EPIC: Platform-as-a-Service Model for Cloud Networking

Theophilus Benson
Aditya Akella
Sambit Sahu
Anees Shaikh

Technical Report #1686

February 2011

EPIC: Platform-as-a-Service Model for Cloud Networking

Theophilus Benson[⋆], *Aditya Akella*[⋆], *Sambit Sahu*[†], *and Anees Shaikh*[†]
[⋆]*UW-Madison*, [†]*IBM Research*

Abstract

Enterprises today face several challenges when hosting line-of-business applications in the cloud. Central to many of these challenges is the limited support for control over cloud network functions, such as, the ability to ensure security, performance guarantees or isolation, and to flexibly interpose middleboxes in application deployments. In this paper, we present the design and implementation of a novel cloud networking system called EPIC. Customers can leverage EPIC to deploy applications augmented with a rich and extensible set of network functions such as virtual network isolation, custom addressing, service differentiation, and flexible interposition of various middleboxes. EPIC primitives are directly implemented within the cloud infrastructure itself using high-speed programmable network elements, making EPIC highly efficient. We evaluate an OpenFlow-based prototype of EPIC and find that it can be used to instantiate a variety of network functions in the cloud, and that its performance is robust even in the face of large numbers of provisioned services and link/device failures.

1 Introduction

Cloud computing is an emerging new model for the delivery and consumption for IT resources. Cloud service providers deliver virtual servers with pre-configured software stacks using standardized and highly automated processes with support for several delivery models, e.g., public, private, and hybrid. Customers of cloud services are able to consume these resources paying only for the resources they use, with the ability to scale usage on demand. Given the economic appeal and agility of this model, it is not surprising that both small and large companies are increasingly leveraging cloud computing for their workloads [40, 41].

Despite the growing adoption of cloud by enterprises, particularly for test/development workloads, key challenges remain when migrating Line-of-Business production applications. A variety of reasons are cited to keep certain production workloads in the enterprise data center. These include, poor isolation, lack of support for security, privacy, and audit compliance, unpredictable performance, and poor reliability of resources [45]. Finally, it is increasingly evident that large enterprises are unwilling to move all workloads to public clouds. This means, for example, that applications running in a cloud must be able to communicate securely with those running behind the enterprise firewall, something which is not well-supported today.

Underlying many of these challenges is the *absent or limited support for control over the network* in current Cloud Computing environments. The goal of our paper is to highlight the support needed in the cloud network layer to overcome these challenges. Our central contribution is the *design, implementation and evaluation of EPIC, a general cloud networking system* that offers network-level controls and services to support the operation and management of a broad range of production enterprise applications. EPIC's novelty arises from the integration of provisioning and management for both cloud-based applications and networks. EPIC brings these together in a unified framework that provides cloud tenants with a simple abstraction for deploying virtual instances and their underlying network functions, and also includes a number of techniques and algorithms to address the scaling and performance issues that surface.

The cloud network model thus far has been to provide basic reachability, based on dynamic or static publicly routable addresses, with basic firewall capabilities available at each virtual server. Several key network functions are generally not available, however, e.g., fine-grained network isolation for security and/or service differentiation, middleboxes for intrusion detection and audit compliance, control over addressing, and optimizations like protocol acceleration, path control and distributed caching for improved performance and availability.

A few cloud providers are starting to provide network-related features that address some of these shortcomings. Cloud providers such as Amazon have extended their network features to include secure VPN-based connectivity to a set of isolated virtual instances with the ability to specify private address ranges and subnets [3, 7]. Third-party companies provide virtual appliances that can be deployed in a cloud to provide, for example, fast data replication [11], application acceleration [19] and content delivery [1, 4].

While these certainly help to bridge the gaps in cloud networking functionality, they represent point solutions that are not well-integrated from a customer point of view. In particular, anecdotal evidence suggests that it is difficult to compose different service offerings in order to faithfully replicate, in clouds, the complex network functionality found within enterprise deployments today [35]. It is also expensive, as each offering is from a different vendor. We believe that what is required is a unified cloud-provided framework for customers to flexibly define and use a rich variety of cloud network services, coupled with corresponding mechanisms for providers to efficiently deploy and manage these services.

In this paper, we describe EPIC, a cloud networking framework with key primitives that customers can leverage to complement their virtual server deployments with a rich set of network functions for virtual network isolation, custom addressing, service differentiation, and a variety of middlebox functions such as intrusion detection, caching, or application acceleration. EPIC does not merely integrate existing solutions and offerings for these services – rather, it is a comprehensive and extensible solution for cloud networking that also provides a simple API for customers to specify their network designs and policies. Unlike solutions based on third-party virtual appliances and overlay networks, EPIC primitives are implemented within the cloud infrastructure, and hence are highly efficient and transparent to cloud tenants and end-users.

Hence, EPIC allows enterprises to easily and effectively replicate key aspects of their application deployments in the cloud, and even enrich them, with little overhead. In effect, our approach can be viewed as a Platform-as-a-Service (PaaS) model for cloud networking, similar to existing PaaS offerings that provide standard application-level services and associated APIs to make it easier to move (or write) applications to the cloud [18, 17, 9].

EPIC is built around several design principles: (1) leveraging software-defined networks to enable fine-grained control over the network, (2) using indirection to maintain the abstraction of control over address allocation, and (3) extending the network edge onto the physical hosts with host-based virtual switches. Although address indirection and programmable networks are established approaches, EPIC integrates all of these techniques at the network layer, and ties them to the cloud provisioning system to streamline and unify the way in which applications and network services are deployed in current clouds.

The design and implementation of EPIC is made challenging by the practical issues that arise when designing for an environment as large and dynamic as a cloud. For example, network devices are limited in the amount of control state they can maintain, and the rate at which state can be updated. Also, the dynamic nature of customer applications and infrastructure failures or downtime require that the cloud network maintain specified policies under varying amounts of churn in the system. EPIC's design and implementation includes algorithms and optimizations designed to reduce the impact of these hardware limitations, and also manage the dynamic nature of cloud-delivered services.

Our prototype implementation of EPIC leverages programmable virtual network devices, including OpenFlow [37], and Open vSwitch [13] (though the design can make use of other programmable network paradigms as we discuss later in the paper). EPIC primitives are specified as part of a cloud deployment, and low-level directives (forwarding rules) are installed in the network data plane automatically, as the corresponding virtual servers are instantiated. In this way, the desired network functions are integrated with the application as a policy file, making it easy for the tenant to tailor the network support for the needs of the workload. The common case of providing basic Internet connectivity to virtual servers is supported by default, so that customers not requiring any specific support need not do anything new.

We show the flexibility of EPIC in supporting a number of network functions in the cloud using a typical multi-tier application model in our lab testbed with commercial OpenFlow-enabled network devices. We validate that fine-grained access control, VLAN-based isolation, service differentiation, IP address mapping, and middlebox interposition can be easily specified and deployed using EPIC. We also evaluate, using emulated scenarios, the performance and scalability of EPIC in the face of dynamics such as network and host failures, and also as the number of customers and size of the cloud varies. EPIC scales to the dynamics of a large cloud with 270K VMs by recovering (i.e., re-establishing network services as well as connectivity) from link failures and device failures in well under .1 seconds and 6 seconds, respectively. We find that EPIC imposes a low overhead on the devices in the cloud, requiring, for example, only 96 MB of memory per endhost. We also show that simple heuristics can be employed to effectively manage limited device-level memory for holding the forwarding state for large numbers of cloud tenants.

2 Background and Design Requirements

In this section, we motivate the need for additional network-level support when moving typical multi-tier enterprise applications to the cloud. We argue that the lack of sufficient network support in clouds today deters operators from redeploying their applications, and identify the design requirements for our system to overcome these challenges.

2.1 Limitation of Current Cloud Networking Mechanisms

Below we focus on three important challenges that arise from limited control over the networking policies in current clouds.

Limitation 1: Application performance. Many tiered applications require some assurances of the bandwidth between server instances to satisfy user transactions within an acceptable time frame and meet predefined SLAs. For instance, the "thumbnail" application described in [29] generates and sends different versions of photos between the business logic servers before they are finally returned to the user. Insufficient bandwidth between these servers, e.g., at times of high cloud utilization, will impose significant latency on user interactions [29]. Also, recent studies [38] point to the slow rise in the average latency within the EC2 cloud, possibly due to oversubscription. Thus, without explicit control, variations in cloud workloads and oversubscription can cause delay and bandwidth to drift beyond acceptable limits, leading to SLA violations for the hosted applications.

Limitation 2: Flexible middlebox interposition. Enterprises deploy a wide variety of security middleboxes in their data centers, such as deep packet inspection (DPI) or intrusion detection systems (IDS), to protect their applications from attacks. These are often employed alongside other middleboxes [23] that perform load balancing [5], caching [27] and application acceleration [15]. When deployed in the cloud, an enterprise application should continue to be able to leverage this collection of middlebox functions.

Today, there are a limited number of solutions to address this need. IDS providers, such as SourceFire [16], have started packaging their security software into virtual appliances that can be deployed within the cloud. Similarly, EC2 provides a virtual load balancer appliance for cloud-based applications [8]. Unfortunately, there is no means today to specify and control middlebox traversal, i.e., the series of virtual appliances that traffic should traverse before arriving at, or after leaving, a node in the enterprise application. A common practice when using virtual appliances is to install all the virtual appliances in the same VM as the application server. However, this approach can degrade application performance significantly. It also increases the cost of the cloud-based deployment as the customer will have to buy as many appliance instances as there are application servers.

Limitation 3: Application rewriting for consistent network operation. The cost and difficulty of application rewriting places a significant barrier to migrating enterprise applications into the cloud. Applications may need to be rewritten or reconfigured before deployment in the cloud to address several network-related limitations. Two key issues are: (i) lack of a broadcast domain abstraction in the cloud and (2) cloud-assigned IP addresses for virtual servers.

Cloud providers such as EC2 do not allow broadcast traffic [21], which precludes important mechanisms such as broadcast-based failover. Applications may have to be rewritten to employ alternate failover mechanisms in the cloud. For example, backend database servers must be rewritten to use other failover mechanisms such as Layer-3 heartbeats [42] and third-party cluster resource managers (e.g., PaceMaker [14]).

When writing configuration files for their applications, some applications may have hardcoded IP addresses for servers in various tiers or for external services on which the applications depend (see examples in [36]). When redeploying applications within the cloud, virtual servers are likely to be given new addresses in the cloud, requiring, at a minimum, updates to their configurations Depending on whether or not the services that the applications depend on have also been migrated into the cloud, further updates to configurations may be necessary. Configurations can be quite complex for production 3-Tier applications [28], hence retooling them to ensure consistency in IP addresses is challenging and difficult to automate.

Network Functions	On-path Middlebox	Layer 2 Broadcast	QoS	ACL	Static Addressing
EC2 [2]	N	N	N	Y	N
EC2+VLAN	N	Y	N	Y	N
EC2 w/VPC [3]	N	N	N	Y	Y
VPN-Cubed [7]	N	Y	N	Y	Y
EPIC	Y	Y	Y	Y	Y

Table 1: Policies supported by the networking layers of various clouds.

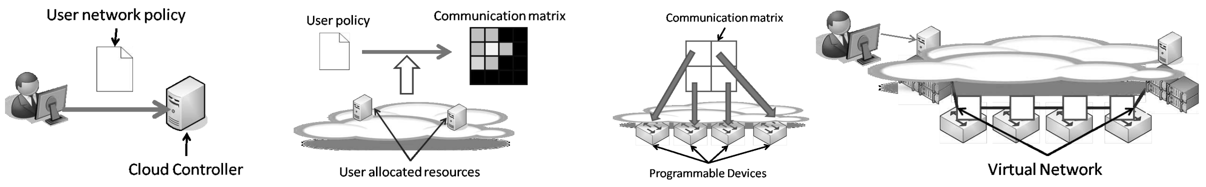
As mentioned in Section 1, some cloud providers do support some specific network-level functionality, but these are generally point solutions that only partially address the limitations described above. For example, in Table 1, we list a number of network functions and consider to what extent they are supported by some commercially available cloud services[1]. We see that each of the available mechanisms addresses a subset of the desired functions, while EPIC provides a framework with more comprehensive support for network-layer policies in the cloud.

2.2 Design Requirements

Our aim is to design a cloud networking framework to effectively support production enterprise applications. Our system should address the basic challenges outlined in Section 2.1, also make it relatively simple to realize a wide variety of network functionality within the cloud. Specifically, we set the following five functional and operational design goals for our framework:

(i) Allow enterprises to realize a cloud deployment with an identical data-plane configuration as when deployed locally; this includes the ability to flexibly interpose a variety of middleboxes such as firewalls, caches, application accelerators, and load balancers.

[1]Note that EC2+VLAN is not actually available, but represents an IaaS service with the ability for customers to create VLANs.



(a) Specify user requirements (b) Convert requirements into a communication matrix (c) Compile matrix entries into network-level rules (d) Install rules and configure paths

Figure 1: Various steps in the EPIC framework.

(ii) Allow enterprises to specify bandwidth required by applications hosted in the cloud, ensuring similar performance to hosting locally.

(iii) Require little or no rewriting of applications (i.e., applications should run "out of the box" as much as possible), in particular for IP addresses and for failover mechanisms.

(iv) Operate effectively at scale and under a variety of dynamic scenarios, extending the benefits of running in the cloud to as large a customer-base as possible and minimizing undesirable interactions among consumers.

(v) Provide a simple, easy-to-use interface to enterprise customers.

To meet these requirements, our ground-up design leverages programmable networks (both at network devices and end-hosts), coupled with a simple interface that allows customers to specify their application configuration at a high level of abstraction. Our use of programmable networks based on OpenFlow enables dynamic reconfiguration of the cloud to efficiently meet our design goals.

3 EPIC System Design

In this section, we describe the architectural components of the EPIC cloud networking framework and their interactions. This high-level description is followed by more details on the design and implementation of each component.

EPIC overview. Figure 1 illustrates the four main actions required in EPIC.

First, a cloud tenant or operator uses a simple cloud interface to specify the network requirements, or a policy, for their application using high level language constructs (Figure 1 (a)). We describe the constructs and the rich set of features that can be specified using them, e.g., QoS required, address remapping, middlebox traversal, etc. in Section 3.1.

Next, the network policy is translated from the high level constructs into a canonical description of the desired network communication patterns and network services; we refer to this as the "communication matrix" (Figure 1 (b)). This represents the logical view of the resource demand of the tenant and it remains invariant as long as the tenant application is running. At the end of this step, the tenant instance is deployed by creating and placing the specified number of VMs. We describe this in Section 3.2.

Then, we translate the logical communication matrix along with knowledge of VM locations into network-level directives (i.e., configuration commands or rules) for devices in the cloud (Figure 1 (c)). This is done based on the knowledge of other tenants' requirements and/or their current levels of activity. This step determines whether it is possible to map the tenants logical requirements into the cloud's physical resources.

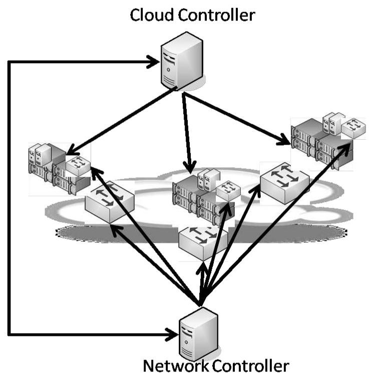


Figure 2: EPIC architecture and components

The final step is to install the configuration commands or rules into the devices within the network (Figure 1 (d)), thereby creating the necessary physical communication paths that satisfy the tenant's needs. In addition, address-rewriting rules are instantiated within various network locations to ensure that applications can use legacy IP address even within the cloud's setting. We describe the last two steps in Section 3.3. The tenant's Cloud instance is then ready to run as per the operator's specifications.

EPIC components. Figure 2 presents the main components of EPIC that implement the above steps: *cloud controller* and *network controller*. The cloud controller manages both the virtual resources and the physical hosts, and externalizes APIs for setting network policies. It addresses the steps described in Figure 1 (a) and (b). The network controller is responsible for monitoring and managing the configuration of network devices. It handles the tasks outlined in Figure 1 (c) and (d). The communication between these two controllers is handled through the use of TCP connections.

3.1 Network Policy Specification

In this section, we describe a simple policy language and its constructs with which cloud customers can specify networking requirements for their applications. We have implemented and evaluated EPIC using this language (Sections 5 and 6) and have found it to be sufficient to realize the requirements outlined in Section 2.2. As we show below, it is also easy and intuitive to use.

Policy example. Our policy language provides a set of constructs for describing the set of VMs that make up an application and the network controls desired for them. We illustrate these constructs by showing how an enterprise operator specifies policies to instantiate the 3-Tier application presented in Figure 3. It is is composed of 1 front-end server which distributes connections across a clustered business logic tier. The backend tier consists of 2 database servers, only one of which is active at any time. A broadcast-based failover mechanism is used for notification in case the current master fails [10]). The frontend server is configured to accept secure connections and it is protected by middleboxes that perform deep packet inspection and intrusion detection. Interposed between the business-logic tier (3 servers used in a load-balancing configuration) and the backend tier, there are middleboxes that perform application classification – specifying that certain traffic receives reserved bandwidth while others be treated as best effort. We assume that the customer wishes to use enterprise-assigned addresses for the backend servers.

In using the policy language to configure the above application, the operator first identifies the specific enterprise-defined addresses to use for VMs in his application; e.g., the operator specifies addresses for backend VMs in lines 1 and 2. Then, the operator proceeds to identify logical groupings (e.g., based on tiers) for the VMs that are part of her application (lines 1-3).

Next the operator defines the properties for the virtual networks to be created between these groups of VMs. In doing so, the operator has the ability to control three network aspects among groups of VMs: (1) quality-of-service or, more specifically, bandwidth reservations, (2) need for layer 2 broadcast, and (3) on-path middleboxes.

The operator configures the application in Figure 3 as follows. For the front end, she first defines a network service called *protectfrontend()* (line 8) that prevents Layer-2 broadcast, provides best effort guarantees, and forces traffic through a DPI middlebox. In line 13, the *protect-frontend()* network service is applied between VMs in

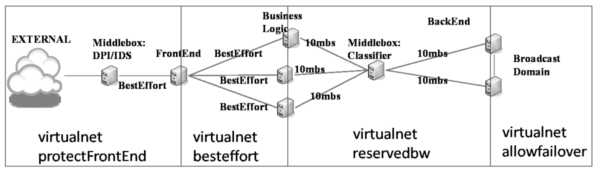


Figure 3: Example 3-tier application

1 `address` *dbserver1* = {128.1.104.103}
2 `address` *dbserver2* = {128.1.104.13}
3 `group` *frontend* = {httpdserver}
4 `group` *businesslogic* = {jboss1,jboss2, jboss3}
5 `group` *backend* = {dbserver1, dbserver2}
6 `middlebox` *Class* = {type=classifier, config=""}
7 `middlebox` *DPI* = {type=dpi, config= ""}
8 `networkService` *protectFrontEnd* = {l2broadcast=no, qos=BestEffort, mb=DPI}
9 `networkService` *besteffort* = {l2broadcast=no, qos=BestEffort, mb=none }
10 `networkservice` *reservedbw* = {l2broadcast=no, qos=10mbs, mb=Class}
11 `networkservice` *allowFailover* = {l2broadcast=yes, qos=BestEffort, mb=none}
12 `virtualnet` *allowfailover* (backend)
13 `virtualnet` *protectFrontEnd(frontend, EXTERNAL)*
14 `virtualnet` *besteffort(frontend,businesslogic)*
15 `virtualnet` *reservedbw(businesslogic,backend)*

Figure 4: Enterprise network policies for our example 3 tier application

frontend and those in the *EXTERNAL* group which refers to external users.

To configure the network for the back-end, the operator creates a network service *allowfailover()* that allows failover through layer 2 broadcast (in line 9). The operator applies this network service among all VMs in the group *backend* (line 12).

Finally, the operator configures the rest of the network by creating a virtual network between the *backend* and the *businesslogic* tiers and applying the *reservedbandwidth()* network service (defined in line 8) to the network (line 13). This service also includes a middlebox "Class" (defined in line 4) to perform application classification, in addition to 10Mbps of bandwidth between a VM in the second tier and one in the 3rd tier and disallowing layer-2 broadcast.

3.1.1 Network Policy Constructs

We now provide more formal definitions of the constructs employed in the example above.

Address. This construct allows the enterprise operator to bound an enterprise assigned address to a VM.

Groups. This construct allows the enterprise operator to logically group a set of virtual machines. The grouping could be on the basis of similarity in their functionality, as in the example above, or on the basis of similarity in policies that apply to the VMs. A group can contain one or more virtual machines and the virtual machines can be identified either by name or by the IP address assigned by the cloud provider.

Middleboxes. With this construct, a user can configure and initialize a virtual middlebox. Any virtual middlebox supported by the cloud provider can be used – the middlebox can be provided directly by the cloud (similar to Amazon's VPN middlebox), or it can be a third-party appliance (similar to NetEx [11]).

To initialize a virtual middlebox, the enterprise operator must specify: (1) the type of middlebox (a list of supported middleboxes can be retrieved from the cloud provider) and (2) a configuration file for the middlebox. The exact format, which is provided by the cloud provider or the third-party vendor, may vary from middlebox to middlebox.

Network Service. This construct allows enterprise operators to specify the set of services to provide among a collection of virtual machines.

Currently, we support 3 different services: a layer 2 broadcast service, a bandwidth reservation service, and a middlebox interposition service. Each of these network services can be specified as a parameter of the `networkservice` command: layer 2 broadcast has two options: yes and no; quality of service has two options: best effort or a numerical value specifying the amount of bandwidth to reserve in Mbps; and middle box interposition takes as arguments a list of 0 or more middlebox names to traverse in sequence.

Virtual Network. We refer to a collection of VMs among which a certain network service should apply as a virtual network. A virtual network can span 1 or two groups. A virtual network is specified by applying a network service to a list of 1 or 2 groups. When a single group is used, it means that the service should apply among all pairs of VMs in the group. When a pair of groups are used it means that the service should apply between an arbitrary VM in the first group and an arbitrary in the second group.

This approach to specifying policies has a few characteristics which makes it intuitive and easy to use: (1) groups makes it simple to modify the policy configuration when, for example, additional VMs with similar functionality or with similar policies applied to them are added to a customer's application. The policy change is then confined to just the group definition and does not impact the rest of the specification. The use of groups is similar to the use of policy units in [24] – it aligns with how operators think of their network services. (2) The use of the network service abstraction means that the cloud can provide a variety of pre-defined services as policy templates and tenants can simply apply them to the desired groups of VMs. (3) Middlebox interposition is easy to specify, and it is also possible to provide canonical middlebox traversals as templates to tenants (e.g., for implementing security policies).

The EPIC specification language complements policy languages and constructs provided by current clouds such as Amazon EC2 and Windows Azure. For example, our policy language can work with EC2's virtual instance identifiers when specifying the policies. Our policy language can also be easily embedded into Azure's current configuration language by extending the "endpoint" element to accept our policies. Such an extension of the Azure configuration will provide tenants with control over properties of the connection between created between virtual instances, something the language does not currently offer.

Finally, note that EPIC's policy language is merely a candidate among many possibilities; it is possible to incorporate alternative, more descriptive languages. We leave this exploration for future work.

3.2 Cloud Controller

In typical cloud, the cloud controller is responsible for managing physical resources, monitoring the physical machines, placing virtual machines, and allocating storage. The controller reacts to changes in workload by migrating virtual machines, scheduling new virtual machines, and allocating physical resources.

EPIC extends the cloud controller in three ways in order to facilitate better control over the network in additional to control over hosts:

(1) It accepts a wide range of network policies (as opposed to simple requests for VMs) and parses it to generate a communication matrix for the tenant's resources. The matrix captures the requirements for the network between two tenant VMs. An entry specifies if the virtual network between the source and the destination VM (row and column, respectively) should permit packets; if so, whether layer 2 broadcast is allowed, or layer 3 traffic is allowed, or both are allowed. And when layer 3 traffic is allowed, the entry also specifies bandwidth reservations and the middlebox traversal required by traffic between the endpoints, if any are set. The matrix is then passed to the network controller which interfaces with the programmable switches.

(2) As the location of virtual resources changes either due to user requests, workload fluctuations, or device failure, the cloud controller updates the network controller with the new locations; this information is used by the network controller when it decides how to re-assign network level directives to switches in order ensure the appropriate connectivity between different tenant VMs.

(3) It instantiates a software programmable switch on each physical host that is integrated into a tenant's application. The software switch is configured to connect any number of virtual machines to the network of physical programmable switches. The software switches are crucial to extend network control beyond the physical switches and into the end-hosts themselves. Once configured, the cloud controller informs the network controller of the location of the software switches and subsequently sends updates about the set of virtual machines attached to the switches.

3.3 Network Controller

The network controller is a new component that EPIC introduces into the cloud. It is responsible for configuring virtual networks throughout the cloud by mapping the logical requirements in the communication matrix onto the physical network resources. It also controls resources, e.g., by performing re-mapping when available resources change, to ensure that tenant requirements are consistently satisfied. Thus, it is crucial to EPIC's overall functioning.

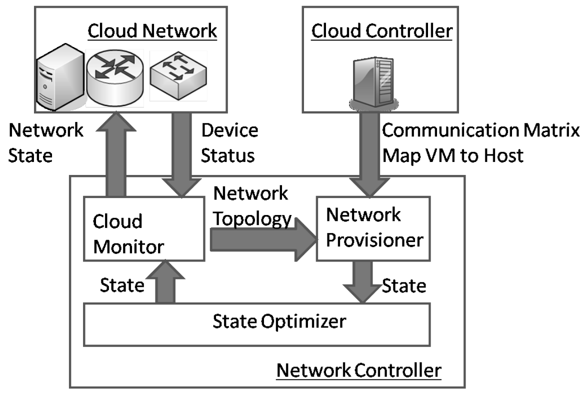


Figure 5: Internal components of the network controller.

In figure 5, we present the internal modules for the network controller. The network controller takes two inputs: communication matrix, and a mapping between a tenant's VMs and physical hosts. In addition, the network controller collects at least two additional pieces of state: the current status of various switches and links (along with link utilizations) and the current mapping of flows corresponding to various virtual networks across the physical cloud network. By default, it uses the cloud monitor module to periodically poll the devices for this state; however, the cloud monitor can also receive triggered state updates from devices when they come up or when their neighbors fails.

Based on the inputs, the controller uses the network provisioner module to generate the set of configuration commands for each of the programmable devices in the network and configures them accordingly to instantiate the tenant's virtual network. A similar set of actions must be taken by the network provisioner when remapping tenant virtual networks during network failures. In addition to these basic actions, the network provisioner is responsible for situations where control over the network is necessary, e.g., tearing down a tenant's application, and providing the necessary support to prevent the tenant from having to rewrite applications to match addresses assigned in the cloud. We discuss these tasks in more detail below.

Provisioning and De-provisioning Virtual Networks To provision a virtual network between a pair of virtual resources (VMs, or a VM and a middlebox), the network controller first determines the constraints that apply to the path between the resources based on the attributed requested. The constraints can range from simply finding a loop-free circuit when the user request simple best effort connectivity between two VMs, to identifying the amount of bandwidth needed along a path when QoS is required. Once the constraints have been gathered, the network controller searches the graph reflecting the current network state and resources for a physical path that satisfies these constraints. We use widest shortest path first for generating Best effort paths and paths with QoS requirements, while we use Spanning Tree algorithms for generating broadcast paths.

If a path is found, the controller generates and pushes the appropriate configuration commands to the network devices on the path. The nature of the configuration commands generated is specific to the type of programmable devices used in the network. We discuss the details of how rules are created and allocated to the appropriate devices while taking into account limited ruleset memory in Section 4.1.

De-provisioning a tenant's application occurs is a similar fashion; we omit the details for brevity.

Addresses Rewriting

The final crucial function of the network provisioner is to perform address mapping to allow enterprises to reuse existing addresses. To achieve this, the cloud controller provides the network controller a map of the VM names to their enterprise-local addresses (provided by the tenant as part of his policy). For each VM in the map, the network controller installs a "mangle" rule in the software switch resident on the host where the VM resides. This rule rewrites the destination address from the enterprise-local address to the cloud-assigned address before forwarding out the appropriate port. For other VMs or traffic using cloud addresses, rules are installed for forwarding without rewriting the destination address. When VMs move, the mangle rules are recreated at the appropriate software switches. Thus, we leverage programmability of the Cloud, in particular, the software switches to avoid address rewriting.

4 Practical Issues

The Network Controller faces several challenges in: (i) installing forwarding state to completely enforce tenant policies under the constraints of network device processing and memory limitations, and (ii) ensuring that network policies persist in the face of cloud dynamics such as device and link failures. In this section, we discuss the techniques used by EPIC to deal with these practical issues.

4.1 Hardware Device Limitations

EPIC uses the fine-grained control provided by programmable devices to enforce quality of service guarantees, middle-box interposition, tenant defined broadcast domains, and address writing. The cost of using fine-grained control is state explosion in the network devices. In using the APIs provided by OpenFlow and NOX to configure the network, EPIC creates $O(V * N^2)$ forwarding entries *per device* within the network, where V is the number of virtual networks and N is the number of virtual machines using these virtual networks.

Several data center grade switches have limited TCAM space for flow table entries, roughly on the order of 2000 entries. Thus, unless this memory is carefully managed, we may not be able to support a large number of virtual networks in the cloud.

Next, we present several optimizations implemented at the network controller to address this issue. These optimizations leverage the distinction between host-based switches, i.e., OpenFlow based software switches on the physical hosts, and OpenFlow-enabled in-network switches: Flow tables in the former are stored in the much larger host memory (DRAM), providing room for many more rules, as opposed to limited TCAMs used in the latter. The goal of these optimizations is to provide EPIC with fine grained control while limiting the in-network state.

Optimization 1: Best effort traffic. Our first optimization applies to configuring flow table rules for best effort traffic. It works simply as follows: We install full flow-based rules in the host switches, and simple destination-based rules in in-network switches (i.e., source-based entries are wild-carded in the latter case). This optimization leverages the insight that best effort traffic can be aggregated along a small collection of network paths. Thus, each in-network device needs only to maintain rules for at most one spanning tree per destination, thereby reducing storage requirements from $O(N^2)$ to $O(N)$ per virtual network, where N is the number of virtual machines. We illustrate this optimization in Figure 6. In (a), we show rule sets at different devices without the optimization. Device D carries 6 flow table rules. In (b), we show that, with the optimization, device D holds 4 flow table rules, a 33% reduction.

This destination based forward prevents Middlebox traversal. To allow for middlebox traversal, EPIC installs rules in the software switches of the source VM and subsequent middleboxes that encapsulate and tunnel the packet from the source VM or current middlebox to the next middlebox.

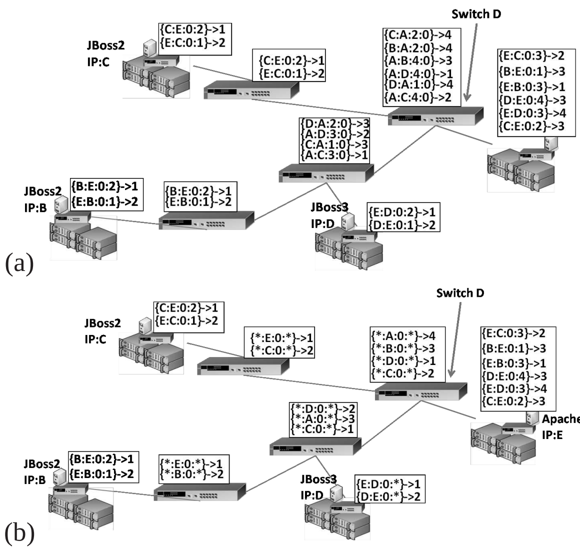


Figure 6: A network with 4 hosts, 4 switches, and 4 VMs. The flow table for each switch is displayed in a white rectangle. The flow entries in each table have the following format: {Src IP:Dest IP:ToS:InPort}-> OutPort with a * indicating a wildcard.

Optimization 2: QoS traffic. Our next optimization extends the above idea to traffic with QoS requirements. The behavior of host-based software switches remains qualitatively the same as above. However, in-network switches forward on the basis of both the destination-based information as well as the type-of-service (ToS) bits in the packet header. ToS bits are used to select the appropriate queue for network traffic.

If multiple reserved paths to the same destination use the same underlying devices and links, then only one entry is need per in-network device. If a pair of paths only share some links and devices, then the controller uses different ToS values for each path leading to separate entries in in-network switches; the optimization is less effective in this situation. Although less effective, this approach reduces the storage requirements from $O(N^2)$ to $O(S * N)$ per virtual network, where N is the number of virtual machines and S is the max number of alternate paths from any switch to the virtual machine.

Optimization 3: Forwarding entry aggregation. Given that the earlier optimizations allow for simple destination based forwarding, we can use the wildcard feature to aggregate forwarding entries with the same output port, in a fashion similar to how IP address aggregation is done in Internet routers. To increase the efficiency, we assign contiguous addresses to VM placed behind the same ToR switch. This results in gains of O((S)*N/P), where S is the number of distinct paths to a ToR, N is the number of virtual machines, and P is the size of prefix assigned to each ToR switch.

4.2 Cloud Dynamics

The validity of the network policies are affected by dynamic cloud events ranging from link failures, device failures, to policy changes. We now show how EPIC ensures that the correct set policies are enforced during these events. EPIC employs a simple design principle of precomputation and caching to reduce the impact of device or link failures on the network.

Policy Changes & Host/VM dynamics: When host conditions change due to oversubscription or failure, the cloud controller may map a tenant's VMs to other hosts and regenerate the communication matrix. The cloud controller also regenerates the communication matrix when a tenant changes his policies. When the matrix is regenerated, the cloud controller informs the network controller of the new matrix. The virtual networks would then need to be reprovisioned as well.

To do this without causing significant disruption to existing tenants, the network controller performs remapping for only the changed portions of the communication matrix.

Device/link failures: When devices or links fail, virtual networks can potentially be rendered invalid. In such situations, EPIC tears down and re-provisions all virtual networks which are dependent on the failed links or devices. To reduce downtime EPIC employs precomputation and caching. EPIC maintains a map between devices/links and the set of dependent virtual networks, thus allowing it to quickly determine the virtual networks to re-provision under link or device failures. To reduce the time to re-provision these virtual networks, EPIC precomputes network state for different failure scenarios. In our current implementation EPIC precomputes network state for the failure of the core and aggregation devices – a small number of devices with significant state. Thus, failure of these devices can be resolved by simply looking up the cache for the new network state and installing this state.

5 Prototype Implementation

In this section, we describe our prototype of the EPIC cloud networking framework.

OpenNebula cloud controller. We leverage the OpenNebula 1.4 cloud framework to implement the cloud controller component of EPIC. We chose OpenNebula as it provides an identical set of abstractions to users as many prominent IaaS providers, such as EC2, 3Tera, and Eucalyptus. We modified the OpenNebula source to accept user requirements specified using the language described in §3.1, to keep the generate the communication matrix, and to instantiate and configure software switches on hosts. Our modifications were limited to 226 lines of code. We also built a parser to convert policy specifications into communication matrices. Our Perl-based parser has 237 lines.

NOX and OpenFlow for network control. We utilize OpenFlow-enabled switches (specifically, HP Procurve 6400 series switches flashed with the OpenFlow 1.0 firmware) within our lab-based set-up. We chose OpenFlow because using OpenFlow does not require a forklift change to the network; in most cases a simple firmware upgrade of switches is sufficient.

The OpenFlow framework provides an API that allows external software control of the flow tables of network switches. In particular, it allows an authenticated software controller running NOX [12] to dynamically install, update and delete flow-level entries in switch flow tables. It also provides a variety of mechanisms to track network state (e.g., switch and link states). The NOX controller can also be configured to read state from external sources.

We implemented the EPIC network controller atop NOX using 2468 lines of C++ code. We interfaced the network controller with cloud controller; the network controller constantly polls the cloud controller and pulls new/updated communication matrices and VM mappings as and when they are available. We implemented the full set of functionality outlined in Section 3.3 including, provisioning and de-provisioning virtual networks, handling host and network dynamics, and providing mechanisms for address rewriting. Our controller runs on a commodity Linux machine (2.4 GHZ, 4 cores, 4GB RAM).

We implemented end-host software switches using Open vSwitch.

For completeness, we also implemented the following functions: (1) NAT functionality at the cloud gateway to allow customers to use cloud-assigned internal IPs for their applications; this function is configured and controlled by the network controller, and (2) ARP functionality within customer applications; similar to Ethane [25], we redirect all ARP traffic to the network controller who then provides the appropriate responses.

The network controller is the most crucial component of the EPIC framework as it directly orchestrates the physical network by mapping customer requirements onto it and by managing the mappings. In what follows, we first provide (in Section 4.1) details of the algorithms used by our network controller in performing the mappings by leveraging the OpenFlow interface to network switch flow tables. We then note (in Section 4.2) that in addition to managing network bandwidth and switch-level queues when provisioning virtual networks, the controller must also manage the limited memory available for flow table entries in network switches. This is crucial to supporting a large number of customers with diverse requirements. We conclude with a description of simple optimizations we have implemented for manag-

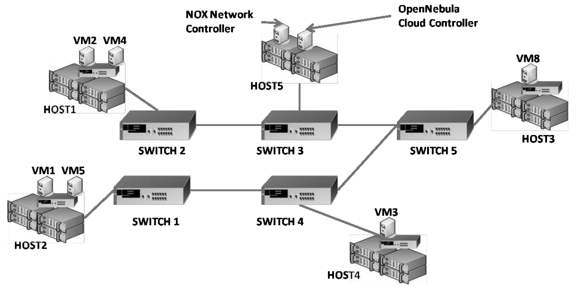


Figure 7: Experimental testbed

ing flow table memories.

6 EPIC System Evaluation

In this section, we present an experimental evaluation of the EPIC prototype in both a lab-based cloud as well as a large-scale emulated cloud. We demonstrate the key primitives supported in EPIC, validating the ability to specify and instantiate a variety of network functions in the cloud. Our experiments also show that EPIC performance scales well as the number of provisioned services grows, and when reconstituting the virtual network after a link or device failure in clouds of varying sizes.

6.1 Functional Validation

We begin by demonstrating the flexibility and functionality of EPIC in implementing several different network functions and policies, including the ability to satisfy user-specified constraints under component failures.

By submitting different network policies to the user interface, we were able to implement, within the confines of the cloud, the different enterprise networking scenarios below.

We perform the testing on a cloud testbed consisting of 5 physical hosts and 5 network switches connected as shown in Figure 7. Four of the five hosts are available for deploying virtual machines, while the fifth (Host5) is used to run the controller services (i.e., cloud controller and network controller each within a different VM). The 5 programmable network devices are 24 port HP Procurve 6400 switches with 20 1Gbps ports.

Our initial application deployment uses a policy that enables point-to-point reachability between VMs 1-3 (which are all part of the application deployment), but not to/from any other VMs in the cloud.

VLAN: In the VLAN scenario, we modify the baseline policy above to place VM2 and VM3 in the same VLAN (broadcast domain) to enable the broadcast-based failover service. We verified that that VM2 and VM3 are able to communicate and then failed the application running in VM2. The VLAN configuration allowed VM3 to correctly detect the failure and take over the role of the primary.

Class-of-Service: To demonstrate the CoS primitive, we modify the baseline policy to reserve 900Mbps for traffic between VM1 and VM2. In this case, the quality-of-service constraint did not result in a change to the underlying paths, though in general a new path may be computed to meet the requirement, as described earlier in Section 5. We instantiate file transfers from VM1 and VM2 and simultaneously from VM5 to VM4 which are deployed on the same hosts as VM1 and VM2, respectively. We observe, with the aid of IPerf, that flows between VM1 and VM2 received the requested share of link bandwidth on the paths shared with flows between VM4 and VM5.

Middlebox Interposition: To validate the correctness of our framework to interpose virtual middleboxes on the network path, we modified our policy between VM1, VM2 and VM3 to force all traffic to and from VM1 through an DPI middlebox implemented in snort. Over several runs of the experiments, we observed that it takes an average of 12ms to modify the path so that traffic from VM2 to VM1 is directed through VM8, where the DPI function is hosted.

Address Rewriting: Finally, we demonstrated the ability of enterprise customers to retain their current IP addressing and connectivity as they move their applications to the cloud. We deployed a simple client-server application with the client in VM3 and server on VM2. The client is configured to refer to the server in VM2 by its original globally routable IP address. Without the address mapping policy installed, VM3 is unable to communicate with VM2 since each VM has been assigned a new private IP address in the cloud. After adding a policy to remap VM2's original address, we observe that traffic from VM3 is able to reach VM2.

6.2 Network Controller Performance

Next, we evaluate the ability of EPIC's network controller to scale to provisioning and managing a large number of virtual networks in a large-scale cloud data center. In these experiments, we use a script to generate a series of user policy files that specify virtual machine connectivity and the associated network functions. The cloud controller functionality is emulated by a script that simulates placement information for each VM (rather than actually provisioning them) and generates a communication matrix for the network controller. The network controller operates as usual, computing the required flow table entries for each switch based on the physical location of the VMs, but does not install the entries. This approach allows us to focus on the performance of the network controller in a large-scale setting unconstrained by the size and topology of our lab testbed. In our experiments, the network controller is deployed on a 2.40GHz quad core Intel Xeon PC with 4GB of memory running Ubuntu 10.

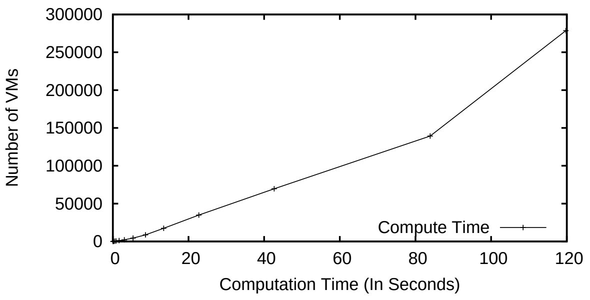


Figure 8: Virtual network computation time

6.2.1 Virtual Network Computation

First, we examine the time taken to initialize services containing a relatively large number of virtual machines and a complex set of virtual networks and policies. In these experiments, we utilize a reference 3-tier services containing 3 virtual middleboxes and 14 application servers (VMs). The service is composed of 6 front-end servers, 5 business logic servers, 3 back-end servers, a VPN middlebox, an IDS middlebox, and a load balancer middlebox. The associated virtual network specification is: (i) traffic to the front-end servers must traverse the IDS, and a load balancer, (ii) traffic between the front-end servers and the business-logic tier receives 10Mbps guaranteed bandwidth, (iii) the back-end servers must be placed within a broadcast domain to enable failover, and (iv) traffic from the enterprise (i.e., external to the cloud) to the back-end must use a VPN.

Figure 8 shows the amount of time taken to simultaneously instantiate network services for as many as 270K VMs (about 16K instances of the reference service). The total time consists of the time to compute corresponding flow table entries and paths in the network. The controller would additionally need to install the flow table entries in the appropriate switches – this time is not captured in our experiments, but is expected to take less than 10ms per flow entry [43]. From the figure, we observe that it takes about 120s to instantiate the virtual network services for the 270K VMs in the cloud. This delay is relatively small when considering the overall service provisioning time, including virtual machine provisioning. For example, experiments in Amazon EC2 showed that provisioning 20 small virtual machine instances can take 180s, and that this time grows with the number of instances being deployed [39].

6.2.2 Failure Handling

When data center elements such as links, switches, or hosts fail, the virtual networks must be remapped and re-installed to restore service. In this series of experiments, we measure the performance of the network controller in re-establishing functional virtual networks in data centers of varying sizes when different components fail. Our data center network model consists of 3-tiers of switches (top-of-rack, aggregation, and core switches). We consider data centers with 200, 500, and 1000 ToR switches, each connected to 30 hosts that can in turn support 9 VMs. These data center models each have 2 core switches, and 20, 50, and 100 switches in the aggregation layer, respectively. As is typical, ToR switches each have two uplinks to the aggregation layer, and aggregation switches are dual-homed to the core layer. In each of these cases, we assume the maximum number of VMs are running, i.e., 54K, 140K, and 270K, respectively.

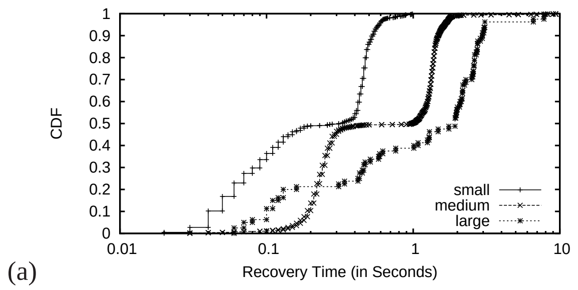
In our failure model, a link, switch, or host is randomly selected to fail. We measure the time taken by EPIC's network controller to recalculate the configuration state to be pushed to the devices. We ignore the time to receive failure notifications which is bounded by the frequency of device polling, and also the time to install state in the device which is, again, assumed less than 10ms. We run each experiment around 150 times to generate the distribution of recomputation times.

Link and Switch Failures. To understand the impact of link failures, we randomly select and delete a link from the topology, triggering the network controller to deprovision paths that use the failed link, and reprovision them on alternate links. We examine the recovery time for links with and without our precomputation and caching. We observe in Figure 9, that without precomputation and caching the median recovery time for the largest cloud with 270K VMs is 2s, and the worst case is under 10s. With caching and precomputation, we observe that the median recovery time for the largest cloud is reduced to 0.2s. In examining the recovery time for device failures, not shown here due to space constraints, we observe that these numbers are in general an order of magnitude worse than the link failure numbers. We note that by extending the precomputation algorithm to precompute for the edge switches we can reduce the recovery for all links and device to a constant time of under 0.2 second (Cache lookup time). However, doing this will require allocating more memory for the cache.

Host Failures. In our experiments with host failures, we randomly select a host to fail and delete it from the topology. This triggers the cloud controller to update the state of the affected VMs and notify the network controller to remap the corresponding virtual networks. Figure 10 shows the time for the network controller to do this; we can see that, compared to provisioning, this take very little time. While provisioning requires searching the graph and calculating paths, remapping requires only a look up in a data structure followed by a control message sent to the appropriate switches (Section 3.2.2).

6.3 Impact on Cloud Infrastructure

The design of EPIC introduces a number of changes to the cloud infrastructure. In this section, we summa-



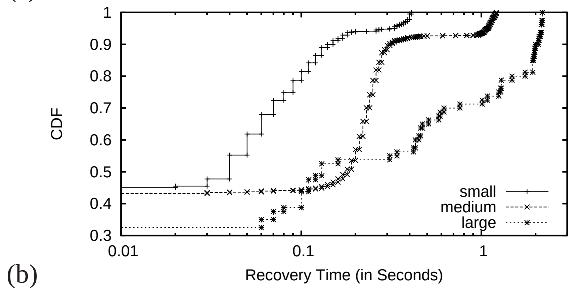


Figure 9: Virtual network recomputation time under link failures. (a) Without caching and precomputation (b) With caching and precomputation of core and aggregation devices

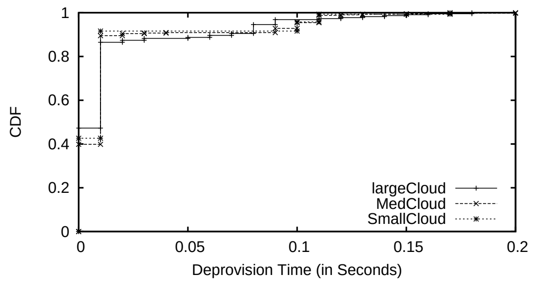


Figure 10: virtual network deprovision time under host failures

rize our observations of the resource overheads of these changes.

One significant change is the modification of the cloud controller to generate and transfer the communication matrix to the network controller. Our experience with EPIC revealed little negative impact in terms of memory and CPU overhead in the cloud management system due to this change. Another change that might raise some concern for a cloud provider is the introduction of the Open vSwitch at each host, which requires instantiation of TAP interfaces in place of the standard kvm vswitch. We observed that the resource usage of the TAP interfaces was minimal, however, and should not impact the number of VMs that can be supported, for example.

In examining the overhead of adding large rulesets in the Open vSwitch, we find that the memory consumption is not significant. Table 2 shows the amount of memory consumed by the virtual switch. We observe that a

Ruleset Size	Memory (in MB)
65536	33
131072	37
196608	57
262144	77
327680	94

Table 2: Resource impact of flow entries in Open vSwitch on hosts.

virtual switch in able to store 300K entries in less than 100MB (of the 4GB available to the Open vSwitch). We note that with a limit of 300K rules, EPIC is able to allocate on average 10K fowarding rules for each VM on the host – hardly limiting the size of a VM's virtual network. We conclude that the host-based Open vSwitches are able to efficiently hold a significant amount of state and thus support our optimizations which increase the amount of forwarding rules and state at the edge.

In Section 4.2, we described several optimizations to the network provisioning algorithm to reduce the number of forwarding rules in the network switches. Here, we show the quantitative impact of those optimizations on the network state for the case of provisioning 16K instances of the reference service (270K VMs) in the largest data center model (i.e., 1000 ToR switches). Table 3 shows the maximum number of flow table entries across the switches in each of the 3 tiers of the data center, plus the Open vSwitches at the hosts. Our goal is to indicate the relative benefits offered by our optimizations. The first row shows the maximum number of flow table entries with no optimizations. Subsequent rows show the number of entries after applying each optimization separately and the last row shows the impact of all of the optimizations taken together.

The best effort and QoS forwarding optimizations achieve substantial benefits each. As the results show, moving up from the host virtual switch layer toward the data center core results in greater benefits from the optimization since there are more flows available for consolidation. On the whole, the optimizations are able to yield between 93% and 99% reduction in flow table usage across different network switches.

Note that the applications we provision have "concentrated" many-to-one or one-to-many communication patterns (e.g., simultaneous communication between a front-end VM and multiple VMs in the business logic layer). Such patterns are quite amenable to our optimizations as they lead to aggregation of entries in in-network switches. As other studies have noted, such many-to-one patterns are highly prevalent in enterprise workloads [29]. In fact, prior studies have found even more highly concentrated communication patterns than what we imposed in our reference application [29]; our optimization can help reduce flow table requirements for such applications to a much greater extent than shown in

Algo	host switch	ToR	Agg	Core
No Optimization	283152	3436	62512	277421
Destination Forwarding	0%	3%	21%	39%
Cos Forwarding	0%	2%	20%	30%
Cos + Destination + Prefix	**0%**	**93%**	**95%**	**99%**

Table 3: Results indicating effect of flow entry optimizations on switches at each tier. The bottom three rows show the percentage reduction in flow table size.

Table 3. This leads us to believe that our optimizations are likely to be quite effective in many practical scenarios.

7 Discussion

In this section, we briefly address some additional considerations toward a more complete network services platform offering in the cloud. As we do not lay out details of how these additional services can be implemented in this limited space, they should be considered as subjects for future work.

Applicability to Non-IaaS clouds: Although we have focused on applying EPIC to the IaaS cloud model, we believe our approach can be generalized to other cloud models which operate at higher levels of abstraction than virtual machines. In PaaS clouds, for example, users write applications to take advantage of specific services and APIs, which are then deployed and managed in the cloud using cloud-assigned ports and IP addresses. In our current design, EPIC allows policies to be specified on a VM basis – in a PaaS model, these policies would be specified on applications. When creating the communication matrix, the cloud controller can map labels to the appropriate address/port assigned to the user application. EPIC network services would then be applied to flows between applications rather than between virtual machines.

Managing Cloud Network Services: In this paper we described EPIC network services and primitives related primarily to the data plane, e.g., traffic isolation, middleboxes, and QoS. An important additional set of services are also needed for enterprise tenants to monitor and manage the cloud virtual network, similar to what they could do in a traditional data center. For example, we can extend the EPIC framework to allow users to attach monitoring, reporting, and logging functions to virtual network devices. The management data can be processed and made available as a continuous feed, or uploaded to a management application in the cloud that provides the ability to visualize the virtual network operations. Careful optimization is of course necessary to ensure privacy of the network data, and to limit the overhead of collecting and transmitting management data.

Extending Enterprise Policies across the WAN: Although cloud providers do not typically control the wide-area portion of the network, enhanced network services that extend from the cloud into the WAN would further benefit applications, particularly those that need to integrate with enterprise-side services, e.g., in a hybrid model. This could be achieved by integrating the virtual network in the cloud with a cooperating ISP or overlay network provider. The EPIC framework can be extended to support new primitives that identify endpoints outside the cloud that are to be integrated. The network controller can negotiate an overlay path, for example, to provision wide-area paths that provide specific services such as service differentiation, WAN acceleration, data deduplication, encryption, etc.

8 Related Work

A fair amount of recent work [22, 44] has been done to examine current and future challenges for both users and providers of cloud computing in running a variety of diverse workloads in the Cloud. In particular, the most closely related works to ours (e.g., [47, 48, 30, 33] focus on individual issues, such as providing better access control, privacy and isolation [47, 48, 30] or reducing disruption of services during migration [47]. Our goals are much more comprehensive and they span security, isolation for performance, and minimal application rewriting.

In our work, we note that the space of security policies utilized by enterprises is much wider than access control or privacy and it is often encapsulated within middleboxes. Our approach accommodates middlebox interposition as a first-class requirements while the above studies do not. Middlebox interposition has been considered in the context of data center architecture before [31], where the goal is to decouple middlebox placement from network location to ensure more robust DC functioning. Conceptually, our approach to accommodating middleboxes is similar to this prior work.

Our work complements efforts on virtualization technologies which is fundamental to cloud computing. Much work in the network virtualization community has focused on providing and enforcing isolation between users [26], on using virtualization to allow better management [46, 20], or on providing primitives to measure aspects of the virtual network [32]. However, none of these studies have highlighted the network layer support needed in typical IaaS cloud computing scenarios and how to offer it.

EPIC considers enterprise security policies on the same footing as the ability to control performance of applications. The latter issue has been addressed in prior work [29, 34], where the focus is on which applications to move into based on expected latency [29] or on deciding which cloud to choose based on performance [34]. We argue that the cloud itself should provide the enterprise with a means to control network performance within the cloud.

Several virtual appliances [7, 6], have been developed to provide point solutions which utilize overlays to accomplish addressing indirection and enforce privacy. Unlike our approach, such solutions do not require provider involvement, but they result in inferior performance as they are external to the cloud and their functioning is subject to variations in wide-area performance.

9 Conclusion

In this paper, we presented EPIC, a network service platform that enables tenants to leverage many of the network functions needed for production enterprise applications to run in IaaS clouds. Our prototype design and implementation of EPIC leverages programmable network devices and supports key features such as isolation, middlebox functions, and quality-of-service. EPIC primitives are specified as part of a Cloud deployment, and are installed in the network data plane automatically, as the corresponding virtual servers are instantiated. We demonstrated the flexibility of EPIC in supporting a number of network functions in the cloud using a typical multi-tier application model in our lab testbed with commercial OpenFlow-enabled network devices. We showed how fine-grained access control, VLAN-based isolation, service differentiation, and middlebox interposition can be easily specified and deployed in several scenarios. We also showed that EPIC performs well in the face of large numbers of provisioning requests, and network and host dynamics.

References

[1] Amazon cloudfront. http://aws.amazon.com/cloudfront/.

[2] Amazon ec2. http://aws.amazon.com/ec2/.

[3] Amazon virtual private cloud. http://aws.amazon.com/vpc/.

[4] Aruba networks: Virtual bracnch network. http://www.arubanetworks.com/solutions/vbn_announcement.php.

[5] Barracuda Load Balancer. http://www.barracudanetworks.com.

[6] Cloudswitch. http://www.cloudswitch.com.

[7] Cohesive ft:vpn-cubed. http://www.cohesiveft.com/vpncubed/.

[8] Elastic load balancing. http://aws.amazon.org/elasticloadbalancing/.

[9] Google app engine. http://code.google.com/appengine/.

[10] How a server cluster works: Server clusters (mscs). http://technet.microsoft.com/en-us/library/cc738051%28WS.10%29.aspx.

[11] Netex. http://www.netex.com.

[12] Nox. http://noxrepo.org/.

[13] Open vswitch project. http://www.vswitch.org/.

[14] The pacemaker linux project. Http://www.clusterlabs.org.

[15] Riverbed Networks: WAN Optimization. http://www.riverbed.com/solutions/optimize/.

[16] Sourcefire. http://www.sourcefire.com.

[17] Vmware springsource. http://www.springsource.com/.

[18] Windows azure platform. http://www.microsoft.com/windowsazure/.

[19] Aryaka Networks: Cloud-based Application Acceleration. http://www.aryaka.com, 1999.

[20] R. Alimi, Y. Wang, and Y. R. Yang. Shadow configuration as a network management primitive. *SIGCOMM Comput. Commun. Rev.*, 38(4):111–122, 2008.

[21] Amazon. Using dhcp options. http://docs.amazonwebservices.com/AmazonVPC /latest/DeveloperGuide/index.html?UsingDHCPOptions.html, 2010.

[22] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, Feb 2009.

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

[24] T. Benson, A. Akella, and D. A. Maltz. Mining policies from enterprise network configuration. In *IMC '09: Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 136–142, New York, NY, USA, 2009. ACM.

[25] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. Mckeown, and S. Shenker. Ethane: Taking control of the enterprise. In *SIGCOMM Computer Comm. Rev*, 2007.

[26] X. Chen, Z. M. Mao, and J. Van Der Merwe. Shadownet: a platform for rapid and safe network evolution. In *USENIX'09: Proceedings of the 2009 conference on USENIX Annual technical conference*, pages 3–3, Berkeley, CA, USA, 2009. USENIX Association.

[27] cisco. Cisco MDS 9000 Series Caching Services Module, 2003.

[28] T. Eilam, M. H. Kalantar, A. V. Konstantinou, G. Pacifici, and J. Pershing. Managing the Configuration Complexity of Distributed Applications in Internet Data Centers. *IEEE Communications Magazine*, pages 166–177, March 2006.

[29] M. Hajjat, X. Sun, Y.-W. E. Sung, D. Maltz, S. Rao, K. Sripanidkulchai, and M. Tawarmalani. Cloudward bound: planning for beneficial migration of enterprise applications to the cloud. In *SIGCOMM '10: Proceedings of the ACM SIGCOMM 2010*, pages 243–254, New York, NY, USA, 2010. ACM.

[30] F. Hao, T. Lakshman, S. Mukherjee, and H. Song. Secure cloud computing with a virtualized network infrastructure. In *HotCloud, 2010*.

[31] D. A. Joseph, A. Tavakoli, I. Stoica, D. Joseph, A. Tavakoli, and I. Stoica. A policy-aware switching layer for data centers. In *SIGCOMM*, 2008.

[32] E. Keller, R. B. Lee, and J. Rexford. Accountability in hosted virtual networks. In *VISA '09: Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*, pages 29–36, New York, NY, USA, 2009. ACM.

[33] E. Keller and J. Rexford. The "platform as a service" model for networking. In *INM/WREN '10*, San Jose, CA, USA, 2010.

[34] A. Li, X. Yang, and S. K. M. Zhang. Cloudcmp: Comparing public cloud providers. In *IMC '10*, Melborne, Australia, 2010.

[35] L. MacVittie. The Question Shouldnt Be Where are the Network Virtual Appliances but Where is the Architecture? http://devcentral.f5.com/weblogs/macvittie, 2010.

[36] A. Mankin and K. Ramakrishnan. Embedding globally-routable internet addresses considered harmful. Request for Comments 4085, Internet Engineering Task Force, June 2005.

[37] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, 2008.

[38] C. Metz. Amazon cloud accused of network slowdown. http://www.thebitsource.com/featured-posts/rackspace-cloud-servers-versus-amazon-ec2-performance-analysis/, January 2010.

[39] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema. A performance analysis of ec2 cloud computing services for scientific computing. In *Cloudcomp 2009*, Munich, Germany, 2009.

[40] S. Palumbo. Is iaas moving beyond just cloud fluff? August 2010. http://www.businesswire.com/news/home/20100823005929/en.

[41] C. Pettey. Gartner identifies the top 10 strategic technologies for 2010. October 2009. http://www.gartner.com/it/page.jsp?id=1210613.

[42] A. L. Robertson. The high-availability linux project. Http://linux-ha.org/.

[43] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker. Applying nox to the datacenter. In *Proc. of workshop on Hot Topics in Networks (HotNets-VIII)*, 2009.

[44] L. M. Vaquero, L. R. Merino, J. Caceres, and M. Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, 2009.

[45] G. Wang and T. S. E. Ng. The impact of virtualization on network performance of amazon ec2 data center. In *INFOCOM'10: Proceedings of the 29th conference on Information communications*, pages 1163–1171, Piscataway, NJ, USA, 2010. IEEE Press.

[46] Y. Wang, E. Keller, B. Biskeborn, J. van der Merwe, and J. Rexford. Virtual routers on the move: live router migration as a network-management primitive. In *SIGCOMM '08: Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, pages 231–242, New York, NY, USA, 2008. ACM.

[47] T. Wood, P. Shenoy, A. Gerber, K. Ramakrishnan, and J. V. der Merwe. The case for enterprise-ready virtual private clouds. In *HotCloud, 2009*.

[48] T. Wood, P. Shenoy, K. K. Ramakrishnan, and J. V. D. Merwe. Cloudnet: A platform for optimized wan migration of virtual machines. Technical Report 2, University of Massachusetts, Amherst, 2010.