A Policy-aware Switching Layer for Data Centers

Dilip A. Joseph
dilip@cs.berkeley.edu

Arsalan Tavakoli
arsalan@cs.berkeley.edu

Ion Stoica
istoica@cs.berkeley.edu

University of California at Berkeley

ABSTRACT

Data centers deploy a variety of middleboxes (*e.g.*, firewalls, load balancers and SSL offloaders) to protect, manage and improve the performance of applications and services they run. Since existing networks provide limited support for middleboxes, administrators typically overload path selection mechanisms to coerce traffic through the desired sequences of middleboxes placed on the network path. These ad-hoc practices result in a data center network that is hard to configure and maintain, wastes middlebox resources, and cannot guarantee middlebox traversal under network churn.

To address these issues, we propose the policy-aware switching layer or *PLayer*, a new layer-2 for data centers consisting of inter-connected policy-aware switches or *pswitches*. Unmodified middleboxes are placed off the network path by plugging them into *pswitches*. Based on policies specified by administrators, *pswitches* explicitly forward different types of traffic through different sequences of middleboxes. Experiments using our prototype software *pswitches* suggest that the *PLayer* is flexible, uses middleboxes efficiently, and guarantees correct middlebox traversal under churn.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design; C.2.5 [**Computer-Communication Networks**]: Local and Wide-Area Networks

General Terms

Design, Performance, Management

1. INTRODUCTION

In recent years, data centers have rapidly grown to become an integral part of the Internet fabric [7]. These data centers typically host tens or even thousands of different applications [16], ranging from simple web servers providing static content to complex e-commerce applications. To protect, manage and improve the performance of these applications, data centers deploy a large variety of *middleboxes*, including firewalls, load balancers, SSL offloaders, web caches, and intrusion prevention boxes.

Unfortunately, the process of deploying middleboxes in today's data center networks is inflexible and prone to misconfiguration. While literature on the practical impact and prevalence of middlebox deployment issues in current data centers is scant, there is growing evidence of these problems. According to [4], 78% of data center downtime is caused by misconfiguration. The sheer number of misconfiguration issues cited by industry manuals [15, 6], reports of large-scale network misconfigurations [3], and anecdotal evidence from network equipment vendors and data center architects complete a gloomy picture.

As noted by others in the context of the Internet [32, 30], the key challenge in supporting middleboxes in today's networks is that there are no available protocols and mechanisms to *explicitly* insert these middleboxes on the path between end-points. As a result, data center administrators deploy middleboxes *implicitly* by placing them in series on the physical network path [16]. To ensure that traffic traverses the desired sequence of middleboxes, administrators must rely on overloading existing path selection mechanisms, such as layer-2 spanning tree construction (used to prevent forwarding loops). As the complexity and scale of data centers increase, it is becoming harder and harder to rely on these ad-hoc mechanisms to ensure the following highly desirable properties:

(i) Correctness: *Traffic should traverse middleboxes in the sequence specified by the network administrator under all network conditions.* Configuring layer-2 switches and layer-3 routers to enforce the correct sequence of middleboxes involves tweaking hundreds of knobs, a highly complex and error-prone process [4, 15, 28, 19]. Misconfiguration is exacerbated by the abundance of redundant network paths in a data center, and the unpredictability of network path selection under network churn [21, 15]. For example, the failure or addition of a network link may result in traffic being routed around the network path containing a mandatory firewall, thus violating data center security policy.

(ii) Flexibility: *The sequences of middleboxes should be easily (re)configured as application requirements change.* Deploying middleboxes on the physical network path constrains the data center network. Adding, removing or changing the order of middleboxes traversed by a particular application's traffic, *i.e.*, modifying the logical network topology, requires significant engineering and configuration changes [16]. For example, adding an SSL offload box in front of web traffic

SIGCOMM'08, August 17–22, 2008, Seattle, Washington, USA.

requires identifying or creating a choke point through which all web traffic passes and manually inserting the SSL offload box at that location.

(iii) Efficiency: *Traffic should not traverse unnecessary middleboxes.* On-path deployment of middleboxes forces all traffic flowing on a particular network path to traverse the same sequence of middleboxes. However, different applications may have different requirements. A simple web application may require its inbound traffic to pass through a simple firewall followed by a load balancer, while an Enterprise Resource Planning (ERP) application may require that all its traffic be scrubbed by a dedicated custom firewall and then by an intrusion prevention box. Since all traffic traverses the same middleboxes, the web traffic will unnecessarily waste the resources of the intrusion prevention box and the custom firewall.

In this paper, we present the policy-aware switching layer (or *PLayer*), a proposal that aims to address the limitations of today's data center middlebox deployments. The *PLayer* is built around two principles: (i) Separating policy from reachability, and (ii) Taking middleboxes off the physical network path. It consists of policy-aware switches, or *pswitches*, which maintain the middlebox traversal requirements of all applications in the form of *policy specifications*. These *pswitches* classify incoming traffic and explicitly redirect them to appropriate middleboxes, thus guaranteeing middlebox traversal in the policy-mandated sequence. The low-latency links in a typical data center network enable off-path placement of middleboxes with minimal performance sacrifice. Off-path middlebox placement simplifies topology modifications and enables efficient usage of existing middleboxes. For example, adding an SSL offload box in front of HTTPS traffic simply involves plugging in the SSL offload box into a *pswitch* and configuring the appropriate HTTPS traffic policy at a centralized policy controller. The system automatically ensures that the SSL box is only traversed by HTTPS traffic while the firewall and the load balancer are shared with HTTP traffic. To ease deployment in existing data centers, the *PLayer* aims to support existing middleboxes and application servers without any modifications, and to minimize changes required in other network entities like switches.

Separating policy from reachability and centralized control of networks have been proposed in previous work [23, 20]. Explicitly redirecting network packets to pass through off-path middleboxes is based on the well-known principle of indirection [30, 32, 22]. This paper combines these two general principles to revise the current ad-hoc manner in which middleboxes are deployed in data centers. Keeping existing middleboxes and servers unmodified, supporting middleboxes that modify frames, and guaranteeing middlebox traversal under all conditions of policy, middlebox and network churn make the design and implementation of the *PLayer* a challenging problem. We have prototyped *pswitches* in software using Click [25] and evaluated its functionality on a small testbed.

2. BACKGROUND

In this section, we describe our target environment and the associated data center network architecture. We then illustrate the limitations of current best practices in data center middlebox deployment.

2.1 Data Center Network Architecture

Our target network environment is characterized as follows:

Scale: The network may consist of tens of thousands of machines running thousands of applications and services.

Middlebox-based Policies: The traffic needs to traverse various middleboxes, such as firewalls, intrusion prevention boxes, and load balancers before being delivered to applications and services.

Low-Latency Links: The network is composed of low-latency links which facilitate rapid information dissemination and allow for indirection-mechanisms with minimal performance overhead.

While both data centers and enterprise networks fit the above characterization, in this paper we focus on data centers, for brevity.

The physical network topology in a data center is typically organized as a three layer hierarchy [15], as shown in Figure 1(a). The access layer provides physical connectivity to the servers in the data centers, while the aggregation layer connects together access layer switches. Middleboxes are usually deployed at the aggregation layer to ensure that traffic traverses middleboxes before reaching data center applications and services. Multiple redundant links connect together pairs of switches at all layers, enabling high availability at the risk of forwarding loops. The access layer is implemented at the data link layer (*i.e.*, layer-2), as clustering, failover and virtual server movement protocols deployed in data centers require layer-2 adjacency [1, 16].

2.2 Limitations of Current Middlebox Deployment Mechanisms

In today's data centers, there is a strong coupling between the physical network topology and the *logical* topology. The logical topology determines the sequences of middleboxes to be traversed by different types of application traffic, as specified by data center policies. Current middlebox deployment practices hard code these policies into the physical network topology by placing middleboxes in sequence on the physical network paths and by tweaking path selection mechanisms like spanning tree construction to send traffic through these paths. This coupling leads to middlebox deployments that are hard to configure and fail to achieve the three properties – correctness, flexibility and efficiency – described in the previous section. We illustrate these limitations using the data center network topology in Figure 1.

2.2.1 Hard to Configure and Ensure Correctness

Reliance on overloading path selection mechanisms to send traffic through middleboxes makes it hard to ensure that traffic traverses the correct sequence of middleboxes under all network conditions. Suppose we want traffic between servers *S1* and *S2* in Figure 1(b) to always traverse a firewall, so that *S1* and *S2* are protected from each other when one of them gets compromised. Currently, there are three ways to achieve this: (i) Use the existing aggregation layer firewalls, (ii) Deploy new standalone firewalls, or (iii) Incorporate firewall functionality into the switches themselves. All three options are hard to implement and configure, as well as suffer from many limitations.

The first option of using the existing aggregation layer firewalls requires all traffic between *S1* and *S2* to traverse the path ($S1$, $A1$, $G1$, $L1$, $F1$, $G3$, $G4$, $F2$, $L2$, $G2$, $A2$,

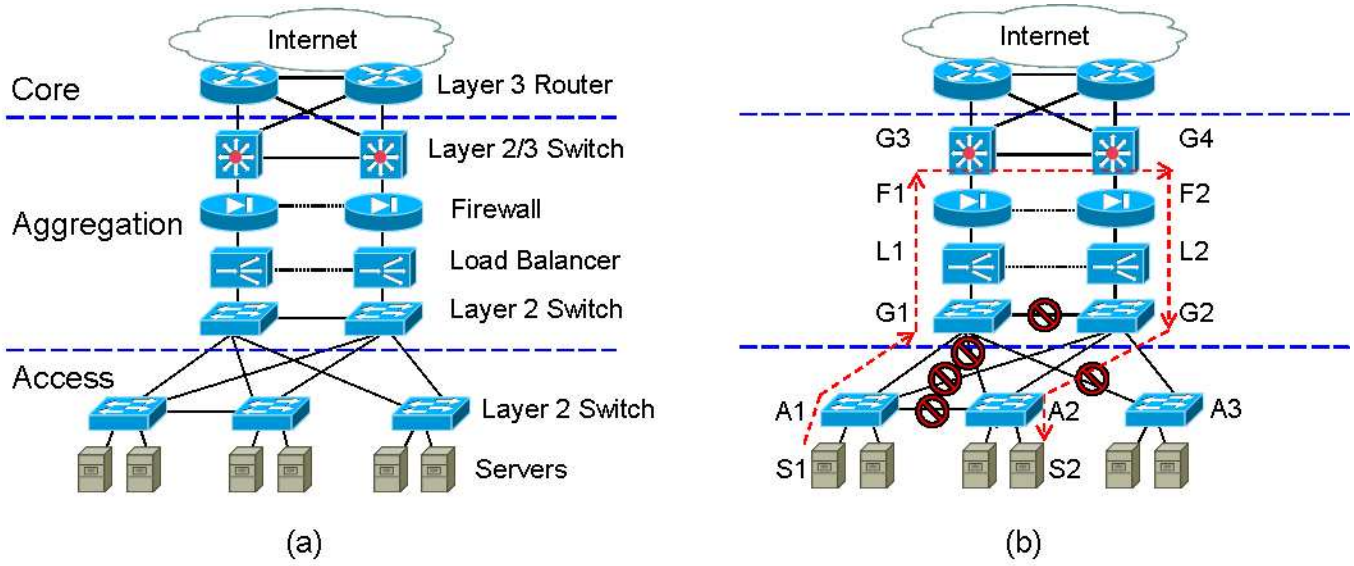

Figure 1: (a) Prevalent 3-layer data center network topology. (b) Layer-2 path between servers *S1* and *S2* including a firewall.

$S2$), marked in Figure 1(b). An immediately obvious problem with this approach is that it wastes resources by causing frames to gratuitously traverse two firewalls instead of one, and two load-balancers. An even more important problem is that there is no good mechanism to enforce this path between *S1* and *S2*. The following are three widely used mechanisms:

- *Remove physical connectivity:* By removing links $(A1, G2)$, $(A1, A2)$, $(G1, G2)$ and $(A2, G1)$, the network administrator can ensure that there is no physical layer-2 connectivity between *S1* and *S2* except via the desired path. The link $(A3, G1)$ must also be removed by the administrator or blocked out by the spanning tree protocol in order to break forwarding loops. The main drawback of this mechanism is that we lose the fault-tolerance property of the original topology, where traffic from/to *S1* can fail over to path $(G2, L2, F2, G4)$ when a middlebox or a switch on the primary path (*e.g.*, *L1* or *F1* or *G1*) fails. Identifying the subset of links to be removed from the large number of redundant links in a data center, while simultaneously satisfying different policies, fault-tolerance requirements, spanning tree convergence and middlebox failover configurations, is a very complex and possibly infeasible problem.

- *Manipulate link costs:* Instead of physically removing links, administrators can coerce the spanning tree construction algorithm to avoid these links by assigning them high link costs. This mechanism is hindered by the difficulty in predicting the behavior of the spanning tree construction algorithm across different failure conditions in a complex highly redundant network topology [21, 15]. Similar to identifying the subset of links to be removed, tweaking distributed link costs to simultaneously carve out the different layer-2 paths needed by different policy, fault-tolerance and traffic engineering requirements is hard, if not impossible.

- *Separate VLANs:* Placing *S1* and *S2* on separate VLANs that are inter-connected only at the aggregation-layer firewalls ensures that traffic between them always traverses a firewall. One immediate drawback of this mechanism is that it disallows applications, clustering protocols and virtual server mobility mechanisms requiring layer-2 adjacency [1, 16]. It also forces all applications on a server to traverse the same middlebox sequence, irrespective of policy. Guaranteeing middlebox traversal requires all desired middleboxes to be placed at all VLAN inter-connection points. Similar to the cases of removing links and manipulating link costs, overloading VLAN configuration to simultaneously satisfy many different middlebox traversal policies and traffic isolation (the original purpose of VLANs) requirements is hard.

The second option of using a standalone firewall is also implemented through the mechanisms described above, and hence suffer the same limitations. Firewall traversal can be guaranteed by placing firewalls on every possible network path between *S1* and *S2*. However, this incurs high hardware, power, configuration and management costs, and also increases the risk of traffic traversing undesired middleboxes. Apart from wasting resources, packets traversing an undesired middlebox can hinder application functionality. For example, unforeseen routing changes in the Internet, external to the data center, may shift traffic to a backup data center ingress point with an on-path firewall that filters all non-web traffic, thus crippling other applications.

The third option of incorporating firewall functionality into switches is in line with the industry trend of consolidating more and more middlebox functionality into switches. Currently, only high-end switches [5] incorporate middlebox functionality and often replace the sequence of middleboxes and switches at the aggregation layer (for example, $F1$,$L1$,$G1$ and $G3$). This option suffers the same limitations as the first two, as it uses similar mechanisms to coerce *S1*-*S2* traffic through the high-end aggregation switches

incorporating the required middlebox functionality. Sending *S1-S2* traffic through these switches even when a direct path exists further strains their resources (already oversubscribed by multiple access layer switches). They also become concentrated points of failure. This problem goes away if all switches in the data center incorporate all the required middlebox functionality. Though not impossible, this is impractical from a cost (both hardware and management) and efficiency perspective.

2.2.2 Network Inflexibility

While data centers are typically well-planned, changes are unavoidable. For example, to ensure compliance with future regulation like Sarbanes Oxley, new accounting middleboxes may be needed for email traffic. The dFence [27] DDOS attack mitigation middlebox is dynamically deployed on the path of external network traffic during DDOS attacks. New instances of middleboxes are also deployed to handle increased loads, a possibly more frequent event with the advent of on-demand instantiated virtual middleboxes.

Adding a new standalone middlebox, whether as part of a logical topology update or to reduce load on existing middleboxes, currently requires significant re-engineering and configuration changes, physical rewiring of the backup traffic path(s), shifting of traffic to this path, and finally rewiring the original path. Plugging in a new middlebox 'service' module into a single high-end switch is easier. However, it still involves significant re-engineering and configuration, especially if all middlebox expansion slots in the switch are filled up.

Network inflexibility also manifests as fate-sharing between middleboxes and traffic flow. All traffic on a particular network path is forced to traverse the same middlebox sequence, irrespective of policy requirements. Moreover, the failure of any middlebox instance on the physical path breaks the traffic flow on that path. This can be disastrous for the data center if no backup paths exist, especially when availability is more important than middlebox traversal.

2.2.3 Inefficient Resource Usage

Ideally, traffic should only traverse the required middleboxes, and be load balanced across multiple instances of the same middlebox type, if available. However, configuration inflexibility and on-path middlebox placement make it difficult to achieve these goals using existing middlebox deployment mechanisms. Suppose, spanning tree construction blocks out the $(G4, F2, L2, G2)$ path in Figure 1(b). All traffic entering the data center, irrespective of policy, flows through the remaining path $(G3, F1, L1, G1)$, forcing middleboxes $F1$ and $L1$ to process unnecessary traffic and waste their resources. Moreover, middleboxes $F2$ and $L2$ on the blocked out path remain unutilized even when $F1$ and $L1$ are struggling with overload.

3. DESIGN OVERVIEW

The policy-aware switching layer (*PLayer*) is a data center middlebox deployment proposal that aims to address the limitations of current approaches, described in the previous section. The *PLayer* achieves its goals by adhering to the following two design principles: *(i) Separating policy from reachability.* The sequence of middleboxes traversed by application traffic is explicitly dictated by data center policy and not implicitly by network path selection mechanisms like layer-2 spanning tree construction and layer-3 routing; *(ii) Taking middleboxes off the physical network path.* Rather than placing middleboxes on the physical network path at choke points in the network, middleboxes are plugged in off the physical network data path and traffic is explicitly forwarded to them. Explicitly redirecting traffic through off-path middleboxes is based on the well-known principle of indirection [30, 32, 22]. A data center network is a more apt environment for indirection than the wide area Internet due to its very low inter-node latencies.

The *PLayer* consists of enhanced layer-2 switches called policy-aware switches or *pswitches*. Unmodified middleboxes are plugged into a *pswitch* just like servers are plugged into a regular layer-2 switch. However, unlike regular layer-2 switches, *pswitches* forward frames according to the policies specified by the network administrator.

Policies define the sequence of middleboxes to be traversed by different traffic. A policy is of the form: *[Start Location, Traffic Selector]→Sequence*. The left hand side defines the applicable traffic – frames with 5-tuples (*i.e.*, source and destination IP addresses and port numbers, and protocol type) matching the *Traffic Selector* arriving from the *Start Location*. We use *frame 5-tuple* to refer to the 5-tuple of the packet within the frame. The right hand side specifies the sequence of middlebox types (not instances) to be traversed by this traffic [1].

Policies are automatically translated by the *PLayer* into *rules* that are stored at *pswitches* in rule tables. A rule is of the form *[Previous Hop, Traffic Selector] : Next Hop*. Each rule determines the middlebox or server to which traffic of a particular type, arriving from the specified previous hop, should be forwarded next. Upon receiving a frame, the *pswitch* matches it to a rule in its table, if any, and then forwards it to the next hop specified by the matching rule.

The *PLayer* relies on centralized *policy* and *middlebox* controllers to set up and maintain the rule tables at the various *pswitches*. Network administrators specify policies at the policy controller, which then reliably disseminates them to each *pswitch*. The centralized middlebox controller monitors the liveness of middleboxes and informs *pswitches* about the addition or failure of middleboxes.

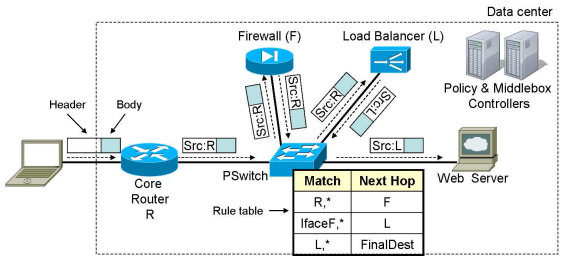

Figure 2: A simple *PLayer* consisting of only one *pswitch*.

To better understand how the *PLayer* works, we present three examples of increasing complexity that demonstrate its key functionality. In practice, the *PLayer* consists of mul-

[1]Middlebox interface information can also be incorporated into a policy. For example, frames from an external client to an internal server must enter a firewall via its *red* interface, while frames in the reverse direction should enter through the *green* interface.

tiple *pswitches* inter-connected together in complex topologies. For example, in the data center topology discussed previously, *pswitches* would replace layer-2 switches. However, for ease of exposition, we start with a simple example containing only a single *pswitch*.

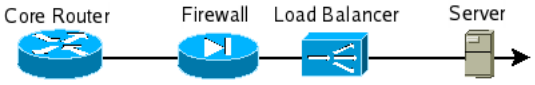

Figure 3: A simplified snippet of the data center topology in Figure 1, highlighting the on-path middlebox placement.

Figure 2 shows how the *PLayer* implements the policy induced by the physical topology in Figure 3, where all frames entering the data center are required to traverse a firewall and then a load balancer before reaching the servers. When the *pswitch* receives a frame, it performs the following three operations: (i) Identify the previous hop traversed by the frame, (ii) Determine the next hop to be traversed by the frame, and (iii) Forward the frame to its next hop. The *pswitch* identifies frames arriving from the core router and the load balancer based on their source MAC addresses (R and L, respectively). Since the firewall does not modify the MAC addresses of frames passing through it, the *pswitch* identifies frames coming from it based on the ingress interface (*IfaceF*) they arrive on. The *pswitch* determines the next hop for the frame by matching its previous hop information and 5-tuple against the rules in the rule table. In this example, the policy translates into the following three rules – (i) $[R, *] : F$, (ii) $[IfaceF, *] : L$, and (iii) $[L, *] : FinalDest$. The first rule specifies that every frame entering the data center (*i.e.*, every frame arriving from core router R) should be forwarded to the firewall (F). The second rule specifies that every frame arriving from the firewall should be forwarded to the load balancer (L). The third rule specifies that frames arriving from the load balancer should be sent to the final destination, *i.e.*, the server identified by the frame's destination MAC address. The *pswitch* forwards the frame to the next hop determined by the matching rule, encapsulated in a frame explicitly addressed to the next hop. It is easy to see that the *pswitch* correctly implements the original policy through these rules, *i.e.*, every incoming frame traverses the firewall followed by the load balancer.

Multiple equivalent instances of middleboxes are often deployed for scalability and fault-tolerance. Figure 4 shows how the *PLayer* can load balance incoming traffic across two equivalent firewalls, $F1$ and $F2$. The first rule in the table specifies that incoming frames can be sent either to firewall $F1$ or to firewall $F2$. Since the firewall maintains per-flow state, the *pswitch* uses a flow- direction-agnostic consistent hash on a frame's 5-tuple to select the same firewall instance for all frames in both forward and reverse directions of a flow.

The more complex example in Figure 5 illustrates how the *PLayer* supports different policies for different applications and how forwarding load is spread across multiple *pswitches*. Web traffic has the same policy as before, while Enterprise Resource Planning (ERP) traffic is to be scrubbed by a dedicated custom firewall (W) followed by an Intrusion Prevention Box (*IPB*). The middleboxes are distributed across the two *pswitches* A and B. The rule table at each *pswitch* has rules that match frames coming from the entities connected

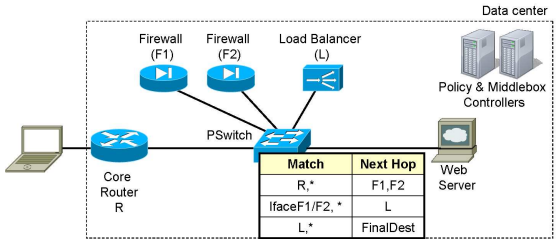

Figure 4: Load balancing traffic across two equivalent middlebox instances.

to it. For example, rules at *pswitch* A match frames coming from middleboxes $F1$ and L, and the core router R. For sake of simplicity, we assume that all frames with TCP port 80 are part of web traffic and all others are part of ERP traffic. A frame (say, an ERP frame) entering the data center first reaches *pswitch* A. *Pswitch* A looks up the most specific rule for the frame ($[R, *] : W$) and forwards it to the next hop (W). The *PLayer* uses existing layer-2 mechanisms (*e.g.*, spanning tree based Ethernet forwarding) to forward the frame to its next hop, instead of inventing a new forwarding mechanism. *Pswitch* B receives the frame after it is processed by W. It looks up the most specific rule from its rule table ($[IfaceW, *] : IPB$) and forwards the frame to the next hop (IPB). An HTTP frame entering the data center matches different rules and thus follows a different path.

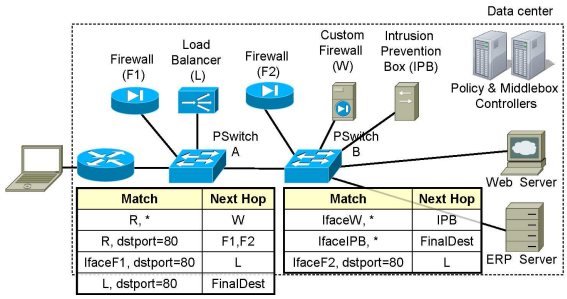

Figure 5: Different policies for web and ERP applications.

The three examples discussed in this section provide a high level illustration of how the *PLayer* achieves the three desirable properties of correctness, flexibility and efficiency. The explicit separation between policy and the physical network topology simplifies configuration. The desired logical topologies can be easily implemented by specifying appropriate policies at the centralized policy controller, without tweaking spanning tree link costs and IP gateway settings distributed across various switches and servers. By explicitly redirecting frames only through the middleboxes specified by policy, the *PLayer* guarantees that middleboxes are neither skipped nor unnecessarily traversed. Placing middleboxes off the physical network path prevents large scale traffic shifts on middlebox failures and ensures that middlebox resources are not wasted serving unnecessary traffic or get stuck on inactive network paths.

The *PLayer* operates at layer-2 since data centers are predominantly layer-2 [16]. It re-uses existing tried and tested layer-2 mechanisms to forward packets between two points in the network rather than inventing a custom forwarding

mechanism. Furthermore, since middleboxes like firewalls are often not explicitly addressable, the *PLayer* relies on simple layer-2 mechanisms described in Section 4.2.3 to forward frames to these middleboxes, rather than more heavy-weight layer-3 or higher mechanisms.

In the next three sections, we discuss how the *PLayer* addresses the three main challenges listed below:

(i) Minimal Infrastructure Changes: Support existing middleboxes and servers without any modifications and minimize changes to network infrastructure like switches.

(ii) Non-transparent Middleboxes : Handle middleboxes that modify frames while specifying policies and while ensuring that all frames in both forward and reverse directions of a flow traverse the same middlebox instances.

(iii) Correct Traversal Under Churn : Guarantee correct middlebox traversal during middlebox churn and conflicting policy updates.

4. MINIMAL INFRASTRUCTURE CHANGES

Minimizing changes to existing network forwarding infrastructure and supporting unmodified middleboxes and servers is crucial for *PLayer* adoption in current data centers. In addition to describing how we meet this challenge, in this section, we also explain a *pswitch*'s internal structure and operations, and thus set the stage for describing how we solve other challenges in subsequent sections.

4.1 Forwarding Infrastructure

The modular design of *pswitches*, reliance on standard data center path selection mechanisms to forward frames, and encapsulation of forwarded frames in new Ethernet-II frames help meet the challenge of minimizing changes to the existing data center network forwarding infrastructure.

4.1.1 *Pswitch* Design & Standard Forwarding

Figure 6 shows the internal structure of a *pswitch* with N interfaces. For ease of explanation, each physical interface is shown as two separate logical interfaces – an input interface and an output interface. A *pswitch* consists of two independent parts – the Switch Core and the Policy Core:

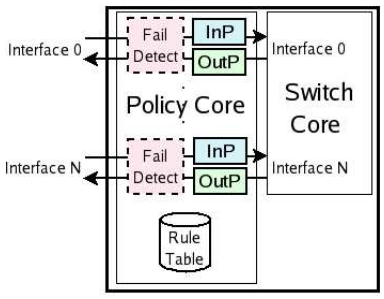

Figure 6: Internal components of a *pswitch*.

(i) Switch Core : The Switch Core provides regular Ethernet switch functionality – it forwards Ethernet frames based on their destination MAC addresses, performs MAC address learning and participates in the Spanning Tree Protocol to construct a loop-free forwarding topology.

(ii) Policy Core : The Policy Core redirects frames [2] to the middleboxes dictated by policy. It consists of multiple modules: The RuleTable stores the rules used for matching and forwarding frames. Each *pswitch* interface has an inP, an outP and a FailDetect module associated with it. An inP module processes a frame as it enters a *pswitch* interface – it identifies the frame's previous hop, looks up the matching rule and emits it out to the corresponding Switch Core interface for regular forwarding to the next hop specified by the rule. An outP module processes a frame as it exits a *pswitch* interface, decapsulating or dropping it as explained later in the section. The FailDetect module of a *pswitch* interface monitors the liveness of the connected middlebox (if any) using standard mechanisms like ICMP pings, layer-7 content snooping, SNMP polling, TCP health checks, and reports to the middlebox controller.

The Switch Core appears like a regular Ethernet switch to the Policy Core, while the Policy Core appears like a multi-interface device to the Switch Core. This clean separation allows us to re-use existing Ethernet switch functionality in constructing a *pswitch* with minimal changes, thus simplifying deployment. The Switch Core can also be easily replaced with an existing non-Ethernet forwarding mechanism, if required by the existing data center network infrastructure.

A frame redirected by the Policy Core is encapsulated in a new Ethernet-II frame identified by a new *EtherType* code. The outer frame's destination MAC address is set to that of the next hop middlebox or server, and its source MAC address is set to that of the original frame or the last middlebox instance traversed, if any. We encapsulate rather than overwrite as preserving the original MAC addresses is often required for correctness (*e.g.*, firewalls may filter on source MAC addresses; load-balancers may set the destination MAC address to that of a dynamically chosen server). Although encapsulation may increase frame size beyond the 1500 byte MTU, it is below the limit accepted by most switches. For example, Cisco switches allow 1600 byte 'baby giants'.

4.1.2 Incremental Deployment

Incorporating the *PLayer* into an existing data center does not require a fork-lift upgrade of the entire network. Only switches which connect to the external network and those into which servers requiring middlebox traversal guarantees are plugged in, need to be converted to *pswitches*. Other switches need not be converted if they can be configured or modified to treat encapsulated frames with the new *EtherType* as regular Ethernet frames. Middleboxes can also be plugged into a regular switch. However, transparent middleboxes must be accompanied by the inline SrcMacRewriter device (described in Section 4.2.2). If the data center contains backup switches and redundant paths, *pswitches* can be smoothly introduced without network downtime by first converting the backup switches to *pswitches*.

4.2 Unmodified Middleboxes and Servers

Pswitches address the challenge of supporting unmodified middleboxes and servers in three ways – (i) Ensure that only relevant frames in standard Ethernet format reach middleboxes and servers, (ii) Use only non-intrusive techniques to identify a frame's previous hop, and (iii) Support varied middlebox addressing requirements.

[2]Only frames containing IP packets are considered. Non-IP frames like ARP requests are forwarded by the Switch Core as in regular Ethernet switches.

4.2.1 Frames reaching Middleboxes and Servers

The OUTP module of a *pswitch* interface directly connected to a middlebox or server emits out a unicast frame only if it is MAC addressed to the connected middlebox or server. Dropping other frames, which may have reached the *pswitch* through standard Ethernet broadcast forwarding, avoids undesirable middlebox behavior (*e.g.*, a firewall can terminate a flow by sending TCP RSTs if it receives an unexpected frame). The OUTP module also decapsulates the frames it emits and thus the middlebox or server receives standard Ethernet frames it can understand.

4.2.2 Previous Hop Identification

A *pswitch* does not rely on explicit middlebox support or modifications for identifying a frame's previous hop. The previous hop of a frame can be identified in three possible ways: (i) source MAC address if the previous hop is a middlebox that changes the source MAC address, (ii) *pswitch* interface on which the frame arrives if the middlebox is directly attached to the *pswitch*, or (iii) VLAN tag if the data center network has been divided into different functional zones using VLANs (*i.e.*, external web servers, firewalls, etc.). If none of the above 3 conditions hold (for example, in a partial *pswitch* deployment where middleboxes are plugged into regular Ethernet switches), then we install a simple stateless in-line device, SRCMACREWRITER, in between the middlebox and the regular Ethernet switch to which it is connected. SRCMACREWRITER inserts a special source MAC address that can uniquely identify the middlebox into frames emitted by the middlebox, as in option (i) above.

4.2.3 Middlebox Addressing

Many middleboxes like firewalls operate inline with traffic and do not require traffic to be explicitly addressed to them at layer-2 or layer-3. Moreover, for many such middleboxes, traffic *cannot* be explicitly addressed to them, as they lack a MAC address. We solve this problem by assigning a fake MAC address to such a middlebox instance when it is registered with the middlebox controller. The fake MAC address is used as the destination MAC of encapsulated frames forwarded to it. If the middlebox is directly connected to a *pswitch*, the *pswitch* also fills in this MAC address in the source MAC field of encapsulated frames forwarded to the next hop. If it is not directly attached to a *pswitch*, this MAC address is used by the SRCMACREWRITER device. In all cases, the middlebox remains unmodified.

In contrast, some middleboxes like load balancers often require traffic to be explicitly addressed to them at layer-2, layer-3 or both. We support middleboxes that require layer-3 addressing by using per-segment policies(Section 5). Middleboxes that require layer-2 addressing are supported by rewriting the destination MAC addresses of frames (if necessary) before they are emitted out to such middleboxes.

5. NON-TRANSPARENT MIDDLEBOXES

Non-transparent middleboxes, *i.e.*, middleboxes that modify frame headers or content (*e.g.*, load balancers), make end-to-end policy specification and consistent middlebox instance selection challenging. By using per-segment policies, we support non-transparent middleboxes in policy specification. By enhancing policy specifications with hints that indicate which frame header fields are left untouched by non-transparent middleboxes, we enable the middlebox instance selection mechanism at a *pswitch* to select the same middlebox instances for all packets in both forward and reverse directions of a flow, as required by stateful middleboxes like firewalls and load balancers.

Middleboxes may modify frames reaching them in different ways. MAC-address modification aids previous hop identification but does not affect traffic classification or middlebox instance selection since they are independent of layer-2 headers. Similarly, payload modification does not affect policy specification or middlebox instance selection, unless deep packet inspection is used for traffic classification. Traffic classification and flow identification mainly rely on a frame's 5-tuple. Middleboxes that fragment frames do not affect policy specification or middlebox instance selection as long as the frame 5-tuple is the same for all fragments. In the remainder of this section, we describe how we support middleboxes that modify frame 5-tuples. We also provide the details of our basic middlebox instance selection mechanism in order to provide the context for how non-transparent middleboxes and middlebox churn (Section 6.2) affect it.

5.1 Policy Specification

Middleboxes that modify frame 5-tuples are supported in policy specification by using *per-segment policies*. We define the bi-directional end-to-end traffic between two nodes, *e.g.*, A and B, as a *flow*. Figure 7 depicts a flow passing through a firewall unmodified, and then a load balancer that rewrites the destination IP address IP_B to the address IP_W of an available web server. Frame modifications by the load balancer preclude the use of a single concise *Selector*. Per-segment policies 1 and 2 shown in Figure 7, each matching frames during a portion of their end-to-end flow, together define the complete policy. Per-segment policies also enable policy definitions that include middleboxes requiring traffic to be explicitly addressed to them at the IP layer.

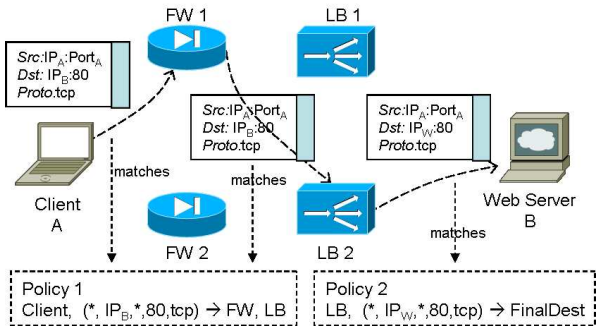

Figure 7: Policies for different segments of the logical middlebox sequence traversed by traffic between A and B.

5.2 Middlebox Instance Selection

The *PLayer* uses consistent hashing to select the same middlebox instance for all frames in both forward and reverse directions of a flow. A frame's 5-tuple identifies the flow to which it belongs. A flow-direction agnostic hash value h is calculated over the frame's 5-tuple. The ids of all live instances of the desired middlebox type are arranged in a ring , and the instance whose id is closest to h in the counter-clockwise direction is selected [31]. Based on middlebox semantics and functionality, network administrators

indicate the frame 5-tuple fields to be used in middlebox instance selection along with the policies. The entire 5-tuple is used for middleboxes that do not modify frames.

When middleboxes modify the frame 5-tuple, instance selection can no longer be based on the entire 5-tuple. For example, in the $A \rightarrow B$ flow direction in Figure 7, the load balancer instance is selected when the frame 5-tuple is (IP_A, IP_B, $Port_A$, $Port_B$, tcp). In the $B \rightarrow A$ reverse direction, the load balancer instance is to be selected when the frame 5-tuple is (IP_W, IP_A, $Port_B$, $Port_A$, tcp). The policy hints that a load balancer instance should be selected only based on frame 5-tuple fields unmodified by the load balancer, *viz.*, IP_A, $Port_A$, $Port_B$ and tcp (although source and destination fields are interchanged).

We assume that a middlebox modifying the entire 5-tuple always changes the source IP address to its own IP address, so that regular layer-3 routing can be used to ensure that reverse traffic reaches the same middlebox instance. Although we are not aware of any middleboxes that violate this assumption, we discuss how *pswitches* enhanced with per-flow state can support such middleboxes in [24].

6. GUARANTEES UNDER CHURN

In this section, we argue that the *PLayer* guarantees correct middlebox traversal under churn. There are several entities in the system that can experience churn, including the network, policies, and middleboxes. Since *pswitches* explicitly forward frames to middleboxes based on policy, churn at the network layer alone (*e.g.*, *pswitch* or link failures, new link activation, link rewiring) will not violate middlebox traversal guarantees. In this section, we discuss the traversal guarantees under policy and middlebox churn. Our technical report [24] presents additional details and a formal analysis of *PLayer* operations and churn guarantees.

6.1 Policy Churn

Network administrators update policies at a centralized policy controller when the logical topology of the data center network needs to be changed. In this section, we first briefly describe our policy dissemination mechanism. We then discuss how we prevent incorrect middlebox traversal during conflicting policy updates.

6.1.1 Policy Dissemination

The policy controller reliably disseminates policy information over separate TCP connections to each *pswitch*. After all *pswitches* receive the complete policy update, the policy controller signals each *pswitch* to adopt it. The signal, which is conveyed in a single packet, has a better chance of synchronously reaching the different *pswitches* than the multiple packets carrying the policy updates. Similar to network map dissemination [26], the policy version number recorded inside encapsulated frames is used to further improve synchronization – a *pswitch* that has not yet adopted the latest policy update will immediately adopt it upon receiving a frame stamped with the latest policy version number.

Policy dissemination over separate TCP connections to each *pswitch* scales well if the number of *pswitches* in the data center is small (a few 100s), assuming infrequent policy updates (a few times a week). If the number of *pswitches* is very large, then the distributed reliable broadcast mechanism suggested by RCP [18] is used for policy dissemination.

6.1.2 Conflicting Policy Updates

Even a perfectly synchronized policy dissemination mechanism cannot prevent some frames from violating middlebox traversal guarantees during a conflicting policy update. For example, suppose that the current policy (version 1) mandates all traffic to traverse a load-balancer and then a firewall, while the new policy (version 2) mandates all traffic to traverse a firewall followed by a load-balancer. During the brief policy transition period, suppose *pswitch A* (using policy version 1) redirects a frame to a load-balancer attached to *pswitch B* (also using version 1). Before the frame arrives back at B after processing by the load-balancer, B adopts policy version 2. It subsequently sends the frame directly to its final destination (thinking that the load-balancer is the last hop in the sequence), bypassing the firewall and causing a security vulnerability.

We prevent the violation of middlebox traversal guarantees by specifying the middlebox sequence of a conflicting policy update in terms of new intermediate middlebox types. For the example above, the middlebox sequence in the new policy is specified as (*firewall'*,*load balancer'*). Although intermediate middlebox types are functionally identical to the original types, they have separate instances. Frames redirected under the new policy traverse these separate instances. Hence, a *pswitch* will never confuse these frames with those redirected under the original policy, and thus avoids incorrect forwarding.

6.2 Middlebox Churn

A *pswitch* identifies flows based on the 5-tuple common to all frames of the flow. A middlebox instance is selected for a frame by using flow direction agnostic consistent hashing on its 5-tuple (Section 5.2). This is sufficient for consistent middlebox instance selection when no new middlebox instances are added. When a running middlebox instance fails, all flows served by it are automatically shifted to an active standby, if available, or are shifted to some other instance determined by consistent hashing. If flows are shifted to a middlebox instance that does not have state about the flow, it may be dropped, thus affecting availability. However, this is unavoidable even in existing network infrastructures and is not a limitation of the *PLayer*.

Adding a new middlebox instance changes the number of instances (n) serving as targets for consistent hashing. As a result, $\frac{1}{2n}$ of the flows are shifted to the newly added instance, on average. Stateful middlebox instances like firewalls may drop the reassigned flow and briefly impede network availability. If n is large (say 5), only a small fraction of flows (10%) are affected. If such relatively small and infrequent pre-planned disruptions are unacceptable for the data center, flows can be pinned to middlebox instances by enhancing *pswitches* with per-flow state [24].

7. IMPLEMENTATION AND EVALUATION

In this section we briefly describe our prototype implementation of the *PLayer* and subsequently demonstrate its functionality and flexibility under different network scenarios, as well as provide preliminary performance benchmarks.

7.1 Implementation

We have prototyped *pswitches* in software using Click [25]. An unmodified Click *Etherswitch* element formed the Switch Core, while the Policy Core was implemented in 5500 lines of

C++. Each interface of the Policy Core plugs into the corresponding interface of the Etherswitch element, maintaining the modular *pswitch* design described in Section 4.

Due to our inability to procure expensive hardware middleboxes for testing, we used commercial quality software middleboxes running on standard Linux PCs: (i) *Netfilter/iptables* [13] based firewall, (ii) Bro [29] intrusion detection system, and (iii) *BalanceNG* [2] load balancer. We used the Net-SNMP [8] package for implementing SNMP-based middlebox liveness monitoring. Instead of inventing a custom policy language, we leveraged the flexibility of XML to express policies in a simple human-readable format. The middlebox controller, policy controller, and web-based configuration GUI were implemented using Ruby-On-Rails [12].

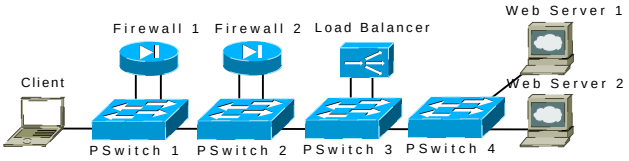

Figure 8: Physical topology on the DETER testbed used to demonstrate *PLayer* functionality.

7.2 Validation of Functionality

We validated the functionality and flexibility of the *PLayer* using computers on the DETER [17] testbed, connected together as shown in Figure 8. The physical topology was constrained by the maximum number of Ethernet interfaces (4) available on individual testbed computers. Using simple policy changes to the *PLayer*, we implemented the different logical network topologies shown in Figure 9, without rewiring the physical topology or taking the system offline. Not all devices were used in every logical topology.

Topology A→B: Logical topology A represents our starting point and the most basic topology – a client directly communicates with a web server. By configuring the policy *[Client, (*,IP_{web1},*,80,tcp)] → firewall* at the policy controller, we implemented logical topology B, in which a firewall is inserted in between the client and the web server. We validated that all client-web server traffic flowed through the firewall by monitoring the links. We also observed that all flows were dropped when the firewall failed (was turned off).

Topology B→C: Adding a second firewall, Firewall 2, in parallel with Firewall 1, in order to split the processing load resulted in logical topology C. Implementing logical topology C required no policy changes. The new firewall instance was simply registered at the middlebox controller, which then immediately informed all four *pswitches*. Approximately half of the existing flows shifted from Firewall 1 to Firewall 2 upon its introduction. However, no flows were dropped as the filtering rules at Firewall 2 were configured to temporarily allow the pre-existing flows. Configuring firewall filtering behavior is orthogonal to *PLayer* configuration.

Topology C→B→C: To validate the correctness of *PLayer* operations when middleboxes fail, we took down one of the forwarding interfaces of Firewall 1, thus reverting to logical topology B. The SNMP daemon detected the failure on Firewall 1 in under 3 seconds and immediately reported it to all *pswitches* via the middlebox controller. All existing and new flows shifted to Firewall 2 as soon as the failure report was received. After Firewall 1 was brought back alive, the *pswitches* restarted balancing traffic across the two firewall instances in under 3 seconds.

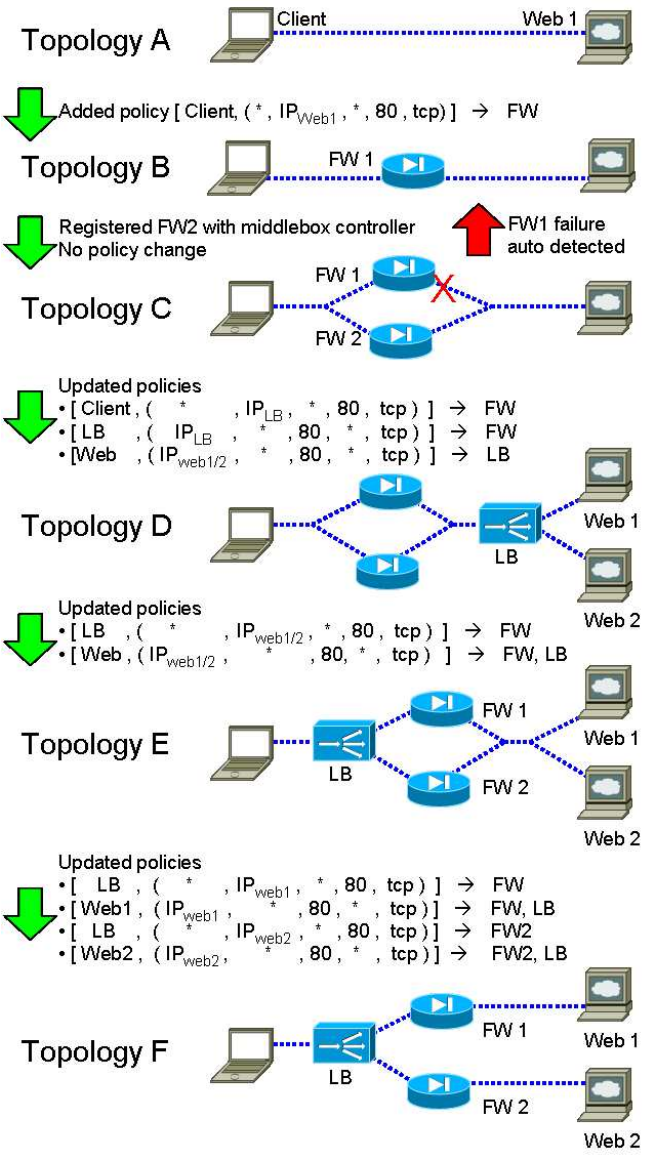

Figure 9: Logical topologies used to demonstrate *PLayer* functionality.

Topology C→D: We next inserted a load balancer in between the firewalls and web server 1, and added a second web server, yielding logical topology D. Clients send HTTP packets to the load balancer's IP address IP_{LB}, instead of a web server IP address (as required by the load balancer operation mode). The load balancer rewrites the destination IP address to that of one of the web servers, selected in a round-robin fashion. To implement this logical topology, we specified the policy *[Client, (*,IP_{LB},*,80,tcp)] → firewall* and the corresponding reverse policy for the client-load balancer segment of the path. The load balancer, which automatically forwards packets to a web server instance, is not explicitly listed in the middlebox sequence because it is the end point to which packets are addressed. We also specified

the policy *[Web, ($IP_{web1/2}$,*,80,*,tcp)]→load balancer*. This policy enabled us to force the web servers' response traffic to pass through the load balancer without reconfiguring the default IP gateway on the web servers, as done in current best practices. We verified that the client-web server traffic was balanced across the two firewalls and the two web servers. We also verified the correctness of *PLayer* operations under firewall, load balancer and web server failure.

Topology D→E: In order to demonstrate the *PLayer* 's flexibility, we flipped the order of the firewalls and the load balancer in logical topology D, yielding topology E. Implementing this change simply involves updating the policies to *[LB, (*,$IP_{web1/2}$,*,80,tcp)] → firewall* and *[Web, ($IP_{web1/2}$,*,80,*,tcp)] → firewall,* load balancer. We do not specify a policy to include the load balancer on the client to web server path, as the HTTP packets sent by the client are addressed to the load balancer, as before.

Topology E→F: To further demonstrate the *PLayer*'s flexibility, we updated the policies to implement logical topology F, in which Firewall 1 solely serves web server 1 and Firewall 2 solely serves web server 2. This topology is relevant when the load balancer intelligently redirects different types of content requests (for example, static versus dynamic) to different web servers, thus requiring different types of protection from the firewalls. To implement this topology, we changed the middlebox type of Firewall 2 to a new type $firewall_2$, at the middlebox controller. We then updated the forward direction policies to *[LB, (*,IP_{web1},*,80,tcp)] → firewall* and *[LB, (*,IP_{web2},*,80,tcp)]* → $firewall_2$, and modified the reverse policies accordingly.

The logical topology modifications and failure scenarios studied here are orthogonal to the complexity of the physical topology. We also validated the *PLayer* on a more complex topology [24] that emulates the popular data center topology shown in Figure 1.

7.3 Benchmarks

In this section, we provide preliminary throughput and latency benchmarks for our prototype *pswitch* implementation, relative to standard software Ethernet switches and on-path middlebox deployment. Our initial implementation focused on feasibility and functionality, rather than optimized performance. While the performance of a software *pswitch* may be improved by code optimization, achieving line speeds is unlikely. Inspired by the 50x speedup obtained when moving from a software to hardware switch prototype in [20], we plan to prototype *pswitches* on the NetFPGA [9] boards. We believe that the hardware *pswitch* implementation will have sufficient switching bandwidth to support frames traversing the *pswitch* multiple times due to middleboxes and will be able to operate at line speeds.

Our prototype *pswitch* achieved 82% of the TCP throughput of a regular software Ethernet switch, with a 16% increase in latency. Figure 10(a) shows the simple topology used in this comparison experiment, with each component instantiated on a separate 3GHz Linux PC. We used *nuttcp* [10] and *ping* for measuring TCP throughput and latency, respectively. The *pswitch* and the standalone Click Etherswitch, devoid of any *pswitch* functionality, saturated their PC CPUs at throughputs of 750 Mbps and 912 Mbps, respectively, incurring latencies of 0.3 ms and 0.25 ms.

Compared to an on-path middlebox deployment, off-path deployment using our prototype *pswitch* achieved 40% of the throughput at double the latency (Figure 10(b)). The on-path firewall deployment achieved an end-to-end throughput of 932 Mbps and a latency of 0.3 ms, while the *pswitch*-based firewall deployment achieved 350 Mbps with a latency of 0.6 ms. Although latency doubled as a result of multiple *pswitch* traversals, the sub-millisecond latency increase is in general much smaller than wide-area Internet latencies. The throughput decrease is a result of frames that arrived on different *pswitch* interfaces traversing the same already saturated CPU. Hardware-based *pswitches* with multi-gigabit switching fabrics should not suffer this throughput drop.

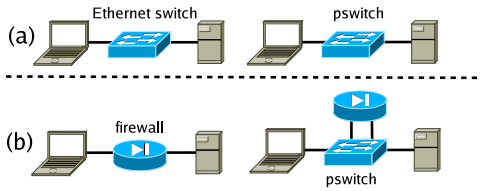

Figure 10: Topologies used in benchmarking *pswitch* performance.

Microbenchmarking showed that a *pswitch* takes between 1300 and 7000 CPU ticks (1 tick $\approx \frac{1}{3000}$ microsecond on a 3GHz CPU) to process a frame. A frame entering a *pswitch* input interface from a middlebox or server is processed and emitted out of the appropriate *pswitch* output interfaces in 6997 CPU ticks. Approximately 50% of the time is spent in rule lookup (from a 25 policy database) and middlebox instance selection, and 44% on frame encapsulation. Overheads of frame classification and frame handoff between different Click elements consumed the remaining INP processing time. An encapsulated frame reaching the *pswitch* directly attached to its destination server/middlebox was decapsulated and emitted out to the server/middlebox in 1312 CPU ticks.

8. LIMITATIONS

The following are the main limitations of the *PLayer*:

(i) Indirect Paths : Similar to some existing VLAN-based middlebox deployment mechanisms, redirecting frames to off-path middleboxes causes them to follow paths that are less efficient than direct paths formed by middleboxes physically placed in sequence. We believe that the bandwidth overhead and slight latency increase are insignificant in a bandwidth-rich low latency data center network.

(ii) Policy Specification : Traffic classification and policy specification using frame 5-tuples is not trivial. However, it is simpler than the current ad-hoc middlebox deployment best practices. Network administrators specify policies using a configuration GUI at the centralized policy controller. Static policy analysis flags policy inconsistencies and misconfiguration (*e.g.*, policy loops), and policy holes (*e.g.*, absence of policy for SSH traffic). Since every *pswitch* has the same policy set, policy specification is also less complex than configuring a distributed firewall system.

(iii) Incorrect Packet Classification : 5-tuples alone may be insufficient to distinguish different types of traffic if it is obfuscated or uses unexpected transport ports. For example, a *pswitch* cannot identify HTTPS traffic unexpectedly sent to port 80 instead of 443, and forward it to an SSL offload box. Since such unexpected traffic is likely to be

dropped by the destinations themselves, classification inaccuracy is not a show-stopper. However, it implies that if deep packet inspection capable firewalls are available, then policies must be defined to forward all traffic to them, rather than allowing traffic to skip firewalls based on their 5-tuples.

(iv) Incorrectly Wired Middleboxes : The *PLayer* requires middleboxes to be correctly wired for accurate previous hop identification and next hop forwarding. For example, if a firewall is plugged into *pswitch* interface 5 while the *pswitch* thinks that an intrusion prevention box is plugged in there, then frames emitted to the intrusion prevention box will reach the firewall. Even existing middlebox deployment mechanisms critically rely on middleboxes being correctly wired. Since middleboxes are few in number compared to servers, we expect them to be carefully wired.

(v) Unsupported Policies : The *PLayer* does not support policies that require traffic to traverse the same type of middlebox multiple times (*e.g.*, *[Core Router, (*,*,*,80,tcp)] → firewall, load balancer, firewall*). The previous hop determination mechanism used by *pswitches* cannot distinguish the two firewalls. We believe that such policies are rare, and hence tradeoff complete policy expressivity for simplicity of design. Note that policies involving different firewall types (*e.g.*, *[Core Router, (*,*,*,80,tcp)] → external firewall, load-balancer, internal firewall*) are supported.

9. RELATED WORK

Indirection is a well-known principle in computer networking. The Internet Indirection Infrastructure [30] and the Delegation Oriented Architecture [32] provide layer-3 and above mechanisms that enable packets to be explicitly redirected through middleboxes located anywhere on the Internet. Due to pre-dominantly layer-2 topologies within data centers, the *PLayer* is optimized to use indirection at layer-2. SelNet [22] is a general-purpose network architecture that provides indirection support at layer '2.5'. In SelNet, endhosts implement a multi-hop address resolution protocol that establishes per flow next-hop forwarding state at middleboxes. The endhost and middlebox modifications required make SelNet impractical for current data centers. Using per-flow multi-hop address resolution to determine the middleboxes to be imposed is slow and inefficient, especially in a data center environment where policies are apriori known. The *PLayer* does not require endhost or middlebox modifications. A *pswitch* can quickly determine the middleboxes to be traversed by the packets in a flow without performing multi-hop address resolution.

Separating policy from reachability and centralized management of networks are goals our work shares with many existing proposals like 4D [23] and Ethane [20]. 4D concentrates on general network management and does not provide mechanisms to guarantee middlebox traversal. Instantiations of 4D like the Routing Control Platform (RCP) [18] focus on reducing the complexity of iBGP inside an AS and not on Data Centers. Unlike 4D, the *PLayer* does not mandate centralized computation of the forwarding table – it works with existing network path selection protocols running at switches and routers, whether centralized or distributed.

Ethane [20] is a proposal for centralized management and security of enterprise networks. An Ethane switch forwards the first packet of a flow to a centralized domain controller. This controller calculates the path to be taken by the flow, installs per-flow forwarding state at the Ethane switches on the calculated path and then responds with an encrypted source route that is enforced at each switch. Although not a focus for Ethane, off-path middleboxes can be imposed by including them in the source routes. In the *PLayer*, each *pswitch* individually determines the next hop of a packet without contacting a centralized controller, and immediately forwards packets without waiting for flow state to be installed at *pswitches* on the packet path. Ethane has been shown to scale well for large enterprise networks (20000 hosts and 10000 new flows/second). However, even if client authentication and encryption are disabled, centrally handing out and installing source routes in multiple switches at the start of each flow may not scale to large data centers with hundreds of switches, serving 100s of thousands of simultaneous flows[3]. The distributed approach taken by the *PLayer* makes it better suited for scaling to a large number of flows. For short flows (like single packet heartbeat messages or 2-packet DNS query/response pairs), Ethane's signaling and flow setup overhead can be longer than the flow itself. The prevalence of short flows [33] and single packet DoS attacks hinder the scalability of the flow tables in Ethane switches. Although Ethane's centralized controller can be replicated for fault-tolerance, it constitutes one more component on the critical path of all new flows, thereby increasing complexity and chances of failure. The *PLayer* operates unhindered under the current policies even if the policy controller fails.

Some high-end switches like the Cisco Catalyst 6500 [5] allow various middleboxes to be plugged into the switch chassis. Through appropriate VLAN configurations on switches and IP gateway settings on end servers, these switches offer limited and indirect control over the middlebox sequence traversed by traffic. Middlebox traversal in the *PLayer* is explicitly controlled by policies configured at a central location, rather than implicitly dictated by complex configuration settings spread across different switches and end servers. Crucial middleboxes like firewalls plugged into a high-end switch may be bypassed if traffic is routed around it during failures. Unlike the *PLayer*, only specially designed middleboxes can be plugged into the switch chassis. Concentrating all middleboxes in a single (or redundant) switch chassis creates a central point of failure. Increasing the number of middleboxes once all chassis slots are filled up is difficult.

MPLS traffic engineering capabilities can be overloaded to force packets through network paths with middleboxes. This approach not only suffers from the drawbacks of on-path middlebox placement discussed earlier, but also requires middleboxes to be modified to relay MPLS labels.

Policy Based Routing (PBR) [11], a feature present in some routers, enables packets matching pre-specified policies to be assigned different QoS treatment or to be forwarded out through specified interfaces. Although PBR provides no direct mechanism to impose middleboxes, it can be used along with standard BGP/IGP routing and tunneling to impose middleboxes. dFence [27], a DoS mitigation system which on-demand imposes DoS mitigation middleboxes on the data path to servers under DOS attack, uses this approach. The *PLayer* does not rely on configurations spread

[3]We estimate that Google receives over 400k search queries per second, assuming 80% of search traffic is concentrated in 50 peak hours a week [14]. Multiple flows from each search query and from other services like GMail are likely to result in each Google data center serving 100s of thousands of new flows/second.

across different routing and tunneling mechanisms. It instead provides a simple and direct layer-2 mechanism to impose middleboxes on the data path. A layer-2 mechanism is more suitable for imposing middleboxes in a data center, as data centers are pre-dominantly layer-2 and many middleboxes cannot even be addressed at the IP layer.

10. CONCLUSION

The recent rapid growth in the number, importance, scale and complexity of data centers and their very low latency, high bandwidth network infrastructures open up challenging avenues of research. In this paper, we proposed the policy-aware switching layer (*PLayer*), a new way to deploy middleboxes in data centers. The *PLayer* leverages the data center network's conduciveness for indirection to explicitly redirect traffic to unmodified off-path middleboxes specified by policy. Unlike current practices, our approach guarantees correct middlebox traversal under all network conditions, and enables more efficient and flexible network topologies. We demonstrated the functionality and feasibility of our proposal through a software prototype deployed on a small testbed.

11. REFERENCES

[1] Architecture Brief: Using Cisco Catalyst 6500 and Cisco Nexus 7000 Series Switching Technology in Data Center Networks. http://www.cisco.com/en/US/prod/collateral/switches/ps9441/ps9402/ps9512/White_Paper_C17-449427.pdf.

[2] BalanceNG: The Software Load Balancer. http://www.inlab.de/balanceng.

[3] Beth Israel Deaconess Medical Center. Network Outage Information. http://home.caregroup.org/templatesnew/departments/BID/network_outage/.

[4] BladeLogic Sets Standard for Data Center Automation and Provides Foundation for Utility Computing with Operations Manager Version 5. Business Wire, Sept 15, 2003, http://findarticles.com/p/articles/mi_m0EIN/is_2003_Sept_15/ai_107753392/pg_2.

[5] Cisco Catalyst 6500 Series Switches Solution. http://www.cisco.com/en/US/products/sw/iosswrel/ps1830/products_feature_guide09186a008008790d.html.

[6] Cisco Systems, Inc. Spanning Tree Protocol Problems and Related Design Considerations. http://www.cisco.com/warp/public/473/16.html.

[7] Microsoft: Datacenter Growth Defies Moore's Law. InfoWorld, April 18, 2007, http://www.pcworld.com/article/id,130921/article.html.

[8] Net-SNMP. http://net-snmp.sourceforge.net.

[9] NetFPGA. http://netfpga.org.

[10] nuttcp. http://linux.die.net/man/8/nuttcp.

[11] Policy based routing. http://www.cisco.com/warp/public/732/Tech/plicy_wp.htm.

[12] Ruby on Rails. http://www.rubyonrails.org.

[13] The netfilter.org project. http://netfilter.org.

[14] US Search Engine Rankings, September 2007. http://searchenginewatch.com/showPage.html?page=3627654.

[15] Cisco Data Center Infrastructure 2.1 Design Guide, 2006.

[16] M. Arregoces and M. Portolani. *Data Center Fundamentals*. Cisco Press, 2003.

[17] R. Bajcsy*, et. al.* Cyber defense technology networking and evaluation. *Commun. ACM*, 47(3):58–61, 2004 http://deterlab.net.

[18] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe. Design and Implementation of a Routing Control Platform. In *NSDI 2005*.

[19] D. Caldwell, A. Gilbert, J. Gottlieb, A. Greenberg, G. Hjalmtysson, and J. Rexford. The Cutting EDGE of IP Router Configuration. In *HotNets 2003*.

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

[21] K. Elmeleegy, A. Cox, and T. Ng. On Count-to-Infinity Induced Forwarding Loops in Ethernet Networks. In *Infocom 2006*.

[22] R. Gold, P. Gunningberg, and C. Tschudin. A Virtualized Link Layer with Support for Indirection. In *FDNA 2004*.

[23] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. A Clean Slate 4D Approach to Network Control and Management. In *ACM SIGCOMM Computer Communication Review. 35(5). October, 2005*.

[24] D. Joseph, A. Tavakoli, and I. Stoica. A Policy-aware Switching Layer for Data Centers. Technical report, EECS Dept., University of California at Berkeley, June 2008.

[25] 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.

[26] K. Lakshminarayanan. *Design of a Resilient and Customizable Routing Architecture*. PhD thesis, EECS Dept., University of California, Berkeley, 2007.

[27] A. Mahimkar, J. Dange, V. Shmatikov, H. Vin, and Y. Zhang. dFence: Transparent Network-based Denial of Service Mitigation. In *NSDI 2007*.

[28] D. Oppenheimer, A. Ganapathi, and D. Patterson. Why do Internet services fail, and what can be done about it. In *USENIX Symposium on Internet Technologies and Systems*, 2003.

[29] V. Paxson. Bro: A system for detecting network intruders in real-time. *Computer Networks*, 31(23–24):2435–2463, 1999.

[30] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet Indirection Infrastructure. In *SIGCOMM 2002*.

[31] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *SIGCOMM 2001*.

[32] M. Walfish, J. Stribling, M. Krohn, H. Balakrishnan, R. Morris, and S. Shenker. Middleboxes No Longer Considered Harmful. In *OSDI 2004*.

[33] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker. On the Characteristics and Origins of Internet Flow Rates. In *SIGCOMM 2002*.