization; e.g., 20% of Load Balancers run at $<5\%$ utilization. Today, however, enterprises must invest resources for peak utilization. With a cloud solution, an enterprise can lease a large load balancer only at peak hours and a smaller, cheaper instance otherwise. Furthermore, a pay-per-use model democratizes access to middlebox services and enables even small networks who cannot afford up-front costs to benefit from middlebox processing.

These arguments parallel familiar arguments for the move to cloud computation [34]. This parallel, we believe, only bolsters the case.

3.3 Design Space

Having established the potential benefits of outsourcing middleboxes to the cloud, we now consider how such outsourcing might be achieved. To start, any solution will require some supporting functionality deployed at the enterprise: at a minimum, we will require some device to *redirect* the enterprise's traffic to the cloud. Hence, we assume that each enterprise deploys a generic appliance which we call an *Appliance for Outsourcing Middleboxes* or *APLOMB*. However, depending on the complexity of the design, the functionality might be integrated with the egress router. We assume that the APLOMB redirects traffic to a Point of Presence (PoP), a datacenter hosting middleboxes which process the enterprise's traffic.

As a baseline, we reflect on the properties of middleboxes as deployed today within the enterprise. Consider a middlebox m that serves traffic between endpoints a and b. Our proposal is to change the *placement* of m – moving m from the enterprise to the cloud. Moving m to the cloud eliminates three key properties of its current placement:

(1) on-path: m lies on the direct IP path between a and b
(2) choke point: all paths between a and b traverse m
(3) local: m is located inside the enterprise.

The challenges we face in outsourcing middleboxes all derive from losing the above properties, and our design focuses on compensating for this loss. More specifically, in attempting to regain the benefits of the above properties, we arrive at three design components, as described below.

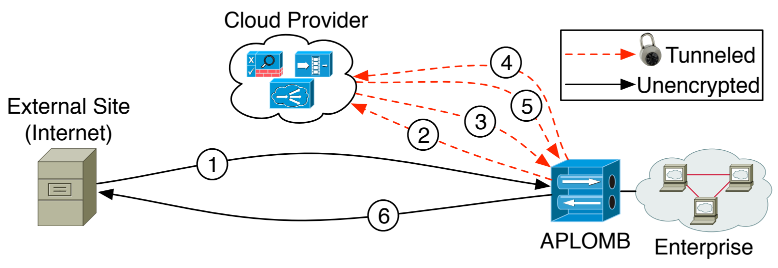
Redirection: Being on-path makes it trivially easy for a middlebox to obtain the traffic it must process; being at a choke point ensures the middlebox sees both directions of traffic flow between two endpoints (bidirectional visibility is critical since most middleboxes operate at the session level). A middlebox in the cloud loses this natural ability; hence we need a redirection architecture that routes traffic between a and b via the cloud, with both directions of traffic consistently traversing the same cloud PoP.

Latency Strategy: A second consequence of being on-path is that the middlebox introduces no additional latency into the path. In contrast, sending traffic on a detour through the cloud could increase path latency, necessitating a practical strategy for low latency operation.

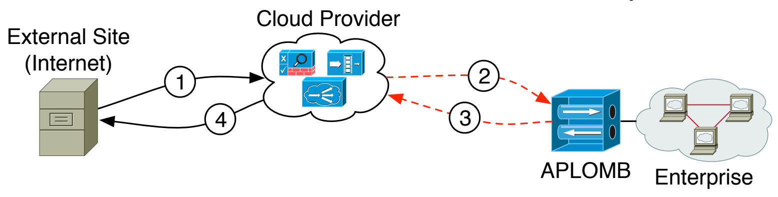
Further, certain 'extremely local' middleboxes such as proxies and WAN optimizers rely on being local to obtain significant reductions in latency and bandwidth costs. Caching proxies effectively terminate communication from an enterprise host a to an external host b thus reducing communication latency from that of path a-m-b to that of a-m. Likewise, WAN optimizers include a protocol acceleration component that achieves significant latency savings (although using very different mechanisms from a proxy).Thus, the latency optimizations we develop also must serve to minimize the latency increase due to taking extremely local middleboxes out of the enterprise.

APLOMB +: 'Extremely local' middleboxes not only reduce latency, but also reduce bandwidth consumption. Caching proxies, by serving content from a local store, avoid fetching data from the wide area; WAN Optimizers include a redundancy elimination component. To retain the savings in bandwidth consumption, we propose what we term APLOMB + appliances that extend APLOMB to provide comparable bandwidth reduction to extremely local appliances.

We explore solutions for the above design components in §3.3.1 (redirection), §3.3.2 (low latency) and §3.3.3 (APLOMB+). Recall from §3.1 that our design goals are to ensure: (i) functional equivalence, (ii) low performance overhead, and (iii) low enterprise-side complexity. We analyze our design options through the lens of these goals and recap the solution we arrive at in §3.3.4.



(a) "Bounce" redirection inflates latency.



(b) A direct architecture reduces latency.

Fig. 3.5: Comparing two redirection architectures.

3.3.1 Redirection

We consider three natural approaches to redirection and discuss their latency vs. complexity tradeoffs.

Bounce Redirection

In the simplest case, the APLOMB gateway at the enterprise tunnels both ingress and egress traffic to the cloud, as shown in Figure 3.5(a). Incoming traffic is bounced to the cloud PoP (1), processed by middleboxes, then sent back to the enterprise (2,3) and delivered to the appropriate hosts. Outgoing traffic is similarly redirected (4-6).

This scheme has two advantages. First, the APLOMB gateway is the only device that needs to be cloud-aware; no modification is required to existing enterprise network or application infrastructure. Second, the design requires minimal gateway functionality and configuration—a few static rules to redirect traffic to the PoP. The obvious drawback of this architecture is the increase in end-to-end latency due to an extra round trip to the cloud PoP for each packet.[2]

[2]We could eliminate a hop for outgoing traffic by routing return traffic directly from the cloud to the external target. However, this would require the cloud provider to spoof the enterprise's IP

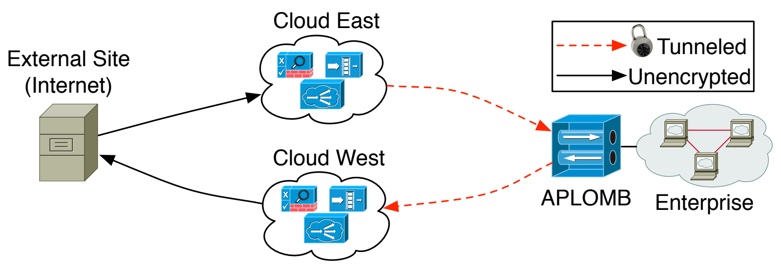


Fig. 3.6: A pure-IP solution cannot ensure that inbound and outbound traffic traverse the same PoP, breaking bidirectional middlebox services.

IP-based Redirection

To avoid the extra round-trips in bounce redirection, we might instead route traffic directly to/from the cloud as in Figure 3.5(b). One approach is to redirect traffic at the IP level: for example, the cloud provider could announce IP prefix P on the enterprise's behalf. Hosts communicating with the enterprise direct their traffic to P and thus their enterprise-bound traffic is received by the provider. The cloud provider, after processing the traffic, then tunnels the traffic to the enterprise gateways, who announce an additional prefix P'. [3]

In practice, enterprises would like to leverage the multi-PoP footprint of a provider for improved latency, load distribution and fault tolerance. For this, the cloud provider might advertise P from multiple PoPs so that client traffic is effectively 'anycasted' to the closest PoP. Unfortunately, IP-based redirection breaks down in a multi-PoP scenario since we cannot ensure that traffic from a client a to enterprise b will be routed to the same cloud PoP as that from b to a, thus breaking stateful middleboxes. This is shown in Figure 3.6 where the Cloud-West PoP is closest (in terms of BGP hops) to the enterprise while Cloud-East is closest to the external site. Likewise, if the underlying BGP paths change during a session then different PoPs might be traversed, once again disrupting stateful processing. Finally, because traffic is redirected at the network layer based on BGP path selection criteria (*e.g.*, AS hops), the enterprise or the cloud provider has little control over which PoP is selected and cannot (for example) pick PoPs to optimize end-to-end latency. Because of these limitations, we reject IP-based redirection as an option.

---

addresses, and such messages may be filtered by intermediate ISPs.

[3]The prefix P would in fact have to be owned by the cloud provider. If the cloud provider simply advertises a prefix assigned to the enterprise, then ISPs might filter the BGP announcements as they would fail the origin authorization checks.

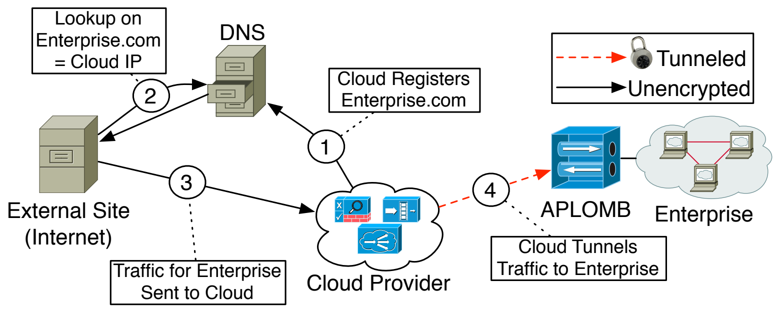

Lookup on Enterprise.com = Cloud IP
DNS
Cloud Registers Enterprise.com
Tunneled
Unencrypted
2
1
External Site (Internet)
3
4
APLOMB
Enterprise
Cloud Tunnels Traffic to Enterprise
Traffic for Enterprise Sent to Cloud
Cloud Provider

Fig. 3.7: DNS redirection step by step.

DNS-based Redirection

DNS-based redirection avoids the problems of IP-based redirection. Here the cloud provider runs the DNS resolution on the enterprise's behalf [3]. We explain this using the example in Figure 3.7. After an enterprise client provides its cloud provider with a manifest of their externally accessible services, the provider registers DNS names on behalf of the client's external services (step 1); *e.g.*, the provider registers '`MyEnterprise.com`'. When a user performs a DNS lookup on `MyEnterprise.com` (step 2), the DNS record directs it to the cloud PoP. The user then directs his traffic to the cloud PoP (step 3), where the traffic undergoes NAT to translate from the public IP address mapped to the cloud PoP to a private IP address internal to the enterprise client's network. The traffic is then processed by any relevant middleboxes and tunneled (step 4) to the enterprise.

This scheme addresses the bidirectionality concerns even in a multi-PoP setting as the intermediate PoP remains the same even if the network-level routing changes. Outbound traffic from the enterprise is relatively easy to control; the gateway device looks up a redirection map to find the PoP to which it must send return traffic. This ensures the symmetric traversal of middleboxes. Finally, Internet traffic initiated by enterprise hosts undergo NAT at the cloud provider. Thus, return traffic is forced to traverse the same PoP based on the public IP the provider assigned this connection.[4]

Redirection Tradeoffs

To compare the latency inflation from bounce redirection *vs.* DNS-based redirection, we use measurements from over 300 PlanetLab nodes and twenty Amazon

[4]Many enterprises already use NATs to external services for other reasons (*e.g.*, flexibility and security); we introduce no new constraints.

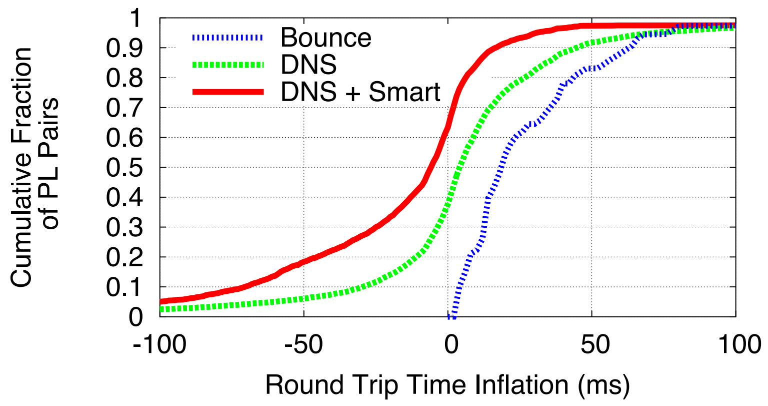


Fig. 3.8: Round Trip Time (RTT) inflation when redirecting traffic between US PlanetLab nodes through Amazon PoPs.

CloudFront locations. We consider an enterprise "site" located at one of fifty US-based PlanetLab sites while the other PlanetLab nodes emulate "clients". For each site e, we pick the closest Amazon CloudFront PoP $P_e^* = \arg\min_P Latency(P, e)$ and measure the impact of tunneling traffic to/from this PoP.

Figure 3.8 shows that the simplest bounce redirection can increase the end-to-end RTT by more than 50ms for 20% of inter-PlanetLab paths. The basic DNS-based redirection reduces the 80^{th} percentile of latency inflation $2\times$ compared to bounce redirection. In fact, for more than 30% of the pairwise measurements, the latency is actually lower than the direct IP path. This is because of well-known triangle inequality violations in inter-domain routing and the fact that cloud providers are very well connected to tier-1/2 ISPs [50]. Hence because the additional enterprise-side complexity required for DNS-based redirection is minimal and yet it achieves significantly lower latencies than Bounce redirection, we choose the DNS-based design.

3.3.2 Low Latency Operation

We now consider additional latency-sensitive PoP selection algorithms and analyze the scale of deployment a cloud provider requires to achieve low latency operation.

Smarter Redirection

So far, we considered a simple PoP selection algorithm where an enterprise site e picks its closest PoP. Figure 3.8 shows that with this simple redirection, 10% of end-to-end scenarios still suffer more than 50ms inflation. To reduce this latency further, we will try to utilize multiple PoPs from the cloud provider's footprint to optimize the *end-to-end* latency as opposed to just the enterprise-to-cloud latency. That is, instead of using a single fixed PoP P_e^* for each enterprise site e, we choose the optimal PoP for each c, e combination. Formally, for each client c and enterprise site e, we identify:

$$P_{c,e}^* : arg \min_P Latency(P, c) + Latency(P, e)$$

We quantify the inflation using smart redirection and the same experimental setup as before, with Amazon CloudFront sites as potential PoPs and PlanetLab nodes as enterprise sites. Figure 3.8 shows that with this "Smart Redirection", more than 70% of the cases have zero or negative inflation and 90% of all traffic has less than 10ms inflation.

Smart redirection requires that the APLOMB appliance direct traffic to different PoPs based on the client's IP and maintain persistent tunnels to multiple PoPs instead of just one tunnel to its closest PoP. This requirement is modest: mappings for PoP selection can be computed at the cloud provider and pushed to APLOMB appliances, and today's commodity gateways can already support hundreds of persistent tunneled connections.

Finally, we note that if communication includes extremely local appliances such as proxies and WAN optimizers, then the bulk of communication is between the enterprise and the middlebox and hence the optimal strategy (which we follow) for such cases is still to simply pick the closest PoP.

Provider Footprint

We now analyze how the middlebox provider's choice of geographic footprint may impact latency. Today's clouds have a few tens of global PoPs and expand as new demand arises [4]. For greater coverage, we could envision an extreme point with a middlebox provider with a footprint comparable to CDNs such as Akamai with thousands of vantage points [80]. While it is clear that a larger footprint provides lower latency, what is not obvious is how large a footprint is required in the context of outsourcing middleboxes.

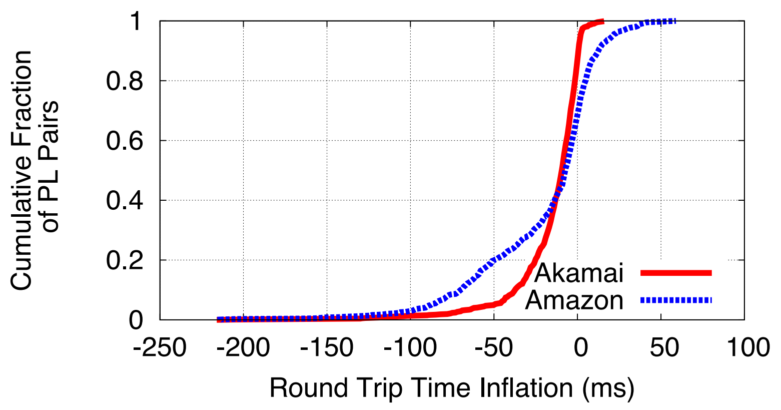


Fig. 3.9: PlanetLab-to-PlanetLab RTTs with APLOMB redirection through Amazon and Akamai.

To understand the implications of the provider's footprint, we extend our measurements to consider a cloud provider with an Akamai-like footprint using IP addresses of over 20,000 Akamai hosts [43]. First, we repeat the the end-to-end latency analysis for paths between US PlanetLab nodes and see that a larger, edge-concentrated Akamai footprint reduces tail latency, but the overall changes are marginal compared to a smaller but well connected Amazon-like footprint. End-to-end latency is the metric of interest when outsourcing most middleboxes – all except for 'extremely local' appliances. Because roughly 70% of inter-PlanetLab node paths actually experience *improved* latency, these results suggest that a middlebox provider can service most customers with most types of middleboxes (*e.g.*, NIDS, firewalls) with an Amazon-like footprint of a few tens of PoPs.

To evaluate whether we can outsource even extremely local middleboxes without a high latency penalty (we discuss bandwidth penalties in §3.3.3), we look at the RTT between each Planetlab node and its closest Akamai node in Figure 3.10. In this case, we see a more dramatic impact of Akamai's footprint as it provides sub-millisecond latencies to 20% of sites, and less than 5 ms latencies to almost 90% of sites. An Amazon-like footprint provides only 30% of sites with an RTT <5 ms. Hence our results suggest that an Amazon-like footprint can serve latency acceleration benefits in only a limited portion of the US; to serve a nation-wide set of sites, an Akamai-like footprint is necessary.

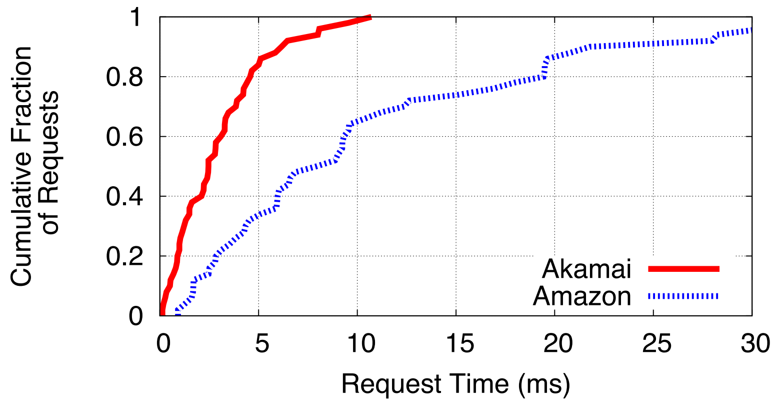


Fig. 3.10: Direct RTTs from PlanetLab to nearest Akamai or Amazon redirection node.

3.3.3 APLOMB+ Gateways

As mentioned earlier, extremely local appliances optimize both latency and bandwidth consumption. Our results above suggest that, with an appropriate provider footprint, these appliances can be outsourced and still offer significant latency savings. We now consider the question of the bandwidth savings they enable. Unfortunately, this is a harder problem since bandwidth optimizations must fundamentally be implemented before the enterprise access link in order to be useful. We thus see three options, described below.

The first is to simply not outsource these appliances. From the enterprises we surveyed and Figure 3.1, we see that WAN optimizers and proxies are currently only deployed in large enterprises and that APLOMB is of significant value even if it doesn't cover proxies and WAN optimizers. Nevertheless, we'd like to do better and hence ask whether a full-fledged middlebox is really needed or whether we could achieve much of their benefit with a more minimal design.

Thus the second option we consider is to embed some *general-purpose* traffic compression capabilities into the APLOMB appliance—we term such an augmented appliance an APLOMB+. In §3.5.3, we evaluate APLOMB+ against traditional WAN optimizers using measurements from a large enterprise and show that protocol-agnostic compression [31] can provide similar bandwidth savings (Figure 3.18). While our measurements suggest that in the specific case of WAN optimization a minimalist APLOMB+ suffices, we do not claim that such a minimal capability exists for every conceivable middlebox (*e.g.*, consider an appliance that encodes outgoing traffic for loss protection), nor that APLOMB+ can fully replicate the behavior of dedicated appliances.

Our third option considers more general support for extremely local appliances at the APLOMB gateway. For this, we envision a more “active” appliance architecture that can run specialized software modules (*e.g.*, a FEC encoder). A minimal set of such modules can be dynamically installed either by the cloud provider or the enterprise administrator. Although more general, this option increases both device and configuration complexity for the enterprise. For this reason, and because APLOMB+ suffices to outsource the extremely local appliances we find in today’s networks, we choose to implement APLOMB+ in our design.

Type of Middlebox	Enterprise Device	Cloud Footprint
IP Firewalls	Basic APLOMB	Multi-PoP
Application Firewalls	Basic APLOMB	Multi-PoP
VPN Gateways	Basic APLOMB	Multi-PoP
Load Balancers	Basic APLOMB	Multi-PoP
IDS/IPS	Basic APLOMB	Multi-PoP
WAN optimizers	APLOMB+	CDN
Proxies	APLOMB+	CDN

Table 3.2: Complexity of design and cloud footprint required to outsource different types of middleboxes.

3.3.4 Summary

We briefly recap our design and its performance and complexity tradeoffs. At the enterprise end, the functionality we require is embedded in an APLOMB appliance. The basic APLOMB tunnels traffic to multiple cloud PoPs and stores a redirection map based on which it forwards traffic to the cloud. The cloud provider uses DNS redirection to redirect traffic from the enterprise’s external contacts to a cloud PoP before forwarding it to the enterprise. APLOMB+ augments this basic functionality with general compression for bandwidth savings.

In addition to middlebox processing, a cloud-based middlebox provider must support DNS translation for its customers, NAT, and tunneling. The key design choice to a provider is the scale of its deployment footprint. We saw that an Amazon-like footprint often *decreases* latency relative to the direct IP path. However, for performance optimization devices, we saw that a larger Akamai-like footprint is necessary to provide extremely local services with nation-wide availability.

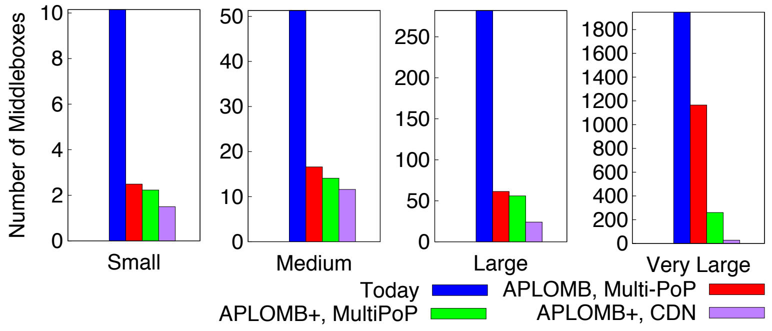


Fig. 3.11: Average number of middleboxes remaining in enterprise under different outsourcing options.

Table 3.2 identifies the design option (and hence its associated complexity) that is needed to retain the functional equivalence of the different middleboxes observed in our survey, *e.g.*, outsourcing an IP firewall requires only a basic APLOMB at the enterprise and an Amazon-scale footprint.[5]

Based on this analysis, Figure 3.11 shows the number of middleboxes that remain in an average small, medium, and large enterprise under different outsourcing deployment options. This suggests that small and medium enterprises can achieve almost all outsourcing benefits with a basic APLOMB architecture using today's cloud providers (we discuss the remaining middleboxes, 'internal firewalls', in §3.5.3). The same basic architecture can outsource close to 50% of the appliances in very large enterprise networks; using APLOMB+ increases the percentage of outsourced appliances to close to 90%.

3.4 APLOMB: Detailed Design

In describing the detailed design of the APLOMB architecture, we focus on three key components as shown in Figure 3.12: (1) a APLOMB gateway to redirect enterprise traffic, (2) the corresponding functions and middlebox capabilities at the cloud

[5]We note that even load balancers can be outsourced since APLOMB retains stateful semantics. One subtle issue is whether load balancers really need to be physically close to backend servers; *e.g.*, for identifying load imbalances at the sub-millisecond granularity. Our conversations with administrators suggest that this is not a typical requirement.

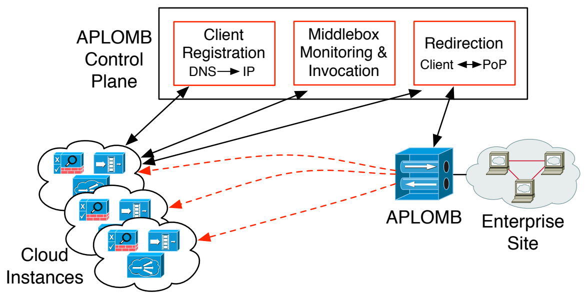


Fig. 3.12: Architectural components of APLOMB.

provider, and (3) a control plane which is responsible for managing and configuring these components.

3.4.1 Enterprise Configuration

Redirecting traffic from the enterprise client to the cloud middlebox provider is simple: an APLOMB gateway is co-located with the enterprise's gateway router, and enterprise administrators supply the cloud provider with a manifest of their address allocations. APLOMB changes neither routing nor switching, and end hosts require no new configuration.

Registration

APLOMB involves an initial registration step in which administrators provide the cloud provider with an *address manifest*. These manifests list the enterprise network's address blocks in its *private* address space and associates each address or prefix with one of three types of address records:

Protected services: Most private IP addresses are registered as protected services. These address records contain an IP address or prefix and the public IP address of the APLOMB device at the gateway to the registered address(es). This registration allows inter-site enterprise traffic to traverse the cloud infrastructure (*e.g.* a host at site A with address 10.2.3.4 can communicate with a host at site B with address 10.4.5.6, and the cloud provider knows that the internal address 10.4.5.6 maps to the APLOMB gateway at site B). The cloud provider allocates no permanent public IP address for hosts with 'protected services' addresses; Internet-destined connections instead undergo traditional NAPT.

DNS services: For hosts which accept incoming traffic, such as web servers, a publicly routeable address must direct incoming traffic to the appropriate cloud

PoP. For these IP addresses, the administrator requests DNS service in the address manifest, listing the private IP address of the service, the relevant APLOMB gateway, and a DNS name. The cloud provider then manages the DNS records for this address on the enterprise client's behalf. When a DNS request for this service arrives, the cloud provider (dynamically) assigns a public IP from its own pool of IP addresses and directs this request to the appropriate cloud PoP and subsequent APLOMB gateway.

Legacy IP services: While DNS-based services are the common case, enterprise may require legacy services that require fixed IP addresses. For these services, the enterprise registers the internal IP address and corresponding APLOMB gateway, and the cloud provider allocates a static public IP address at a single PoP for the IP service. For this type of service, we fall back to the single-PoP Cloud-IP solution rather than DNS redirection discussed in §3.3.

APLOMB gateway

The APLOMB gateway is logically co-located with the enterprise's gateway router and has two key functions: (1) maintaining persistent tunnels to multiple cloud PoPs and (2) steering the outgoing traffic to the appropriate cloud PoP. The gateway registers itself with the cloud controller (§3.4.3), which supplies it with a list of cloud tunnel endpoints in each PoP and forwarding rules (5-tuple $\rightarrow$ cloud PoP Identifier) for redirection. (The gateway router blocks all IP traffic into the network that is not tunneled to a APLOMB gateway.) For security reasons, we use encrypted tunnels (e.g., using OpenVPN) and for reducing bandwidth costs, we enable protocol-agnostic redundancy elimination [31]. Note that the functionality required of the APLOMB gateway is simple enough to be bundled with the egress router itself or built using commodity hardware.

For scalability and fault tolerance, we rely on traditional load balancing techniques. For example, to load balance traffic across multiple APLOMB gateways, the enterprise's private address space can be split to direct traffic to, *e.g.* 10.1.0.0/17 to one gateway, and 10.1.128.0/17 to another. To handle gateway failures, we envision APLOMB hardware with fail-open NICs configured to direct the packets to a APLOMB replica under failure. Since each APLOMB box keeps almost no per-flow state, the replica receiving traffic from the failed device can start forwarding the new traffic without interruption to existing flows.

3.4.2 Cloud Functionality

The cloud provider has three main tasks: (1p publicly addressable IP addresses to the appropriate enterprise customer and internal private address, (2) apply middlebox processing services to the customers' traffic according to their *policies* (§3.4.3), and (3) tunnel traffic to and from the appropriate APLOMB gateways at enterprise sites. Thus, the core components at the cloud PoP are:

- *Tunnel Endpoints* to encapsulate/decapsulate traffic from the enterprise (and to encrypt/decrypt and compress/decompress if enabled)
- *Middlebox Instances* to process the customers' traffic
- *NAT Devices* to translate between publicly visible IP addresses and the clients' internal addresses. NAT devices manage statically configured IP to IP mappings for DNS and Legacy IP services, and generate IP and port mappings for Protected Services (§3.4.1).
- *Policy switching* logic to steer packets between the above components.

Fortunately, it is possible to realize each of these components with existing solutions and there are many research and commercial solutions to provide these features (*e.g.* [57, 39, 11]). These solutions differ along two key dimensions depending on whether the middlebox services are: (1) provided by the cloud infrastructure provider (e.g., Amazon) or by third-party cloud service providers running within these infrastructure providers (e.g., [26]), and (2) realized using hardware- (e.g., [20, 16]) or software-based middleboxes (e.g., [73, 27, 18, 75]. Our architecture is agnostic to these choices and accommodates a broad range of deployment scenarios as long as there is some feasible path to implement the four components described above. The specific implementation we present in this paper runs as a third-party service using software-based middleboxes over an existing infrastructure provider.

3.4.3 Control Plane

A driving design principle for APLOMB is to keep the new components introduced by our architecture that are on the critical path – *i.e.*, the APLOMB gateway device and the cloud terminal endpoint – as simple and as stateless as possible. This not only reduces the enterprise's administrative overhead but also enables seamless transition in the presence of hardware and network failures. To this end, the APLOMB Control Plane manages the relevant network state representing APLOMB gateways, cloud PoPs, middlebox instances, and tunnel endpoints. It is responsible for determining optimal redirection strategies between communicating parties, managing and pushing middlebox policy configurations, and dynamically scaling cloud middlebox capacity to meet demands.

In practice, the control plane is realized in a *cloud controller*, which manages every APLOMB gateway, middlebox, tunneling end point, and the internals of the cloud switching policy.[6] Each entity (APLOMB device, middlebox, *etc.*) registers itself with the controller. The controller sends periodic 'heartbeat' health checks to each device to verify its continued activity. In addition, the controller gathers RTTs from each PoP to every prefix on the Internet (for PoP selection) and utilization statistics from each middlebox (for adaptive scaling). Below we discuss the redirection optimization, policy management, and middlebox scaling performed by the cloud controller.

Redirection Optimization. Using measurement data from the cloud PoPs, the cloud controller pushes the current best (as discussed in §3.3.2) tunnel selection strategies to the APLOMB gateways at the enterprise and mappings in the DNS. To deal with transient routing issues or performance instability, the cloud controller periodically updates these tunneling configurations based on the newest measurements from each cloud PoP.

Policy Configuration. The cloud controller is also responsible for implementing enterprise- and middlebox-specific *policies*. Thus, the cloud provider provides a rich policy configuration interface that exports the available types of middlebox processing to enterprise administrators and also implements a programmatic interface to specify the types of middlebox processing required [56]. Enterprise administrators can specify different *policy chains* of middlebox processing for each class of traffic specified using the traditional 5-tuple categorization of flows (i.e., source and destination IPs, port values and the protocol). For example, an enterprise could require all egress traffic to go through a firewall $\rightarrow$ exfiltration engine $\rightarrow$ proxy. and require that all ingress traffic traverse a firewall $\rightarrow$ IDS, and all traffic to internal web services further go through an application-level firewall. If appropriate, the provider may also export certain device-specific configuration parameters that the enterprise administrator can tune.

Middlebox Scaling. APLOMB providers have a great deal of flexibility in how they actually implement the desired middlebox processing. In particular, as utilization increases on a particular middlebox, the APLOMB provider simply increases the number of instances of that middlebox being utilized for a client's traffic.

Using data from heartbeat health checks on all middleboxes, the cloud con-

[6]While the cloud controller may be in reality a replicated or federated set of controllers, for simplicity this discussion refers to a single logically centralized controller.

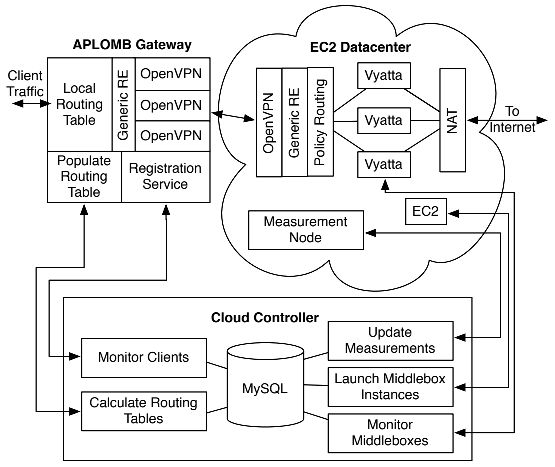


Fig. 3.13: Software architecture of APLOMB.

troller detects changes in utilization. When utilization is high, the cloud controller launches new middleboxes and updates the policy switching framework; when utilization is low, the cloud controller deactivates excess instances. While scaling is simpler if all middlebox processing is performed in software on standard virtual machines, providers using hardware middleboxes could achieve the same result using policy switching alone. Techniques for dynamic scaling under load are well-known for cloud computing applications like web servers [17]; as such we do not go into detail here.

3.4.4 Implementation

We built a prototype system for cloud middlebox processing using middlebox processing services running on EC2 and APLOMB endpoints in our lab and at the authors' homes. We consciously choose to use off-the-shelf components that run on existing cloud providers and end host systems. This makes our system easy to deploy and use and demonstrates that the barriers to adoption are minimal. Our APLOMB endpoint software can be deployed on a stand-alone software router or as a tunneling layer on an end host; installing and running the end host software is as simple as connecting to a VPN.

Figure 3.13 is a software architecture diagram of our implementation. We implement a cloud controller on a server in our lab and use geographically distributed EC2 datacenters as cloud PoPs. Our cloud controller employs a MySQL database to store data on middlebox nodes, RTTs to and from cloud PoPs, and registered clients. The cloud controller monitors APLOMB devices, calculates and pushes routing tables to the APLOMB devices, requests measurements from the cloud PoPs, monitors middlebox instances, and scales middlebox instances up or down as demand varies.

At the enterprise or the end host, the APLOMB gateway maintains several concurrent VPN tunnels, one to a remote APLOMB at each cloud PoP. On startup, the APLOMB software contacts the cloud controller and registers itself, fetches remote tunnel endpoints for each cloud PoP, and requests a set of initial tunnel redirection mappings. A simple tunnel selection layer, populated by the cloud controller, directs traffic to the appropriate endpoint tunnel, and a redundancy elimination encoding module compresses all outgoing traffic. When run on a software router, ingress traffic comes from an attached hosts for whom the router serves as their default gateway. Running on a laptop or end host, static routes in the kernel direct application traffic to the appropriate egress VPN tunnel.

EC2 datacenters host tunnel endpoints, redundancy elimination decoders, middlebox routers, and NATs, each with an inter-device switching layer and controller registration and monitoring service. For tunneling, we use OpenVPN [15], a widely-deployed VPN solution with packages for all major operating systems. We use a Click [60] implementation of the redundancy elimination technique described by Anand et al [31]. For middlebox processing, we use Vyatta [24], a customizable software middlebox. Our default Vyatta installation performs firewalling, intrusion detection, caching, and application-layer web filtering. Policy configurations (§3.4.3) are translated into Vyatta configurations such that each client can have a unique Vyatta configuration dependent on their needs. Finally, each cloud PoP also hosts one ‘measurement node’, which periodically issues ping measurements for RTT estimation to assist in PoP selection.

3.5 Evaluation

We now evaluate APLOMB. First, we present performance benchmarks for three common applications running over our implementation (§3.5.1).We then demonstrate APLOMB’s dynamic scaling capability and its resilience to failure (§3.5.2). Having shown that APLOMB is practical, we return to our goal of outsourcing all middlebox functionality in an enterprise with a trace-driven evaluation of middlebox outsourcing using APLOMB, applied to data from a middlebox deployment in a large enterprise (§3.5.3).

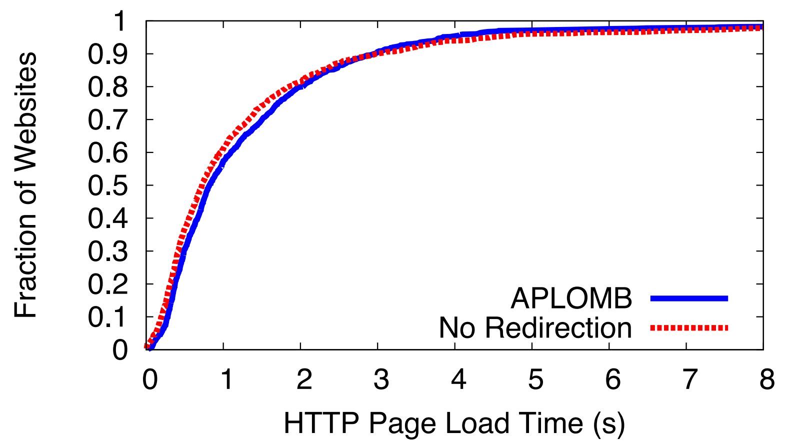


Fig. 3.14: CDF of HTTP Page Load times for Alexa top 1,000 sites with and without APLOMB.

3.5.1 Application Performance

We first demonstrate that APLOMB's architecture is practical for enterprise use with performance benchmarks for common applications using our APLOMB implementation.

HTTP Page Loads: In Figure 3.14, we plot page load times (fetching the front page and all embedded content) from a university network for the Alexa top 1,000 most popular web pages with and without APLOMB processing. We performed this experiment with a vacant cache. For pages at the 50^{th} percentile, page loads without APLOMB took 0.72 seconds, while page loads with took 0.82 seconds. For pages at the 95^{th} percentile, using APLOMB results in shorter page load times: 3.85 seconds versus 4.53 seconds.

BitTorrent: While we don't expect BitTorrent to be a major component of enterprise traffic, we chose to experiment with Bit Torrent because it allowed us to observe a bulk transfer over a long period of time, to observe many connections over our infrastructure simultaneously, and to establish connections to non-commercial endpoints. We downloaded a 698MB public domain film over BitTorrent with and without APLOMB from both a university network and from a residential network, five times repeatedly. The average residential download took 294 seconds without APLOMB, with APLOMB the download speed increased 2.8% to 302 seconds. The average university download took 156 seconds without APLOMB, with APLOMB the average download took 165 seconds, a 5.5% increase.

Voice over IP: Voice over IP (VoIP) is a common enterprise application, but unlike the previously explored applications, VoIP performance depends not only on low latency and high bandwidth, but on low *jitter*, or variance in latency. APLOMB easily accommodates this third demand: we ran VoIP calls over APLOMB and for each call logged the jitter estimator, a running estimate of packet interarrival variance developed for RTP. Industry experts cite 30ms of one-way jitter as a target for maximum acceptable jitter [10]. In the first call, to a residential network, median inbound/outbound jitter with APLOMB was 2.49 ms/2.46 ms and without was 2.3 ms/1.03 ms. In the second, to a public WiFi hotspot, the median inbound/outbound jitter with APLOMB was 13.21 ms/14.49 ms and without was 4.41 ms/4.04 ms.

In summary, these three common applications suffer little or no penalty when their traffic is redirected through APLOMB.

3.5.2 Scaling and Failover

To evaluate APLOMB's dynamic scaling, we measured traffic from a single client to the APLOMB cloud. Figure 3.15 shows capacity adapting to increased network load over a 10-minute period. The client workload involved simultaneously streaming a video, repeatedly requesting large files over HTTP, and downloading several large files via BitTorrent. The resulting network load varied significantly over the course of the experiment, providing an opportunity for capacity scaling. The controller tracks CPU utilization of each middlebox instance and adds additional capacity when existing instances exceed a utilization threshold for one minute.

While middlebox capacity lags changes in demand, this is primarily an artifact of the low sampling resolution of the monitoring infrastructure provided by our cloud provider. Once a new middlebox instance has been allocated and initialized, actual switchover time to begin routing traffic through it is less than 100ms. To handle failed middlebox instances, the cloud controller checks for reachability between itself and individual middlebox instances every second; when an instance becomes unreachable, APLOMB ceases routing traffic through it within 100ms. Using the same mechanism, the enterprise APLOMB can cope with failure of a remote APLOMB, re-routing traffic to another remote APLOMB in the same or even different cloud PoP, providing fault-tolerance against loss of an entire datacenter.

3.5.3 Case Study

We set out with the goal of outsourcing as many middleboxes as possible and reducing enterprise costs, all the while without increasing bandwidth utilization or

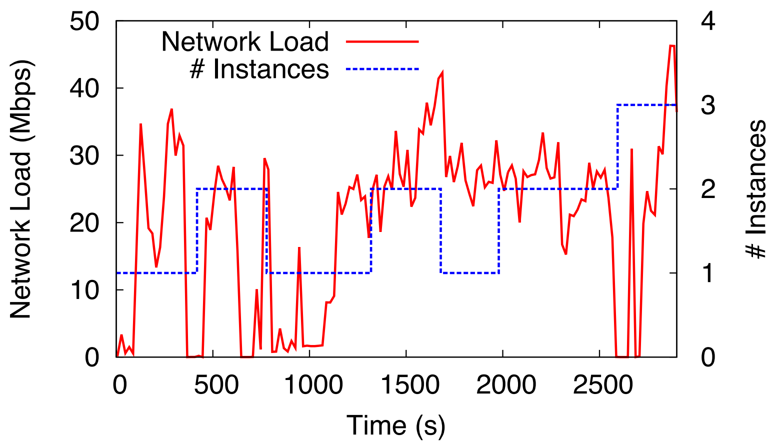


Fig. 3.15: Network load (Y_1) and number of software middlebox instances (Y_2) under load. Experiment used low-capacity instances to highlight scaling dynamics.

latency. We revisit this using the data from the very large enterprise to determine:

- How many middleboxes can the enterprise outsource?
- What are the gains from elastic scaling?
- What latency penalty will inter-site traffic suffer?
- How much does the enterprise's bandwidth costs increase?

Middleboxes Outsourced: Figure 3.16 shows that the large enterprise can outsource close to 60% of the middleboxes under a CDN footprint with APLOMB+.

This high fraction of outsourceability comes despite an atypically high deployment of "internal" firewalls and NIDS at this enterprise. Internal firewalls protect a host or subnetwork not only from Internet-originated traffic, but from traffic originated within the enterprise; the most common reason we found for these deployments was PCI compliance for managing credit card data. While the average enterprise of this size deploys 27.7 unoutsourceable internal firewalls, this enterprise deploys over 100 internal firewalls. From discussions with the network's administrators, we learned these were installed in the past to protect internal servers against worms that preferentially scanned internal prefixes, *e.g.* CodeRed and Nimda. As more IT infrastructure moves to the cloud (see §3.6), many internal firewalls will be able to move to the cloud as well.

Cost Reduction: To evaluate benefits from elastic scaling, in Figure 3.17 we focus on each site of the enterprise and show the ratio of peak-to-average volumes for total inter-site traffic. We use sites across three continents: North America (NA-

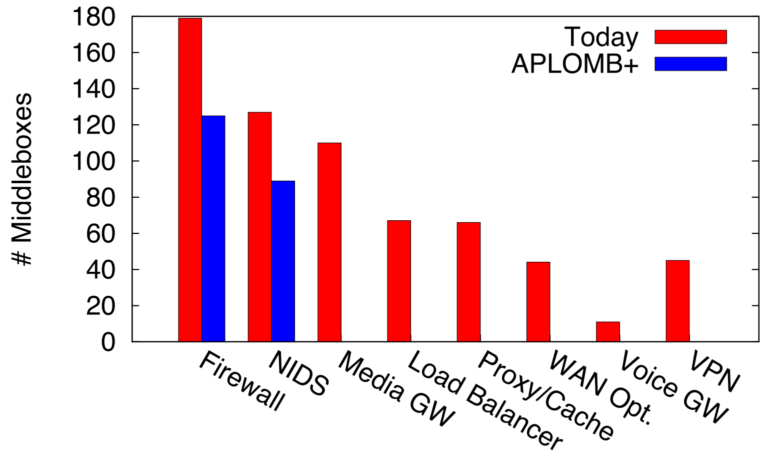


Fig. 3.16: Number of middleboxes in the enterprise with and without APLOMB+. The enterprise has an atypical number of 'internal' firewalls and NIDS.

x), Asia (AS-x), and Europe (EU-x). The peak represents a conservative estimate of the traffic volume the enterprise has provisioned at the site, while the average is the typical utilization; we see that most sites are provisioned over $2\times$ their typical load, and some of the smaller sites as much as $12\times$! In addition, we show peak-to-average values for the top four protocols in use. The per-protocol numbers are indicative of elasticity savings per middlebox, as different protocols are likely to traverse different middleboxes.

Latency: We measured redirection latency for inter-site traffic between the top eleven sites of the enterprise through the APLOMB infrastructure by pinging hosts at each site from within EC2. We found that for more than 60% of inter-site pairs, the latency with redirection is almost identical to the direct RTT. We found that most sites with inflated latency were in Asia, where EC2 does not have a wide footprint.

We also calculated a weighted inflation value, weighted by traffic volume and found that in expectation a typical redirected packet experiences only 1.13 ms of inflation. This results from the fact that the inter-site pairs with high traffic volume actually have negative inflation, by virtue of one or both endpoints being in the US or Europe, where EC2's footprint and connectivity is high.

Bandwidth: Last, we evaluate bandwidth inflation. We ran a traffic trace with full packet payloads collected at a different small enterprise [12] through our APLOMB prototype with and without generic redundancy elimination. Without Generic RE, the bandwidth utilization increased by 6.2% due to encryption and en-

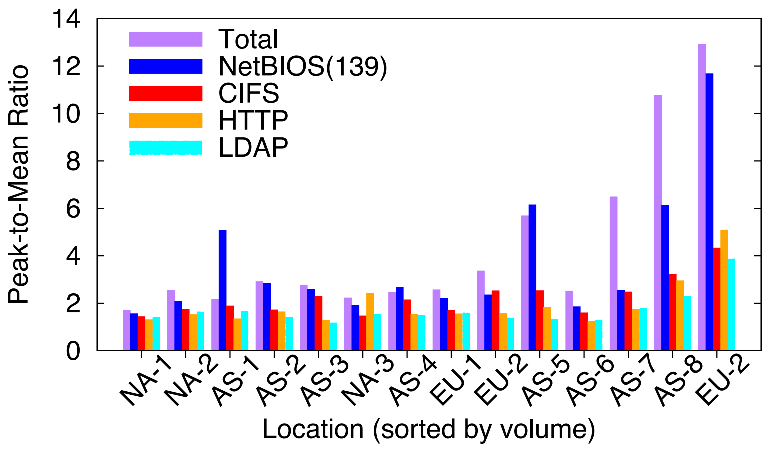


Fig. 3.17: Ratio of peak traffic volume to average traffic volume, divided by protocol.

capsulation overhead. With Generic RE, the bandwidth utilization reduced by 28%, giving APLOMB+ a 32% improvement over basic APLOMB.

As we observed in §3.2, many larger enterprises already compress their inter-site traffic using WAN optimizers. To evaluate the impact of switching compression for inter-site traffic from a traditional WAN optimizer solution to APLOMB+, we compared our observed benefits to those provided by WAN optimizers at eight of the large enterprise sites. In Figure 3.18, we measure the bandwidth cost of a given site in terms of the 95^{th} percentile of the total traffic volume with a WAN Optimizer, with APLOMB, and with APLOMB+. With APLOMB, the worst case inflation is 52% in the median case and at most 58%; APLOMB+ improves this to a median case of 3.8% inflation and a worst case of 8.1%.

3.6 Discussion

Before concluding, we mention some final thoughts on the future of "hybrid" enterprise/cloud architectures, potential cost models for bandwidth, and security challenges that continue to face APLOMB and cloud computing.

IT Outsourcing and Hybrid Clouds: APLOMB complements the ongoing move by enterprises from locally-hosted and managed infrastructure to outsourced cloud infrastructure. A network administrator at one large enterprise we surveyed reported their company's management had issued a broad mandate to moving a sig-

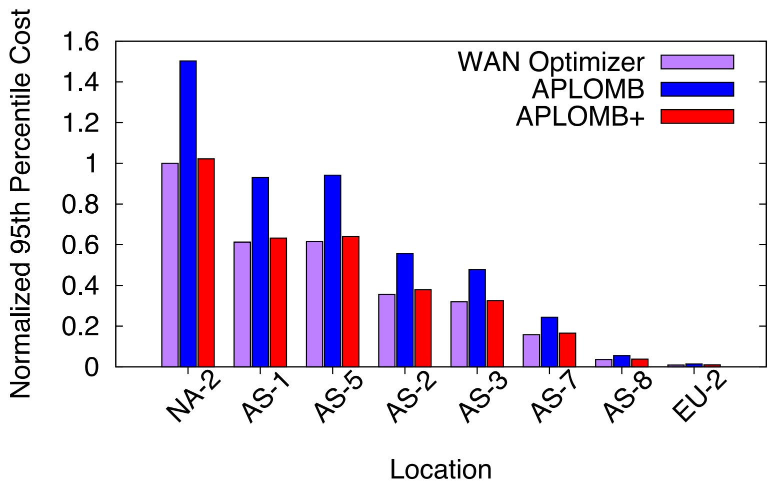


Fig. 3.18: 95th percentile bandwidth without APLOMB, with APLOMB, and with APLOMB+.

nificant portion of their IT infrastructure to the cloud. Federal government agencies are also rapidly moving their IT infrastructure to the cloud, in compliance with a mandate to adopt a ”cloud first” policy for new services and to reduce the number of existing federal datacenters by 800 before 2015 [62]. As these services move to the cloud, the middleboxes protecting them (including internal firewalls, which APLOMB itself cannot outsource) will move to the cloud as well.

Nevertheless, many enterprises plan to keep at least some local infrastructure, citing security and performance concerns for applications currently deployed locally [28]. Further, user-facing devices such as laptops, desktops, smartphones, and printers will always remain within the enterprise – and the majority of middlebox services benefit these devices rather than servers. With some end hosts moving to the cloud, and the majority remaining behind in the enterprise, multiple vendors now offer services for integrating public cloud services with enterprises' existing infrastructure [2, 23], facilitating so-called “hybrid clouds” [51]. APLOMB allows administrators to evade the middlebox-related complexity in this hybrid model by consolidating middleboxes in only one deployment setting.

Bandwidth Costs: APLOMB reduces the cost of middlebox infrastructure, but it may increase bandwidth costs due to current cloud business models. Today, tunneling traffic to a cloud provider necessitates paying for bandwidth twice – once for the enterprise network's access link, and again at the cloud provider. Nevertheless, this does not mean that APLOMB will double bandwidth costs for an enterprise. We ob-

Pricing Model	Total Cost	$/GB	$/Mbps
Standard EC2	30003.20	0.0586	17.58
Amazon DirectConnect	11882.50	0.0232	6.96
Wholesale Bandwidth	6826.70	0.0133	4.00

Table 3.3: Cost comparison of different cloud bandwidth pricing models given an enterprise with a monthly transfer volume of 500TB (an overestimate as compared to the very large enterprise in our study); assumes conversion rate of 1Mbps of sustained transfer equals 300GB over the course of a month.

served earlier that redundancy elimination and compression can reduce bandwidth demands at the enterprise access link by roughly 30%. This optimization is not possible without redirection through a cloud PoP, and could allow a lower capacity, less expensive access link to satisfy an enterprise's needs.

The largest factor in the cost of APLOMB for an enterprise is the bandwidth cost model used by a cloud provider. Today, cloud providers price bandwidth purely by volume; for example, Amazon EC2 charges between $0.05-$0.12 per GB of outgoing traffic, decreasing as volume increases (all incoming traffic is free). On the other hand, a dedicated APLOMB service provider would be able to take advantage of wholesale bandwidth, which is priced by transfer rate. We convert between the two pricing strategies (per-GB and per-Mbps) with the rough conversion factor of 1Mbps sustained monthly throughput equaling 300GB per month. This is in comparison with "wholesale" bandwidth prices of $3-$5 per Mbps for high-volume customers. As a result, though current pricing strategies are not well-suited for APLOMB, a dedicated APLOMB provider could offer substantially lower prices. Indeed, Amazon offers a bulk-priced bandwidth service, "DirectConnect", which offers substantially lower per-GB costs for high-volume customers [2]. Table 3.3 provides a comparison of the bandwidth costs for a hypothetical enterprise which transfers 500TB of traffic per month to and from a cloud service provider under each of these models. These charges a minimal compared to expected savings in hardware, personnel, and other management costs.

Security Challenges: Adopting APLOMB brings with it the same security questions as have challenged cloud computing. These challenges have not stopped widespread adoption of cloud computing services, nor the willingness of security certification standards to certify cloud services (for example, services on Amazon EC2 can achieve PCI-1 compliance, the highest level of certification for storing credit card data). However, these challenges remain concerns for APLOMB and cloud computing in general. Just as cloud storage services have raised questions about providing a cloud provider unencrypted access to data, cloud middlebox services give the cloud provider unencrypted access to traffic flows. Although VMs and other isolation techniques aim to protect customers of a cloud service from other, malicious, customers, some have demonstrated in the cloud computing context information leakage, *e.g.* through side-channels [72]. APLOMB encrypts tunneled traffic to and from the enterprise to protect against man-in-the-middle attacks, and allocates each client it's own set of VMs for middlebox processing, but ultimately it will not appeal to companies whose security policies restrict them from cloud computing in general.

3.7 Related Work

Our work contributes to and draws inspiration from a rich corpus of work in cloud computing, redirection services, and network management.

Cloud Computing: The motivation for APLOMB parallels traditional arguments in favor of cloud computing, many of which are discussed by Armbrust et al. [34]. APLOMB also adapts techniques from traditional cloud solutions, *e.g.* utilization monitoring and dynamic scaling [17], and DNS-based redirection to datacenters with optimal performance for the customer [79].

Middlebox Management: Others have tackled middlebox management challenges within the enterprise [56, 57, 37, 45, 75]. Their solutions offer insights we can apply for managing middleboxes within the cloud – *e.g.*, the policy-routing switch of Joseph et al. [57], the management plane of Ballani et al. [37], and the consolidated appliance of Sekar et al. [75]. None of these proposals consider moving middlebox management out of the enterprise entirely, as we do. Like us, ETTM [45] proposes removing middleboxes from the enterprise network but, where we advocate moving them to the cloud, ETTM proposes the opposite: pushing middlebox processing to enterprise end hosts. As such, ETTM still retains the problem of middlebox management in the enterprise. Sekar et al [75] report on the middlebox deployment of a single large enterprise; our survey is broader in scope (covering a range of management and failure concerns) and covers 57 networks of various scales. They also propose a consolidated middlebox architecture that aims to ameliorate some of the administrative burden associated with middlebox management, but they do not go so far as to propose removing middleboxes from the enterprise network entirely.

Redirection Services: Traffic redirection infrastructures have been explored in prior work [32, 78, 82] but in the context of improving Internet or overlay routing architectures as opposed to APLOMB's goal of enabling middlebox processing in the cloud. RON showed how routing via an intermediary might improve latency; we report similar findings using cloud PoPs as intermediaries. Walfish et al. [82] propose a clean-slate architecture, DOA, by which end hosts explicitly address middleboxes. Gibb et al. [49] develop a service model for middleboxes that focuses on service-aware routers that redirect traffic to middleboxes that can be in the local network or Internet.

Cloud Networking: Using virtual middlebox appliances [24] reduces the physical hardware cost of middlebox ownership, but cannot match the performance of hardware solutions and does little to improve configuration complexity. Some startups and security companies have cloud-based offerings for specific middlebox services: Aryaka [6] offers protocol acceleration; ZScalar [26] performs intrusion detection; and Barracuda Flex [7] offers web security. To some extent, our work can be viewed as an extreme extrapolation of their services and we provide a comprehensive exploration and evaluation of such a trend. CloudNaaS [39] and startup Embrane [11] aim at providing complete middlebox solutions for enterprise services that are *already* in the cloud.

3.8 Conclusion

Outsourcing middlebox processing to the cloud relieves enterprises of major problems caused by today's enterprise middlebox infrastructure: cost, management complexity, capacity rigidity, and failures. Our survey of 57 enterprise network managers guides the design of APLOMB, a practical system for middlebox processing in the cloud. APLOMB succeeds in outsourcing the vast majority of middleboxes from a typical enterprise network without impacting performance, making scalable, affordable middlebox processing accessible to enterprise networks of every size.

4 Discussion and Conclusion

Netcalls and APLOMB both present deployment models where ISPs and cloud providers can export middleboxes 'as a service' to external clients. As we've shown, breaking the 'unilateral model' of middlebox deployment today – where enterprise administrators deploy middleboxes for local use only and with a single administrative policy – can present new business models, save on costs for clients, and create new usage capabilities for clients.

Netcalls and APLOMB each present a very different scenario, however, for who will deploy these middleboxes and how they will be used. Netcalls envisions ISPs as deploying middlebox services, where APLOMB envisions deployment in the cloud. Netcall users are end host applications with relatively fine-grained configuration policies; APLOMB users are enterprise administrators with relatively course-grained configuration policies.

When it comes to likelihood of adoption, it is clear that APLOMB is the more deployable of the two, for example:

- Implementing APLOMB can be done in software with out-of-the-box components. Implementing Netcalls requires transitioning network infrastructure to SDN, with an entirely new management plane for individual middleboxes.
- Deploying in the cloud means that any new start-up can deploy APLOMB services on multi-purpose infrastructure; and hence that there are countless possible players who may decide to serve as providers. Deploying on the Internet means that existing ISPs must move to adoption – Netcalls cannot be deployed by 'any startup', but only a very specific set of companies.
- The client base for APLOMB exists today: enterprise network administrators. Applications that invoke netcalls are nonexistent (besides the prototypes presented earlier).

Given the greater promise for APLOMB's adoption, we might ask, can we implement netcalls on top of an APLOMB deployment? For many middlebox services, implementing netcalls-style function calls on top of APLOMB is entirely feasible. VPN tunneling can be performed not only at the edge of an enterprise, but directly from any laptop. Thus, any individual user may become an APLOMB client. Individual applications might use the netcalls protocol to communicate with resolution servers that resolve client requests to middleboxes in the cloud rather than ISPs.

While this deployment model might serve to deploy netcalls sooner than we may expect any ISP deployment, the cloud is, however, limited in the set of network processing services that it can deploy. For example, the cloud can not implement 'hop-by-hop' services such as any form of QoS guarantee, as these services must be deployed in the network. Further, the cloud cannot protect against network-layer DDoS, as netcalls can. To obtain the full capabilities promised by netcalls, we must rely on an eventual ISP deployment.

Although ultimately the complete success of netcalls relies on an ISP deployment, a gradual deployment evolution from APLOMB and a more limited version of netcalls isn't a dismal prospect. Early-stage startups are today deploying limited versions of APLOMB [26, 6] which focus solely on a single type of middlebox appliance. At the same time, ISPs are starting to deploy some, limited middlebox services to enterprises, *e.g.* [35] for DDoS defense. Should these initial services prove useful, it is not hard to imagine that these companies will expand to a full suite of enterprise middlebox services in the cloud. Adding limited netcalls-style programmability is not an outrageous step forward – once the middlebox infrastructure is in place for a multi-enterprise middlebox service. Hence, there is a path from today's deployments towards even netcalls-style services.

In this thesis, we have shown how middlebox services, where third party providers deploy middlebox services and expose them for usage and configuration, can lead to new business models and new capabilities for middlebox usage. APLOMB presents many benefits, primarily in terms of cost, to enterprise administrators who may outsource their middlebox processing to cloud providers. Netcalls presents new programmability features for application developers, who explicitly configure middlebox services in the networks their traffic traverses. We've demonstrated the possible benefits from these architectures (or a hybrid of the two); from here forward one can only only evaluate the varying forms that middlebox services may take as the market drives adoption.

Bibliography

[1] Akamai Security Solutions. http://www.akamai.com/security.

[2] Amazon Direct Connect. http://aws.amazon.com/directconnect.

[3] Amazon Route 53. http://aws.amazon.com/route53.

[4] Amazon Web Services Launches Brazil Datacenters for Its Cloud Computing Platform. http://tinyurl.com/amazonbrazil.

[5] Androguard. http://code.google.com/p/androguard/.

[6] Aryaka WAN Optimization. http://www.aryaka.com.

[7] Barracuda Web Security Flex. http://www.barracudanetworks.com/ns/products/web_security_flex_overview.php.

[8] CAIDA AS Relationships. http://www.caida.org/data/active/as-relationships/.

[9] CAIDA: The Internet Topology Data Kit. http://www.caida.org/data/active/internet-topology-data-kit/.

[10] CISCO: Quality of Service Design Overview. http://www.ciscopress.com/articles/article.asp?p=357102.

[11] Embrane. http://www.embrane.com/.

[12] M57 packet traces. https://domex.nps.edu/corp/scenarios/2009-m57/net/.

[13] mod_qos. http://opensource.adnovum.ch/mod_qos/.

[14] Network Monitoring Tools. http://tinyurl.com/nmtools.

[15] OpenVPN. http://www.openvpn.com.

[16] Palo Alto Networks. http://www.paloaltonetworks.com/.

[17] RightScale Cloud Management. http://www.rightscale.com/.

[18] Riverbed Virtual Steelhead. http://tinyurl.com/6o3vn9k.

[19] RouteViews Project. http://www.routeviews.org.

[20] Symantec: Data Loss Protection. http://www.vontu.com.

[21] The CAIDA AS Relationships Dataset, Jan 2010. http://www.caida.org/data/active/as-relationships/.

[22] Tivoli Monitoring Software. http://tinyurl.com/7ycs5xv.

[23] VMWare vCloud. http://vcloud.vmware.com.

[24] Vyatta Software Middlebox. http://www.vyatta.com.

[25] World Enterprise Network and Data Security Markets. https://www.abiresearch.com/research/product/1006059-world-enterprise\
\-network-and-data-security/.

[26] ZScaler Cloud Security. http://www.zscaler.com.

[27] Transparent caching using Squid. http://www.visolve.com/squid/whitepapers/trans_caching.pdf, 2006.

[28] Cloud Computing - 31 companies Describe Their Experiences. http://www.ipanematech.com/information-center/download.php?link=white-papers/White%20Book_2011-Cloud_Computing_OBS_Ipanema_http://www.ipanematech.com/information-center/download.php?link=white-papers/White%20Book_2011-Cloud_Computing_OBS_Ipanema_Technologies_EBG.pdf, 2011.

[29] M. Allman and V. Paxson. TCP Congestion Control. RFC 5681.

[30] E. Amir, S. McCanne, and R. Katz. An Active Service Framework and its Application to Real-Time Multimedia Transcoding. In *Proc. ACM SIGCOMM*, 1998.

[31] A. Anand, A. Gupta, A. Akella, S. Seshan, and S. Shenker. Packet Caches on Routers: The Implications of Universal Redundant Traffic Elimination. In *Proc. ACM SIGCOMM*, 2008.

[32] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proc. ACM Symposium on Operating Systems Principles*, 2001.

[33] T. Anderson, L. Peterson, S. Shenker, and J. Turner. Overcoming the Internet Impasse Through Virtualization. *IEEE Computer*, 38(4):34–41, April 2005.

[34] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A View of Cloud Computing. *Communications of the ACM*, 53(4):50–58, April 2010.

[35] AT&T. DDoS Defense. `http://tinyurl.com/attddos`.

[36] H. Balakrishnan, K. Lakshminarayanan, S. Ratnasamy, S. Shenker, I. Stoica, and M. Walfish. A Layered Naming Architecture for the Internet. In *Proc. ACM SIGCOMM*, 2004.

[37] H. Ballani and P. Francis. CONMan: a Step Towards Network Manageability. In *Proc. ACM SIGCOMM*, 2007.

[38] T. Benson, A. Akella, and A. Shaikh. Demystifying Configuration Challenges and Trade-offs in Network-Based ISP Services. In *Proc. ACM SIGCOMM*, 2011.

[39] T. Benson, A. Akella, A. Shaikh, and S. Sahu. CloudNaaS: a Cloud Networking Platform for Enterprise Applications. In *Proc. ACM Symposium on Cloud Computing*, 2011.

[40] R. Beverly, A. Berger, Y. Hyun, and K. Claffy. Understanding the Efficacy of Deployed Internet Source Address Validation Filtering. In *Proc. ACM SIGCOMM Internet Measurement Conference*, 2009.

[41] R. Braden, L. Zhang, S. Benson, S. Herzog, and S. Jamin. Resource Reservation Protocol (RSVP). RFC 2205.

[42] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking Control of the Enterprise. In *Proc. ACM SIGCOMM*, 2007.

[43] D. R. Choffnes and F. E. Bustamante. Taming the Torrent: a Practical Approach to Reducing Cross-ISP Traffic in Peer-to-Peer Systems. In *Proc. ACM SIGCOMM*, 2008.

[44] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C.-G. Liu, and L. Wei. An Architecture for Wide-Area Multicast Routing. In *Proc. ACM SIGCOMM*, 1994.

[45] C. Dixon, H. Uppal, V. Brajkovic, D. Brandon, T. Anderson, and A. Krishnamurthy. ETTM: a Scalable Fault Tolerant Network Manager. In *Proc. USENIX Network Systems Design and Implementation*, 2011.

[46] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy. RouteBricks: Exploiting Parallelism to Scale Software Routers. In *Proc. ACM Symposium on Operating Systems Principles*, 2009.

[47] N. Dukkipati and N. McKeown. Why Flow-Completion Time is the Right Metric for Congestion Control. *ACM SIGCOMM Computer Communications Review*, 36(1):59–62, January 2006.

[48] S. Floyd. HighSpeed TCP for Large Congestion Windows. RFC 3649.

[49] G. Gibb, H. Zeng, and N. McKeown. Outsourcing Network Functionality. In *Proc. ACM Workshop on Hot Topics in Software Defined Networking*, 2012.

[50] K. P. Gummadi, H. V. Madhyastha, S. D. Gribble, H. M. Levy, and D. Wetherall. Improving the Reliability of Internet Paths with One-hop Source Routing. In *Proc. USENIX Operating Systems Design and Implementation*, 2004.

[51] M. Hajjat, X. Sun, Y.-W. E. Sung, D. A. Maltz, S. Rao, K. Sripanidkulchai, and M. Tawarmalani. Cloudward Bound: Planning for Beneficial Migration of Enterprise Applications to the Cloud. In *Proc. ACM SIGCOMM*, 2012.

[52] S. Han, K. Jang, K. Park, and S. Moon. PacketShader: A GPU-Accelerated Software Router. In *Proc. ACM SIGCOMM*, 2010.

[53] S. Han, S. Marshall, B.-G. Chun, and S. Ratnasamy. Megapipe: A New Programming Interface for Scalable Network I/O. In *Proc. USENIX Operating Systems Design and Implementation*, 2012.

[54] M. Honda, Y. Nishida, C. Raiciu, A. Greengalgh, M. Handley, and H. Tokuda. Is it still possible to extend TCP? In *Proc. ACM SIGCOMM Internet Measurement Conference*, 2011.

[55] J. P. John, E. Katz-Bassett, A. Krishnamurthy, T. Anderson, and A. Venkataramani. Consensus Routing: The Internet as a Distributed System. In *Proc. USENIX Network Systems Design and Implementation*, 2008.

[56] D. Joseph and I. Stoica. Modeling middleboxes. *IEEE Network*, 22(5):20–25, September 2008.

[57] D. A. Joseph, A. Tavakoli, and I. Stoica. A Policy-Aware Switching Layer for Data Centers. In *Proc. ACM SIGCOMM*, 2008.

[58] D. Katabi, M. Handley, and C. Rohrs. Congestion Control for High Bandwidth-Delay Product Networks. In *Proc. ACM SIGCOMM*, 2002.

[59] E. Katz-Bassett, H. Madhyastha, V. K. Adhikari, C. Scott, J. Sherry, P. van Wesep, T. Anderson, and A. Krishnamurthy. Reverse Traceroute. In *Proc. USENIX Network Systems Design and Implementation*, 2010.

[60] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click Modular Router. *ACM Transactions on Computer Systems*, 18(3):263–297, August 2000.

[61] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutevski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker. Onix: A Distributed Control Platform for Large-scale Production Network. In *Proc. USENIX Operating Systems Design and Implementation*, 2010.

[62] V. Kundra. 25 Point Implementation Plan to Reform Federal Information Technology Management. Technical report, US CIO, 2010.

[63] N. Kushman, S. Kandula, D. Katabi, and B. Maggs. R-BGP: Staying Connected in a Connected World. In *Proc. USENIX Network Systems Design and Implementation*, 2007.

[64] G. Lu, C. Guo, Y. Li, Z. Zhou, T. Yuan, H. Wu, Y. Xiong, R. Gao, and Y. Zhang. ServerSwitch: A Programmable and High Performance Platform for Data Center Networks. In *Proc. USENIX Network Systems Design and Implementation*, 2011.

[65] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling High Bandwidth Aggregates in the Network. *ACM SIGCOMM Computer Communication Review*, 32(3):62–73, July 2001.

[66] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling Innovation in Campus Networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, March 2008.

[67] C. Muthukrishnan, V. Paxson, M. Allman, and A. Akella. Using Strongly Typed Networking to Architect for Tussle. In *Proc. Workshop on Hot Topics in Networks (HotNets)*, 2010.

[68] A. Pesterev, J. Strauss, N. Zeldovich, and R. T. Morris. Improving Network Connection Locality on Multicore Systems. In *Proc. ACM EuroSys*, 2012.

[69] L. Popa, N. Egi, S. Ratnasamy, and I. Stoica. Building Extensible Networks with Rule-Based Forwarding (RBF). In *Proc. USENIX Operating Systems Design and Implementation*, 2010.

[70] B. Raman, S. Agarwal, Y. Chen, M. Caesar, W. Cui, P. Johansson, K. Lai, T. Lavian, S. Machiraju, Z. M. Mao, G. Porter, T. Roscoe, M. Seshadri, J. Shih, K. Sklower, L. Subramanian, T. Suzuki, S. Zhuang, A. D. Joseph, Y. H. Katz, and I. Stoica. The SAHARA Model for Service Composition Across Multiple Providers. In *Proc. IEEE Pervasive*, 2002.

[71] J. Rexford, J. Wang, Z. Xiao, and Y. Zhang. BGP Routing Stability of Popular Destinations. In *Proc. ACM SIGCOMM Internet Measurement Workshop*, 2002.

[72] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off of my cloud: Exploring Information Leakage in Third-party Compute Clouds. In *Proc. ACM Computer and Communications Security*, 2009.

[73] M. Roesch. Snort - Lightweight Intrusion Detection for Networks. In *Proc. USENIX Large Installations Systams Administration*, 1999.

[74] V. Sekar, N. Egi, S. Ratnasamy, M. Reiter, and G. Shi. Design and Implementation of a Consolidated Middlebox Architecture. In *Proc. USENIX Network Systems Design and Implementation*, 2012.

[75] V. Sekar, S. Ratnasamy, M. K. Reiter, N. Egi, and G. Shi. The Middlebox Manifesto: Enabling Innovation in Middlebox Deployment. In *Proc. ACM Workshop on Hot Topics in Networking (HotNets)*, 2011.

[76] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making Middleboxes Someone Else's Problem: Network Processing as a Cloud Service. In *Proc. ACM SIGCOMM*, 2012.

[77] M. Stiemerling, J. Quittek, and T. Taylor. Middlebox Communication (MIDCOM) Protocol Semantics. RFC 5189.

[78] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet Indirection Infrastructure. *IEEE/ACM Transactions on Networking*, 12(2):205–218, April 2004.

[79] A. Su, D. Choffnes, A. Kuzmanovic, and F. Bustamante. Drafting behind Akamai (Travelocity-Based Detouring). In *Proc. ACM SIGCOMM*, 2006.

[80] V. Valancius, N. Laoutaris, L. Massouli'e, C. Diot, and P. Rodriguez. Greening the Internet with Nano Data Centers. In *Proc. ACM CoNEXT*, 2009.

[81] A. Vahdat, M. Dahlin, T. Anderson, and A. Aggarwal. Active Names: Flexible Location and Transport of Wide-Area Resources. In *Proc. USENIX Symposium on Internet Technologies and Systems*, 1999.

[82] M. Walfish, J. Stribling, M. Krohn, H. Balakrishnan, R. Morris, and S. Shenker. Middleboxes No Longer Considered Harmful. In *Proc. USENIX Operating Systems Design and Implementation*, 2004.

[83] D. Wetherall, U. Legedza, and J. Guttag. Introducing new Internet services: Why and how. *IEEE Network*, 12(3):12–19, May 1998.

[84] W. Xu and J. Rexford. MIRO: Multi-Path Interdomain Routing. In *Proc. ACM SIGCOMM*, 2006.

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