# PHYSICAL RESOURCE MANAGEMENT AND ACCESS MEDIATION WITHIN THE CLOUD COMPUTING PARADIGM

A Thesis

by

HUTSON KEITH BETTS

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2012

Major Subject: Computer Engineering

# PHYSICAL RESOURCE MANAGEMENT AND ACCESS MEDIATION WITHIN THE CLOUD COMPUTING PARADIGM

A Thesis

by

HUTSON KEITH BETTS

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

| | |
|---|---|
| Chair of Committee, | Riccardo Bettati |
| Committee Members, | A. L. Narasimha Reddy |
| | Radu Stoleru |
| Head of Department, | Duncan M. Walker |

August 2012

Major Subject: Computer Engineering

# ABSTRACT

Physical Resource Management And Access Mediation Within The Cloud Computing Paradigm. (August 2012 )
Hutson Keith Betts, B.S., Texas A&M University
Chair of Advisory Committee: Dr. Riccardo Bettati

Cloud computing has seen a surge over the past decade as corporations and institutions have sought to leverage the economies-of-scale achievable through this new computing paradigm. However, the rapid adoptions of cloud computing technologies that implement the existing cloud computing paradigm threaten to undermine the long-term utility of the cloud model of computing. In this thesis we address how to accommodate the variety of access requirements and diverse hardware platforms of cloud computing users by developing extensions to the existing cloud computing paradigm that afford consumer-driven access requirements and integration of new physical hardware platforms.

# NOMENCLATURE

| | |
|---|---|
| API | Application Programming Interface |
| AWS | Amazon Web Services |
| SCADA | Supervisory Control And Data Acquisition |
| VLAN | Virtual Local Area Network |
| VNEL | Virtual Network Engineering Lab |
| VPC | Virtual Private Cloud |

# TABLE OF CONTENTS

Page

Page

# LIST OF TABLES

# LIST OF FIGURES

FIGURE Page

# 1. INTRODUCTION

## 1.1 Motivation

Cloud computing has seen a surge over the past decade as corporations and institutions have sought to leverage the economies-of-scale achievable through this new computing paradigm. Large-scale virtualization has been the predominant form of cloud computing, with its low cost of entrance and ability to scale quickly. Unlike traditional economies-of-scale, which allow for low-cost manufacturing of large quantities of a single resource type, the cloud computing paradigm allows for a wide variety of resource types to be created. Furthermore, the resources offered by cloud providers are accessible to consumers from any point on the Internet. However, the rapid adoptions of cloud computing technologies that implement the existing cloud computing paradigm threaten to undermine the long-term utility of the cloud model of computing. Foremost, these technologies hinder the ability for projects dependent on specific physical hardware platforms to integrate with cloud tool-kits. This inhibits the ability for those projects to leverage the benefits of cloud computing. Also, to make resources universally accessible, cloud computing technologies have taken a generalized approach towards providing consumers access to their resources, further hindering consumers that require special access accommodations. In this thesis we address how to accommodate the variety of access requirements and diverse hardware platforms of cloud computing users by developing extensions to the existing cloud computing paradigm that afford consumer-driven access requirements and integration of new physical hardware platforms.

This thesis follows the style of *IEEE Security & Privacy*.

## 1.2 Background

Several well-known cloud providers such as Amazon's Elastic Compute Cloud [1], Rackspace [2], and GoGrid [3] support large-scale deployments of *virtualized* systems across commodity hardware. Some characteristics that define the cloud computing paradigm include elasticity of consumer-owned resources, scalability to meet demand, multitenancy of a cloud provider's infrastructure, and many more. Other providers, such as Emulab [4] and VNEL [5] offer support for the allocation and consumption of *physical* hardware by end-users. Management of physical assets for consumption by customers is handled through mechanisms distinct from those used by traditional cloud providers, but still supporting a similar life-cycle management paradigm.

To assist with the adoption of the cloud model of computing, and the setup of cloud computing installations, several publicly available tool-kits have been developed, such as [6] [7] [8] [9] [10]. Deployments of these tool-kits by private entities for the purpose of leveraging the cloud model of computing for internal use are called *private clouds* [11]. However, as noted by [12], private clouds are assumed, by some individuals, to violate the economies of scale achievable by pooling resources since each private entity continues to use their existing infrastructure. This has lead to a unified approach, called *hybrid cloud computing*, that allows for enterprises to scale beyond their internal resources into publicly available clouds [11]. An example of this type of computing can be seen in Figure 1.1. The interoperability between clouds that has allowed hybrid cloud computing, or the leveraging of multiple clouds, has lead to the coining of the term *intercloud* [13], or "cloud of clouds" [14] [15]. In some cases, resources deployed to the cloud must communicate with one another privately over direct virtual or physical connections. Two or more resources communicating directly over dedicated, private connections form a special type of cloud called a *virtual private cloud*, or VPC, [16]. Amazon's EC2 platform supports virtual private clouds for its customers through their Virtual Private Cloud [17] service. Cloud

computing tool-kits also facilitate virtual private clouds by supporting the ability to create virtual private networks between virtualized resources.

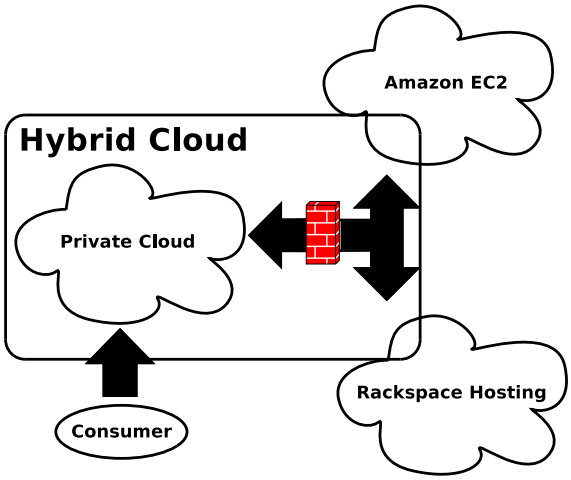


**Fig. 1.1.**: Hybrid cloud computing.

In a number of settings, such as training, educational learning, high-performance cyber-physical systems, or emergency recovery systems, it would be beneficial to integrate physical components into traditional cloud computing platforms, making those components accessible in the same manner as virtualized components. For example, a supervisory control and data acquisition (SCADA) security training facility may want to integrate physical SCANDA components into a virtual private cloud environment. Unfortunately, commercial and educational large-scale resource providers do not offer a unified abstraction for both physical and virtualized resources. On one hand, providers like EC2 offer virtual machine instances through an interface which requires users to choose an instance size, each size specifying the amount of memory or processors to be allocated. Cloud computing tool-kits can be leveraged more openly by allowing users to specify the amount of memory or processors they desire directly. On the other hand, providers like Emulab require their customers to specify exactly the type of hardware they wish to reserve. This is achieved by a catalog-like list of available hardware, along with the hardware specifications. Users

must determine the appropriate hardware platform from the catalog when issuing their request to reserve resources. Also, providers like Emulab treat virtualization as a by-product of their hardware allocation process, leaving it to the user to deploy a cloud computing tool-kit on their reserved physical systems to effectively leverage large-scale virtualization. In addition to having different ways for allocating and managing resources for consumers, providers also differ in the methods they use for permitting access to those resources. Each provider restricts their customers to a particular method, or methods, for accessing the resources they have been allocated; one provider makes resources accessible directly through a network interface, another provider connects resources through both a network proxy and a console server.

One important reason for supporting physical resources along with virtualized resources within a cloud computing environment is so that specialized hardware, which may be impossible to virtualize, can co-exist with virtualizable systems using the same deployment and management infrastructure of a traditional cloud computing platform. Furthermore, leveraging specialized hardware through a cloud computing platform has a number of other benefits: First, allowing the hardware to be shared through a cloud platform allows development costs to be amortized for per-hardware tool-kits. Second, acquisition, management and replacement costs can be more easily amortized as new hardware is introduced and old hardware depreciates. In many environments, costs associated with the acquisition and management of specialized hardware is not economical. For example, within academia specialized hardware might only be required for a short duration, such as a short-term grant-funded project. Similarly, newly formed businesses partaking in a hardware-related venture might not posses adequate start-up funds. Instead, those hardware systems could be operated by cloud providers, who could then offer those assets on-demand to consumers at a greatly reduced cost, effectively achieving economies-of-scale through global resource sharing.

## 1.3 Purpose

Despite the prevalence of cloud providers and the growing availability of open source tool-kits, no existing platform bridges the gap between hardware-based and virtualization-based platforms or supports consumer-driven access requirements. If, as proof-of-concept, an open-source tool-kit could be extended with support for hardware and virtualization abstraction and support for multiple access methods, the benefits could be substantial. For instance, consumers of cloud computing could leverage the properties and peripherals of physical hardware using the cloud computing paradigm. Also, consumers can dictate how access should be granted to their cloud-enabled resources, tailoring access based on need or purpose.

These improvements to the cloud computing paradigm can enhance the underlying resource management infrastructure for the Web Access Exercise System (WAES), version 2.0, more concisely known as WAES2 [18]. WAES2 is a course material authoring, course management, and case-based instructional learning tool, supported by research at Texas A&M University, for the benefit of instructors and students at community colleges. Opening up the allocation of cloud resources to include physical assets and consumer-driven access requirements facilitates a greater diversity of environments for teaching.

## 1.4 Objective

In this research, we will first evaluate existing methods for accessing resources deployed within cloud computing environment. Next, we will extend the cloud computing paradigm in two ways: (a) resource abstraction for interfacing with both *physical and virtual* systems, and (b) support for *multiple access methods*. We will also focus on describing, demonstrating, and evaluating how support for multiple access methods, and allocation of virtualized and physical resources would benefit users of the cloud computing paradigm. Finally, we will elaborate on our implemen-

tation of a new cloud computing layer that allows consumers to deploy and manage virtual private clouds as a whole rather than as a collection of individual resources.

# 2. CLOUD COMPUTING PARADIGM WORK

## 2.1 Requirements

Prior to discussing our solution, we must establish a set of minimum requirements for a system capable of providing a cloud computing mechanism, resource abstraction, and multiple access methods. For our purpose we leverage the taxonomy developed by the RESERVOIR project [19]. RESERVOIR consists of a set of requirements [20] defining the characteristics of cloud computing infrastructures, management of virtualized resources, and service architectures for large-scale service deployments to the cloud. We will extend these requirements to handle resource abstraction and multiple access methods. We separate the requirements into five major categories: isolation, scalability, federation, separation, and management. Each category has distinct points that are elaborated upon in greater detail.

### 2.1.1 Isolation

Isolation requirements are established with the intention to mitigate risk associated with the cloud provider's infrastructure and the virtual private clouds deployed within it; risks associated with the use of resources by end-consumers, environmental factors related to physical infrastructure security and accidental network misconfiguration. Isolation pertains to the isolation of virtual private clouds from one another, and implies that the infrastructure must always remain under the control of the cloud provider, even when multiple virtual private clouds co-exist within the same physical infrastructure. The following items define the requirements for isolation:

**Black Box** Each resource allocated by a cloud provider must be considered a black box; the provider need not be concerned with the internal workings of the resource. Once a resource has been allocated to a consumer, that resource should be under the control of the consumer; control includes the internal workings of

the resource, along with actions such as starting, restarting and shutting down the resource. Physical and virtual resources allocated to a consumer should have no impact on the provider's infrastructure beyond the consumption of resources expressly allocated to them.

**Sand Box** Consumers should be unaware of the physical infrastructure that instantiates and connects their resource. A user should not be able to interact directly with resources outside of their own virtual private clouds, except when those resources are connected through a public network such as the Internet. Furthermore, a user should not be able to interact directly with a provider's physical hardware except for hardware allocated to the user as part of their virtual private cloud.

**Naming Space** A virtual private cloud must be able to use a naming space independent of other virtual private clouds or the underlying infrastructure. Multiple virtual networks across one or more virtual private clouds should be able to utilize the same addressing scheme, be it at the data-link layer or higher. Furthermore, a virtual private cloud should be able to utilize the same addressing scheme as the underlying infrastructure without conflicting with traffic on the cloud provider's network.

### 2.1.2 Scalability

Foremost, to achieve scalability, a cloud provider must meet the requirements of the consumer regardless of the quantity of resources requested or the topology of the consumer's requested network. Since scalability applies to effects on both the consumer and cloud provider, we break scalability requirements into two subcategories: 1) scalability as it relates to the scale and diversity of a consumer's virtual private cloud, and 2) scalability relating to a cloud provider's ability to facilitate the consumer.

Scalability, as it relates to the consumer, should be achievable both at instantiation time of the virtual private cloud and throughout the lifetime of the VPC, that is, when the consumer adjusts their virtual private cloud to meet on-going needs. Virtual private clouds should scale both in size and numbers:

**Scaling in size** A virtual private cloud, when instantiated, must meet the requirements of the consumer in their entirety, or not at all. In some cases, a single provider may not have sufficient resources available to satisfy a request. In such cases, cloud bursting may be used to leverage resources of other providers [11]. Cloud bursting allows a cloud provider to leverage unique resource types offered by another provider or to exceed their own resource availability to meet consumer needs.

**Scaling in number** In order to foster a diverse ecosystem of virtual private clouds, consumers, and providers, we believe cloud provider's should support an arbitrary large number of virtual private clouds within the same infrastructure, provided sufficient physical and virtual resources are available.

Scalability of the cloud provider is the ability of the provider to meet the current and future demands of its consumers. Cloud providers should scale through federation and through diversity of resource types:

**Federating** When resources at one provider are exhausted, virtual private clouds should be able to extend to other cloud providers for their demands to be meet. Cloud bursting could be accomplished by either linking resources together using applications running within the virtual private cloud, such as transparent VPNs, or by dedicated links between cloud providers at the infrastructure level. In deriving a cloud provider's decision whether to scale a virtual private cloud across providers should not be dependent on knowledge of the internal workings of the VPC.

**Diversity** Diversity of resource types offered by a cloud provider should scale linearly with the time and effort it takes to integrate new resource types into the infrastructure. Also, a cloud provider's configuration and management complexity should be decoupled from the size and diversity of the virtual private cloud.

### 2.1.3 Federation

The practice of scaling horizontally across cloud providers, leveraging resources from each provider, is known as cloud federation. In order to scale virtual private clouds across multiple cloud providers, the latter must be able to contribute resources to form a cohesive entity even when individual resources of the VPC operate on different infrastructures. Additionally, physical and virtual resources must be interchangeable between providers. The following items define the requirements for federation:

**Distributed Migration** Migration of virtual resources between cloud providers and across administrative domains should be possible without requiring changes to the virtual resource. Physical migration of physical resources between cloud providers and across administrative domains should be possible without requiring physical modification of a cloud provider's existing infrastructure. Lastly, physical and virtual resources should not be aware of a change in their physical location, nor require modifications to accommodate that change.

**Distributed Deployment** It should be possible to deploy a virtual private cloud consisting of physical and virtual resources across multiple cloud provider infrastructures and administrative domains. Federating should not require additional knowledge on the part of those resources, nor on the part of the consumer.

**Administrative Privacy** A virtual private cloud that spans multiple cloud providers or administrative domains must not reveal to one cloud provider the inter-

nal structure or configuration of an infrastructure belonging to another cloud provider. This includes the identities of physical machines, networking devices, and the network's topology. From this requirement we further stipulate that the internal nature of an infrastructure provider's network, that which is supporting a part of the virtual private cloud, should be transparent to outside entities.

**Administrative Security** The security and stability of a cloud provider should not have an effect on any other cloud provider, even when those providers are supporting part of the same federated virtual private cloud. If a security breach occurs within the confines of a virtual private cloud, cloud providers hosting any part of the virtual private cloud must remain immune to the breach. [1]

**Administrative Independence** Administration of a cloud provider's infrastructure should not have an effect on the administration of any other cloud provider, even when those providers serve parts of the same virtual private cloud. Providers should be able to add or remove physical assets from their network, and make network alterations, without coordinating with other providers. Additionally, providers should handle, transparently, alterations to the routing between components of a virtual private cloud when configuration changes occur to the physical network.

### 2.1.4 Management

Cloud providers must be able to manage physical and virtual resources uniformly, along with the virtual networks they form; failure to do so limits the versatility of

[1] The internal security of a virtual private cloud is outside the scope of the cloud provider's responsibility. It would, therefore, be possible for a virtual private cloud to become compromised, in its entirety, even when it expands across multiple infrastructures and administrative domains. When the physical infrastructure of a cloud provider is compromised, virtual private clouds hosted by the provider are typically assumed vulnerable, including parts of the virtual private cloud hosted within other cloud providers. However, as noted earlier, compromised virtual private clouds should not imply a higher risk to cloud providers whose infrastructures are not compromised.

the provider, along with the diversity of virtual private clouds that could be hosted. Management also extends to the ability of consumers to leverage the resources of a cloud provider through mechanisms that allow for the expressive, yet concise, description of the consumer's requirements, and the management of a resource's life-cycles. The following items define the requirements for management:

**Life-Cycle Management** A resource should be manageable by the consumer throughout its life-cycle, that is, from deployment to termination. Transitioning a resource from one state to another should be at the sole control of the consumer. Only when the maintenance of the cloud provider's infrastructure take precedents may control of a resource's life-cycle be preempted. At the conclusion of maintenance, the consumer's resource should be in a state before the preemption took place.

**Ad-Hoc Configuration** The arrangement of physical and virtual resources into arbitrary topologies should be possible if each resource in the topology supports the necessary interfaces.

**Resource Migration** Live and off-line migration of virtual resources should be supported within a cloud provider's infrastructure. Migration should also apply to the contents of storage devices associated with physical resources, thereby allowing the operating system of a physical resource to be stored off-line, migrated while live, or migrated to a live system from an off-line copy.

**Resource Access** A cloud provider must be able to meet a consumer's *access requirements*, a description on how the consumer wishes to gain access to their resource, or resources. A consumer's access requirements may include access methods such as network interfaces, or console ports.

## 2.2 Prior Work

In this section we summarize the issues that traditional cloud providers and researchers have encountered with the cloud computing paradigm, their analysis of each issue's importance and solutions. Specifically, we will focus on the following areas: cloud computing design principles, solutions for achieving cloud computing, isolation of virtual private clouds and the security of cloud provider's infrastructures, security of the cloud and, finally, how mixed environments consisting of physical and virtual resources are managed.

Overlay-based testbeds like PlanetLab [21] offer researchers access to geographically distributed resources, to run services they have developed, to be tested in a wide-area network environment. Outlined in [22] are four design principles the authors believe should be adhered to if testbeds are to be useful to service developers and service consumers. Those design principles are: 1) the ability to divide computing resources into slices, 2) distributed authority and control of resources within the overlay, 3) separation of management functionality into independent agents, and 4) long-lived application programming interfaces. One testbed that achieves the four design principles along with support for arbitrary network topologies is a testbed and tool-kit developed by the Emulab project out of the University of Utah [4]. Unlike PlanetLab described earlier, Emulab is typically deployed as a centralized infrastructure consisting of hundreds of physical servers and dozens of switches. Projects like Emulab address a need for providers to offer physical assets and not simply virtual machines for large scale research projects. Emulab was developed with the intent of leveraging the physical servers for constructing virtual networks that emulate real network environments for researchers, instructors and student alike.

To allocate resources, Emulab uses an on-demand model; when sufficient resources are available, requests are acceptable, but when resources are scarce, requests are immediately rejected. To offer a best-effort approach to resource allocation, a batch processing system is available for users to queue their requests until sufficient

resources are available [23]. Because Emulab, in its original form, only allocated physical resources, it was naturally vulnerable to resource exhaustion during periods of high demand. To overcome resource exhaustion Emulab began leveraging FreeBSD's jails [24] [25], which were used to segment the operating system on a physical machine into logical systems offering separate file systems, and execution separation between users. Emulab now offers support for virtualization using the Xen paravirtualization hypervisor [26]. Virtualized environments allow the resources of the physical system, such as processors and memory, to be segmented into discrete blocks allowing for several operating systems to run in-tandem on the same physical system. With FreeBSD's jails and Xen, Emulab can multiplex physical systems into several virtual nodes to meet the demands of users [27]. However, when a user requests a resource, he or she must specify whether the resource is to be physical or virtual. They must further specify that the virtual resource is to leverage Xen, using special syntax integrated into the "NS" ("Network Simulator") language used by Emulab for describing network topologies [23]. Virtual nodes supported by Xen are instantiated with fixed memory and disk space, as if they were physical nodes, without exploiting Xen's ability to be customized for each node.

Emulab's physical servers also support full virtualization, in contrast to paravirtualization. Users are encouraged to take advantage of the full virtualization using Eucalyptus [28]. Eucalyptus is a cloud computing tool-kit similar to OpenStack [7] and OpenNebula [6]. Cloud computing tool-kits must be installed by the user and configured to treat the physical servers allocated to the user as virtual machine hosts. One component of Emulab's management tool-kit is a custom hardware restart mechanism called "Whack-on-LAN" [29], an approach that mirrors the ability of hypervisors to create and terminate virtual machines.

When accessing resources of a traditional cloud provider, that access is facilitated through a network interface not firewalled by the provider. Emulab is similarly open to its users, though they go further by providing additional methods to users for

gaining access to resources. Every resource, called a node in Emulab terminology, is attached to a control network that allows for several methods of access to the node by users. Establishing a connection to an Emulab resource is possible by three different methods: 1) by a canonical Emulab name address accessible from the Internet, 2) by the same address from Emulab's user management server, or 3) console access using telnet wrapped by a custom application on Emulab's user management server [23] [30]. Emulab relies upon users to secure the access methods to their resources from infiltration by malicious adversaries and to insure their resources to not pose a danger to the Internet. In order to mitigate risk to the Internet, Emulab automatically applies port filtering to all connections on their control network [31], restricting the utility of the Emulab network for applications and protocols designed to use those restricted ports by default.

Corporations that desire to exploit the scalability of cloud computing are inhibited by the same security concerns that cloud providers are forced to deal with when hosting security sensitive research [32]. Those concerns include, but are not limited to, confidentiality, integrity and availability. DETER Lab [33] [34] [35] supports research in the same manner as Emulab, through the provisioning of physical resources, but extends the Emulab software to facilitate a more robust and secure environment for security related research. The security apparatus supported by DETER includes dynamically configurable firewalls, intrusion detection on control networks, a mechanism for cleansing nodes after each experiment and a set of procedures for classifying the security sensitivity of experiments prior to their deployment [36]. DETER Lab achieves network isolation using two approaches: 1) a dynamic assignment system that leverages VLAN-capable switches, as does Emulab, and 2) their own VLAN tagging mechanism for assigning VLAN tags based on MAC addresses Lahey:2008:EIS:1496662.1496666.

One approach taken to run live malware on DETER [37] achieves isolation of experiments, and security of the testbed, by focusing on four parts of an experi-

ment's life-cycle: 1) the one-time setup of the physical experiment environment, 2) pre-deployment steps, 3) experiment run-time, and 4) post-experiment clean-up. A proposal to improve the security and architecture of DETER Lab is given in [38]. DETER Lab has also begun a transition to a more robust installation capable of handling the diversity of security related research projects [39]. Newer developments in DETER have lead to enhanced experimental validity, diversity in experiments and improved scalability [40]. Outside of DETER, the authors in [41] discuss a technique for mitigating risks associated with the security of hosted virtual machines. In [42] a mechanism is described for cleansing potentially compromised virtual machines, similar to those mechanisms used by DETER Lab. However, the mechanism is approached from the perspective of integration into a cloud computing tool-kit such as OpenStack. An approach for countering security vulnerabilities associated with transferring work between providers is examined in [43]. Other work [44], looks at leveraging kernel-based virtual machines for scalability and for the recording and replay of security related events within the virtual machine. This work supports experiments such as *distributed denial of service* within the DETER infrastructure [45].

Federation is the provisioning of resources across multiple providers in a manner that allows those resources to operate as if they were provisioned within a single provider. Federation has been achieved by DETER using a tool called SEER, or Security Experimentation EnviRonment, that leverages the existing DETER Lab infrastructure [46]. Another approach to federation relies upon the concept of *Cloud Brokers* [47] to split a consumer's request across multiple cloud providers, achieving the lowest cost possible, while insuring network connectivity between resources. The *cloud broker* takes responsibility for interacting directly with each provider. OpenNebula supports federation by scaling horizontally, issuing requests for resources from other cloud providers when internal resources are scarce [48].

When federating across providers, one must insure the security of the infrastructure and proper authentication of the consumer; a point that has been vocal-

ized in [49]. Issues that are raised include authentication of consumers, issues with name spaces within each testbed or cloud provider, and finally, the decomposition of services across those providers. Efforts to federate Emulab-like testbeds [50] [51] concentrate on overcoming the limitation of small network security testbeds, the isolation required to conduct experiments involving malware, and facilitate access to resources. One approach for achieving an abstraction of the proprietary interfaces exposed by cloud providers required to support federation is the cloud-provider neutral abstraction layer developed in [52].

Even though DETER is capable of handling security sensitive experiments, their deployment requires prior approval by DETER Lab's administrators. For security-sensitive applications the authors in [53] propose a two-constraint approach by which providers and consumers negotiate constraints that insure a provider is comfortable with the risk, and that the consumer is satisfied that the provider can facilitate the needs of the experiment.

## 2.3 Our Contributions to the Cloud Computing Paradigm

One of our primary contributions to the cloud computing paradigm is the introduction of a robust platform for supporting the diverse access requirements of cloud computing consumers. To this end, we developed an access management system, known formally as the Access Pathway Manager, to compliment the existing functionality of cloud computing tool-kits. Our access management system supports the ability for consumers to specify the exact methods they require to access their cloud-enabled resources. We call these methods Access Methods, which describe how access should be grated to a cloud resource. One or more access methods associated with a resource are bundled together into what we call the consumers access requirements. Access requirements are associated with a single cloud-enabled resource, typically compute nodes. One particularly important property of an access requirements set is the purpose of the consumer's resource. A purpose must be chosen from a list of

categories that describe how a resource could be used by the consumer. The purpose affects how, or even if, access methods are deployed within a cloud provider's infrastructure. Depending on the purpose, cloud providers may apply different administrative policies to the requested method, effecting accessibility, or the cloud provider may even outright reject the requested access method.

Upon extracting access requirements from a resource request our access management system makes a decision on whether the request is valid and if it can be satisfied. If the consumer's access requirements can be meet, the access management system will make the appropriate configuration modifications on the cloud provider's infrastructure. Actual modification is handled by a policy engine. The policy engine takes into consideration the consumer's requirements, the purpose of the resource to which those requirements are associated, and the administrative policies of the cloud provider.

Our second contribution illustrates how existing capabilities of cloud computing frameworks can be leveraged to allocate and manage physical hardware resources. We accomplish this illustration by extending the framework of an existing cloud computing tool-kit to: support requests for specific hardware types, integrate drivers for managing the hardware, and support the allocation of hardware to consumers.

The primary benefit achieved by extending an existing framework is that we can demonstrate how physical hardware can be integrated into virtual private clouds using the same cloud computing paradigm primitives to deploy, update and delete resources. Though our approach is applicable to all cloud platforms, our work is directly beneficial to the OpenNebula community, whose cloud computing tool-kit we use in our work.

Our third contribution is a system to deploy multiple identical virtual private clouds without modification and without issuing requests for each component of the VPC. Our system operates as a web service at the layer above traditional cloud providers or cloud computing tool-kits, leveraging cloud APIs to manage the compo-

nents that constitute the virtual private cloud. To further this end we use a software library called Libcloud to manage the life-cycle of cloud resources. However, that library and similar libraries, do not support the management of virtual private networks. Virtual private networks are a key component required to create a private LAN within the cloud. We contribute, therefore, to Libcloud by implementing support for managing virtual private networks in the cloud.

## 2.4 Tool-Sets Used in Our Work

We use an open source cloud computing tool-kit known as OpenNebula [6] for our cloud computing platform. As part of its default installation, OpenNebula uses the best-effort resource allocation model of cloud computing, instantiating virtual machines when resources become available. Resource allocation occurs at the direction of OpenNebula's built-in scheduler, which reserves system resources on a physical host and then allocates those resources to a virtual machine instance. Furthermore, OpenNebula supports the provisioning of virtual private networks between virtual machine instances by instantiating virtual network interfaces and managing VLAN assignments on the physical host. Virtual private networks are made possible by employing either data-link layer filtering of MAC addresses, or VLAN tagging as speIEEE 802.1Q [54] and IEEE 802.1ad [55]. Of those, the latter two are important for their ability to isolate network traffic in a secure and reliable manner. Lastly, OpenNebula includes a web service component that adheres to the Open Cloud Computing Interface (OCCI) standard [56] for managing cloud resources.

Our work also requires a language-agnostic, human-readable, and structured format capable of conveying structure and relationships. This format must require little effort to incorporate into existing cloud computing tool-kits. We decided upon JavaScript Object Notation [57], or better known as its acronym JSON. JSON providers a lightweight standard for data-interchange using a form that is simple, human-readable, and capable of serializing and de-serializing data structures.

We also use four Python libraries in our work: DJANGO [58], Django-Piston [59], SQLAlchemy [60] and Libcloud [61]. We require these libraries for their ability to provide a RESTful [62] web service that consumers use for managing their virtual private clouds, interaction with cloud APIs and maintaining persistence of each user's request.

# 3. EVALUATION FRAMEWORK

For formulating our evaluation criteria and methodology, it is important to note that this research contains very few metrics that can be gathered and analyzed in a meaningful way. Rather, this research consists of functional attributes which will require a different analysis other than raw performance analysis or number comparisons.

## 3.1 Success Criteria

Because of the qualitative nature of this work, we will take an inventory of the functional requirements of this research to be considered successful, while not failing to also include a few metrics that could also be useful, or potentially meaningful. Metrics which we will account for must be viewed critically since only numerical values gathered from system-as-a-whole testing would demonstrate the performance characteristics and limitations of the research implementation. However, those values could potentially be skewed as a result of the particular functional limitations of any tool-kit, or hardware used as a part of the while system.

For our qualitative success criterion, we inventory the functional requirements of each component of our research deemed necessary for validating the approach taken by this research:

**Table 3.1**: OpenNebula Success Criteria

| Criteria | Valid If |
|---|---|
| Ability to deploy two virtual resources connected by a virtual network. | Resource is queued for deployment.<br><br>Resources are deployed to cluster.<br>Network traffic can flow between resource devices. |

**Table 3.2**: Libcloud Success Criteria

| Criteria | Valid If |
|---|---|
| Ability to poll for existing virtual resources, and virtual networks. | All existing virtual resources are displayed.<br>All existing virtual networks are displayed. |
| Ability to instantiate a service consisting of a single virtual resource device. | Resource is queued for deployment.<br><br>Resource is deployed to cluster. |
| Ability to instantiate two virtual resource devices connected by a shared virtual network. | Virtual network is registered by the cloud management tool-kit, OpenNebula.<br>Resources are queued for deployment. |
| Ability to remove resources deployed through the Libcloud tool-kit. | Virtual resources, and system resources associated with hose resources, are destroyed. |

**Table 3.3**: Service Management System Success Criteria

| Criteria | Valid If |
|---|---|
| Ability to instantiate a service consisting of a single virtual resource device. | Resource is queued for deployment.<br><br>Resource is deployed to cluster. |
| Ability to instantiate two virtual resource devices connected by a shared virtual network. | Virtual network is registered by the cloud management tool-kit.<br><br>Resources are queued for deployment.<br>Resources are deployed to cluster. |
| Ability to remove services deployed through SMS. | Virtual resources, and system resources associated with hose resources, are destroyed.<br>Resources are queued for deployment.<br>Resources are deployed to cluster. |

**Table 3.4**: Access Pathway Manager Success Criteria

| Criteria | Valid If |
|---|---|
| Ability to modify virtual resource deployment definitions. | Given a particular policy directive, the appropriate modifications are made to a virtual resource deployment definition. |
| Ability to apply different deployment policies based on service requirements. | Service requirements are correctly interpreted by policy engine.<br>Unique service requirements lead to the correct application of unique policy directives.<br>Virtual resources are deployed with appropriate modifications resulting from policy directives. |

First, a test must be conducted using the cloud management tool-kit, OpenNebula. Most capabilities of OpenNebula are assumed to be functional, and their behaviors adequately documented by their official developers for the purpose of this research. We therefore restrict our success criterion for OpenNebula to its ability to deploy virtual networks across a cluster of systems, which is a primary focus of this research, and will validate virtual networking as an acceptable solution for facilitating network security and isolation.

After tests relating to OpenNebula are completed, it will fall upon Libcloud to be validated. Libcloud's compute driver for OpenNebula will be tested to verify work on the driver matches the requirements of the OpenNebula OCCI API. Furthermore, the network driver developed for OpenNebula will be tested against OpenNebula to insure that user requirements are properly interpreted by the driver, network attributes properly extrapolated, and the virtual networks successfully deployed within a cloud computing environment.

Next, our Access Pathway Manager will be tested to determine if it meets our success criterion. Each success criterion will be tested in part to verify the Access Pathway Manager in both the low-level instantiation processes, and in high-level policy evaluation process. For success criterion (1), a policy is fixed, and the test focuses on APMs ability to deploy the policy. Success criterion (2) focuses on APMs ability to successfully interpret the requirements of a service, and chose the appropriate policy.

Lastly, the Service Management System, which is dependent on the success of Libcloud and OpenNebula, will be tested. These tests will verify that SMS can successfully extrapolate the service components from a service description, and deploy them to OpenNebula through OCCI.

## 3.2 Evaluation Methodology

To evaluate the success of our research contributions, a step-wise approach will be taken to assess the degree to which each component of our cloud implementation meets its success criterion. The step-by-step testing and evaluation process will focus in-detail on each component, beginning with the component on which most other components are dependent, and transitioning to the component with the most implementation dependencies.

As noted earlier, there are metrics that could be meaningful to other in support on their own evaluations of the particular implementation demonstrated here. Therefore, those values will be included as part of the evaluation and shown in perspective.

# 4. ACCESS PATHWAY MANAGEMENT FOR CLOUD COMPUTING

## 4.1 Introduction

Leveraging large-scale cloud installations is possible because consumers are able to gain access to resources therein in a reliable and secure manner. Such remote access is supported by traditional cloud providers and Emulab-like installations alike. For example, traditional cloud providers attach the virtual equivalent of a Ethernet cable between a cloud-enabled resource and the Internet. Those connections support network protocols layered over Ethernet with few if any traffic restrictions. Those traffic restrictions are typically in place to prevent network abuse, such as unsolicited e-mail traffic or malware. Emulab-like installations offer additional remote access in the form of a serial-like interface. This interface is accessible directly from the Internet, facilitated through a transparent proxy and from a system housed within the Emulab infrastructure. Both the network and serial-like interfaces are attached to a consumer's resource without input from the consumer. Consumers only have control over the services that listen for in-bound connection requests on those interfaces.

Naturally, access to cloud resources is less versatile than access to locally owned and operated resources. Resources under the physical control of the consumer can be configured to use any method of access desired as long as the physical hardware and operating system support it. As a result of the minimal access methods offered by cloud providers and Emulab-like installations, those platforms impose a limitation on the utility of resources by consumers [1]. Restricted utility results from the limitation on services that use unsupported access methods and from the unnecessary exposing of resources to malicious activity on the Internet, and the burden placed on consumers to mitigate that risk.

[1]The term *consumer* denotes the owner and operator of one or more virtual private clouds, while the term *client* will denote the user of services provided by a VPC.

In many cases these traditional access methods are not sufficient to meet the needs of the consumer. Instead, consumers may require a variety of access methods to gain access to their cloud-enabled resources. For example, software running on a consumer's resources may pose a risk to the cloud provider's infrastructure or Internet at large, but only over certain ports or a network interface. Similarly, a network interface may be unnecessary to meet the needs of the consumer or their purpose. Finally, resources may possess multiple services, with each service requiring its own dedicated access method. A consumer's needs and the access methods that satisfy them are considered to be the *access requirements* of the consumer. If the consumer's requirements are not meet, then their needs and expectations are not fulfilled.

In section 4.2 we define access methods and describe how they are constructed, in section 4.3 we describe how consumers define sets of access methods per resource, in section 4.4 we discuss how access points are presented to consumers, in section 4.5 we describe access pathways and what they represent and in section 6.4 we discuss our software architecture and implementation.

### 4.1.1 Scenario

To illustrate the importance of custom access requirements, we present an example currently used in a real-world scenario involving a virtual private cloud with three resources connected in a series by two virtual private networks. In this scenario, students at academic institutes are required to configure a firewall on one of these resources for the purpose of learning how to filter traffic and protect network services. Because traffic will be affected on the network interfaces attached to the resource, the training scenario precludes students from connecting to that resource over any network-like interface. Instead, the student must access the resource through another medium. In this scenario, students connect to the VPC through a console port. An illustration of this VPC setup used by WAES2 [18] can be seen in Figure 4.1. Stu-

dents connect to the console port of Machine B and then configure the firewall to filter traffic between Machines A and C. This scenario illustrates how virtual private clouds may require the cloud infrastructure to support diverse access requirements to achieve a VPC's full potential.

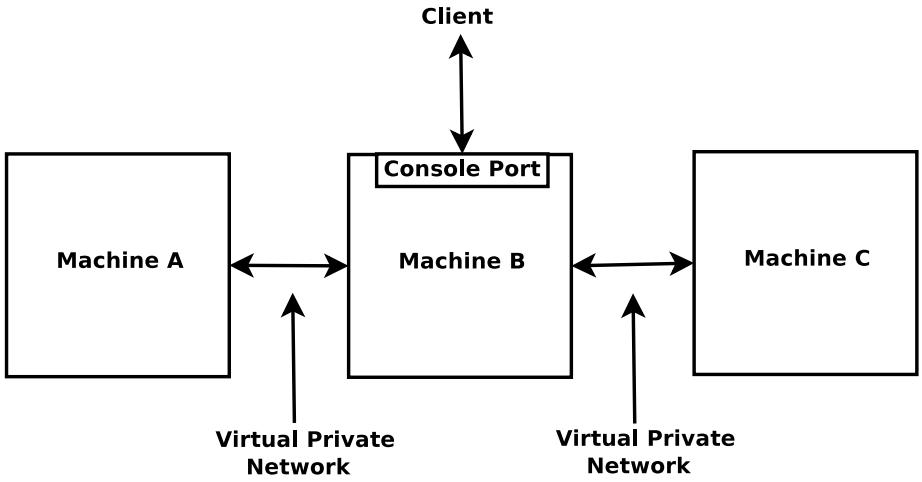


**Fig. 4.1.**: A WAES2 firewall VPC with a single user-accessible console port on the firewall resource.

### 4.1.2 Considerations

When allowing consumers to request their own access methods, cloud providers must taken into consideration the consumer's intention. As part of the resource request process, each requested resource must be marked with the consumer's purpose for that resource. At this time, we assume that resources only fall into a single category. We can derivepse adequate categories based on on-going research leveraging cloud computing for malware analysis [63], the establishment of expectations for storage confidentiality as elaborated upon in [64] and testbeds for network protocol development [4]. We don't impose a restriction on the concrete types that may be used for the purpose. A consumer might describe the purpose of their virtual private cloud as a *exploitable platform*, *malware platform*, *secure platform*, *testbed platform*,

or *confidential platform*. If a consumer's access methods can not be meet because of the purpose, then the entire request is rejected.

To determine the appropriate method for accessing a resource, three approaches can be taken. First, access methods can be derived from the properties of a virtual private cloud and applied automatically by the cloud provider. Second, consumers can be restricted to a pre-defined set of access methods offered by the cloud provider from which the consumer may choose on a per-resource basis. Third, a more relax approach may be supported in which the consumer may poll for valid access method types, and then build their own access method definition around that type on a per-resource basis.

The first way is hazardous, as the mapping of VPC properties to a set of access methods may not coincide with the expectations of the consumer. The second way establishes well-defined behavior but severely limits the utility of cloud computing as the consumer must expend effort to locate a cloud provider that offers the resources and access methods they require. Therefore, we support the third approach.

Our motivation for supporting a more relaxed approach is to remedy the problems of the other two alternatives. By allowing consumers to define their own access methods, we relax the requirement that services deployed to the cloud must adhere to the access methods imposed by the cloud provider. Furthermore, supporting consumer defined access methods allow consumers to request only those access methods they require, subsequently limiting the extent to which their cloud resources and services are exposed to the Internet.

In the following we argue that access methods should be treated as first-class objects, just as traditional cloud-enabled compute nodes are treated; consumers should be able to retrieve supported access method types and build access method definitions that can then be assigned to the consumer's resources. In this chapter we will also define what we call *consumer-defined access methods*, explain their relationship to a consumer's access requirements, explain how access points are derived from ac-

cess methods and introduce the concept of access pathways. Furthermore we will demonstrate how access pathways exceed the current standards in cloud computing, providing a unified approach toward deploying firewalls, port and protocol filtering, yet improving the utility of cloud computing to consumers. Lastly, we will outline a software architecture capable of facilitating the management of access pathways and examine how our solution can be incorporated into the cloud computing tool-kit OpenNebula.

## 4.2 Access Method

An access method describes how a resource within the cloud should be made accessible to the Internet. For each resource requested by the consumer, the consumer may associate one or more access methods with the resource. Each access method is defined by an access type and one or more access ports. A diagram of the access method syntax can be seen in Figure 4.2.

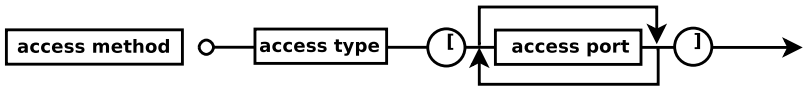


**Fig. 4.2.**: Syntax diagram for the access method specification.

The *access type* defines the connection medium and the control capabilities that the access method should provide. For example, a serial connection may be established over a serial interface to communicate with the virtual or physical equivalent of a console port on a resource. Alternatively, the consumer may need access over a Ethernet network to support services that require TCP/IP for traffic management. Different access types provide different levels of control over the traffic that can propagate across the access medium. A network access type implies that the access method will behave like any network interface, though the actual medium could be a virtual or physical interface, of which the physical medium could be wireless, Eth-

ernet, or Fiber, among others. A serial access type implies that the access method will behave like a serial cable attached to the console port of a physical system, though again, the actual medium could be a virtual console, DB-9, or similar type of interface. A diagram of the access type syntax can be seen in Figure 4.3.

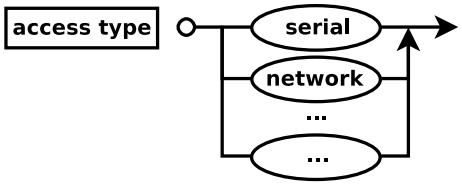


**Fig. 4.3.**: Syntax diagram for the access type specification.

The *access port* of the access method specifies the *transport-layer access port* on the resource and the set of *protocols* allowed to pass through that port. A diagram of the access port syntax can be seen in Figure 4.4.

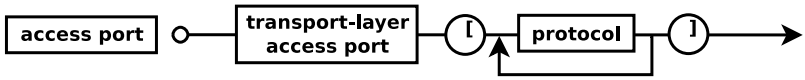


**Fig. 4.4.**: Syntax diagram for the access port specification.

In the case of network access, the transport-layer access port is defined by a port number. When the access port is implied by the access type, or is not applicable to the access type, this parameter can be omitted. When access is desired on all ports for a network-type access method, no ports should be included as part of the access method definition. A diagram of the transport-layer access port syntax can be seen in Figure 4.5.

*Protocols* specify the transport-layer and application-layer traffic that a consumer wants to pass through the transport-layer access port. This stipulates that any traffic not matching a protocol in the *protocols* list will be dropped. Dropping traffic is handled by the cloud provider's own infrastructure in the form of firewalls and deep-

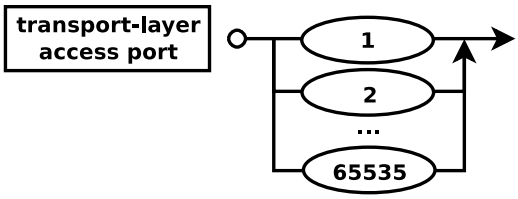


**Fig. 4.5.**: Syntax diagram for the transport-layer access port specification.

packet-inspection tools, all of which should remain transparent to the consumer. A diagram of the protocol syntax can be seen in Figure 4.6.

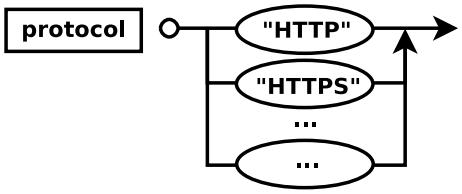


**Fig. 4.6.**: Syntax diagram for the protocol specification.

Only a single instance of the *type* and *ports* properties may be defined for each access method. In addition to a transport-layer access port, each port has a protocols set. Handling duplicate transport-layer access ports is implementation dependent, with one approach forbidding duplicate transport-layer access ports, while another may merge port protocol lists together.

We illustrate our access method specification in listing 4.1 through an example of a *network*-type access method [2]. This example specifies that only ports 22 and 80 should allow traffic to pass through. Furthermore, only connections established using the application-layer HTTP and HTTPS protocols are allowed on port 80, and connections established using the application-layer SSH protocol are allowed on port 22.

[2]In this example we use JavaScript Object Notation [57], better known as its acronym JSON, a the representation for the access method specification. JSON providers a lightweight standard for data-interchange using a form that is simple, language-agnostic, human-readable, and capable of serializing and de-serializing data structures. JSON is used to specify access methods in our implementation of the Access Pathway Manager (See Section 4.6.1).

```
{"type": "network", "ports": {"80": ["HTTP", "HTTPS"], "22": ["SSH
    "]}}
```

**Listing 4.1: JSON-encoded Access Method**

Though beneficial to both cloud providers and cloud consumers, consumer-defined access methods raise their own issues primarily due to the fact that consumers have only a limited knowledge of the environment in which their VPC is deployed. First, consumers may not know the security implications of using a particular access method. For example, consumers may assume that application-layer protocols that are sufficient for their own private networks, such as Telnet, should suffice for VPCs, even though the clear-text transmission of data could be exploited by nefarious third-parties. Second, consumers do not have an incentive to achieve the minimum-privilege principle, or may believe the use of complimentary mechanisms for establishing firewalled network interfaces may be to cumbersome. Fortunately, offering the ability to specify more restrictive access requirements as part of the resource request process and by bundling access requirements into the resource request process will only facilitate stronger control over the security of that resource. Third, consumers have no knowledge of the cloud provider's infrastructure, which may preclude certain access methods. This last issue can be partially mitigated by rejecting requests that contain an access method, or methods, that are not supported by the cloud provider, and by providing a discovery mechanism for consumers to learn what access methods are support by the cloud provider.

## 4.3 Access Method Sets

So far we have discussed how to construct the definition of a single access method. In some cases consumers may want to associate more than one access method with a resource. A bundle of one or more access methods is called an *access method set*. An

access method set may contain access methods of identical definitions associated with a single resource. Each resource that has access methods will have its own access method set. A formal definition of an access method set is shown in Figure 4.7.

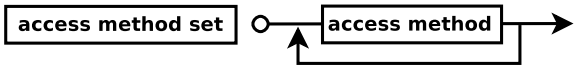


**Fig. 4.7.**: Syntax diagram for the access method set specification.

As an example, in listing 4.2 we show the specification of the set for a single resource. This set consists of two access methods that shall be referred to by their types: *network* and *serial*. To maintain consistency, we use the same *network* access method defined earlier.

```
{"access_methods": ["{'type': 'network', 'ports': {80: ('HTTP', '
    HTTPS'), 22: 'SSH'}}", "{'type': 'serial'}"]}
```

**Listing 4.2: Access Method Set**

Establishment of access methods in an access method set follows transactional semantics: either all access methods in the set are established or the entire request, including the request for resource establishment, is rejected. In the latter case the requesting consumer is returned an error response indicating what requested access methods could not be established and what lead to the failure. By providing a detailed report of the failure consumers can re-issue their request with a modified set of access methods, and so *de-facto* enter into a resource-negotiation round with the cloud provider.

Part of the negotiation process may result as a consequence of cloud providers establishing access requirements that consumers must adhere to when formulating their access method sets. If a consumer fails to meet the requirements of the cloud provider, they may be required to make alterations to their access methods. For example, a consumer may request access to certain ports that are blocked by the

provider, as illustrated in example 4.3. Also, consumers may be required to include additional access methods, as illustrated in example 4.4. Provider's may also restrict transport-layer and application-layer protocols, as illustrated in example 4.5.

```
{"access_methods": ["{'type': 'network', 'ports': {80: ('HTTP', '
    HTTPS'), 22: 'SSH'}, 'error': 'Port 80 is blocked.'}", "{'type':
    'serial'}"]}
```

**Listing 4.3: Access Method with Port Blocking Error**

```
{"access_methods": ["{'type': 'network', 'ports': {80: ('HTTP', '
    HTTPS')}, 'error': 'An SSH accessible port is required.'}", "{'
    type': 'serial'}"]}
```

**Listing 4.4: Access Method with Inclusion Error**

```
{"access_methods": ["{'type': 'network', 'ports': {80: ('HTTP', '
    HTTPS'), 22: 'SSH'}, 'error': 'HTTP protocol is not allowed.'}",
    "{'type': 'serial'}"]}
```

**Listing 4.5: Access Method with Forbidden Protocol Error**

Resource negotiation can be further supported by an appropriate resource discovery interface, where consumers can query the provider about available resources and access methods. Also, this request-response mechanism can allow the consumer to request access methods that are satisfactory for the purpose of a consumer's virtual private cloud. This mechanism, however, would not be a requirement prior to issuing a request for resources.

## 4.4 Access Point

Access method sets convey the consumer's desired methods for accessing their cloud-based resources, but they do not offer information required to actually establish

a connection. Information relevant to establishing a connection to an end-point consists of an address of the end-point, the port on which a service is listening and the protocol used for the communication channel. These end-points are known as service access points. Each access method requested by the consumer may be converted into one or more access points. Access points are not serviced by the cloud provider, but only indicates the location, and subsequent protocol that is allowed, to establish a connection to the resource to which it is associated. Handling connections is the sole responsibility of the services running within resources controlled by the consumer. Each access point is described in a manner similar to an access method, except that the *type* attribute is replaced with an *address* attribute. The address can be any routable IP address or Fully Qualified Domain Name. Because an access point may contain a list of ports and the protocols allowed on each port, it is possible to map access methods to access points on a one-to-one basis. A diagram of the access point specification can be seen in Figure 4.8

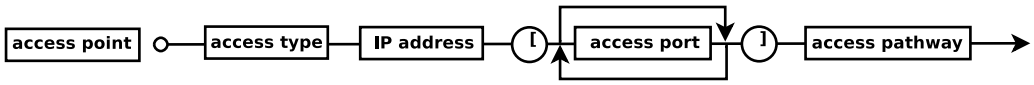


**Fig. 4.8.**: Syntax diagram for the access point specification.

As an example, Figure 4.6 shows the description of a single access point. This access point contains the address and the ports required for applications to establish connections using the HTTP, HTTPS and SSH application-layer protocols.

```
{"type": "network", "ports": {"80": ["HTTP", "HTTPS"], "22": ["SSH
    "]}, "address": "192.168.0.1"}
```

**Listing 4.6: Access Point**

Contained within the definition of the access point is an IP address, a list of ports on which connections may be established, and a list of protocols that are allowed over the connections. In this example, the consumer might leverage the

access point by pointing a web browser to the specified *address* 192.168.0.1 and *port 80*. Alternatively, the consumer may leverage port 22 to establish a secure remote terminal.

Though accepting establishing connections, or accepting in-bound connections is the responsibility of the consumer, both the consumer and cloud provider have the joint responsibility to facilitate the connection from point-to-point. By point-to-point we mean from a service running on a resource allocated to a consumer by the cloud provider, to the end-user of the connection. The joint responsibility is broken down as follows: (1) the consumer must have a service, such as an application, running on the resource that will respond to connection attempts at the given port, and using the protocol, specified in the access point definition, (2) the cloud provider must establish a path between the publicly available access point and the consumer's resource. That path must allow traffic destined for the access point to pass through the cloud provider's infrastructure and arrive on the port specified in the consumer's original request for an access method.

We call this path from the publicly accessible access point made available by the cloud provider to the access point on the cloud-enabled resource the *access pathway*. In the following section we describe how to specify the access pathway.

## 4.5 Access Pathways

An access pathway connects consumers to their cloud-enabled resources, and clients to the services that run on those resources, through a serial of access points. There are three forms of access pathways: (1) Access pathways that consist of only those access points within the cloud provider's infrastructure through which traffic travels transparently from the consumer to the resource. These access points are established on systems within the cloud provider's physical infrastructure and traffic from the first access point is automatically forwarded to the last access point along the pathway. (2) In other cases, access pathways that consist of access points that the

consumer must establish connections to in succession to gain access to their resources. In these cases consumers are required to establish a connection to the first access point in the pathway with the intention of directly, or indirectly, establishing a new connection to the access point next in the chain towards the consumer's resource. (3) Finally, access pathways that are combinations of the two cases above. In this last case, consumers are required to establish connections to two or more access points, but one of more of those access points may itself be a set of access points that tunnel traffic between each other, transparent to the consumer.

To represent a pathway, we have add an *access_pathway* property to each access point. If the consumer must connect to multiple access points in succession, the next access point in the chain will be contained within the *access_pathway* property of that access point. Furthermore, the *access_pathway* property may contain multiple access points that are accessible from a single access point. For example, a user may first be required to log into a resource using the top most access point. From there they can gain access to any of the access points listed within the next lowest level. A diagram of the access point specification can be seen in Figure 4.9


**Fig. 4.9.**: Syntax diagram for the access pathway specification.

An illustration of a pathway can be seen in listing 4.7. In our example a consumer receives an access pathway consisting of one access point. That access point is sufficient to gain access to the consumer's resource.

```
{'access_point': {'address': '192.168.0.1', 'ports': {'80': ('http',
    'https'), '22': ('ssh')}, 'access_pathway': []}}
```

**Listing 4.7: Access Pathway**

The formulation of an access pathway as a series of access points that a consumer has to establish connections to in succession is necessary to overcome security related issues for cloud providers. A cloud provider may wish to offer access to the consumer's resource using the access methods specified but direct access to the Internet may not be warranted as a result of the consumer's purpose for their virtual private cloud. A cloud provider could offer an intermediate node as part of the pathway if that additional node would allow for the consumer's access requirements to be satisfied.

In some cases an additional property will be required by access points when an intermediate node is used. This additional property is known as the access point's *credentials*. Credentials are only required by those access protocols that support a username, password, encryption key, or combination of those, and pass through an intermediate node. For example, credentials required to establish an SSH connection, or credentials required by the Basic Authentication mechanism of a web server that proxies an HTTP connection between the other access point and the inner access point. This scenario is illustrated in 4.8.

```
{'access_point': {'address': '192.168.0.1', 'ports': {'80': ('http',
    'https'), '22': ('ssh')}, 'credentials':{'username': 'bob', '
   password': 'bob', 'key': None}, 'access_pathway': [{'address':
   '10.1.0.1', 'ports': {'80': ('http', 'https'), '22': ('ssh')}}]}}
\end{center}
```

**Listing 4.8: Access Pathway**

In all forms of an access pathway, the cloud provider is aware of its existence for the purpose of maintaining the pathway. Only when there are access points along a pathway that the consumer must establish connections to in sequential order must the consumer know about the pathway. Those cases in which a consumer has only one access point and no pathway then the pathway is represented in the form shown earlier in listing 4.7. In all other cases the consumer is presented with an access pathway as shown in listing 4.8.

## 4.6 Implementation

To demonstrate the utility of the access pathway model we developed a complimentary tool to the OpenNebula cloud computing tool-kit. This new tool facilitates the creation and management of access pathways between the Internet and the resources belonging to the consumers of a cloud provider. We refer to this tool as the *Access Pathway Manager* (APM). In its current iteration, our Access Pathway Manager is built against the OpenNebula tool-kit, though that decision was for convenience and not to impose a limitation on the cloud computing tool-kit that the APM can be integrated into.

### 4.6.1 Access Pathway Manager

To support the management of access pathways within the cloud, the Access Pathway Manager is coupled to the life-cycle of cloud resources. Creation of new pathways occur as part of the resource request process. During that process the access method set provided by the consumer is extracted from the resource request and submitted to the Access Pathway Manager. In addition to the consumer's requirements the Access Pathway Manager also takes into consideration the administrative policies of the cloud provider; those policies potentially affecting the deployment of access pathways.

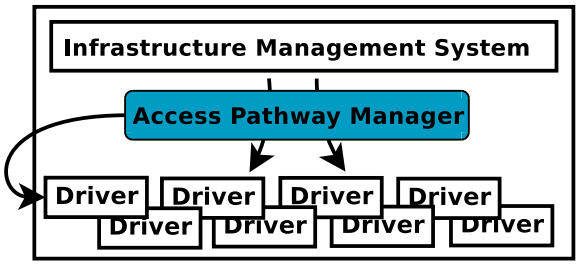


**Fig. 4.10.**: Access Pathway Manager integration into a generic infrastructure management tool-kit.

Different *access types* are handled by the Access Pathway Manager through a set of dedicated, type-specific, handlers. Dispatching each access method from an access method set to the appropriate handler is accomplished by the APM's *Policy Engine*. It is the policy engine that, prior to dispatching access methods to handlers, also validates the consumer's request, insuring the request adheres to the administrative policies of the cloud provider. Furthermore, to maintain state of all access pathways across the physical infrastructure of the cloud provider, a storage back-end is used and managed by the APM. A generalized diagram showing how our Access Pathway Manager integrated into the cloud architecture can be seen in Figure reffig:apm-integration-general. The Access Pathway Manager is called during the following three transition states of a typical cloud resource life-cycle:

**Deploy** State transition that occurs when the cloud provider initiates the process of provisioning a resource, or resources, to a consumer as part of a previous request by that consumer.

**Resubmit** State transition that occurs when the owner of a resource that already exists submits a new description with the expectation that the existing resource will be destroyed and provision again with properties from the new description. During the *resubmit* transition, the existing resource is destroyed and the request enters into a *pending* state prior to entering the *deploy* state transition. During the *resubmit* transition, the existing pathway is destroyed in a manner identical to the behavior of the APM during a *stop* state transition.

**Stop** State transition that occurs when the cloud provider begins the clean-up process for a resource following the release of that resource by the consumer. In this state all access pathways between the Internet and the resource are destroyed by destroying access points that constitute the pathway.

Our integration of the Access Pathway Manager into OpenNebula takes advantage of the latter's hierarchical and compartmentalized architecture. That architecture

is designed to map high-level life-cycle management commands to the underlying hypervisor used to provision a new resource. Provisioning is accomplished through a set of *virtual machine hooks*, which are scripts that align with the transition between life-cycle states. Though OpenNebula refers to their hooks as virtual machine hooks, they are merely scripts written for particular hypervisors. Our Access Pathway Manager is integrated into OpenNebula's dispatch mechanism that maps the generic life-cycle transitions to the scriptable commands for the underlying hypervisor. This allows for a minimally invasive integration, and allows the Access Pathway Manager to capture all calls to these hooks. The APM is passed the transition state, the OpenNebula hypervisor driver being called and a definition of the resource. Our Access Pathway Manager in relation to OpenNebula's architecture can be seen in Figure 4.11.

When a resource's definition is passed to the appropriate OpenNebula driver, the consumer's access method set and VPC purpose are contained within the resource's definition. For OpenNebula's KVM driver, for example, the resource definition is encoded in XML with the access method set and VPC purpose embedded as elements within the definition. The Access Pathway Manager, using the type of hypervisor driver used by OpenNebula to determine how the access requirements and purpose are embedded, will extract both for the purpose of creating a new access pathway. Figure 4.9 gives an example of an embedded access method set and the VPC purpose:

```
<>
<access_point>
{'access_point': {'type': 'network', 'ports': {'80': ('http', 'https
    '), '22': ('ssh')}}}</access_point>
<purpose>Web Hosting</purpose>
</>
```

**Listing 4.9: Access Method Request**

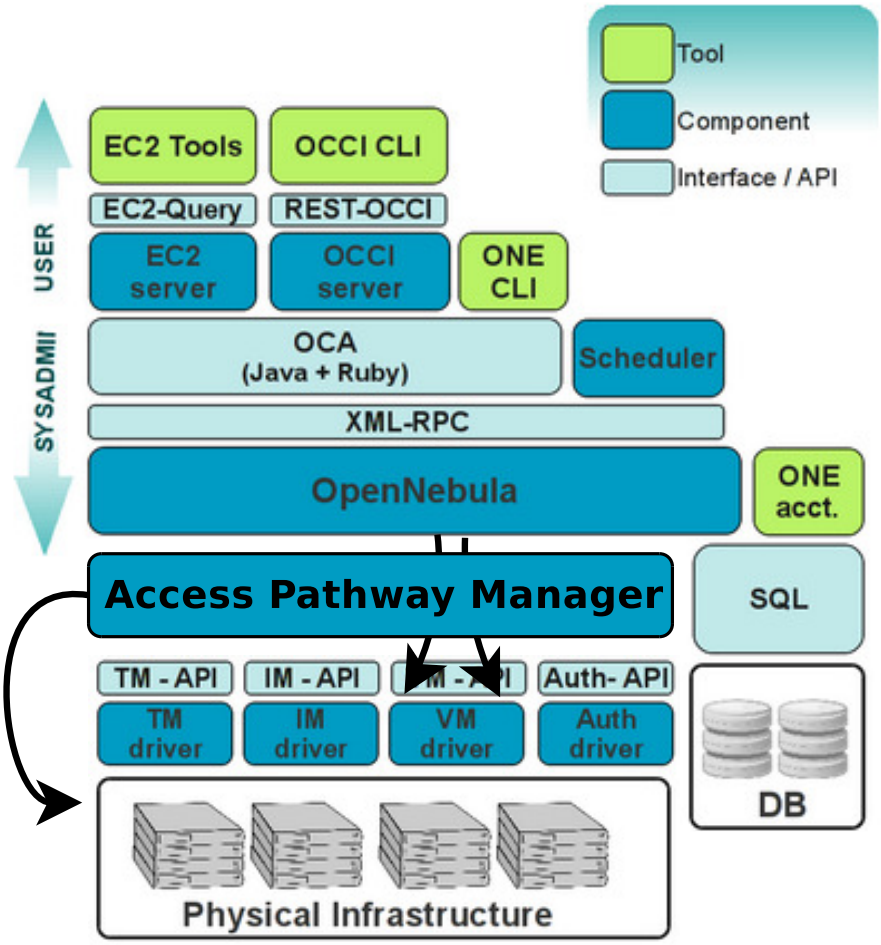


**Fig. 4.11.**: Access Pathway Manager integration into the OpenNebula software stack. Figure adapted from [65].

We avoid modifying the core of OpenNebula directly, avoiding both OpenNebula's resource request interpreter, and the specification used to define a resource for provisioning by OpenNebula. We avoid these modifications by embedding the consumer's access requirements in a manner that causes the requirements to be passed through OpenNebula without modification so that those requirements can be extracted by the Access Pathway Manager. In this way, the consumer's access requirements are handed to the underlying drivers without further interpretation by

OpenNebula. Those requirements are intercepted by the APM prior to the resource definition being handed over to the underlying hypervisor driver for provisioning.

To embed the consumer's access requirements we exploit the ability of OpenNebula's XML-RPC interface to pass raw XML to OpenNebula along with the consumer's resource request, formatted in OpenNebula's proprietary format. Passing raw XML causes the raw XML to become embedded within the resource definition when OpenNebula converts the consumer's request from the proprietary format to a hypervisor-specific format. Raw XML is enabled by the *RAW* attribute allowed in a resource template, a template that defines the properties of a resource, passed to OpenNebula's XML-RPC interface.

Though OpenNebula does not need to be modified, OpenNebula's OCCI interface must be modified to handle two additional properties within an XML-encoded definition of a resource. Those properties are the access method set and the VPC purpose. Our modifications to the OCCI API causes the OCCI interface to extract the two new properties, embed them within XML and assign them to the *RAW* attribute used by the OpenNebula XML-RPC interface. At that point, the access methods and purpose will be transparently embedded into the hypervisor-specific XML and captured by the Access Pathway Manager.

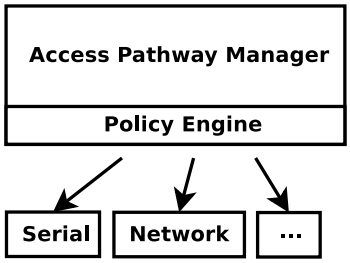


**Fig. 4.12.**: Access Pathway Manager architecture.

Each access method type is associated with a handler that is responsible for incorporating the appropriate access point into the resource and constructing the access pathway between the consumer and that resource. Handlers are part of the

Access Pathway Manager's Policy Engine. These handlers can be seen in Figure 4.12. Furthermore, a sequence diagram demonstrating the flow control through the Access Pathway Manager can be seen in Figure 4.13.

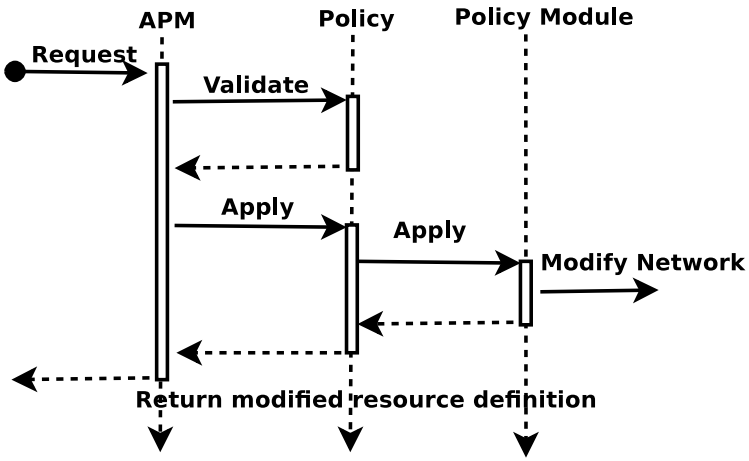


**Fig. 4.13.**: Access Pathway Manager flow control sequence.

### 4.6.2 Policy Engine

An Access Pathway Manager's *Policy Engine* drives the instantiation of pathways based on the administrative policies of the provider, the requirements of the consumer, the type of each resource requested and the consumer's intended purpose. Each type of access method requested by the consumer is associated with a policy handler. Each policy handler will generate a pathway satisfying the requirements just mentioned. Because pathways may overlap in their use of systems within the provider's infrastructure, policy handlers share a common library for managing resource types and executing common access modifiers, such as iptable rules. Three possible access methods that would require their own policy handlers, such as: (1) network, (2) serial, and (3) USB.

It was our decision to associate policy handlers with each access method type and to take sole responsibility for driving the establishment of an access pathway from the

resource to an Internet-accessible interface. Incorporating appropriate access points into the resource would seem like the only responsibility of the policy handler. However, it was determined that each policy handler, possessing the knowledge on how a particular access method is instantiated, would take further responsibility in driving the construction of a pathway that best facilitates that access method. Resource handlers for interacting with infrastructure components, such as switches, routers or intermediate compute nodes, would be facilitated through a common library available to each policy handler.

A final requirement construct an access pathway is an understanding of the cloud provider's network infrastructure. To facilitate this understanding, a model capable of capturing the particulars of the cloud provider's infrastructure would need to be incorporated into the Policy Engine. All physical systems within the infrastructure must be captured in model, including compute servers, switches, routers, switched power supplies and gateways. Our model must also capture the relationships between physical systems, including but not limited to Ethernet, USB, serial and power connections. Lastly, it is important to capture the resources that are allocated to consumers since those resources represent one of the end-points for an access pathway. Though physical systems can be captured at installation time, virtualized resources must be captured in our model when they are instantiated at the request of a consumer. This is accomplished by updating the model when the Access Pathway Manager is called during the deployment of a resource.

### 4.6.3 Access Information Retrieval

Access information should be retrieved though the same mechanism consumers use to retrieve information about their resources. When retrieving information about a resource through OpenNebula's OCCI interface, an XML-encoded definition is return containing information such as: status, memory, disks, network interfaces, ect. Access information should be embedded within the resource definition as additional

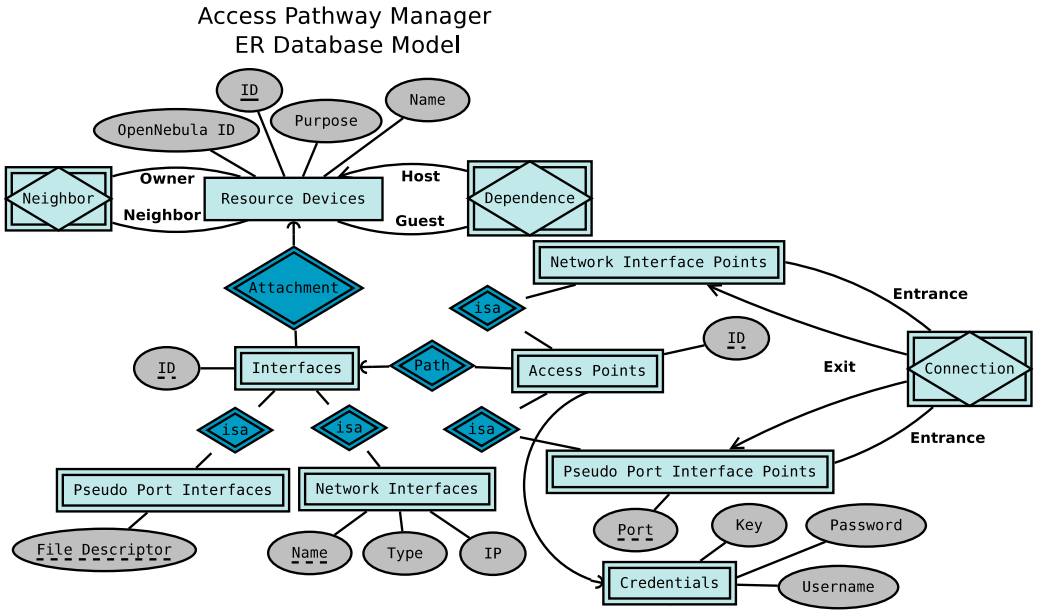


**Fig. 4.14.**: Data model for building and managing access pathways.

top-level properties underneath the root element. Access information is nothing more than the access point, and embedded access pathway, information. An example of a resource definition returned as part of a information request containing embedded access information is shown in the following example. Also shown is the purpose of the virtual private cloud embedded as another top-level property.

```
<>
<access_point>
{'access_point': {'type': 'network', 'address': '192.168.0.1', '
    ports': {'80': ('http', 'https'), '22': ('ssh')}, 'access_pathway
    ': []}}</access_point>
<purpose>Web Hosting</purpose>
</>
```

**Listing 4.10: Access Information**

## 4.7 Results

To validate our work, we decided upon a single test that should be sufficient to demonstrate the utility of an Access Pathway component to the cloud computing paradigm, and to the consumer's ability to leverage cloud computing for non-traditional uses. Our two tests are summarized below.

For our test we demonstrate that our proposed Access Pathway Management system is capable of modifying resources during the deployment process that are accessible over serial connections similar to how the Virtual Network Engineering Lab is currently used for teaching. As part of VNEL, virtual private clouds are provided to students to learn how to configure iptables-based firewalls. Providing a standard network interface is not feasible. Therefore, access is facilitated through a serial connection. We wish to verify that a request for a virtual private cloud, consisting of three resources, each resource requested along with a serial access method, can be deployed to the cloud and instantiated with the appropriate access points. This test is accomplished using OpenNebula's OCCI interface and a set of network and compute node templates. The network templates are used to establish network connections between the three resources of the virtual private cloud as seen in Figure 4.1.

# 5. PHYSICAL RESOURCE MANAGEMENT WITHIN THE CLOUD COMPUTING PARADIGM

## 5.1 Introduction

Traditional cloud providers offer consumers the ability to leverage virtualization to achieve large-scale computing, while smaller, non-traditional, providers offer consumers the use of physical resources. No single cloud provider offers the allocation of both virtual and physical resources to a consumer. Subsequently no provider offers a unified ontology for that provisioning. Though no unified ontology exists, Emulab was the first to develop an infrastructure capable of scaling to the demands of consumers by leveraging virtualization in conjunction with their existing physical resource provisioning system [26]. However, their ontology is built on the premiss that consumers must leverage the physical hardware provisioning system to deploy virtual assets to meet those needs. This further requires the consumer to specify the physical hardware they wish to leverage for virtualization and the virtual nodes to be instantiated on each physical system. In addition to the requirement that consumers map virtual nodes onto physical systems, Emulab also restricts virtual nodes to a pre-specified configuration that dictates a particular amount of dedicated memory and processor time. Therefore, scaling is only achieved through the direct intervention of the consumer and not through a deployment mechanism that automatically maps resource requests to resource availability.

To leverage the traditional cloud computing paradigm, consumers are required to deploy their own hypervisors, cloud computing tool-kits, or other supporting tools. It is this cloud computing infrastructure, an infrastructure installed on top of physical systems, that consumers rely upon for handing the deployment of virtual machines in a manner that allows for disassociation between virtual instances and physical hardware, and allows for consumers to dictate the properties of each virtual machine.

Possessing the capability to deploy cloud tool-kits on top of a virtual private cloud is a useful technique for leveraging the scalability of cloud computing to test new features incorporated into a tool-kit without having to deploy the tool-kit to physical hardware; and thereby incur the overhead of managing physical assets. However, to achieve scalability initially should not require additional effort on the part of the consumer. Rather, the ability to scale quickly with virtualized machines should be an intrinsic property of the infrastructure.

Expecting the consumer to manage their own cloud implementation, even with the help of an existing tool-kit, is to much of a burden. The consumer should not need to manage their own cloud layer on top of a physical layer just to manage virtual instances. Rather, that should be an existing feature of the underlying provider that already offers physical resources.

Supporting a mechanism that is capable of provisioning both virtual and physical resources allows greater utility for the consumer while avoiding the overhead of a dedicated mechanism for physical resources. Furthermore, we avoid the need to deploy additional software on top of physical hardware and support the ability to scale when the quantity of physical resources become scarce.

## 5.2 Resource Types

A major difference between cloud providers and Emulab-like providers is the inherent property that physical resources are immutable, consisting of fixed hardware components that can not be modified pragmatically. Therefore, rather than specifying the properties of the resources, such as is the case with virtual machines, the properties of the physical hardware are intrinsic properties of the resource. Emulab-like installations construct their resource request language such that consumers request the specific type of hardware, quantity, and their association to other resources in a network topology. Traditional cloud providers also restrict consumer requests

for compute nodes to set sizes, or flavors, with each instance consisting of hard-coded attributes such as memory, and processors.

Though this practice is not necessary for virtual machines, it's common practice for cloud providers since it allows them to apply price points to each size. This part of the cloud computing paradigm can be exploited since the restriction on customization by consumers allows for physical hardware to be integrated through the same request mechanism.

One important aspect to managing both virtual and physical resources from the consumer's perspective is the ability to retrieve information about resource availability and capability. To extend OpenNebula's ability to report information about physical resources we must first determine the appropriate properties that can adequetly decribe the resource to the consumer.

Thse properties that define a virtual resource include:

- CPU
- VCPU
- RAM
- Disk Image

For physical resources, there are only two properties that all physical resources have. Those include:

- Type of Physical Resource
- Power Interfaces

Any other attributes of a physical resource are associated with the type, or model, of that physical hardware. Furthermore, physical resources are considered to remain immutable. Once installed in a cloud provider's infrastructure, there should not exist

the expectation that the physical resource would, or should, be physically modified. Those modifications would include the addition of network interface, memory or hard disk space, or the attachment of peripheral devices. Any of these modifications would require the addition of a new resource type that OpenNebula would report information on.

Also, by making physical resource immutable we alleviate the requirement for on-going human intervention to maintain a fixed quantity of resources that match the day-to-day requirements of the consumer. Instead, we establish the expectation that is physical resources matching a particular configuration are required by consumers, then the resource provider should incorporate the additional resources into their infrastructure, or replace existing physical resources with the modified ones.

## 5.3 Implementation Strategy

### 5.3.1 Resource Management and Provisioning

To facilitate a mechanism capable of provisioning physical and virtual resources, we must establish how that mechanism can interpret the consumers request and provision the correct resource. Fortunately, there is already a mechanism for accomplishing this. Traditional cloud computing providers offer a mechanism for retrieving a list of pre-define virtual resource sizes. These sizes specify the amount of memory, the number of processors and number of network interfaces a virtual resource is allocated. Pre-defined sizes are restrictive but they allow cloud providers to develop price tiers. We propose to leverage this mechanism by re-purposing for the problem of virtual-physical resource management. We purpose that virtual resource sizes represent resource types, and that one or more instances for each type may exist to represent the different properties of each type.

First, there is a standard virtual resource type consisting of the standard sizes already presented through the aforementioned mechanism. The other three resource

types are all physical types. There are two sizes for our third type, switches, representing switches categorized by number of network ports. For our last device, the NSLU2 is categorized by whether it is powered by a traditional power plug or whether it is powered over the USB cable.

We approach physical resource management in a manner similar to Euclypus. However, we only apply this approach to some types of resources based on special consideration for connection types that require special management. For those that have this approach applied, we assign one or more management nodes to handle the execution of life-cycle steps. Life-cycle steps include restarting physical hardware, and virtual machines, deploying disk images, and provisioning resources based on demand.

### 5.3.2 Management Node

Management nodes are pre-configured physical servers that act as the care-takers of physical devices offered to consumers. A major reason for leveraging this architecture is the ability to integrate with the cloud computing paradigm. Cloud tool-kits provision compute nodes across physical servers by instantiating virtual machines within hypervisors on those servers. Provisioning is only possible when adequate resources exist within the servers that act as hosts. Those resources are typically the available memory, or CPU.

To support the provisioning of physical resources, we must make them known to the cloud tool-kit. For provisioning virtual machines ,physical host are registered with the cloud tool-kit and periodically polled for resource availability and current virtual machine state. Therefore, we first need a method to project availability. Also, provisioning is carried out by the cloud tool-kit by issuing commands remotely on the physical host. Those commands execute actions of the hosts's hypervisor, such as creating, starting, pausing, resuming, and destroying virtual machines. Therefore, in addition to projecting availability, we must also support life-cycle management.

Lastly, because some physical connections can not be routed using network-based services, an intermediary must provide the ability to either route the traffic from those connections to other virtual machines hosts, or instantiate virtual machines directly. A summary of these requirements is given below:

- Resource availability
- Connection management
- Resource life-cycle management

We propose that an intermediary is required to satisfy these requirements. This intermediary we shall call a management node. Management nodes are pre-configure to carry out the tasks required to manage physical resources.

To provide resource availability, the management node must maintain a persistent database of physical resources that are under its control.

Another benefit of using management nodes is the ability to scale. If a greater quantity of of a particular physical resource type if requires, and existing management nodes for that type cannot handle additional resources, then a new management node can be provisioned and added to list of active hosts. Interaction between the cloud tool-kit is identical between virtual machine hosts and physical resource hosts, except for the value polled. For management nodes, only the resource quantity would be different from virtual machine hosts. Resource state information would be reported by both virtual machine hosts and management nodes alike.

### 5.3.3 USB-Devices

One example where a management node is required is for physical resources that have USB ports that we, as the cloud provider, wish to offer to consumers. To facilitate consumer access to those USB ports, we management one or more physical

servers that act as end-points for the USB cables emanating from the physical resources. Those physical servers, which are the management nodes, double as virtual machine hosts.

As an example of this situation, a consumer may wish to reserve two resources, one generic, and the other consisting of the physical resource with a USB port.

We do not provide an approach for establishing a physical connection between the USB ports on two separate servers. Leveraging USB between physical servers does not significantly benefit consumers. We believe communication between servers can be achieved adequately using existing network-based services. Services offered by USB, such as booting from USB, and external storage, can also be achieved using network boot or network-accessible storage on other networked servers.

## 5.4 Intra-VPC Access Management

One important consideration that must be taken before integrating physical systems into a network infrastructure is how those physical systems will be networked. This task is typically easier than indicated since the owner of the network already knows how the physical systems will be used, and therefore, how they should be connected to offer the intended services. Unfortunately, when it is the intended of that owner to offer those physical systems as resources to cloud consumers, the owner now must anticipate all possible connections, and combinations, that consumers might desire. Towards accommodating consumers, it is straight forward to attach cables from every access interface on the physical system, such as serial ports, Ethernet interfaces, and USB ports, to equivalent access interfaces within the network infrastructure. Ethernet interfaces can be connected to switches with VLAN tag support, and serial ports can be connected to serial console servers, which typically support telnet or SSH to serial connections.

Since it is difficult to route certain interfaces, such as USB traffic, we propose that all connections from a type of physical system be connected to the same set of

systems, one or more virtual machine hosts. Therefore, the virtual machine hosts will act as the sole end-point for those types of connections. To accommodate the the need for accessing the USB port on a physical system, we leave it either to the consumer to request a virtual machine attached to the physical end-point of the USB cable, or to the cloud provider to establish a software-based tunneling application for projecting the USB end-point to another location within their infrastructure.

The underlying mechanism for projecting a physical serial port into a virtual machine is already supported by hypervisors such as KVM. What we wish to propose is the method that consumers use to indicate the need for a cloud-based resource to be attached to a cloud-based physical resource. The semantics behind the resource allocation and configuration it important to insure consumers can leverage physical resources within their virtual private clouds, and to insure that cloud providers can appropriately accommodate the consumer's request.

In the manner that access methods allow consumers to fully specify how a resource is accessible from the Internet, so to could it allow consumers to specify how one resource should be accessible by another. We therefore propose to extend our use of access method requests to include requests for methods of access between resources within the cloud. The same semantics used to request an access method for a resource could also apply to requesting access methods between resources.

Between virtualized resources, attaching network interfaces is trivial and applying network traffic conditioning is possible as demonstrated by Emulab [23]. Furthermore, attaching serial devices between virtualized resources is also possible, though the method may require the two attached devices to be co-located on the same host.

# 6. SERVICE MANAGEMENT SYSTEM

## 6.1 Introduction

Our third contribution to the cloud computing paradigm is a new service layer that can deploy, update, and destroy a virtual private cloud (VPC) from a single request. Such a self-contained VPC request would trigger the uploading of virtual disk images, the creation of virtual private networks, and the instantiation of compute resources. In section 6.2 we outline the cloud computing architecture and our service layer's relationship to it, in section 6.3 we describe the design for our new service layer, in section 6.4 we discuss our implementation, and in section 4.7 we discuss our results.

A major inhibiting factor preventing the easy deployment of virtual private clouds in a single request are the inter-dependencies between VPC components. Dependencies include: 1) Virtual disk images that must be uploaded prior to instantiating compute nodes, 2) Networks that must be configured prior to instantiating compute nodes, and 3) the unique identifiers associated with the previous two that must be known when constructing the description of a compute configuration.

To overcome the tight dependence between VPC components, we designed a system that takes a VPC description with fully articulated dependencies and instantiates all the components in the proper order. The instantiation process also retains an accounting of information only available after instantiation of each component, including: the unique name or numerical identifier of the component. In some cases, the unique identifier created during the instantiation of one component is required as part of the definition for another component. Releasing the consumer from having to manage these dependencies between components is one major benefit of using a system that abstracts away the process of deploying virtual private clouds.

The system allows for a virtual private cloud to be defined within a single configuration description and for all its components to be deployed as a single transaction.

A single transaction affords the following benefits: 1) it eliminates the need to issue multiple requests to deploy a single VPC, and 2) the consumer does not need to maintain a detailed state about the progress of of deploying a VPC other than at transaction level.

A direct approach towards deploying resources within the cloud can be achieved through a more pragmatic solution by leveraging application-level libraries that allow scriptable interaction. Libcloud [61] is such an application-level library for the Python language. Each component of a virtual private cloud must be fully specified and must be instantiated in the proper order such that dependencies are satisfied first. This leads to the necessity that the deployment of a virtual private cloud must be written each time a different VPC is deployed.

In order to enable the deployment of VPCs in a language, and implementation independent, fashion, we developed a *Service Management System*, which using the primitives discussed in section 6.2 enables consumers to first describe a VPC and then deploy the VPC to a cloud provider.

## 6.2 Cloud Architecture

Our Service Management System is a service layer supporting consumer-driven deployment of virtual private clouds. In general, cloud architectures are described in the form a three-level stack. At the lowest level is the *physical infrastructure*, which is then leveraged by each proceeding higher level in the stack. At the top of this stack we place the Service Management System, which the underlying complexity of the cloud computing infrastructure is hidden in a manner supportive of large-scale virtual private cloud computing. A cloud stack with our Service Management System layer as the top layer can be seen in Figure 6.1.

*Physical hardware* forms the lowest layer of a traditional cloud stack. This layer of the cloud stack consists of servers, networking devices, power switches, and the connections between them all. Modification at this layer requires the physical manip-

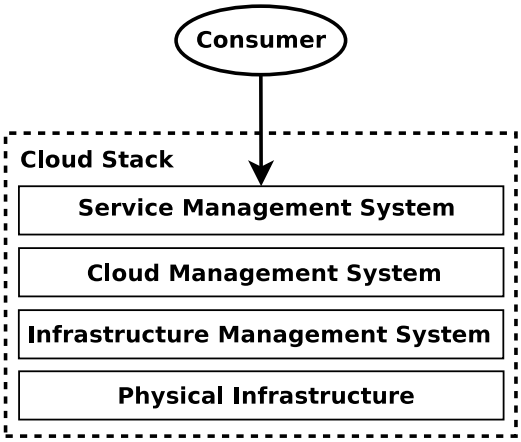


**Fig. 6.1.**: Cloud stack with the Service Management System as the top layer.

ulation of hardware and the configuration of individual hardware devices. Using this layer requires direct interaction with physical hardware by users wishing to utilize the hardware's capabilities.

The next higher layer in the cloud stack is the *infrastructure management system* formed by a cloud computing tool-kit's core management software. This layer provides the functionality necessary to manage the physical infrastructure for the provisioning of virtual machines and virtual networks. This could also include the provisioning and configuration of physical hardware.

Above the infrastructure management system is the *cloud management system* that affords the cloud computing paradigm. Only the minimal infrastructure management related actions, and subsequent properties, are exposed to the users of this layer. Decisions regarding the infrastructure are left to the underlying second layer. Only those actions related to controlling the life-cycle of resources deployed to the cloud, and the configuration of those resources, is supporting by the cloud computing interface. By hiding unnecessary cloud infrastructure management interface commands and abstracting the complexity of those that remain, the cloud computing interface affords the true power of the cloud computing paradigm; shifting users

away from managing physical infrastructures to managing the services that leverage that infrastructure.

Our Service Management System forms the fourth layer of the cloud stack, closest to the consumer, providing services necessary for consumer's to manage VPCs. This additional virtual private cloud layer enables the combination of VPC component definitions into a single request for deployment. This is in contrast to deploying a virtual private cloud through a sequence of request-response pairs.

## 6.3 Service Management System Design

### 6.3.1 Architecture

Our Service Management System consists of two independent, but complementary, applications. The *Service Management Portal* is a web service that accepts virtual private cloud management requests from the consumer and passes those requests on to the second component of the Service Management System. The *Service Management Daemon*, our second application, handles requests by dispatching those requests to independent service agents. Dispatching is supported by a shared queue from which service agents periodically poll and retrieve VPC management requests. existing virtual private clouds to insure that their current state matches the description provided by the consumer.

Virtual private cloud management is facilitated by the Service Management Portal through a RESTful web service as seen in Figure 6.3. Consumers control virtual private clouds by issuing one of four actions. Each VPC management action is mapped to an HTTP method. The four management actions supported by the Portal are:

- GET: Retrieve information on one or more VPCs.
- POST: Request a new VPC.

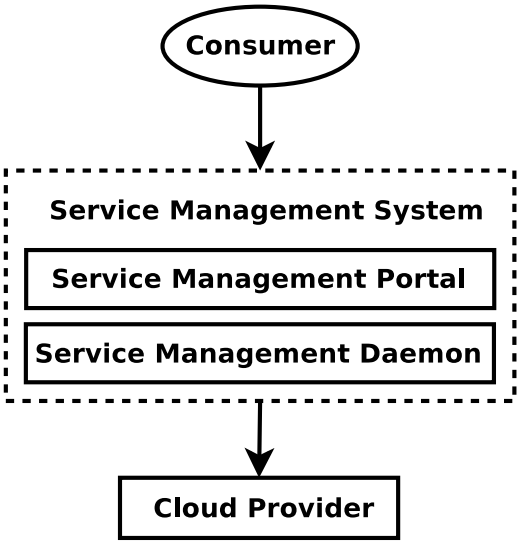


**Fig. 6.2.**: Service Management System architecture.

- PUT: Update an existing VPC.
- DELETE Destroy a VPC.

POST and PUT actions require information specific to the VPC that is being managed to be embedded within the data portion of an HTTP packet. For GET and DELETE actions the ID of the VPC is embedded as part of the URI. Information embedded within the data portion of an HTTP packet pertains to a virtual private cloud's description; specifically the resources, virtual networks and disk images requested by the consumer. The contents of that description are encoded in the expressive language called JavaScript Object Notation [57] (JSON).

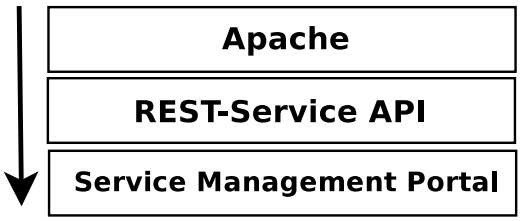


**Fig. 6.3.**: Service Management Portal architecture.

Each virtual private cloud definition contains a collection of attributes and a set of intra-dependencies for each resource and their supporting components. Within a virtual private cloud request, storage devices and virtual networks form the independent components while the resources, physical or virtual, that rely on those components within the cloud provider's infrastructure, are the dependent components. Intra-dependencies are expressed symbolically as named references from dependent components to independent components. A virtual private cloud's naming space is perpetuated only within the VPC's description. Names associated with resources, and their supporting components, must remain unique within a VPC's description, but not within the space of all virtual private clouds.

Our Service Management Portal and Service Management System interact with one another through a local socket that serializes JSON-encoded VPC descriptions between applications. If the Service Management Daemon (SMD) is not active when a request is forwarded to the SMD, then the request will fail and the consumer will be notified. Requests that do not require the Service Management Daemon, such as retrieving information about a virtual private cloud, can be processed from the persistent storage mechanism used by both the Service Management Daemon and Service Management Portal.

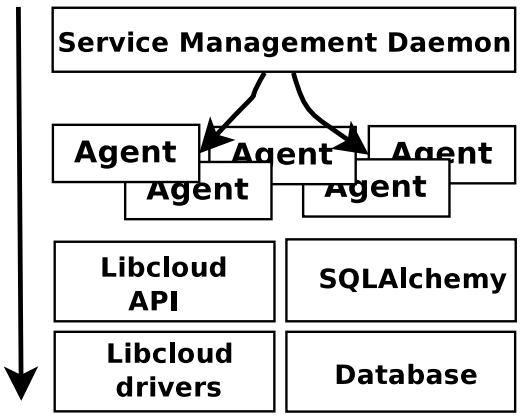


**Fig. 6.4.**: Service Management Daemon architecture.

Each virtual private cloud request is handled by a dedicated Service Agent (SA). Service Agents act on the request by either creating, updating or deleting the virtual private cloud. Our Service Management Daemon queues VPC requests sent by the SMP and, upon the availability of a Service Agent, dispatches the request to that SA. An overview of the SMD architecture can be seen in Figure 6.4. Interaction with cloud providers is accomplished by leveraging the APIs exposed by cloud providers through their publicly available web services.

### 6.3.2 Supporting Components

To support the Service Management System applications, we exploited the features of existing software libraries.

We used three Python libraries to support our handling of consumer requests and interaction with the underlying database. These libraries are DJANGO [58], Django-Piston [59] and SQLAlchemy [60]. No modification of these libraries were required to support our applications.

A fourth Python library, called Libcloud [61], was required was required by our applications to interface with cloud providers. Libcloud has driver support for several dozen cloud provider APIs that allow our application to take a cloud provider agnostic approach. However, because of our need to deploy virtual networks, and our choice to use OpenNebula, we encountered a limitation of Libcloud library. Libcloud has no support for managing virtual networks. Furthermore, Libcloud has a limited driver for OpenNebula's OCCI API. Therefore, we took the latest release of Libcloud, incorporated missing functionality into the library and leveraged that functionality within our new software applications. The lack of network functionality is not a result of a lack of community support, but rather, a consequence of the current cloud computing paradigm.

Libcloud offers an existing abstraction layer for interacting with the public interfaces of cloud providers such as Amazon, OpenNebula and OpenStack. Contained

within the Libcloud library are several sub-libraries for managing different cloud resources such as: compute nodes, storage, load balancers, and DNS.

### 6.3.3 Service Management Portal

Our Service Management Portal supports a mapping of HTTP codes to VPC actions. That mapping is shown below:

- GET: Retrieve either a list of all virtual private clouds owned by the consumer, or, retrieve a symbolic representation of a particular virtual private cloud. In the latter, additional, instance-specific information such as access information, may be included.
- POST: Request the deployment of a virtual private cloud that matches the symbolic representation specified by the consumer.
- PUT: Request that an existing virtual private cloud be modified to match the symbolic representation specified by the consumer.
- DELETE: Delete the specified virtual private cloud. Only those virtual private clouds owned by the consumer making the request may deleted. Deleting a virtual private cloud removes all components that were deployed to create the VPC.

### 6.3.4 Service Management Daemon

Our second component, the Service Management System, is a daemon that accepts and handles requests, maintains a persistent database of virtual private cloud information, and manages the life-cycle of those VPCs. Requests are initiated by the consumer and relate to three particular life-cycle changes: 1) deploy a new virtual private cloud, 2) modify an existing virtual private, or 3) delete all components of

an existing virtual private cloud. Maintaining existing virtual private clouds is initiated by the Service Management System, which schedules periodic tasks intended to verify that all components of a virtual private cloud are running. A maintenance task will destroy any failed components and deploy replacements.

After a component of the virtual private cloud has been deployed to a cloud provider's infrastructure, state information, including a cloud provider unique identifier, is returned. The Service Management System stores that information and then builds a map between the instance information and the symbolic representation.

A map between symbolic and instance representations link the symbolic state of a virtual private cloud to its realized instance.

Management tasks are handled by service agents of the Service Management System. Tasks include deploying, deleting, modifying, and life-cycle management. Each service agent is a separate process that pulls tasks from a shared queue for processing. Upon completion of the task, the service agent pulls the next task in the queue.

## 6.4 Implementation

### 6.4.1 Libcloud

Individual components of a virtual private cloud, such as compute nodes and storage devices, are already supported by the Libcloud tool-kit. The same functionality is supported by similar tool-kits, such as Deltacloud [66]. Neither supports the management of networks.

Virtual private network support has historically been lacking from traditional cloud providers. Only as late as 2009 has Amazon begun to offer virtual private networks between compute nodes [67]. In addition to Amazon, cloud computing tool-kits have begun introducing virtual private networks as a new feature. Due to the recent introduction of virtual private networks, cloud management tool-kits

**Table 6.1**: Network Management Action Arguments

| Action | Aguments |
|---|---|
| List Existing Networks | location |
| Destroy a Network | network |
| Create Network | name, CIDR |

have only now had the option to implement support for deploying, updating and destroying virtual networks.

A common API was derived by determing the least common denominator with regard to support for network management from cloud providers and cloud computing tool-kits. Our common API will act as the base class on which to implement individual cloud network drivers. Further support for cloud tool-kits and providers is achievable by extending the bass class with cloud tool-kit and provider-specific management functionality.

We determined the common functionality was focused around: 1) listing existing networks, 2) destroying a network, and 3) creating a new network. A short description of the functionality is given below:

- Listing Existing Networks: Listing existing networks lists all existing networks owned by a consumer.
- Destroy a Network: Destroys an existing network.
- Create Network: Creates a new network within the infrastructure managed by a cloud provider or cloud computing tool-kit.

Next, arguments for each network management action needed to be determined. Arguments would need to be derived from the properties that define a network by various cloud providers and cloud computing tool-kits.

We conclude with a set of Python-specific method definitions using the aforementioned actions and arguments:

- list_networks(location=none)
- destroy_network(network)
- create_network(name, cidr)

Therefore, with the support of the Libcloud community, we extended Libcloud to include a network library. Our initial effort focused on supporting OpenNebula's OCCI network interface.

### 6.4.2 Service Management System

All functionality specified in the Portal's design were implemented using the appropriate RESTful abstractions and libraries.

We only implemented the ability to handle requests for the instantiation virtual private clouds to cloud providers supporting OpenNebula's OCCI interface.

We have not extended the implementation of the Service Management System to support features beyond instantiating a virtual private cloud. Not implemented handlers return a failure notice indicating to the consumer that the functionality has not been implemented.

# 7. CONCLUSION

## 7.1 Results

With regard to Libcloud, the OpenNebula driver has been re-factored, and now supports OpenNebula v3.0, which is the version of OpenNebula used for this research. Preliminary work has been completed on a Libcloud networking component. There is a working OpenNebula networking driver for Libcloud. However, there are design decisions that require further consideration to insure that the networking component of Libcloud can support driver for other cloud providers.

We constructed the framework for our Access Pathway Manager and one of its two major parts. Implementing the first part of the Access Pathway Manager included the support for injecting access points into virtual machine descriptions, the re-configuration of switch ports, and the routing of traffic from resources to publicly accessible access points. Updates to the OCCI interface for extracting access method requirements and returning access point information will require additional future work.

Our Service Management System is operational and supports the ability to deploy virtual private clouds to OpenNebula clouds. Also, access method sets can be included as part of the virtual private cloud. Work is forthcoming to support retrieving access point information for resources, and supporting a mechanism by which the Service Management System can extract provider supported access methods.

# REFERENCES


[1] A. W. S. LLC, "Amazon elastic compute cloud," February 2012. [Online]. Available: http://aws.amazon.com/ec2/

[2] R. Hosting, "Cloud computing, managed hosting, dedicated server hosting by rackspace," February 2012. [Online]. Available: http://www.rackspace.com/

[3] GoGrid, "Complex infrastructure made easy," May 2012. [Online]. Available: http://www.gogrid.com/

[4] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," *SIGOPS Oper. Syst. Rev.*, vol. 36, pp. 255–270, 2002. [Online]. Available: http://doi.acm.org/10.1145/844128.844152

[5] S. Liu, W. Marti, and W. Zhao, "Virtual networking lab (vnl): its concepts and implementation," in *2001 ASEE Annual Conference Proceedings*, June 2001. [Online]. Available: http://www.umac.mo/rectors_office/docs/weizhao_-cv/pub_refereed_conferences/2001/0106-ASEE-LMZ.pdf

[6] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Virtual infrastructure management in private and hybrid clouds," *IEEE Internet Computing*, vol. 13, no. 5, pp. 14–22, September/October 2009. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/MIC.2009.119

[7] OpenStack and et al., "Openstack," February 2012. [Online]. Available: http://openstack.org/

[8] Eucalyptus and et al., "Eucalyptus," February 2012. [Online]. Available: http://www.eucalyptus.com/

[9] CloudStack and et al., "Cloudstack," February 2012. [Online]. Available: http://cloudstack.org/

[10] Nimbus and et al., "Nimbus," February 2012. [Online]. Available: http://www.nimbusproject.org/

[11] P. Mell and T. Grance, "The nist definition of cloud computing," National Institute of Standards and Technology, Tech. Rep. 800-145, September 2011. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[12] G. Haff, "Just don't call them private clouds," January 2009. [Online]. Available: http://news.cnet.com/8301-13556_3-10150841-61.html

[13] D. Bernstein, E. Ludvigson, K. Sankar, S. Diamond, and M. Morrow, "Blueprint for the intercloud - protocols and formats for cloud computing interoperability." Los Alamitos, CA, USA: IEEE Computer Society, May 2009, pp. 328–336. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/ICIW.2009.55

[14] K. Kevin, "A cloudbook for the cloud," May 2012. [Online]. Available: http://www.kk.org/thetechnium/archives/2007/11/a_cloudbook_for.php

[15] S. Johnston, "The intercloud is a global cloud of clouds," May 2012. [Online]. Available: http://samj.net/2009/06/intercloud-is-global-cloud-of-clouds.html

[16] T. Wood, A. Gerber, K. K. Ramakrishnan, P. Shenoy, and J. Van der Merwe, "The case for enterprise-ready virtual private clouds," in *Proceedings of the 2009 conference on Hot topics in cloud computing*, ser. HotCloud'09. Berkeley, CA, USA: USENIX Association, 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855533.1855537

[17] A. W. S. LLC, "Amazon virtual private cloud," February 2012. [Online]. Available: http://aws.amazon.com/vpc/

[18] L. Cifuentes, R. Mercer, O. Alverez, and R. Bettati, "An architecture for case-based learning," *TechTrends*, vol. 54, pp. 44–50, 2010. [Online]. Available: http://dx.doi.org/10.1007/s11528-010-0453-9

[19] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Cáceres, M. Ben-Yehuda, W. Emmerich, and F. Galán, "The reservoir model and architecture for open federated cloud computing," *IBM J. Res. Dev.*, vol. 53, pp. 535–545, July 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1850659.1850663

[20] B. Rochwerger, A. Galis, E. Levy, J. A. Caceres, D. Breitgand, Y. Wolfsthal, I. M. Llorente, M. Wusthoff, R. S. Montero, and E. Elmroth, "Reservoir: Management technologies and requirements for next generation service oriented infrastructures," in *Integrated Network Management, 2009. IM '09. IFIP/IEEE International Symposium on*, June 2009, pp. 307–310. [Online]. Available: http://dx.doi.org/10.1109/INM.2009.5188828

[21] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "Planetlab: an overlay testbed for broad-coverage services," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 3, pp. 3–12, 2003. [Online]. Available: http://doi.acm.org/10.1145/956993.956995

[22] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A blueprint for introducing disruptive technology into the internet," *SIGCOMM Comput. Commun. Rev.*, vol. 33, pp. 59–64, January 2003. [Online]. Available: http://doi.acm.org/10.1145/774763.774772

[23] Emulab, "Emulab tutorial," March 2012. [Online]. Available: https://users.emulab.net/trac/emulab/wiki/Tutorial

[24] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau, "Large-scale virtualization in the emulab network testbed," in *USENIX 2008 Annual Technical Conference on Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2008, pp. 113–128. [Online]. Available: http://dl.acm.org/citation.cfm?id=1404014.1404023

[25] ——, "Feedback-directed virtualization techniques for scalable network experimentation," Tech. Rep., 2002. [Online]. Available: http://www.cs.utah.edu/flux/papers/virt-ftn2004-02.pdf

[26] Emulab, "Xen-based emulab virtual nodes," March 2012. [Online]. Available: http://users.emulab.net/trac/emulab/wiki/xen

[27] ——, "Multiplexed virtual nodes in emulab," March 2012. [Online]. Available: https://users.emulab.net/trac/emulab/wiki/vnodes

[28] ——, "A short tutorial to setting up eucalyptus cloud system on emulab nodes," March 2012. [Online]. Available: https://users.emulab.net/trac/emulab/wiki/eucalyptus

[29] G. Ayers, K. Webb, M. Hibler, and J. Lepreau, "Whack-on-lan: Inexpensive remote pc reset," University of Utah, Tech. Rep., December 2005. [Online]. Available: http://www.cs.utah.edu/flux/papers/whol-ftn-draft1.pdf

[30] Emulab, "Emulab faq: Using the testbed: Do my nodes have consoles i can look at?" March 2012. [Online]. Available: https://users.emulab.net/trac/emulab/wiki/kb24

[31] ——, "Emulab faq: Security issues: Is emulab firewalled?" April 2012. [Online]. Available: https://users.emulab.net/trac/emulab/wiki/kb58

[32] S. A. Almulla and C. Y. Yeun, "Cloud computing security management," in *Engineering Systems Management and Its Applications (ICESMA), 2010 Second International Conference on*, April 2010, pp. 1–7. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5542654

[33] T. U. of Southern California, "Deter network security testbed," March 2012. [Online]. Available: http://isi.deterlab.net/index.php3

[34] ——, "The deter project," March 2012. [Online]. Available: http://deter-project.org/

[35] T. Benzel, R. Braden, D. Kim, C. Neuman, A. Joseph, K. Sklower, R. Ostrenga, and S. Schwab, "Experience with deter: a testbed for security research," in *Testbeds and Research Infrastructures for the Development of Networks and Communities, 2006. TRIDENTCOM 2006. 2nd International Conference on*, 2006, pp. 388–398. [Online]. Available: http://dx.doi.org/10.1109/TRIDNT.2006.1649172

[36] ——, "Design, deployment, and use of the deter testbed," in *Proceedings of the DETER Community Workshop on Cyber Security Experimentation and Test on DETER Community Workshop on Cyber Security Experimentation and Test 2007*. Berkeley, CA, USA: USENIX Association, 2007, pp. 1–1. [Online]. Available: http://dl.acm.org/citation.cfm?id=1322592.1322593

[37] C. Neuman, C. Shah, and K. Lahey, "Running live self-propagating malware on the deter testbed," in *DETER Community Workshop*, Arlington, VA, June 2006.

[38] R. Ostrenga, S. Schwab, and R. Braden, "A plan for malware containment in the deter testbed," in *Proceedings of the DETER Community Workshop on Cyber Security Experimentation and Test on DETER Community Workshop on Cyber Security Experimentation and Test 2007*, ser. DETER. Berkeley,

CA, USA: USENIX Association, 2007, pp. 10–10. [Online]. Available: http://dl.acm.org/citation.cfm?id=1322592.1322602

[39] T. Benzel, B. Braden, T. Faber, J. Mirkovic, S. Schwab, K. Sollins, and J. Wroclawski, "Current developments in deter cybersecurity testbed technology," in *Conference For Homeland Security, 2009. CATCH '09. Cybersecurity Applications Technology*, March 2009, pp. 57–70. [Online]. Available: http://dx.doi.org/10.1109/CATCH.2009.30

[40] J. Mirkovic, T. V. Benzel, T. Faber, R. Braden, J. T. Wroclawski, and S. Schwab, "The deter project: Advancing the science of cyber security experimentation and test," in *Technologies for Homeland Security (HST), 2010 IEEE International Conference on*, November 2010, pp. 1–7. [Online]. Available: http://dx.doi.org/10.1109/THS.2010.5655108

[41] A. TaheriMonfared, "Securing the iaas service model ofcloud computing againstcompromised components," Ph.D. dissertation, NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY, 2011. [Online]. Available: http://org.ntnu.no/cloudsecurity/thesis/Docs/Report/thesis.ps

[42] A. TaheriMonfared and M. G. Jaatun, "As strong as the weakest link: Handling compromised components in openstack," in *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, December 2011, pp. 189–196. [Online]. Available: http://dx.doi.org/10.1109/CloudCom.2011.34

[43] M. Christodorescu, R. Sailer, D. L. Schales, D. Sgandurra, and D. Zamboni, "Cloud security is not (just) virtualization security: a short paper," in *Proceedings of the 2009 ACM workshop on Cloud computing security*, ser. CCSW '09. New York, NY, USA: ACM, 2009, pp. 97–102. [Online]. Available: http://doi.acm.org/10.1145/1655008.1655022

[44] D. Duchamp and G. De Angelis, "A hypervisor based security testbed," in *Proceedings of the DETER Community Workshop on Cyber Security Experimentation and Test on DETER Community Workshop on Cyber Security Experimentation and Test 2007*, ser. DETER. Berkeley, CA, USA: USENIX Association, 2007, pp. 3–3. [Online]. Available: http://dl.acm.org/citation.cfm?id=1322592.1322595

[45] J. Mirkovic, B. Wilson, A. Hussain, S. Fahmy, P. Reiher, R. Thomas, and S. Schwab, "Automating ddos experimentation," in *Proceedings of the DETER Community Workshop on Cyber Security Experimentation and Test on DETER Community Workshop on Cyber Security Experimentation and Test 2007*, ser. DETER. Berkeley, CA, USA: USENIX Association, 2007, pp. 4–4. [Online]. Available: http://dl.acm.org/citation.cfm?id=1322592.1322596

[46] S. Schwab, B. Wilson, C. Ko, and A. Hussain, "Seer: a security experimentation environment for deter," in *Proceedings of the DETER Community Workshop on Cyber Security Experimentation and Test on DETER Community Workshop on Cyber Security Experimentation and Test 2007*, ser. DETER. Berkeley, CA, USA: USENIX Association, 2007, pp. 2–2. [Online]. Available: http://dl.acm.org/citation.cfm?id=1322592.1322594

[47] I. Houidi, M. Mechtri, W. Louati, and D. Zeghlache, "Cloud service delivery across multiple cloud platforms," in *Services Computing (SCC), 2011 IEEE International Conference on*, July 2011, pp. 741–742. [Online]. Available: http://dx.doi.org/10.1109/SCC.2011.107

[48] A. Celesti, F. Tusa, M. Villari, and A. Puliafito, "How to enhance cloud architectures to enable cross-federation," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, July 2010, pp. 337–345. [Online]. Available: http://dx.doi.org/10.1109/CLOUD.2010.46

[49] T. Faber and J. Wroclawski, "Access control for federation of emulab-based network testbeds," in *Proceedings of the conference on Cyber security experimentation and test*, ser. CSET'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 6:1–6:6. [Online]. Available: http://dl.acm.org/citation.cfm?id=1496662.1496668

[50] K. Sklower and A. D. Joseph, "Very large scale cooperative experiments in emulab-derived systems," in *Proceedings of the DETER Community Workshop on Cyber Security Experimentation and Test on DETER Community Workshop on Cyber Security Experimentation and Test 2007*, ser. DETER. Berkeley, CA, USA: USENIX Association, 2007, pp. 12–12. [Online]. Available: http://dl.acm.org/citation.cfm?id=1322592.1322604

[51] T. Faber, J. Wroclawski, and K. Lahey, "A deter federation architecture," in *Proceedings of the DETER Community Workshop on Cyber Security Experimentation and Test on DETER Community Workshop on Cyber Security Experimentation and Test 2007*, ser. DETER. Berkeley, CA, USA: USENIX Association, 2007, pp. 11–11. [Online]. Available: http://dl.acm.org/citation.cfm?id=1322592.1322603

[52] T. Harmer, P. Wright, C. Cunningham, and R. Perrott, "Provider-independent use of the cloud," in *Euro-Par 2009 Parallel Processing*, ser. Lecture Notes in Computer Science, H. Sips, D. Epema, and H.-X. Lin, Eds. Springer Berlin / Heidelberg, 2009, vol. 5704, pp. 454–465, 10.1007/978-3-642-03869-3-44. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-03869-3_44

[53] *A Two-Constraint Approach to Risky Cybersecurity Experiment Management*, April 2008. [Online]. Available: http://www.isi.edu/ mirkovic/publications/sarnoff.pdf

[54] I. of Electrical and E. Engineers, "Ieee standard for local and metropolitan area networks–media access control (mac) bridges and virtual bridged local area networks," Institute of Electrical and Electronics Engineers, Tech. Rep., August 2011. [Online]. Available: http://www.techstreet.com/standards/ieee/802_1q_2011?product_id=1778388

[55] ——, "Ieee standard for local and metropolitan area networks—virtual bridged local area networks—amendment 4: Provider bridges," Institute of Electrical and Electronics Engineers, Tech. Rep., May 2006. [Online]. Available: http://www.techstreet.com/standards/ieee/802_1ad_2005?product_id=1270166

[56] O. C. C. I. W. Group, "Open cloud computing interface," Open Grid Forum, February 2012. [Online]. Available: http://occi-wg.org/

[57] D. Crockford, "The application/json media type for javascript object notation (json)," RFC 4627 (Informational), 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4627.txt

[58] D. S. Foundation and et al., "Django," February 2012. [Online]. Available: https://www.djangoproject.com/

[59] J. Noehr and et al., "Django-piston," February 2012. [Online]. Available: https://bitbucket.org/jespern/django-piston/wiki/Home

[60] SQLAlchemy and et al., "Sqlalchemy," May 2012. [Online]. Available: http://www.sqlalchemy.org/

[61] T. Muraus and et al., "Libcloud," May 2012. [Online]. Available: http://libcloud.apache.org/

[62] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000. [Online]. Available: http://www.ics.uci.edu/ fielding/pubs/dissertation/top.htm

[63] N. Weaver, S. Staniford, and V. Paxson, "Very fast containment of scanning worms, revisited," in *Malware Detection*, ser. Advances in Information Security, M. Christodorescu, S. Jha, D. Maughan, D. Song, and C. Wang, Eds. Springer US, 2007, vol. 27, pp. 113–145. [Online]. Available: http://dx.doi.org/10.1007/978-0-387-44599-1_6

[64] A. Haeberlen, "A case for the accountable cloud," *SIGOPS Oper. Syst. Rev.*, vol. 44, pp. 52–57, April 2010. [Online]. Available: http://doi.acm.org/10.1145/1773912.1773926

[65] OpenNebula and et al., "Opennebula," July 2012. [Online]. Available: http://opennebula.org/

[66] M. Fojtik and et al., "Deltacloud," May 2012. [Online]. Available: http://deltacloud.apache.org/

[67] L. Dignan, "Amazon launches virtual private cloud service," May 2012. [Online]. Available: http://www.zdnet.co.uk/news/cloud/2009/08/27/amazon-launches-virtual-private-cloud-service-39730194/

# VITA


| | |
|---|---|
| Name: | Hutson Keith Betts |
| Address: | Department of Computer Science and Engineering<br>Texas A&M University<br>TAMU 3112<br>College Station, TX 77843-3112 |
| Email Address: | hut101@tamu.edu |
| Education: | B.S., Computer Science, Texas A&M University, 2009<br>M.S., Computer Engineering, Texas A&M University, 2012 |