CloudNaaS: A Cloud Networking Platform for Enterprise Applications

Theophilus Benson, Aditya Akella
University of Wisconsin–Madison

Anees Shaikh, Sambit Sahu
IBM TJ Watson Research Center

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 CloudNaaS. Customers can leverage CloudNaaS 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. CloudNaaS primitives are directly implemented within the cloud infrastructure itself using high-speed programmable network elements, making CloudNaaS highly efficient. We evaluate an OpenFlow-based prototype of CloudNaaS 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.

Categories and Subject Descriptors

C.2.3 [**COMPUTER-COMMUNICATION NETWORKS**]: Network Operations

General Terms

Design, Performance

Keywords

Datacenter, Virtual Network

1. INTRODUCTION

Cloud computing is an emerging new model for the delivery and consumption for IT resources. Given the economic appeal and agility of this model, both small and large companies are increasingly leveraging cloud computing for their workloads [42, 43]. Despite this growing adoption, however, key challenges remain when migrating line-of-business production applications, including lack of fine-grained security, privacy, audit compliance, unpredictable performance, and poor reliability [48].

Underlying many of these challenges is the *absent or limited control available to customers to configure the network* in current cloud computing environments. The cloud network model has largely focused on providing basic reachability using dynamic or static IP addresses assigned to customer VMs, with basic firewall capabilities available at each virtual server. Several key network functions are generally not available, e.g., fine-grained network isolation for security or service differentiation, policy-based routing through middleboxes (for intrusion detection or audit compliance), control over addressing, and optimizations like protocol acceleration, path control, and distributed caching for improved performance and availability.

In this paper, we present the design, implementation, and evaluation of CloudNaaS (Cloud Networking-as-a-Service), a networking framework that extends the self-service provisioning model of the cloud beyond virtual servers and storage to include a rich set of accompanying network services. CloudNaaS gives customers deploying their applications on the cloud access to virtual network functions such as network isolation, custom addressing, service differentiation, and the ability to easily deploy a variety of middlebox appliances to provide functions such as intrusion detection, caching, or application acceleration. Unlike solutions based on third-party add-on virtual appliances and overlay networks, CloudNaaS primitives are implemented within the cloud infrastructure, and hence are highly efficient and transparent to cloud tenants and end-users. In CloudNaaS, all of these network services are unified in a single, extensible framework. This model has the potential to save cost and complexity compared to the current approach in which cloud customers must integrate a variety of point solutions from cloud providers and third parties to implement networking services.

The design of CloudNaaS leverages techniques such as software-defined networking to provide flexible and fine-grained control of the network (e.g., with OpenFlow-enabled devices), indirection to provide added control over addressing, and host-based virtual switches to extend the network edge into hypervisors. In an environment as large and dynamic as a cloud, however, a number of challenging issues must be addressed. For example, network devices are limited in the amount of control state, such as ACL-based tables, that they can maintain, and the rate at which state can be updated. Also, the dynamic nature of customer applications and infrastructure failures or downtime requires that network services be maintained or re-established under varying amounts of churn in the system. Our design and implementation of CloudNaaS includes algorithms and optimizations that reduce the impact of these

SOCC'11, October 27–28, 2011, Cascais, Portugal.

hardware limitations, and also improve its ability to manage the dynamic nature of cloud-delivered services.

The contributions of CloudNaaS may be summarized as follows:

- design of an integrated provisioning system for cloud applications and network services with simple and flexible interfaces for customers to specify network requirements
- optimizations to improve scalability and overcome hardware limitations of network devices to support cloud-scale multitenancy with tens of thousands of application deployments and hundreds of thousands of VMs
- an implementation of the system, with experimental and simulation-based evaluation using a variety of common cloud application workload models

We demonstrate the benefits of CloudNaaS using extension experiments and simulation experiments on a prototype implementation. The flexibility of CloudNaaS in supporting network services in the cloud is demonstrated through evaluation in a lab testbed with commercial programmable network switches. Using this environment, 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 CloudNaaS.

We evaluate the performance and scalability of CloudNaaS using a number of emulated scenarios with typical multi-tier interactive and batch application models. We focus on the performance of CloudNaaS in the face of dynamics such as network and host failures, and also as the number of customers and the size of the cloud varies. We use realistic reference applications including interactive n-tiered and batch applications. CloudNaaS 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. By using techniques such as caching and precomputation, the processing overhead and recomputation time for recovery is reduced significantly.

Our evaluation shows that CloudNaaS imposes low memory overhead on the network 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 network device memory for holding the forwarding state for large numbers of cloud tenants. These heuristics help reduce network switch memory usage by 96-99% compared to naive routing and forwarding, thereby helping CloudNaaS scale to host many enterprise applications. Our experiments show that our network aware VM placement strategy, in which VMs belonging to the same application deployment are placed topologically near each other, is better able to accommodate network service requests, particularly for applications with many VMs. Network-aware placement reduces the pathlength between VMs belonging to such applications by a factor of 3, and can support nearly 10% more applications than non network-aware algorithms.

2. BACKGROUND

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

Network Functions	On-path Middlebox	Layer 2 Broadcast	QoS	ACL	Static Addressing
EC2 [1]	N	N	N	Y	N
EC2+VLAN	N	Y	N	Y	N
EC2 w/VPC [2]	N	N	N	Y	Y
VPN-Cubed [5]	N	Y	N	Y	Y
CloudNaaS	Y	Y	Y	Y	Y

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

2.1 Limitation of Current Cloud Networking Mechanisms

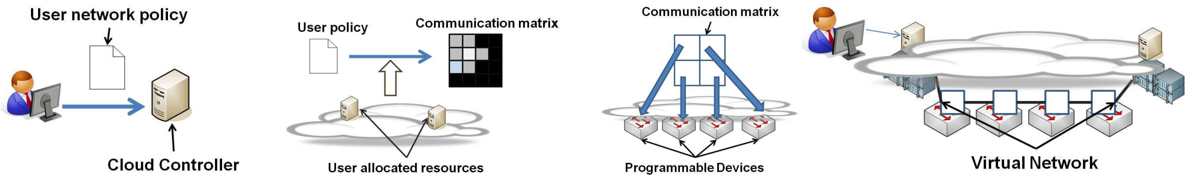
Below, we focus on three important challenges that arise from limited control over the networking capabilities in current clouds. In each case, we suggest a design requirements to address each limitation.

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 [32] 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 [32]. Also, recent studies [37] 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.
Requirement: Cloud tenants should be able to specify bandwidth requirements for applications hosted in the cloud, ensuring similar performance to on-premise deployments.

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 [3], caching [27] and application acceleration [13]. 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 [14], 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 [6]. 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.
Requirement: Ideally, tenants should have the ability to realize an identical data-plane configuration in the cloud to on-premise; this includes the ability to flexibly interpose a variety of middleboxes such as firewalls, caches, application accelerators, and load balancers.

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



(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 CloudNaaS framework.

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 [20], 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 [45] and third-party cluster resource managers (e.g., PaceMaker [12]).

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 [29], hence retooling them to ensure consistency in IP addresses is challenging and difficult to automate. *Requirement:* Applications should require little or no rewriting to handle networking (i.e., applications should run “out of the box” as much as possible), in particular for IP addresses and for network-dependent failover mechanisms.

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 CloudNaaS provides a framework with more comprehensive support for network-layer policies in the cloud.

3. RELATED WORK

Network services have started to receive more attention recently from cloud providers, but the network support is primarily targeted at a small set of capabilities. For example, Amazon recently extended its VPN services to include secure connectivity to isolated virtual instances with the ability to segment them into subnets and specify private address ranges and more flexible network ACLs [2]. Similarly, the Microsoft Windows Azure virtual network provides services for customers to integrate on-premise applications [16]. Both Amazon and Azure also provide network-related add-on services such as traffic load balancing across clustered VMs, and content distribution services using their distributed platforms.

There are also a number of third-party providers of network-related services delivered as virtual cloud appliances. Some available functions include fast data replication [8], application acceleration [17] and intrusion prevention [14]. Another delivery model is via overlays using nodes in the cloud to provide services such as custom addressing and encrypted communications [4, 5].

Both of these types of cloud network services (i.e., cloud-provided or third-party) address some of the gaps discussed in Section 2. But neither offers a single cloud networking framework that supports a wide variety of services without the need to integrate multiple offerings from multiple vendors, each with its own service model and management interface. Overlays have the advantage of supporting a greater variety of services, but usually with a negative impact on performance. In CloudNaaS, an extensive list of services can be provided under a single framework (from both customer and cloud provider perspectives), while also retaining the performance and efficiency of a network-level solution.

The research community has also advanced its view of the requirements and challenges in deploying diverse workloads in the cloud [22, 47]. Some network-related issues have been specifically addressed, including better control over routing [35], over bandwidth [24, 31], architectures for access control, privacy and isolation in the cloud [51, 33], or reducing disruption of services during migration [50]. Our goals are broader in the sense that CloudNaaS spans services that include access control, performance isolation, and control over network paths, e.g., through intermediaries. Control over middlebox placement in data centers has been also been considered in prior research [34] – our approach for managing middleboxes is conceptually similar to this work. Other recent work has also suggested frameworks for network services, e.g., access control in the cloud [44], or distributed network management [28], but these focus primarily on security issues and are not integrated with cloud provisioning.

Some experimental platforms also have similar goals to CloudNaaS, in terms of providing some measure of control over the testbed network [21, 30]. In Emulab, for example, users can specify the network topology and link characteristics using a modeling language similar to popular network simulation tools. Since these environments are designed for experimentation, they expose considerable low-level control over the network. In a multi-tenant cloud environment the network services exposed to customers need to be standardized, and designed to support application needs rather than low-level control over the network infrastructure for emulation.

4. CloudNaaS SYSTEM DESIGN

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

CloudNaaS overview. Figure 1 illustrates the sequence of main operations in CloudNaaS.

First, a cloud customer or tenant uses a simple policy language

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

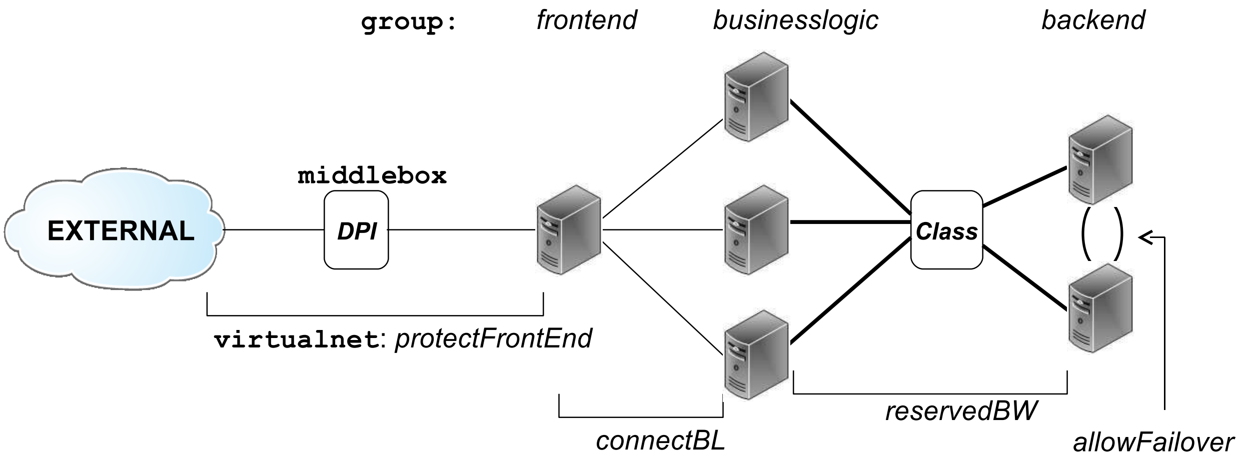


Figure 2: Example 3-tier application

to specify the network services required by his application (Figure 1(a)). We describe the syntax and semantics of the network policy specification below in Section 4.1.

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 customer. At the end of this step, the communication matrix is used to determine the optimal placement of the new VMs such that the cloud is able to satisfy the largest number of global policies in an efficient manner. This is done based on the knowledge of other customers' requirements and/or their current levels of activity. This step determines whether it is possible to map the new customer's logical requirements into the cloud's physical resources.

We then translate the logical communication matrix along with knowledge of the placement of VM locations into network-level directives (i.e., configuration commands or rules) for devices in the cloud (Figure 1 (c)). The customer's VM instances are deployed by creating and placing the specified number of VMs. We describe this in Section 4.2.

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 customer's needs. In addition, address-rewriting rules are instantiated within appropriate network locations to ensure that applications can use custom IP addresses within the cloud. We describe the last two steps in Section 4.3. The new cloud application deployment is then ready to run as per the customer's specifications.

CloudNaaS components: The CloudNaaS architecture consists of two primary communicating components, namely the *cloud controller* and the *network controller*. The cloud controller manages both the virtual resources and the physical hosts, and supports APIs for setting network policies. It addresses the steps shown in Figure 1 (a) and (b). The network controller is responsible for monitoring and managing the configuration of network devices as well as for deciding placement of VMs within the cloud. It handles the tasks outlined in Figure 1 (c) and (d).

4.1 Network Policy Specification

As part of the CloudNaaS system, we have developed a policy language that cloud customers can use to specify network services associated with their application deployments. The CloudNaaS specification language complements user-facing constructs in current clouds such as Amazon EC2 and Windows Azure. For example, our policy language could work with EC2's virtual instance identifiers when specifying network policies.

While the CloudNaaS policy language is just one candidate among a number of possibilities, we have found it to be sufficient to realize the requirements outlined in Section 2, and also extensible and intuitive to use. The constructs in the CloudNaaS policy language could also be leveraged in ongoing work to develop standard APIs for cloud networking in efforts such as OpenStack [11].

4.1.1 Network Policy Constructs

Our policy language provides a set of constructs for identifying the set of VMs that comprise an application and their network services. The basic abstraction is that of a virtual network segment that connects VMs together. Various functions and capabilities may be attached to a virtual network segment to define its behavior. Traffic is only allowed to reach a VM over an explicitly defined virtual network segment, hence providing a secure "default-off" model. This approach can be used to provide standard templates of network services and segments that implement pre-defined policies (e.g., for security). A brief description of the main constructs is given below. A more detailed description may be found in [25].

- `address`: specify a custom (e.g., private) address for a virtual machine. Other VMs on the same virtual network segment will be able to reach it using either the specified private address or the cloud address.
- `group`: create a logical group containing one or more virtual machines. Grouping VMs with similar function, e.g., members of a cluster, makes it possible for modifications to apply across the entire group without requiring changing the service attached to individual VMs.
- `middlebox`: name and initialize a new virtual middlebox by specifying its type and a configuration file. The list of available middleboxes and their configuration syntax is supplied by the cloud provider.
- `networkservice`: specify a set of capabilities to attach to a virtual network segment. The current CloudNaaS implementation supports 3 services: i) layer 2 broadcast domains, ii) link QoS (either standard best-effort or bandwidth reservation in Mbps), and iii) middlebox interposition (list of middleboxes that must be traversed in sequence). A virtual network segment may contain a combination of these three services.
- `virtualnet`: virtual network segments connect groups of

```
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=standard, mb=DPI}
9   networkservice connectBL =
    {l2broadcast=no, qos=standard, mb=none }
10  networkservice reservedBW =
    {l2broadcast=no, qos=10mbs, mb=Class}
11  networkservice allowFailover =
    {l2broadcast=yes, qos=standard, mb=none}
12  virtualnet allowFailover (backend)
13  virtualnet protectFrontEnd(frontend, EXTERNAL)
14  virtualnet connectBL(frontend,businesslogic)
15  virtualnet reservedBW(businesslogic,backend)
```

Figure 3: Network policies for example 3-tier application

VMs and are associated with network services. A virtual network can span 1 or 2 groups. With a single group, the service applies to traffic between all pairs of VMs in the group. With a pair of groups, the service is applied between any VM in the first group and any VM in the second group. Virtual networks can also connect to some pre-defined groups, such as `EXTERNAL`, which indicates all endpoints outside of the cloud.

4.1.2 Network Policy Example

To illustrate how the policy language is used in practice, we show an example specification for the 3-tier application deployment shown in Figure 2. In this example, middleboxes are used to perform deep packet inspection for incoming traffic and also to perform packet classification to distinguish high priority requests between the business logic and backend database servers. In the backend, a broadcast-based failover mechanism is used for notification in case the current master fails [7]. The database servers also use customer-specified addresses.

Figure 3 shows the corresponding network service specification using the constructs described above. The customer assigns the enterprise addresses to the database VMs (lines 1–2), and defines groups for the VMs in each tier (lines 3–5). The two middleboxes are named next, both using the default configuration (lines 6–7). The next lines define the network services required for each virtual network segment (lines 8–11). Note that the `standard`, i.e., best-effort, service is also defined to establish basic connectivity between the front-end server and the business logic tier. The other services specify middlebox traversal, bandwidth reservations, or layer 2 broadcast services. Finally, the last lines attach network segments between or within corresponding groups of VMs (lines 12–15).

4.2 Cloud Controller

In a 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 new requests or changes in workload by provisioning new virtual machines and allocating physical resources. CloudNaaS extends the cloud controller in several ways in order to facilitate better control over the network:

(1) accepts network policy specifications (in addition to requests for VMs) and parses them to generate a communication matrix for the tenant's resources. The matrix captures the requirements for the network between tenant VMs. An entry in the matrix indicates whether 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 any middlebox traversal required by traffic between the endpoints. The matrix is then passed to the network controller which interfaces with the programmable switches.

(2) prior to placing a VM on a physical host, the cloud controller consults the network controller to determine which hosts are candidates for placing the VM. The network controller utilizes a placement algorithm designed to minimize the network state and maximize the performance and the number of virtual networks that can be supported in the cloud (described further in Section 4.3).

(3) manages a software programmable virtual switch on each physical host that supports network services for tenant applications. The software switch is configured to connect any number of virtual machines to the physical network. The software switches are crucial for extending 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 (e.g., if a VM is removed or moves to a different host).

4.3 Network Controller

The network controller is a new component that CloudNaaS introduces into the cloud management system. It is responsible for configuring virtual network segments throughout the cloud by mapping the logical requirements in the communication matrix onto the physical network resources. It also controls resources, e.g., by determining VM placement and performing re-mapping when available resources change, to ensure that tenant requirements are consistently satisfied in an efficient manner.

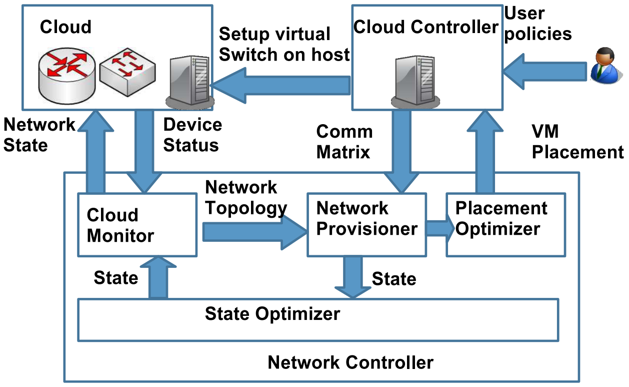


Figure 4: Internal components of the network controller.

Figure 4 shows the main modules of the network controller. It takes two inputs from the cloud controller: the communication matrix for a new request, and a list of physical hosts and available resources on each physical host. In addition, the network controller collects the current status of switches and links (along with link utilizations) and the current mapping of flows corresponding to the virtual network segments deployed in the cloud's physical network. The cloud monitor module periodically polls the devices for this state, but it can also receive triggered state updates from devices when they come up or when they detect a neighbor failure.

Based on these inputs, the controller first invokes the placement optimizer to determine the best location to place VMs within the cloud (and reports it to the cloud controller for provisioning). The controller then 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 segment. A similar set of actions must be taken by the network provisioner when remapping tenant virtual network segments in case of network failures. In addition to these basic actions, the network provisioner performs other control functions in the network, such as tearing down a tenant's application, and installing address rewriting rules in host virtual switches. We discuss these tasks in more detail below.

Provisioning and De-provisioning Virtual Network Segments To provision a virtual network segment 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 requested attributes. The constraints can result in various computations. They might involve simply finding a loop-free path when the user requests best-effort connectivity between two VMs, or 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 [49] computations for generating paths with QoS requirements, standard shortest-paths for best-effort paths, and spanning tree algorithms for generating broadcast paths.

If a path is found, the controller generates the appropriate configuration commands and executes them on 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 6.1.

De-provisioning the virtual network segments belonging to a customer application happens in a similar fashion; we omit the details for brevity.

VM Placement using bin-packing One of the key optimizations in CloudNaaS is the joint placement of virtual machines with virtual network segment provisioning. The programmable network devices used in our design provide the fine-grained control required for per-customer network services, but, as we discuss below in Section 6, they have limited resources available to store state for the virtual network segments. Hence, the objective of the optimization is to place a VM so that the number of networking entries that the controller installs on the physical network devices is minimized. We further try to minimize the number of network devices between communicating VMs to reduce network latency and limit the impact of oversubscription in the cloud network topology (e.g., between server racks located behind different aggregation switches)[2]. This has the benefit of improving application performance in addition to reducing state in the network devices.

We formulate the placement algorithm as an optimization problem that searches through the set of available physical hosts for an optimal location to place a VM. The constraints are to (1) place VMs on a physical host with sufficient free capacity to support the VM's minimum resource requirements and (2) to ensure that a path exists between all communicating VMs. For efficiency and speed, we employ a *bin-packing heuristic* (first-fit decreasing) that sorts virtual network segments according to the number of communicating VMs. The virtual network segments to be created are processed in order, starting by determining placement of the VMs in the larger virtual network segments. The algorithm attempts to pack VMs for a specific virtual network segment on the same physical host, then within the same rack, then on hosts behind the same aggregation devices, and finally on any available physical host with sufficient capacity. This ensures that VMs on the same virtual network segment are co-located within the same region of the data center whenever possible. To distribute load and improve tolerance to localized failures, when handling a new virtual network segment, the algorithm starts placing VMs into a randomly chosen rack with sufficient free capacity (or else at the rack with the highest available capacity).

Addresses Rewriting The final important function of the network provisioner is to perform address mapping to allow enterprises to reuse existing addresses (i.e., custom addressing). To achieve this, the cloud controller provides the network controller with a map of the VM names to their custom addresses as well as the set of VMs communicating within them as indicated in the policy specification by the cloud customer. For each VM in the list, the network controller installs a rewriting rule in the software switch resident on the hosts for the set of VMs communicating with it. This rule translates the destination address from the custom address to the cloud-assigned address before forwarding. For other VMs or traffic using cloud addresses, rules are installed for forwarding without rewriting the destination address. In cases where VMs are migrated, the rewriting rules are recreated at the appropriate software switches on the new hosts. Thus, we leverage programmability of the cloud, in particular the software switches to enable customer applications to use their own custom addressing schemes.

5. PROTOTYPE IMPLEMENTATION

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

OpenNebula cloud controller. We leverage the OpenNebula 1.4 cloud framework to implement the cloud controller component of CloudNaaS. 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 §4.1, to keep the generate the communication matrix, to defer VM placement decisions to the network controller 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 setup. 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 [9] 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 CloudNaaS 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 4.3 includ-

[2] Recently proposed network topology designs can also help to improve the "east-west" VM-to-VM bandwidth in cloud data centers by reducing or eliminating oversubscription.

ing, provisioning and de-provisioning virtual networks, handling host and network dynamics, VM placement 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 [26], we redirect all ARP traffic to the network controller who then provides the appropriate responses.

6. ADDRESSING PRACTICAL ISSUES

In managing the mapping of customer virtual network segment to the physical network the network controller in CloudNaaS has to handle several challenges, namely: (i) installing forwarding state to implement tenant policies while being constrained by network device processing and memory limitations, and (ii) ensuring that network policies persist in the face of dynamics such as device and link failures. In this section, we discuss the techniques used by CloudNaaS to deal with these practical issues.

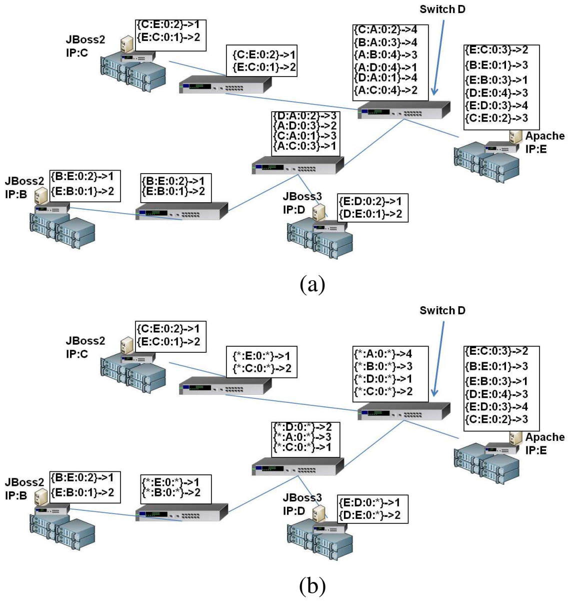
6.1 Hardware Device Limitations

CloudNaaS uses the fine-grained control provided by programmable devices to provide Quality-of-Service guarantees, middlebox interposition, tenant-defined broadcast domains, and address rewriting. The drawback of using fine-grained controls to realize these services is the state explosion they create in network devices. In using the APIs provided by OpenFlow and NOX to configure the network, CloudNaaS 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.

Forwarding entries are stored in Ternary Content-Addressable Memories (TCAMs) which are limited in size, ultimately limiting the number of forwarding entries and virtual networks that can be instantiated. Unless this memory is carefully managed, it may not be possible to support a large number of virtual networks in the cloud.

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

Optimization 1: Best-effort traffic Our first optimization is used for configuring flow table rules for best effort traffic. It works simply as follows: we install full flow-based rules in the host's virtual switch, and simple destination-based rules in the physical switches (i.e., source addresses are wild-carded in the latter case). This optimization leverages the observation 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 in each virtual network. We illustrate this optimization in Figure 5. In (a), we show the rule-set at different devices without the optimization. Device D carries 6 flow table rules. As shown in (b), with the optimization device D holds 4 flow table rules, a 33% reduction.



(a)

(b)

Figure 5: 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.

Using destination-based forwarding prevents middlebox traversal, however. To allow middlebox traversal, CloudNaaS installs rules in the software switches of the source VM and subsequent middleboxes that encapsulate and tunnel packets from the source VM or current middlebox to the next middlebox.

Optimization 2: QoS traffic. The second optimization extends the idea for best-effort traffic 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 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 needed per physical device. If a pair of paths only share some links and devices, then the controller uses different ToS values for each path, which leads to separate entries in the 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 maximum 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 Top-of-Rack (ToR) switch. This results in gains of $O((S) * N/P)$, where S is the number of distinct paths to a ToR switch, N is the number of virtual machines, and P is the size of prefix assigned to each ToR switch.

As we show in Section 7, the above optimizations coupled with our bin-packing placement heuristic (Section 4.3) result in substantial savings in network device switching memory, thereby helping support several thousands of complex enterprise services.

6.2 Cloud Dynamics

Network services must be able to withstand dynamic events in the cloud, such as link failures, device failures, or changes to the network policy specification. To handle these events, CloudNaaS employs precomputation and caching to reduce the impact of device or link failures on network services.

Policy Changes and Host/VM dynamics: When host conditions change due to oversubscription or failure, the cloud controller may migrate a customer's VM to another host and regenerate the communication matrix. The cloud controller also regenerates the communication matrix when a customer submits changes to his network policies. When the matrix is regenerated, the cloud controller informs the network controller of the new matrix, which then triggers reprovisioning of the corresponding virtual networks. To do this without causing significant disruption to existing tenants, the network controller performs reprovisioning for only the changed portions of the communication matrix.

Device/link failures: When devices or links fail, virtual networks may be rendered invalid. In such situations, CloudNaaS tears down and re-provisions all virtual networks which are dependent on the failed links or devices. To reduce downtime CloudNaaS employs precomputation and caching of alternate paths. CloudNaaS maintains an association between devices / links and the set of dependent virtual networks, thus allowing it to quickly determine the virtual networks to re-provision when a particular link or device fails. To reduce the time to re-provision these virtual networks, CloudNaaS precomputes network state for different failure scenarios. In our current implementation CloudNaaS precomputes network state to handle failure of core and aggregation layer devices – a small number of devices having significant state. Failure of these devices can be resolved by simply looking up and installing the precomputed and cached network rules.

7. CloudNaaS SYSTEM EVALUATION

In this section, we present an experimental evaluation of the CloudNaaS prototype in both a lab-based cloud as well as a large-scale emulated cloud. In Section 7.1, we describe our simulator and experimental setup.

Our first goal is to demonstrate the key primitives supported in CloudNaaS, validating the ability to flexibly specify and instantiate a variety of network functions in the cloud and to help minimize application rewrites and reconfigurations due to addressing changes in the cloud (Section 7.2).

We then conduct a variety of experiments examining various key aspects of CloudNaaS. In Section 7.3, we examine the impact of various optimizations described in Section 6 and their ability to help CloudNaaS operate in a scalable fashion under network device resource constraints. In Section 7.4, we study the performance of the network controller in terms of the overhead of virtual network computation at scale, and the ability to ensure that the cloud operates gracefully under failures. We also examine the impact of the bin-packing placement heuristic in facilitating the admission of a large number of virtual network requests from applications and in supporting high application performance.

On the whole, our experiments also show that CloudNaaS is flexible in that it can support a variety of enterprise application requirements, and its 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, despite host and network device resource constraints.

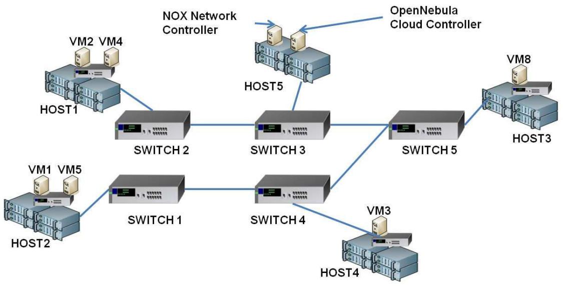


Figure 6: Experimental testbed.

7.1 Experiment Setup

We deployed and validated CloudNaaS on a cloud testbed consisting of 5 physical hosts and 5 network switches connected as shown in Figure 6. 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.

Simulator: In the absence of a real large-scale testbed to study the impact of the proposed optimizations and the performance of the network controller, we instead developed a simulator to model the network controller. We emulate various network events as well as the messages exchanged from the cloud controller and the network devices to the network controller (e.g., the control messages sent to the network controller by the switches when a link failure occurs or when the switch is powered on). We also simulate the user policy files and the equivalent communication matrix that the cloud controller would send to the network controller. The network controller operates as usual, determining VM placement and computing the required flow table entries for each switch based on the placement 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 simulations, the network controller is deployed on a 2.40GHz quad core Intel Xeon PC with 4GB of memory running Ubuntu 10.

Workloads: In these experiments, we use two types of reference enterprise applications: interactive multi-tiered applications and batch multi-tiered applications. We generate network policies and communication matrices consisting of varying sizes (number of VMs) and numbers of each type of service.

Our interactive application model is typical of many enterprise applications which separate presentation (front-tier), application (business-logic tier), and data (database) components for scalability and performance (e.g., SAP R/3 [18]).

For a "small"-sized interactive application, we use 5 VMs, including 2 VMs for redundant databases, and a single VM for each of the other 2 tiers. For the medium sized interactive application, we expand this model to 12 VMs allowing for redundancy at all tiers and introducing middle-boxes for IDS and DPI. Finally, for the large sized 3-tier application, we build on the medium sized model by further increasing the number VMs in the front-tier and the business-logic tiers to accommodate a larger user base; the total number of VMs is 19.

For the interactive applications, the following network primitives are employed: (1) a VLAN isolating the application from other applications in the cloud, (2) QoS requirements between the front-tier and business-logic tier, and (3) on path middlebox traversal between the front-end and end-users. The QoS requirements for the different interactive applications are proportional to the size of the applications with the small requiring 20Mbps, the middle requiring 70Mbps, and the large requiring 100Mbps on the paths connecting the front-tier to the business-logic servers.

For the batch application, we model Microsoft's SharePoint deployments – we derive the networking requirements from the reference architecture [38] for small, medium, and large enterprises which (as with the interactive applications described earlier) differ in the number of VMs hosting employee facing IIS and also in the number of VMs in the service that are used to crawl, index, and search data. The batch application consists of a cluster of VMs used to host IIS and MSSQL applications and a second cluster of VMs used to crawl and index the data in websites and databases stored in the first cluster stored. The network requirements for the batch applications are a VLAN isolating the service from other services in the cloud.

Topology: For our data center network model, we consider a canonical 3-tier network (consisting of Top-of-Rack (TOR), aggregation, and core switches) and Fat-Tree [19]. For each network model, we generate three topologies with 6K, 20K, and 30K physical hosts capable of running a maximum of 54K, 140K, and 270K VMs. In each topology, a physical hosts can support at most 9 VMs and each physical host is connected to an edge switch with a 1Gbps link.

We consider 3-tier topologies with 200, 500, and 1000 ToR switches, each connected to 30 hosts. For the 3-tier network, these data center models each have 2 core switches, and 20, 50, and 100 switches in the aggregation layer, respectively. As is typical, each ToR switch has two 10Gbps uplinks to the aggregation layer, and aggregation switches are dual-homed to the core layer with 10Gbps links.

For the Fat-Tree topologies, we follow [19], varying the K parameter to accommodate varying numbers of physical hosts. All links in the Fat-Tree topologies are assumed to be 1Gbps.

7.2 Functional Validation

We begin by demonstrating the flexibility and functionality of CloudNaaS in implementing several different network functions and policies.

By submitting different network policies to the user interface, we were able to implement, within the confines of the cloud, the different enterprise application networking scenarios below. We perform the testing on the cloud testbed discussed earlier.

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 VMs belonging to other applications (VMs 4–6). For the purpose of this evaluation, we force the VMs to be placed as shown – this set-up helps us study how effective CloudNaaS's primitives are under various interesting situations. Note that we evaluate the placement algorithm (Section 4.3) and the benefits it offers later in this section.

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 fail-over service. We verified 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.

Quality-of-Service: To demonstrate the QoS 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 4. We instantiated file transfers from VM1 to 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.

Size of Ruleset	# of Large Interactive Apps.	Memory (in MB)
65536	3276	33
131072	6553	37
196608	9830	57
262144	13107	77
327680	16384	94

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

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 middle-box 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.

7.3 Impact on Cloud Infrastructure

The design of CloudNaaS introduces a number of changes to the cloud infrastructure. In this section, we summarize 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 CloudNaaS 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 [10] at each host, which requires instantiation of TAP interfaces [15] in place of the standard KVM public bridges. We observed that the resource usage of the TAP interfaces was minimal, and should not impact the number of VMs that can be supported.

In examining the overhead of installing rulesets into 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 virtual switch is able to store 300K entries, sufficient to support the virtual network segments for 16K large 3-tier applications when viewed across the entire data center, in less than 100MB (of the 4GB available to the Open vSwitch) of memory per host. With a limit of 300K rules, CloudNaaS is able to allocate on average 10K forwarding rules, or 1.8K virtual network segments for each VM on the host – indicating that most practical virtual network segments sizes can be supported for each VM deployed on the host for even large applications. 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.

Algorithms	Virtual Switch	ToR	Agg	Core	# of Large Apps
Default Placement w/o optimizations	365	13K	235K	1068K	4K
Default Placement + Destination Forwarding	0%	3%	21%	39%	6.7K
Default Placement + Qos Forwarding	0%	2%	20%	30%	5.4K
Default Placement **+ Qos + Destination + Prefix**	**0%**	**93%**	**95%**	**99%**	**12.2K**

Table 3: Results indicating effect of flow entry optimizations and the default VM placement strategy on switches at each tier. The bottom four rows show the percentage reduction in flow table size.

Algorithms	Virtual Switch	ToR	Agg	Core	# of Large Apps
Bin-Packing w/o optimizations	313	5K	13K	20K	15K
Bin-Packing + Destination Forwarding	0%	49%	47%	46%	15.7K
Bin-Packing + Qos Forwarding	0%	41%	40%	40%	15.6K
Bin-Packing + **Qos + Destination + Prefix**	**0%**	**99.8%**	**99%**	**99%**	**15.9K**

Table 4: Results indicating effect of flow entry optimizations and the bin-packing VM placement strategy on switches at each tier. The bottom four rows show the percentage reduction in flow table size.

In Section 6, 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 large instances of the reference interactive application (i.e., 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. Finally, we observe that employing careful placement decisions using our bin-packing heuristic results in further reduction of the state for the ToR and Agg devices (Table 4). With all our optimizations taken together, we can support 15.9K large interactive applications simultaneously, which is 4× more that what can be supported without any optimizations (Table 3).

7.4 Network Controller Performance

Next, we evaluate the ability of CloudNaaS's network controller to scale to provisioning and managing a large number of virtual networks in a large-scale cloud data center.

7.4.1 Impact of Placement

Placement of VMs plays a crucial role in determining the performance between VMs, the number of QoS requests that can be satisfied, and the amount of state within the network. In this section, we examine the benefits of careful VM placement. We compare our bin-packing heuristic against the default placement strategy used by current cloud platforms, namely OpenNebula [39] and Eucalyptus [40]. The default placement algorithm used by both platforms is a striping algorithm which aims to spread VMs across physical hosts in a round robin fashion. Due to space constraints, we only present our findings for deploying Large interactive and batch services (Lg), medium interactive and batch services (Med), and an equal combination (Eq) of small, medium, and large interactive and batch services on the canonical DC topology; however, we observed similar findings for the Fat-Tree topology.

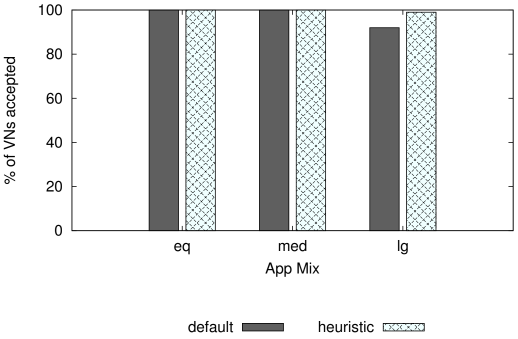


Figure 7: Number of virtual network segments successful supported by CloudNaaS under the bin-packing and the default placement strategies as a percentage of those supported by an optimal placement strategy.

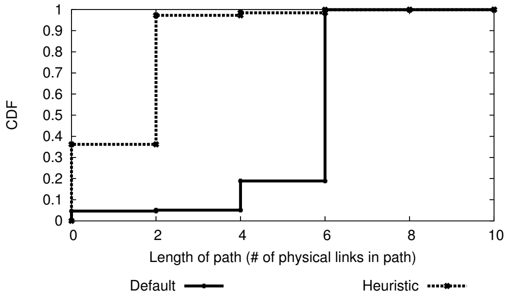


Figure 8: Length of paths, in # of links, between different VMs under different VM placement strategies.

First, we examine the impact of placement on the ability of the network to satisfy the varying QoS requirements placed on it by the different applications. Figure 7, presents a bar graph of the number of virtual network segments admitted as a fraction of the ideal number that can be accommodated by an optimal placement algorithm. The figure shows that both placement schemes satisfy all QoS requirements when only small and medium-sized interactive applications are run within the cloud. However, when only large-sized services are used then our placement algorithm is able to satisfy all requests whereas as uninformed placement approach denies roughly 10% of the requests: in each case when a virtual network segment is denied, the default placement algorithm is unable to satisfy one or more of the network QoS requirements specified by the virtual network segment.

To determine the performance implications of the different placement algorithms, we examine the length of the paths between communicating VMs in Figure 8 for the requests that were accepted in each case. For simplicity, we show our results only for large interactive applications. We observe that paths are in general shorter

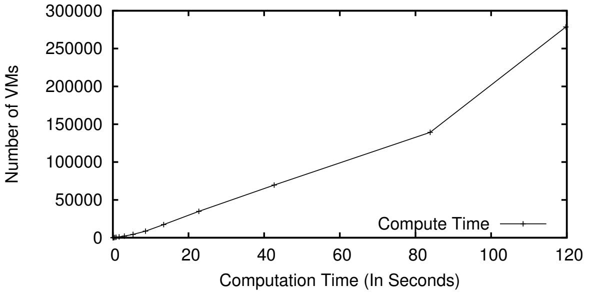


Figure 9: Virtual network segment computation time for Large interactive applications on a tree topology with 30K hosts.

with our placement algorithm: 99% of the paths created using the heuristic are less than 2 links long indicating that these paths never leave the ToR. The naive strategy results in longer paths, potentially resulting in poorer application performance due to greater interaction with cross traffic.

7.4.2 Virtual Network Computation

First, we examine the time taken to initialize several large interactive applications. Recall that each 3-tier application contains a relatively large number of virtual machines and a complex set of virtual networks and policies. We assume the canonical tree interconnect.

Figure 9 shows the amount of time taken to simultaneously instantiate network services for as many as 270K VMs in total spread across nearly 16K instances of the large interactive application (this is the number of large interactive application instances that can be "packed" onto the physical hosts using our placement algorithm; details later in this section). 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 [46]. From the figure, we observe that it takes about 120s in total 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 [41]. We found similar scaling properties for other reference applications and application mixes, as well as for the Fat-Tree interconnect (omitted for brevity).

7.4.3 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. For simplicity, we focus on a scenario consisting of large interactive application instances and a 3-tier network topology.

In our failure model, a link, switch, or host is randomly selected to fail. We measure the time taken by CloudNaaS'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.

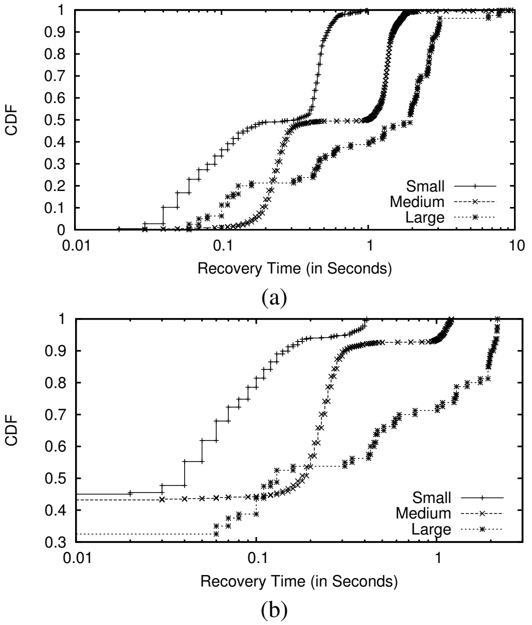


Figure 10: Virtual network segment recomputation time under link failures for large interactive applications on a tree topology with 30K hosts. (a) Without caching and precomputation (b) With caching and precomputation of core and aggregation devices.

We run each experiment around 150 times to generate the distribution of re-computation 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 10, 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 look-up 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 11 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 4).

8. ADDITIONAL CONSIDERATIONS

In this section, we briefly address some additional considerations

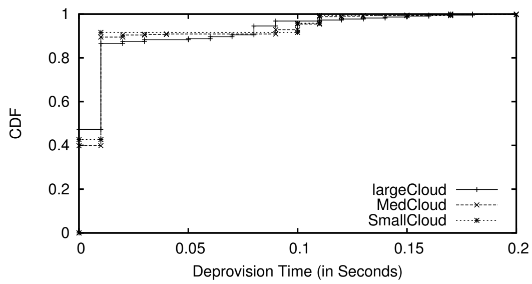


Figure 11: Virtual network deprovision time under host failures for large interactive applications on a tree topology with 30K hosts.

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.

OpenFlow Programmable Devices In our current prototype we use Openflow switches as the programmable devices within the network. However, our design is not tied to the OpenFlow platform and API. As stated earlier, the choice of programmable devices affects the set of configuration commands that are generated, how these commands are pushed to the devices, and how the effects of the commands are undone from the devices. However, we believe irregardless of the type of programmable devices used, these devices will have physical limitations and the general optimizations described in section 6.1 will be required to overcome such restrictions.

Furthermore, unlike other programmable devices, the OpenFlow platforms offers the advantage of easy deployment. Many device vendors including Cisco, HP, and NEC have developed firmware patches that transform existing data center grade switches into OpenFlow enabled switches. Given these firmware patches, we believe that out current implementation of CloudNaaS can be easily adopted and deployed by many existing cloud providers.

Managing Cloud Network Services: In this paper we described CloudNaaS 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 CloudNaaS 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.

WAN Extension: 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 service. This could be achieved by integrating the virtual network in the cloud with a cooperating ISP or overlay network provider. The CloudNaaS 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.

9. CONCLUSION

We presented CloudNaaS, 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 CloudNaaS leverages programmable network devices and supports isolation, middlebox functions, and Quality-of-Service, and helps minimize application rewrites and reconfigurations by allowing applications to use existing address spaces. CloudNaaS 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 CloudNaaS 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 CloudNaaS performs well in the face of large numbers of provisioning requests, and network and host dynamics. We showed that our optimizations for VM placement and for forwarding table aggregation help in making more effective use of the resources of the cloud's physical hosts and network devices, thus helping the cloud scale to support a multitude of enterprise applications.

10. REFERENCES

[1] Amazon EC2. http://aws.amazon.com/ec2/.

[2] Amazon Virtual Private Cloud. http://aws.amazon.com/vpc/.

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

[4] CloudSwitch. http://www.cloudswitch.com.

[5] Cohesive FT:VPN-Cubed. http://www.cohesiveft.com/vpncubed/.

[6] Elastic Load Balancing. http://aws.amazon.org/elasticloadbalancing/.

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

[8] NetEx. http://www.netex.com.

[9] NOX. http://noxrepo.org/.

[10] Open vSwitch project. http://www.vswitch.org/.

[11] OpenStack Cloud Software. http://wiki.openstack.org.

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

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

[14] SOURCEfire. http://www.sourcefire.com.

[15] TUN/TAP device driver. http://vtun.sourceforge.net/.

[16] Windows Azure Platform. http://www.microsoft.com/windowsazure/.

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

[18] "sap web application server, sap r/3 enterprise release 4.70". http://help.sap.com, 2004.

[19] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *SIGCOMM*, 2008.

[20] Amazon. Using DHCP Options. http://docs.amazonwebservices.com/AmazonVPC /latest/DeveloperGuide/index.html?UsingDHCPOptions.html, 2010.

[21] D. Andersen, H. Balakrishnan, M. Kaashoek, and R. Morris. Resilient Overlay Networks. In *SOSP '01*.

[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] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Towards Predictable Datacenter Networks. In *SIGCOMM '11*, Toronto, Canada.

[25] T. Benson, A. Akella, S. Sahu, and A. Shaikh. EPIC: Platform-as-a-service model for cloud networking. Technical Report TR1686, University of Wisconsin, Dept. of Computer Science, 2011.

[26] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. Mckeown, and S. Shenker. Ethane: Taking control of the enterprise. In *SIGCOMM '07*.

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

[28] C. Dixon, H. Uppal, V. Brajkovic, D. Brandon, T. Anderson, and A. Krishnamurthy. ETTM: a scalable fault tolerant network manager. In *NSDI '11*.

[29] 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.

[30] Emulab. Emulab - network emulation testbed. `http://www.emulab.net/`.

[31] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang. Secondnet: a data center network virtualization architecture with bandwidth guarantees. In *Co-NEXT '10*.

[32] 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*.

[33] F. Hao, T. Lakshman, S. Mukherjee, and H. Song. Secure Cloud Computing with a Virtualized Network Infrastructure. In *HotCloud '10*.

[34] D. A. Joseph, A. Tavakoli, I. Stoica, D. Joseph, A. Tavakoli, and I. Stoica. A policy-aware Switching Layer for Data Centers. In *SIGCOMM '08*.

[35] E. Keller and J. Rexford. The "platform as a service" model for networking. In *INM/WREN '10*, San Jose, CA, USA, 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] C. Metz. Amazon cloud accused of network slowdown. http://www.thebitsource.com/featured-posts/rackspace-cloud-servers-versus-amazon-ec2-performance-analysis/, January 2010.

[38] Microsoft. Planning and architecture for sharepoint server 2010. http://technet.microsoft.com/en-us/library/cc261834.aspx.

[39] D. Milojicic, I. M. Llorente, and R. S. Montero. Opennebula: A cloud management tool. *IEEE Internet Computing*, 15:11–14, 2011.

[40] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The Eucalyptus Open-Source Cloud-Computing System. In *CCGRID '09*.

[41] 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.

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

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

[44] L. Popa, M. Yu, S. Y. Ko, S. Ratnasamy, and I. Stoica. CloudPolice: taking access control out of the network. In *HotNets '10*.

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

[46] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker. Applying nox to the datacenter. In *HotNets '09*.

[47] 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.

[48] 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.

[49] Z. Wang and J. Crowcroft. Bandwidth-delay based routing algorithms. In *Global Telecommunications Conference, 1995. GLOBECOM '95., IEEE*, volume 3, nov 1995.

[50] T. Wood, K. K. Ramakrishnan, P. Shenoy, and J. van der Merwe. CloudNet: dynamic pooling of cloud resources by live WAN migration of virtual machines. In *VEE '11*.

[51] T. Wood, P. Shenoy, A. Gerber, K. Ramakrishnan, and J. V. der Merwe. The Case for Enterprise-Ready Virtual Private Clouds. In *HotCloud, 2009*.