LEVERAGING OPENFLOW FOR RESOURCE PLACEMENT OF VIRTUAL
DESKTOP CLOUD APPLICATIONS

THESIS

Presented in Partial Fulfillment of the Requirements for the Degree Master of Science in
the Graduate School of The Ohio State University

By
Sudharsan Rajagopalan

Graduate Program in Computer Science and Engineering

The Ohio State University

2013

Master's Examination Committee:

Dr. Rajiv Ramnath, Advisor

Dr. Jay Ramanathan

Dr. Prasad Calyam.
Abstract

Popular applications such as email, photo/video galleries, and file storage are increasingly being supported by cloud platforms in residential, academia and industry communities. The next frontier for these user communities will be to transition ‘traditional desktops’ that have dedicated hardware and software configurations into ‘virtual desktop clouds’ that are accessible via thin-clients. In this paper, we describe an Intelligent resource placement framework for thin-client based virtual desktops. The framework leverages principles of software defined networking and features a ‘unified resource broker’ that uses special ‘marker packets’ for: (a) “route setup” when handling non-IP traffic between thin-client sites and data centers, (b) “path selection” and “load balancing” of virtual desktop flows to improve performance of interactive applications and video playback, and to cope with faults such as link-failures or Denial of Service cyber-attacks. The Framework has the ability to provisioning OpenFlow paths with less Service Response times for VD Requests. Our Framework In addition, we detail our framework implementation within a virtual desktop cloud (VDC) setup in a multi-domain Global Environment for Network Innovations (GENI) Future Internet testbed spanning backbone and access networks with a automation and centralized control using a tool called VDC-Sim. We present empirical results from our experimentation that leverages OpenFlow programmable networking, as well as cross-traffic capabilities for validating our framework in GENI under realistic settings. Our results demonstrate the importance
of scheduling regulated measurements that can be used for intelligent resource placement decisions. Our results also show the feasibility and benefits of using OpenFlow controller applications for path selection and load balancing between thin-client sites and data centers in VDCs. The thesis also shows how our OpenFlow Framework can used for other cloud applications using GridFTP application over WAN as a Case Study.
I dedicate this work to my family and friends.
Acknowledgments

I would like to acknowledge Dr. Prasad Calyam for being my primary advisor and supporting me during my thesis research. I also thank Dr. Rajiv Ramnath, for giving me the opportunity to be part of their research group. I am extremely grateful for their mentoring me through the entire course of my research with their constant guidance, insight and encouragement. I would also like to thank Dr. Khaled Salah of Khalifa University, UAE and my colleagues at Ohio Supercomputer Center/OARNet, Alex Berryman, Arunprasaath Selvadurai and Saravanan Mohan who collaborated with me at various stages of my thesis and gave valuable inputs. I would like to express my gratitude to Dr. Jay Ramanathan for consenting to be part of my thesis committee.
Vita

2008.........................................................B.E. Computer Science and Engineering

Anna University, MIT Campus.

2011 to present ........................................M.S. Computer Science and Engineering,

The Ohio State University

Publications


Fields of Study

Major Field: Computer Science and Engineering
Table of Contents

Abstract ............................................................................................................................... ii

Acknowledgments ............................................................................................................. v

Vita ...................................................................................................................................... vi

Publications ......................................................................................................................... vi

Fields of Study ................................................................................................................... vi

Table of Contents .............................................................................................................. vii

List of Figures ..................................................................................................................... ix

Chapter 1 Introduction ........................................................................................................ 1

1.1 Traditional networks, Software Defined Networking and OpenFlow ...................... 1

1.2 Advantages of OpenFlow ............................................................................................. 5

1.3 OpenFlow for Cloud Computing Environments ......................................................... 6

1.4 Virtual Desktop Clouds Overview .............................................................................. 8

1.5 Components of Virtual Desktop Cloud ................................................................. 10

1.6 OpenFlow and Virtual Desktop Clouds Motivation .............................................. 12

1.7 Thesis Outline .............................................................................................................. 14

Chapter 2 Related Work ................................................................................................. 15

Chapter 3 Problem Formulation And Solution Approach ........................................ 19
3.1 The Problem

3.2 Solution Approach

3.3 VDC Resource Placement Framework Architecture

Chapter 4 The Marker Packet Scheme

4.1 Marker Packet Structure

4.2 Marker Packet Scheme Workflow

4.3 Experiment Topology Setup and Configuration

4.4 Application Performance Measurements

4.5 Discussion

Chapter 5 Validation of the Analytical Framework for Service Response Time Estimation

5.1 The Analytical Model

5.2 Validation of the Analytical Model Using a MiniNet Simulation

5.3 Comparison of Analytical and Simulated Results

Chapter 6 Network Adaptive Load Balancing

6.1 Need for Load Balancing

6.2 Topology, Experiment and Results

Chapter 7 Automation Using VDC-Sim

7.1 VDC-Sim Modules for Running Experiments
List of Figures

Figure 1.1 Control and Data Planes in Traditional Network ......................... 1
Figure 1.2 Control and Data Planes in OpenFlow Network ........................... 3
Figure 1.3 OpenFlow Forwarding Rule ......................................................... 4
Figure 1.4 OpenFlow Interactions ................................................................. 5
Figure 1.5 Components of Virtual Desktop Clouds ........................................ 10
Figure 3.1 Intelligent VDC Resource Allocation Architecture ...................... 23
Figure 4.1 Marker Packet Structure ............................................................. 26
Figure 4.2 Marker Packet Scheme Workflow ................................................ 27
Figure 4.3 GENI OpenFlow Slice for VDC with Marker Packets in GENI .......... 30
Figure 4.4 Routes for measuring application performance ............................ 32
Figure 4.5 Paths Performance Comparison ................................................... 33
Figure 5.1 Four stages of service in the Analytical Model ............................................ 35
Figure 5.2 Service Response times for provisioning VMs by adding a user ..................... 37
Figure 5.3 Service Response times for provisioning VMs by creating a new .................... 38
Figure 5.4 MiniNet simulation Topology ....................................................................... 39
Figure 5.5 Service Response Times for Data center Virtual Desktop Provisioning for a load of 500 .................................................................................................................. 41
Figure 5.6 Service Response Times for Network path provisioning ................................ 42
Figure 5.7 Service Response Times for Network path provisioning from MiniNet Simulation .......................................................................................................................... 42
Figure 6.1 Virtual Desktop Cloud flows in GENI network ............................................. 44
Figure 6.2 Virtual Desktop Cloud flows in GENI network ............................................. 45
Figure 6.3 VDC-Sim Experiment with Load Balancer off ............................................. 48
Figure 6.4 VDC-Sim Experiment with Load Balancer off ............................................. 48
Figure 6.5 VDC-Sim Experiment with Load Balancer on ............................................. 49
Figure 7.1 VDC-Sim User Interface. a) The Cloud Broker. b) The Workload Generator. c) Running a Simulation. d) Running an Experiment. e) Resource allocation Graph. f) Utility Graph g) Service Response Times ................................................................. 53
Figure 7.2 VDC-Sim Experiment Automation Workflow ................................................. 55
Figure 8.1 GridFTP Transfers ......................................................................................... 58
Figure 8.2 GridFTP in GENI without Load balancing .................................................. 59
Figure 8.3 GridFTP in GENI Data transfer Rates ......................................................... 60
Figure 8.4 GridFTP in GENI with Load balancing ....................................................... 61
Figure 9.1 Virtual Tenancy Using OpenFlow

xi
Chapter 1 Introduction

1.1 Traditional networks, Software Defined Networking and OpenFlow

Traditional networking devices such as routers and switches are made up of two major components; a Control Plane and a Data Plane as shown in Figure 1.1. A control plane in a networking device decides the egress port for an ingress packet. In a router a routing algorithm such as OSPF (Open Shortest Path) , RIP (Routing Information Protocol) makes the decision and in a traditional switch it is done by ARP (Address Resolution Protocol) learning. This decision is called a packet forwarding rule. For routing, a rule can state that if an incoming packet has an IP address that belongs to particular IP network then it has to be egressed via a particular port. For Ethernet switching, a rule can state that if a Ethernet frame has a particular destination Ethernet address the frame has to be egress via a particular port.

Figure 1.1 Control and Data Planes in Traditional Network
Once a rule is computed at the control plane it is informed to the data plane. The data plane has a set of such rules which is stored in a data structure called the forwarding table. The data plane is the component of a networking device that decides what to do with packets arriving on an inbound interface. It matches the packet fields such as IP addresses or Ethernet addresses to the rules in the forwarding table, determines the proper egress interface and sends it out.

Software Defined Networking (SDN) is a new form of network architecture in which the control and data planes are decoupled. In traditional network the network intelligence was distributed among the network devices. But with Software Defined networking this intelligence is logically centralized. As a result, network administrators in various sectors such as enterprises and carriers gain programmability, mechanization, and network control, enabling them to build highly scalable, elastic networks that readily adapt to dynamic load levels and changing customer and business needs. OpenFlow is a standard interface designed specifically for SDN, providing high-performance, granular traffic control across network devices.
With the introduction of OpenFlow[1] the control plane component of these networking devices can run on a remote machine that calculates these rules for all the networking devices in the network. This remote control plane is called the OpenFlow Controller. The rules as shown in Figure 1.3 for forwarding packets are computed in this remote machine and are communicated to an OpenFlow capable network device through a secure channel as shown in Figure 1.2 and are installed in the forwarding tables of these devices. The OpenFlow controller is capable of running applications that implement sophisticated algorithms to calculate a path for set of packets that have similar values in their packet header fields in a Network. An OpenFlow controller also allows network administrators to build applications to guide the packets based on their needs, the physical network topology and the load in the network, allowing networks to more flexible in handling data traffic.
OpenFlow capable network devices such as OpenFlow switches as shown in Figure 1.4 consists of three parts; a flow table with a set of rules on how to process a flow which is implemented in the hardware, a secure channel that allow the switch to communicate with the controller and the OpenFlow protocol that defines a standard interface that allows the controller to manage the entries in the flow table. The last two parts are implemented in software.
1.2 Advantages of OpenFlow

- OpenFlow allows centralized automated management of the network through customized applications developed with OpenFlow controller APIs that run on the controller.
- OpenFlow uses the concept of flows to identify network traffic based on rules that can be programmed statically or dynamically by the OpenFlow controller. A rule is a set of fields in the packet header that has to be matched in a packet to determine a packet flow. For example packets with destination IP addresses in the IPv4 header that belong to the same IP same subnet can be classified as a flow. Based on the flow certain actions such as forwarding the packet via a particular interface can be programmed in the flow table of the network device using a OpenFlow controller. Since OpenFlow allows the device to be programmed on a per flow basis, it provides a granular control over the network.
- The granularity helps in applying various network policies at a user, session, and device or at application level. For example application performance issues attributed to
the network can be eliminated by application based flow classification. In current IP networks flows from two different applications moving between same endpoints will follow the same path. But with OpenFlow the network can be programmed with two different paths for these applications such that the flow from one application does not affect other. Thus by decoupling the control and data planes, by using flow based traffic management mechanisms OpenFlow has the ability to transform computer networks.

- OpenFlow enables better service delivery as applications can exploit centralized network state information to seamlessly and intelligently adapt network behavior to user needs.

### 1.3 OpenFlow for Cloud Computing Environments.

The characteristics of the network traffic in cloud computing environments and the features of OpenFlow makes OpenFlow networking the ideal infrastructure for clouds. In this section the issues in current cloud networks and how OpenFlow can tackle them are discussed.

The load in the cloud network infrastructure is erratic. The traffic patterns change unpredictably. In a client-server application the bulk data transfer is between single client and the server. But in a cloud there is a bustle of traffic moving between different database and fileservers before the data is returned to the user. This traffic patterns change based on application. An application that involves multimedia data moves more traffic than an application that involves text data. The load changes during the time of day and can be characterized to take a random diurnal form. The traditional static IP networks
become a bottleneck for applications as it will affect the end user experience. On the other hand with OpenFlow the network can be exploited to adapt for these kinds of dynamic loads and increase bandwidth utilization by provisioning network resources on the fly based the load and Service Level Agreements thereby satisfying the end user. The granularity of OpenFlow, i.e. ability to manage the traffic at the application level allows optimized use of network resources by automating provisioning the network resources dynamically based on the Application needs.

The Cloud requires its network to be scalable. There is rapid growth in number and the nature of cloud services from Video streaming, Software as a Service (SaaS) and Desktop as a Service (DaaS) that can be provided. This has led to increase in the number of users using these services. The advances in mobile computing bring more devices that will be accessing cloud resources. This growth requires cloud network to be elastic at the same time it should be easier to enforcing Quality of Service (QoS) and security policies in the network. In the current static network infrastructure adding network device involves change configuration of several other network devices which makes scaling a cumbersome manual task. Since OpenFlow uses a centralized controller to impose the data forwarding rules, QoS and Security policies on the network device can be automated making management of the network scaling easier. The OpenFlow controller can control devices from different vendors making interoperability one less attribute to worry about. Server virtualization technologies have made the cloud environment more dynamic. resource allocation and optimization techniques contributes to consistent increase in the number of Virtual Machines (VM) in these servers. VMs migrate across geographically
distributed servers to balance server loads. This requires the underlying network infrastructure to be dynamic as the physical location end-points keep changing. This is in contrast with the static nature of current IP network infrastructure. On the other hand, OpenFlow with its inherent dynamic qualities can substantiate server virtualization.

Network Virtualization has become an important aspect of cloud computing. Hosts from same data center are being used by several tenants. Network Virtualization is necessary to provide traffic isolation for these tenants, thereby enforcing policies specific to the tenants’ business needs and at the same use the underlying physical network effectively in such a way that the clientele traffic do not affect each other. Decoupling control and data planes and a centralized network intelligence make it easier to exploit the physical network by implementing virtual networks on top it, in which network resources can be allocated and de-allocated based on tenant’s needs and provide traffic isolation at the same time.

OpenFlow with easier network resource provisioning, granularity, scalability and traffic isolation opens up the cloud network that was limited by the current networking infrastructure characterized by cumbersome management and static nature.

1.4 Virtual Desktop Clouds Overview

Nowadays online applications such as email, photo/video galleries and file storage are being hosted on cloud platforms providing these software as services (SaaS) in both academia and enterprise due to performance and scalability issues in client-server models. This has contributed increase in number of users using cloud applications. With
Server Virtualization, data centers hosting several Virtual Machines (VM) provides Desktop as a Service (DaaS). This has motivated these user communities to opt for virtual desktops hosted on a cloud infrastructure. Virtual Desktops (VD) can be accessed through thin-clients that are machines with minimal hardware. This makes DaaS as ideal candidate for a domestic utility such as similar to the model we have today for other common computing and communication needs such as VoIP (e.g., Vonage), and IPTV (e.g., Roku). Just like SaaS applications such as Google Docs, Netflix, Spotify that are popular among residential users, we can envisage home users signing-up for VDs. With such a utility service, a thin-client i.e., a set-top-box can be shipped by the VD Cloud service provider (CSP) to residential user to access a VD. This box can be connected to a television, or computer monitors, and multiple residential users can securely access their VDs. Convenience for users, consistent user-perceivable peak performance, and cost-savings contribute a lot to this change.

With the growth in number of users, managing and allocating processor, storage and network resources in an optimal way can be cumbersome. Previous research works on Virtual Desktop Clouds targeting allocation and management of processor, storage and network resources show that CSPs can have complete control over these processor and storage resources. The shortcomings of current IP network infrastructure such as static nature leading to underutilization of bandwidth, vendor dependency for configuration of devices, interoperability between devices of different vendors, longer network resource provisioning cycles, scalability of the network makes it difficult for
CSPs to exercise control over the network in such way that network does not degrade the performance of cloud application.

1.5 Components of Virtual Desktop Cloud

The components of a Virtual Desktop Cloud are shown in figure 1.5. Like any cloud systems the VDC consists of several data centers distributed geographically. These Data Centers are configured with a hypervisor framework such as VMware ESXi on top of which VMs run. Processor, Storage and Network resources are allocated for these VMs from the data center and softwares are installed on them based on the user requirement.

![Figure 1.5 Components of Virtual Desktop Clouds](image-url)
The Unified Resource Broker (URB)[4] receives VD requests, computes the resource requirement, and does the job of resource placement (location of the resource) and provisioning (amount of resource) for the VD request. It acts as the central intelligence system for the VDC. The URB is responsible for optimized resource allocation in such way that it is not over allocated or under allocated and scalability of VDC in terms of number of VD Requests handled. It has to allocate resources as soon as possible reducing the Service Response Times (SRT) for the users at the same time maintaining Service Delivery Quality (SDQ) which is perceived through users Quality of Experience (QoE). It has the information regarding user groups and their application profiles which is used in provisioning a VD for a user. ‘Virtual Desktop pools’ are created based on these profiles and a Virtual Desktops are allocated from these pools accordingly. To satisfy the Service Level Agreements (SLA) between the CSP and the user the URB maintains the minimum and the maximum amount of resources that is to be provisioned based on the profiles. The URB maps the VDs to Data Centers based on thin client latency, server load and cost constraints.

In previous research placement and provisioning problems are addressed with several schemes and are implemented within the URB. One such scheme is a Fixed Resource Allocation Model (F-RAM) in which all VDs are provisioned with same amount of resources. With human-aware, system-aware and network-aware frameworks and tools to optimize resource allocation in virtual desktop clouds in order to improve scalability and increase user Quality of Experience (QoE). An Utility-directed Resource Allocation Model (U-RAM)[3] that provides optimal resource allocation has been
implemented. An Economic Resource Allocation Model (E-RAM) is implemented that take economic policies (such as maximizing profit or reputation, giving preference to premium customers) into account when allocating resources.

1.6 OpenFlow and Virtual Desktop Clouds Motivation.

The Performance of Virtual Desktop Clouds is very much affected by current networking model. In this section we discuss How Virtual Desktop Clouds are affected by IP networks and can OpenFlow overcome these issues.

Static IP Networks does not provide granular traffic control for Virtual Desktop Cloud Applications. For example if there are two users from the same client site accessing two different VDs in the VDC, traffic from both the clients will have the same path throughout the network and the traffic from each of these clients is going to affect one another. Or if a single user uses a couple of applications that utilize high bandwidth, traffic from these two applications are going to affect one another static IP networks may not provide user level or application level granular control over this traffic. But with OpenFlow forwarding rules can give this kind of granular control on the traffic on network.

IP Networks have distributed network intelligence. Each network device in the network calculates routes based on its own position in the topology. Paths are calculated in way each router assumes itself as the source and the paths are calculated from that router. Each router will have its own version of the topology. In a OpenFlow network the OpenFlow controller an agent being external to the physical network, has a better view of
the network topology, that helps in better control over the physical network.

In an IP network, once the routes are converged there exists only one route that is the best route between two end points, even though the physical network has more paths between these destinations. Performance of an application like Virtual Desktop Cloud that uses high bandwidth will affected by other traffic in this path. In case of network congestion IP networks fail to use an another path automatically, thereby underutilizing the existing bandwidth making the performance of the application to suffer a downfall. With OpenFlow such anomalies can be eliminated with a proper path selection application that can select paths based on the existing network’s traffic, running on the controller.

Alternate path selection in IP networks is possible. To do this the network administrator will have to install static routes in all the devices in the network. This job is manual and takes a lot of time. With OpenFlow this can be done on the fly without any hassle.

Network management including adding new devices to support scalability of Virtual Desktop Clouds, in the networks, configuring QoS policies is extremely cumbersome and involves a lot of manual work. In a OpenFlow network with a OpenFlow controller configuring new devices in the network and enforcing policies can be automated and can be done easily.

Considering all these issues in Traditional Networks, OpenFlow can transform the current network infrastructure; making is more suitable for Virtual Desktop Clouds.

13
1.7 Thesis Outline

The remainder of this thesis is organized as follows: In Chapter 2, we describe prior literature related to our work. In Chapter 3, we formulate the problem we are solving in this thesis and a brief introduction to our solution approach. In Chapter 4, we describe Marker Packet scheme. In Chapter 5, we validate our Service Response time Estimation Framework. In Chapter 6, we present our Network Adaptive Load Balancing Experiments and Results. Chapter 7 talks about the Automation of the VDCloud experiments in GENI[5] using a Novel Toolkit called the VDC-Sim. In Chapter 8 we show how our OpenFlow Framework can be used for other cloud Applications using GridFTP[25] as a Case Study. Chapter 9 concludes the thesis.
Chapter 2 Related Work

The rapid development of cloud computing environments and recent advances in virtualization have created new requirements for traditional networks and distributed computing. There have been several recent works that have proposed novel network virtualization and resource adaptation techniques between geographically distributed data centers supporting cloud computing environments.

OpenFlow technology [1] that is built upon software defined networking principles and supported in several vendor switches has emerged as a prominent solution for programmable control of routing and other services (e.g., virtual machine migration) for application flows. Authors in [9] show how programmable flows can be realized by matching packet flows based on the IP addresses, MAC addresses and the port numbers. In their packet forwarding scheme, packets are forwarded to a particular output port if a flow entry for a client is found. Else, the packet is forwarded to a controller application, which then makes forwarding decisions. The demonstrations done in [10] and [11] are based up on this approach for the controller application implementation. In our work, we also adopt a similar approach for installing flows at the time of a thin-client's request to the URB for a VD connection. Once the flows are setup, the traffic between the thin-client and corresponding VD do not pass through the controller application and thus no additional delays are introduced within the application flows.
Alternate technologies such as Overlay Transport Virtualization (OTV) [12], and more recently, Virtual Extensible LAN (VXLAN) [13] have been proposed within industry groups that are in contrast with the open-standards used in OpenFlow solutions. VXLAN [13] has been adopted within several industry vendor products for scalable LAN segmentation and for automated provisioning of logical networks between data centers across Layer 3 networks. A 24-bit LAN segment identifier and MAC-in-UDP encapsulation are used within VXLANs for achieving segmentation between data centers and to realize Layer 2 elasticity and IP address localization to enable for e.g., multi-tenancy and migration of virtual machines across multiple Layer 2 domains. In our work that relies on OpenFlow technology, we use the concept of ‘marker packets’ within packet headers that are processed by the controller application to handle traffic (i.e., non-IP traffic) within VLAN extensions across multiple Layer 3 networks.

Currently, there are several tools which simulate Cloud infrastructures and provide testbeds for simulating and verifying resource allocation schemes on Clouds. CloudSim tool kit [18] offers a Cloud computing framework for modeling and simulating applications provisioning policies on the Cloud. Network CloudSim [20] is an extension of CloudSim which provides a networking model amongst data centers to simulate complex application services and improve scalability by maximizing the network bandwidth. GreenCloud [21] tool is built on the network simulator NS2 [16] and includes simulation of energy aware data centers on Cloud environments and provides a detailed modeling of energy consumed by the servers, data centers, switches and links. Similarly, Jrad et al. [19] have developed a simulation framework for validating and evaluating
interoperable Cloud service brokers on heterogeneous Cloud environments. iCanCoud [22] is another simulation platform for modeling and simulating Cloud infrastructures which leverages system performance in determining the costs involved on the infrastructures. On the other hand, SimGrid [23] and GridSim [24] frameworks are focused towards simulating resource allocation policies for Grid Computing of peer-to-peer (P2P), parallel and distributed systems. While all of the above works are targeted towards applications oriented Clouds, the VDC-Sim tool is unique in simulating resource allocation policies targeted towards VDC infrastructures where user QoE and Service Response Times play predominant roles on the system performance. The tool also provides Run Simulation and Run Experiment allowing users to test their schemes in both environments on a single simulator, which has not been exploited previously. The tool allows users to plugin resource allocation modules through intelligible Perl scripting to test various provisioning and placement schemes on the VDC. The tool provides a simple Graphical User Interface (GUI) to select desired values of loads, cross traffic and faults on the VDC application similar to other related works. Network quality metric plays a critical role in determining the perceived user QoE of a VDC application. There exist many network simulators like NS-2 and OPNET IT Guru [16] [17] which emulate a networking testbed for measuring application performance under varied network states. But rerouting of network packets and bandwidth allocations based on user perceived user QoE cannot be controlled through such simulators and hence there has been limited work in controlling network for maximizing utility on Cloud Computing infrastructures. In our work, we harness the SDN ingenuity using the OpenFlow Protocol to monitor the
network health and tweak the network allocation mechanism based on the user QoE, to maximize the NetUtility of the VDC application.
Chapter 3 Problem Formulation And Solution Approach

3.1 The Problem

Any cloud platform’s capability to support large user workloads is a function of both the server side desktop performance as well as the remote user-perceived QoE. In other words, a CSP can provision adequate CPU and memory resources to a VD in the cloud, but if the thin-client protocol configuration or VD placement does not account for network health degradations and application context, the VD may become unusable for the user. Hence, there is a need to couple “human-and-network awareness” within Internet-scale resource allocation frameworks for: (a) minimizing costly cloud resource over-provisioning, (b) avoiding guesswork in configuring thin-client protocols and VD placement, (c) adapting resource allocation and placement to changing fault-levels and (d) ultimately delivering optimum user QoE of virtualized desktop applications.

Moreover, VD delivery using VDCs can be considered as a ‘Future Internet’ application due to its specialized infrastructure needs that the current Internet infrastructure does not inherently support. Thin-client based VD protocols require several tens of Mbps of network bandwidth for supporting few tens of users, and low network latency is critical for smooth user interactions. As shown from VD protocol characterization studies in [2], one of the most common thin-client protocol in today’s enterprises namely, PCoIP will use as much bandwidth given to it, reacts sensitively to
varying network health, and delivers corresponding user quality of experience (QoE). Better quality translates to smoother video, crisp keyboard/mouse control actions for user applications in VDs. In addition, resource requirements for Desktop-as-a-Service (DaaS) are much higher (e.g., every VD may need resources on the order of 2 GHz CPU, 2 GB RAM and 2Mbps end-to-end network bandwidth) and predicting resource allocation needs is challenging due to bursty nature of workload profiles. In comparison, current cloud infrastructures that support online applications (e.g., photo/video sharing web-sites, customer relationship management software) are transaction-oriented and their workloads are more predictable, and approaches such as overprovisioning are not cost-prohibitive, opposed to the DaaS case. Hence, DaaS platforms are not widely supported by Cloud Service Provider (e.g., Amazon, Microsoft, Dell) due to above challenges in delivering optimal user QoE with current cloud infrastructures on the Internet. To develop and demonstrate validity of intelligent resource allocation methodologies for VDCs, researchers need to experiment on multi-data center VDC settings with realistic user, network and system loads, and with geographically distributed thin-client sites.

With geographically distributed thin client sites, the network between the thin client site and the data center becomes an issue of major concern when it comes to application performance. Application protocols like PCoIP were designed to use the bandwidth available completely. But the network in between underutilizes the bandwidth resources by improper path selection and lack of capabilities to provide an alternate path quickly in case of congestion. As a result the application slows down. Earlier the
Network was a bottleneck and applications where designed to adapt to network related issues, (Example : VNC/RDP supporting multiple levels of encoding.). With OpenFlow the scenario changes. OpenFlow allows us to design Networks for Applications and allows the Application provider (VDCloud Provider in our case) to adaptively manage network resources in such a way that application performance is not affected. This can be achieved through centralized and automated control over the network.

With this we arrive at our problem which we are trying to solve in our thesis which is “to use OpenFlow for Resource Placement in Virtual Desktop Clouds by featuring a URB for

- **Provisioning of Non-IP traffic flows between thin client sites and data centers efficiently.**
- **Path selection and load-balancing of virtual desktop flows to improve performance of interactive applications and video playback, and to cope with cross traffic in network.**
- **Automated Management and Centralized Control of the Network.”**

### 3.2 Solution Approach

In this paper, we present a novel, intelligent resource placement framework that uses human-and-network awareness (i.e., user group profiles and network health measurements) for thinclient based VD delivery in VDCs. The framework leverages principles of software-defined networking and features a URB component that uses special ‘marker packets’ for: (a) “route setup” when handling non-IP traffic within
VLANs between thin-client sites and data centers, (b) “path selection” and “load-balancing” of virtual desktop flows to improve performance of interactive applications and video playback, and to cope with faults such as link-failures or Denial-of-Service Cyber-attacks. URB orchestrates “control plane”, “data plane” and “measurement plane” components to handle corresponding flows within VDCs in a Future Internet infrastructure. We implement our framework in a multi-domain GENI[5] testbed spanning backbone and access networks connecting data center as well as thin-client user sites. Our emulation experimentation leverages OpenFlow programmable networking. The URB framework will consist of an analytical model to estimate this Service Response Times for VD requests when the VD requests arrive in a increasing Poisson rate. In our solution approach we do validate the model using a OpenFlow Network simulated in MiniNet[14] and show how Network provisioning times affect the overall Service Response Times. In Our Solution we also show how the performance of the Application can be improved by doing Adaptive load balancing with OpenFlow when cross traffic exists in the Network. And the last part of the solution includes the VDCloud experiment automation including automated network resource management and Load balancing in case of cross traffic through VDC-Sim toolkit, a cloud simulation tool useful for students and experimenters studying cloud resource allocation models.

### 3.3 VDC Resource Placement Framework Architecture

Figure 3.1 shows our proposed intelligent VDC resource allocation architecture that specifies how the various components interact with each other when handling VD
requests and application flows. The URB module functions are orchestrated by the “measurement engine”, “service engine” and “routing engine” sub-modules via the “control”, and “measurement” planes. When a new VD request arrives and its security token is validated, a resource allocation decision is made by the service engine based on

![Figure 3.1 Intelligent VDC Resource Allocation Architecture](image)

the performance intelligence provided by the measurement engine. Using a secure channel, the service engine provisions the system resources at the preferred data center that can provide the best utility to the VD. The service engine also instructs the routing engine to program the corresponding flow tables or group tables in the intermediate switches using the OpenFlow protocol on a secure channel. Once such a provisioning is
in place, the thin-client and VD application communications involving RDP/PCoIP protocols occur via the “data” plane and do not involve the URB. Such a decoupling of the data plane achieved by leveraging OpenFlow technology for forwarding traffic avoids additional delays while handling VD application flows. It also allows any desired actions or adaptations to be taken in “real-time” at intermediate OpenFlow-capable network devices on packets that match these flows. The measurement engine relies on instrumentation-and measurement frameworks that are open-source as well as proprietary (e.g., VMware Power Tools or LiquidWare Labs Stratosphere UX that provide hypervisor related status measurements in terms of number of active VD connections, CPU and memory measurements). The collected measurements via the measurement plane are analyzed to detect congestion and faults in system and network devices, and such performance intelligence is passed to the service engine along with on-going monitoring information of the various status measurements. The service engine authenticates user’s VD requests using standard mechanisms (e.g., Active Directory or LDAP), and allows them to access their entitled virtual desktops at a data center that provides the highest utility. The decision to provision VD resources at a data center for a VD request is provided by the resource optimization module in the service engine. The decision is based on the information available (through the measurement engine) regarding the resource slack levels obtained from hypervisor APIs, network health measurements, and based on the user group to which the VD request belongs to. For argument sake, we can consider two user groups: Engineering Site, and Distance Learning Site, each having a custom set of applications that use different amounts of
CPU, memory and network bandwidth resources. We can generally assume that the Distance Learning Site uses more video streaming related applications, and hence the profile would indicate relatively higher amounts of network bandwidth provisioning. Similarly, the Engineering site can be generally assumed to use advanced applications (e.g., Matlab, Moldflow) that require relatively higher amounts of CPU and memory provisioning. The VD request is then assigned to a ‘desktop pool’ corresponding to the resource profile for the corresponding user group created by the system provisioning module at the chosen data center.
4.1 Marker Packet Structure

To setup routes in the routing engine, a ‘marker packet’ handler module is used. The ‘marker packet’ concept is our novel contribution in this paper that is needed for orchestrating the workflow between the thin-client, URB and OpenFlow capable switches. The marker packet format as shown in Figure 4.1 is basically a UDP packet with the following header information:

- Length: Marker packet payload length
• Group ID: User group (e.g., Engineering Site, Distance Learning Site) that the VD request belongs to

• Transport Protocol: Remote desktop protocol used by the thin-client

• OpCode: Used to determine the services to be enabled on the packet flow

• Connection Broker IP/MAC Address: IP/MAC address of the connection broker

• Thin-client IP/MAC Address: IP/MAC address of the thin-client

• Server Port: Port number on the server to which the thin client connects

• Client Port: Port number used at the thin-client

4.2 Marker Packet Scheme Workflow

![Diagram of Marker Packet Scheme Workflow](image)

Figure 4.2 Marker Packet Scheme Workflow

27
As shown in Figure 4.2, there is a six-step workflow to setup routes for thin-clients to connect to their provisioned VDs at the chosen data center. We can assume that the CSP has shipped a ‘smart’ thin-client that is pre-configured to connect to the CSP’s VDC infrastructure. Steps 1 and 2 are completed prior to any VD request arrival, and involve the OpenFlow switch joining the network controlled by the OpenFlow controller application running atop for e.g., POX Controller. When a thin-client actually requests a VD in Step 3 by using the IP/MAC address of the connection broker pre-programmed by the CSP in the thin client, it sends a marker packet that is recognized by the edge switch of the OpenFlow network, which then punts it in Step 4 to the URB of the VDC. The marker packet handler in the routing engine is invoked to parse the tuples in the header. Based on the parsing and the service engine instructions, the controller application in Step 5 sets up the client/server flows forwarding decisions from the thin-client location to the chosen data center. It also configures the intermediate network switches with any other service parameters such as actions upon link failure or events that trigger load balancing, flow traffic classification and QoS settings. The service engine then responds back to the thin-client with an updated marker packet that overwrites the IP/MAC address of the connection broker with the IP/MAC address of the entitled VD. In Step 6, the thin-client begins accessing the VD applications through the flows set up in the network by the controller application. Note that, in the event the URB migrates the VD as part of the resource optimization or system load balancing, it ensures that the controller application is consistent by updating the flow tables suitably.
4.3 Experiment Topology Setup and Configuration

We now describe our “VDCloud” experiment slice topology setup and configuration shown in Figure 4.3 that we developed to VDCloud Experiment Slice Setup in GENI[5] to validate our intelligent VDC resource allocation architecture. The VDCloud experiment is part of a multi-domain GENI testbed spanning backbone networks (e.g., Ohio Regional Network - OARnet, Utah Regional Network - UEN, GENI meso-scale backbone - TangoGENI that is operated in part by Internet2/NLR) and access networks (campus networks of e.g., The Ohio State University, University of Utah, University of Wisconsin). Two parallel slices were configured on top of this common infrastructure testbed, one for the VDCloud experiment traffic, and another for the instrumentation and measurement traffic, which provides performance intelligence for resource adaptation decisions in the VDCloud experiment slice. Given that the system platforms are different for the instrumentation and measurement in comparison to the experiment, and to ensure better isolation and manageability, we chose to implement them in separate parallel slices.
There are four major components of our VDCloud experiment slice: (i) data centers, (ii) thin-client sites, (iii) programmable network, and (iv) unified resource broker. For the experiment results presented in this paper, we had a 2 data center configuration comprising of OSU VMLab and Utah Emulab resources with extended VLAN connectivity (Layer 2) on the Internet. The “wide-area ProtoGENI” (WAPG) [6] nodes located at meso-scale campuses (i.e., Stanford, Georgia Tech, Wisconsin, Rutgers) served as thin-client sites. One of the Emulab nodes was used to host our VDCloud experiment controller application. The traffic from the thin-client sites were made to flow within VLANs (Layer 2) through OpenFlow enabled network segments in the
TangoGENI meso-scale backbone [14] network before reaching the data centers. Each of the thin-client sites at the different universities also has a separate interface that allows IP routing (Layer 3) to the data centers. However, the Layer 3 paths as shown in Figure 4.4 comprise of additional router hops and multiple firewalls, and share network bandwidth resources at the edge with other campus core traffic. We configured the thin-clients to access both interactive applications (e.g., Excel file handling with repeatable mouse clicks and keyboard strokes) and video playback (e.g., Windows Media Player file handling for a repeatable video file content) at the data center VDs.

The slice topology has multiple network paths with long and short geographical distances between the thin-client sites and data centers. As a result, the testbed is suitable to investigate the use of OpenFlow for resource placement experiments (e.g., path selection, load balancing) as described in Chapter 6, involving multiple thin-clients connecting to the data centers on diverse paths.

In addition, we can note the use of our ‘marker packets’ sent from the thin-client sites as part of non-IP traffic sessions in the VDCloud experiment slice. We ensured that our OpenFlow controller application implementation in conjunction with the marker packets handler processed the marker packets and related actions (as described in Section 4.2) in the GENI infrastructure routers/switches at the hardware-level, as opposed to at the software-level, where performance can be very slow. Currently, most actions in OpenFlow controller application implementations that pertain to Layer 2 headers, such as
those used in case of our marker packets, can be performed at hardware-level in the GENI infrastructure routers/switches. However, any controller application actions that involve manipulating IP headers are mostly done at the software-level. Further, we remark that - before actual deployment in the testbed, we recreated our slice topology in MiniNet [14] and validated our controller application functionality with simulated traffic.

4.4 Application Performance Measurements

Figure 4.5 shows the performance comparison between the Internet and OpenFlow paths as experienced by the VD applications at VMLab being accessed by thin-clients at the Rutgers site. As stated earlier, the VD applications we chose in our
experiments correspond to interactive applications (e.g., Excel file handling with repeatable mouse clicks and keyboard strokes) and video playback (e.g., Windows Media Player file handling for repeatable video file content). We remark that these two applications are representative of the broad ‘interactive’ and ‘streaming’ user QoE types, respectively.

![Figure 4.5 Paths Performance Comparison](image)

We can observe that the interactive applications consume more bandwidth and in turn take higher task time in the Internet path, in comparison with the corresponding measurements on the OpenFlow path. This suggests higher cross traffic congestion levels on the Internet path that cause more effort on the user side in terms of mouse clicks and keyboard strokes, and consequently higher number of packets being sent and received. In
the case of video playback, we can observe that the video playback consumes more bandwidth, and in turn provides higher video quality in the OpenFlow path, in comparison with the corresponding measurements on the Internet path. This suggests that there is more end-to-end available bandwidth being observed at the media player client on the OpenFlow path that consequently leads to a higher resolution video encoding to be selected that has a higher packet rate. Such a performance intelligence information is available in real-time at the controller application during can be used for path selection that ensures improved VD application performance.

4.5 Discussion

It should be evident that the performance improvements perceived in our limited GENI testbed scale of VD sessions can be extrapolated to obtain much greater benefit in more realistic VDC settings with VD requests on the order of several hundreds. The performance intelligence data related to human-and-network awareness, if collected effectively, can greatly help in defining triggers that invoke OpenFlow actions in intermediate switches that can cope with faults such as link-failures or Denial-of-Service cyber-attacks. Given that Future Internet infrastructures will rely on hypervisor and software-defined networking technologies, our framework can be generally implemented within slices of DaaS offerings serving different user groups. In addition, the path performance characteristics are likely to be more diverse, and thus our optimization will notably improve the scalability of VDCs, and will result in higher profit to CSPs or potentially cheaper price per VD to DaaS users.
Chapter 5 Validation of the Analytical Framework for Service Response Time Estimation

5.1 The Analytical Model

The behavior of our system can be captured by a finite queuing model with four stages of service namely: Resource Usage Estimation, Network Path Provisioning, Data Center Selection and VD placement and provisioning. As shown in Figure 5.1, incoming requests get first queued in a buffer of size K-1 and then get served in N stages with each stage having a different mean service rate, i.e., μ1, μ2, ..., μ4. It is to be noted that incoming requests are served sequentially in 4 stages. A new request enters Stage 1, the Input queue for VD Requests for processing only after the previous request has departed the queuing system, i.e. left Stage 4 Virtual Desktop and Path Provisioning. The execution of all stages is mutually exclusive. That is, if one of the stages is running, no other stage will be running. We assume that incoming requests follow a Poisson arrival λ, and all of the service times are independent and exponentially distributed. The service discipline of requests is FCFS (First Come First Served).

Figure 5.1 Four stages of service in the Analytical Model
Throughout our analysis in this paper, our analytical model assumes the request arrivals are Poisson with fixed request sizes, and the services time are all exponentially distributed. For certain types of network requests, assuming Poisson arrivals is adequate. An analytical solution becomes intractable and not a trivial task when considering non-Poisson arrivals with non-exponentially distributed service times.

The Model can calculate the Service Response Times assuming that the URB can provision VMs in two different ways. If a previously created VM is available for provisioning and matches user requirements this VM will be provisioned for the request by just adding an user profile to this VM. For this, the URB takes 0.9 minutes. If such a VM is not available a new VM will be created and an user profile is added to the VM database and the VM is provisioned. In performing this operation, the URB takes 15.05 mins on an average. For Path provisioning in GENI The URB takes 1.03 minutes. The URB values are calculated based on empirical methods using apache VSL and VMware VDI (hypervisors).

The Results from the Analytical model are plotted in Matlab. Figure 5.2 Aand Figure 5.3 shows the service response times for different VM provisioning schemes for a load of 500. From Figure 5.2, in which service response times are plotted for provisioning VMs by adding a user profile to an existing VM, it can be seen that the difference between the lowest provisioning time and the highest provisioning times is around 2 minutes whereas in Figure 5.4 where service response times are plotted for provisioning VMs by creating a
Figure 5.2 Service Response times for provisioning VMs by adding a user profile in a previously created VM.

new VM and adding a user profile, the difference is less than 0.5 minutes. The reason for this can be attributed to the fact the in the second case the VD request spends more time in the request queue and hence the difference in the provisioning times is less as compared to the first case. Using these values in the model the Service Response times for VD requests arriving at an increasing Poisson rate is calculated and plotted using Matlab.
5.2 Validation of the Analytical Model Using a MiniNet Simulation

The Analytical model assumes the four stages are mutually exclusive. But in reality these four stages may be mutually exclusive. In a scenario where a desktop is allocated from the desktop pool, desktop provisioning and the network path provisioning can happen at the same time. This can reduce the Service Response Times for VD requests considerably. Though the model assumes that the Virtual desktop and Path provisioning components processes only one request for a unit time it may be different in
a real world scenario and the real units can process more units at the given time thereby reducing the Service Response Time even more. To address these issues of the model and to ascertain the contribution of path provisioning component to the total service response time we validate the model against a simulated network with thin client sites and data centers.

Using MiniNet[14] the topology shown in Figure 5.4 is created with a thin client node and a data center node and an OpenFlow network in between. When VD request arrives from the thin client a Marker packet is sent to the OpenFlow controller to notify the request arrival as a part of the Marker Packet scheme which we saw Chapter 4. To simulate this we pump marker packets to the controller using a tool called the MGEN[26]. MGEN is used to generate marker packets at an increasing Poisson rate as assumed in the analytical model. As the controller receives the packets it processes them.

Figure 5.4 MiniNet simulation Topology
and installs flows in the simulated OpenFlow network. The taken to process the VD request is the difference between the times of sending the marker packet and completion of flow installation in the switches. This time difference is calculated for all the marker packets which are sent to represent VD Requests and is plotted.

5.3 Comparison of Analytical and Simulated Results

Here we compare the Service Response Time graphs we plotted for the analytical model using Matlab and the graph we plotted for simulated network using MiniNet.

Figures 5.5 shows the service response times for Data center Virtual Desktop Provisioning for a load of 500 where for a single VD request the SRT is 15.05 minutes. From this graph we can see that the response times grow to a higher value even for a load below 50. Figures 5.6 shows the service response times for Network path provisioning for a load of 500 where for a single VD request the SRT is 1.03 minutes. Even From this graph we can see that the response times grow to a higher value even for a load below 50. This model assumes only request is processed in the system at a time but in a real scenario, the Path provisioning can process a request when the previous request is having
its flow installed in the network and the growth in the SRT can happen only for larger load requests. This fact is confirmed by the graph in Figure 5.7 where we can see the SRT starts increasing only at 30,000 VD Requests. This proves OpenFlow provisioning is fast but presents bottlenecks at Internet-scale.
Figure 5.6 Service Response Times for Network path provisioning for a load of 500

Figure 5.7 Service Response Times for Network path provisioning from MiniNet Simulation
Chapter 6 Network Adaptive Load Balancing

6.1 Need for Load Balancing

Traditional Networks avoid using multiple paths on a physical network to prevent looping of traffic. Hence there exists only one path between two endpoints. Traditional networks lack the flexibility to apply real-time traffic management policies dictated by the client’s operational needs. With Programmable Networks through OpenFlow we provide load balancing service that with which we can give the flexibility to manage VDCloud traffic. VDCloud URB with our OpenFlow controller allows the CSP to configure paths for VDCloud traffic based on real-time conditions and location of Thin Client sites. This empowers you to react to Cross traffic in the Network while never compromising on availability, application performance and operational efficiency.

In chapter 3 we did show how we use Marker Packets for route setup. In this Chapter we show how we can use the same concept to do “path selection” and “load-balancing” of virtual desktop flows to improve performance of interactive applications and video playback, and to cope with faults such as link-failures or Denial-of-Service cyber-attacks.

The slice topology has multiple network paths with long and short geographical distances between the thin-client sites and data centers. As a result, the testbed is suitable to investigate the use of OpenFlow for resource placement experiments (e.g., path
selection, load balancing), involving multiple thin-clients connecting to the data centers on diverse paths. We can see from Figure 6.1 how the testbed can be used to form a congestion point at Atlanta for Clemson thin-client flows, and how this congestion point can be avoided by our load balancing scheme to load balance the flows through the Boston and Washington hops’ path.

6.2 Topology, Experiment and Results.

![Image](image.png)

Figure 6.1 Virtual Desktop Cloud flows in GENI network

Without Load Balancing

To run the experiment for load balancing we reserved the OpenFlow Topology of GENI network with the OpenFlow switches as shown in Figure 6.1. We had the VMLab
data center connected to the GENI OpenFlow network. We also reserved PG46 and PG47 nodes in Clemson to serve the purpose of thin client machines with VMware Smart Thin Client Software running on these machines. Additionally, we reserved WAPG nodes PG48, PG49 in Stanford; PG50 and PG51 in Rutgers to act as cross traffic generators.

From the topology in Figure 6.1, we observe that the path between Stanford and the Rutgers nodes overlap with the path between the thin client node at Clemson and the VMLab Data-Center. This was done to generate cross traffic using high bandwidth to
disrupt the VD traffic between the thin client node and the data center. On initiating the connection from the Thin Client Site at Clemson without load balancing, shows the flow rules that were installed in the OpenFlow switch for the thin client traffic. Once the VD resource (in this case it was the VMLab Data Center) was allocated by the URB component to the client, we ran the entire High Definition wildlife video on Windows to show the Media Player capability. We observed that the video ran smoothly, without any lag or jitter and the Windows GUI was responsive. When we opened other applications in parallel, GUI interactions worked in a smooth manner. The graph in Figure 6.3 shows the average bandwidth consumed by the thin client traffic in the network. Next, we introduced cross traffic in the network by setting the cross traffic levels to 100 in the VDC-Sim Window, without load balancing ran the experiment again. The shortest path chosen by the URB between the thin client node and the data center overlaps with the path between Stanford and the Rutgers WAPG nodes used for cross traffic generation. The OpenFlow Controller installed the flow rules for the cross traffic in the OpenFlow network devices. Once these flows were installed, the Iperf client and server at Stanford and Rutgers started to communicate with traffic of 100Mbps bandwidth respectively. The paths of VDCloud Flow and the cross-traffic flow converge at Atlanta and both the flows follow the same path till Washington as shown in the Figure. The cross traffic disrupted the thin client traffic on the same path and made the thin client node PG47 at Clemson to freeze, and its GUI became unresponsive. The path used by both these flows is shown in 6.1 with the congestion path highlighted. Figure 6.4 shows the system characteristics - the flow tables in the OpenFlow switches with new entries for the cross traffic flows, the
diminishing video quality at the thin client site and the average bandwidth usage statistics as collected at the end points for the thin client i.e. the WAPG node PG47 at Clemson and the end point of the cross traffic i.e. PG50 at Rutgers. We also observed from the Utility graph that the Net Utility went down when we introduced cross-traffic into the network without load balancing. We enabled load balancing at our URB kept the cross traffic as 100 and ran the experiment again. With load balancing activated by the URB on the network and an alternate path was determined by our load balancer enabled OpenFlow controller in the multipath GENI OpenFlow core for the cross traffic generation. The new path is represented in Figure 6.2 and does not overlap with the path used by the thin client flow. The new path was communicated to the OpenFlow network controller and the controller installed the flow rules on the network devices for the path that was to be followed by the cross traffic. Once the path was rerouted, the cross traffic packets that have been queued at the ports of network devices shared between the two traffic flows were flushed out, the thin client became responsive again, and the video resumed. The GUI interaction was smooth and was similar to the working scenario when there was no cross traffic. Figure 6.5 shows the Flow tables with the new flow rules, and the average bandwidth usage statistics collected at the end points of the two flows.
Figure 6.3 VDC-Sim Experiment with Load Balancer off and Cross Traffic off

Figure 6.4 VDC-Sim Experiment with Load Balancer off and Cross Traffic on
Figure 6.5 VDC-Sim Experiment with Load Balancer on and Cross Traffic on

<table>
<thead>
<tr>
<th>OpenFlow Switch</th>
<th>Client</th>
<th>In Port</th>
<th>Out Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATLANTA</td>
<td>PG46</td>
<td>20</td>
<td>52</td>
</tr>
<tr>
<td>ATLANTA</td>
<td>PG47</td>
<td>20</td>
<td>52</td>
</tr>
<tr>
<td>SUNNW</td>
<td>PG48</td>
<td>50</td>
<td>52</td>
</tr>
<tr>
<td>SUNNW</td>
<td>PG49</td>
<td>50</td>
<td>52</td>
</tr>
</tbody>
</table>

Video runs smooth, GUI applications are responsive

Bandwidth Consumed (Mbytes/s)

Application: 4.45
Cross-Traffic: 14.8
Chapter 7 Automation Using VDC-Sim

In this Chapter we introduce a novel simulator called the VDC Sim tool which is designed to test and simulate our U-RAM[3] schemes for visualizing and comparing the Net Utility results of the VDC system under various schemes. The tool simulates a scalable, distributed VDC system in two modes namely "Run Simulation" and "Run Experiment". In "Run Simulation", researchers can plug in their resource allocation schemes and simulate the Service Response Times and Net Utility of the VDC system under varying loads, faults and cross traffic. "Run Experiment” allows researchers to experiment their simulation results on a real network test bed with network cross traffic generation and load balancing. The tool provides control knobs for researchers to fine tune their network allocation policies and maximize network utility. The tool connects to the Global Environment for Network Innovations (GENI) OpenFlow network [8] to establish the SDN for our experiments. To the best of our knowledge, our work is the first to propose a simulator tool targeted towards simulating VDC systems. Our tool is unique in seamlessly integrating both simulation and experiment environments to achieve the optimal Net Utility of VDC systems. The tool includes an intelligent VDC component called the "Unified Resource Broker”(URB) module developed by our research group in [3] which performs the optimization and defragmentation decisions across the VDC system. The tool comes with a easy to setup simple intuitive User Interface for
researchers to provide load, faults and cross-traffic inputs and analyze the results graphically.

7.1 VDC-Sim Modules for Running Experiments

As discussed earlier, one of the major goals of the VDC-Sim is to provide utility measurements of perceived user QoE in defining the Net Utility of the VDC system. Service Response Times of VD requests should be captured in order to analyze and benchmark the overall performance of the system using several provisioning and placement schemes. VDC-Sim should provide visual representations of the system states for varying random VD requests, loads and cross-traffic on the network. The simulator should be scalable in provisioning VD requests and perform repeatable simulations and experiments for diverse input parameters.

For running VDCloud experiments in GENI consists of the following components a) Configuration modules for Wide Area ProtoGENI (WAPG)[6] thin client nodes on GENI to send VD requests and generate cross traffic on the GENI network b) OpenFlow controller on the GENI for “programming” the OpenFlow switches on the network c) Cost Aware Utility Maximal Resource Allocation Algorithm d) Network load balancing and VD provisioning on the data center. The Service Response Times and Net Utility calculated in the "Run Simulation" mode is fed as input parameters for the "Run Experiment" mode to analyze the VDC system performance on a real network test bed subjected to cross traffic generation and network latencies. The new Net Utility and Service Response Times are measured in the Run Experiment mode. If the values are
unsatisfactory in comparison to the Run Simulation mode, these values are looped back to the VDC application in the Run Simulation mode and the simulation is started all over again. The iterative architecture helps researchers in consolidating the global optimum Net Utility of the VDC system by incorporating both simulation and experiment environments for evaluating resource allocation schemes. The main component of the simulator is the URB module which forms the backbone of the entire VDC system and drives the VDC application. The URB is responsible for handling all optimization and routing decisions across the VDC system. The URB evaluates the utility functions of the provisioning and placement schemes through the U-RAM[3] module to determine the server side utility. Once the VDs are provisioned, the URB monitors and measures the interactive response times (a.k.a. timeliness) of the VD’s through the VDBench[2] toolkit “and” adapts the resource allocations to maximize the user QoE. The URB also comprises the Defragmentation module to defrag the resources resulted from allocating VD requests opportunistically to maximize user QoE. Defragmentation of resources increases the number of VD requests served and hence reduces the operational costs of the CSPs. The URB monitors network health in case of network degradation due to cross traffic and link failures. The URB communicates the network state to the OpenFlow controller on the GENI by virtue of marker packets. The OpenFlow controller is responsible for initializing the routing tables of VD network paths and perform load balancing and packet rerouting in case of network bottleneck as notified by the URB in the Run Experiment mode. The OpenFlow controller provides control knobs for
researchers to program and control the network routes for VD traffic to maximize network utility of their resource allocation schemes.

7.2 VDC-Sim User Interface

The tool’s User Interface (UI) is as shown in Figure 7.1. The UI is built using Matlab. The provisioning and placement schemes to be simulated are selected as part of the Cloud Broker panel (frame a of Figure 7.1). The input parameters include loads, faults and cross traffic values can be set through the Workload Generator panel (frame b of Figure 7.1). The status panel(frame d of Figure 7.1 displays the current steps executed...
by the simulator in Run Simulation or Run Experiment mode. The 2D Plot graph of the VD requests points and the data centers on the UI gives a holistic understanding of placement of VDs across the data centers (frame e of Figure 7.1). This helps us interpret how different placement schemes place VD requests across data centers to maximize Net Utility. The color code associated with the VD request points represents the profile type of the VD user. In our work, we have assumed VD requests can arise from three kinds of user profiles – a student in campus computer lab, a distance learning user or a scientist in Engineering Site, each having specific resource requirements. The UI also plots the Net-Utility graph for the past five runs (frame f of Figure 7.1). The last panel consists of the Service Response Times plotted against increasing loads on the VDC system to measure the system’s behavior in servicing VD requests (frame g of Figure 7.1).

7.3 VDC-Sim Experiment Automation Workflow

The workflow for Run Experiment is as follows: a) Thin client WAPG nodes on the GENI are reserved and configured for triggering VD requests and cross traffic generation b) The OpenFlow controller is instantiated with the initial flow tables for OpenFlow switches comprising the VDC network. c) Once the VD requests are initiated from the thin client nodes, the Run Experiment triggers the URB component to allocate network paths on the GENI and resources on the VDC data centers using the U RAM module and based on the utility values obtained in the Run Simulation mode. d) The URB communicates the routing information to the OpenFlow controller through marker packets. e) The controller programs the OpenFlow switches to route the VD data packets
between the data centers and the thin client and vice versa. f) The URB then allocates resources on the selected data center in order to provision the VD. g) The URB monitors the user QoE through the VDBench[2] toolkit. In case the interactive response time of VD applications between the VD client and data center goes below a certain threshold benchmarked by the VDBench[2], it tries to accommodate the resource allocations on the data center while maintaining the Net Utility of the system. h) In case of network cross traffic or down links, the URB detects the network degradation and switches the VD path by communicating the control information to the OpenFlow
controller. The URB thus performs load balancing on the network and maximizes the Net Utility of the VDC system.
Chapter 8 A Case Study: Gridftp With OpenFlow

In this thesis we have shown how OpenFlow can be leveraged for network resource placement for broker based cloud applications. But parts of the OpenFlow framework can be used for other applications which are bandwidth intensive, and the current networking model becomes a bottleneck for the application performance. As we discussed in Chapter 3 we are attempting to design networks for applications deviating from the traditional way if creating application for networks.

When using OpenFlow to allocate network resources for any cloud application the OpenFlow controller should be informed of the endpoints of the network. Using this information the controller can install the flows in the OpenFlow switches before applications in these endpoints start communicating. To give this information to the controller we can use the “Marker packet scheme” described in Chapter 4. When Application end point initiates the connection it sends the marker packet to the controller.

One of the main bottle necks of networks today is underutilization of available bandwidth. Networks today are not designed to completely utilize the bandwidth available in the physical topology. But with OpenFlow we can overcome this by providing better path selection and load balancing as described in chapter 6, thereby improving the application performance in this chapter we show how we can do this using GridFTP[25] application as a case study.
8.1 Introduction to GridFTP.

GridFTP[25] is a high-performance, secure, reliable data transfer protocol optimized for high-bandwidth wide-area networks. The GridFTP protocol is based on FTP, the highly-popular Internet file transfer protocol. It includes a set of protocol features and extensions defined already in IETF RFCs as well as a few additional features to meet requirements from current data grid projects.

![GridFTP Transfers](image)

Figure 8.1 GridFTP Transfers

GridFTP uses basic Grid security on both control (command) and data channels as shown in figure 8.1. Features include multiple data channels for parallel transfers, partial file transfers, third-party transfers, and more. GridFTP can be used to move files (especially large files) across a network efficiently and reliably. These files may include the executable required for an application or data to be consumed or returned by an
application. Higher level services, such as data replication services, could be built on top of GridFTP.

The aim of GridFTP is to provide a more reliable and high performance file transfer for Grid computing applications. This is necessary because of the increased demands of transmitting data in Grid computing - it is frequently necessary to transmit very large files, and this needs to be done fast and reliably. This makes GridFTP a bandwidth intensive application and an apt application for our case study.

8.2 OpenFlow experiments with GridFTP.

Figure 8.2 GridFTP in GENI without Load balancing

Figure 8.2 shows the GENI test setup we used to test GridFTP over our multipath OpenFlow WAN network. We run two GridFTP servers at PG48 and PG49 in Stanford and two GridFTP clients at PG50 and PG51 at Rutgers. We transfer a file of size 1GB from both the clients to the servers. Before the transfer we send Marker packets to the controller for path provisioning in the OpenFlow network. Now we start the GridFTP file transfer in from client1 at PG49 to server1 at PG51. The graph in Figure 8.3 shows the...
instantaneous bandwidth usage for this flow at 1). At 2) in Figure 8.3 we Initiate the flow from the client2 at PG48 to the server2 at PG50. The paths used by both the flows are shown in Figure 8.2. We can see that paths for both flows overlap with each other. This affects the data transfer rates for both the GridFTP flows which is evident from part 3) of Figure 8.3 GridFTP in GENI Data transfer Rates.

Figure 8.3. Now we trigger our load balancer at the controller. The controller installs a different path for the data flow from client2 to server2. This causes the data transfer to go down for some time at part 4) of the figure 8.3. It takes almost 2 minutes for the controller to install new paths and the data flow to be restarted. The different paths for the flows are shown in Figure 8.4. When the flow from client 2 to server is restarted both the flows follow different paths though their end points are in same broadcast domains. This increases the data transfer rate for both the flows which is evident from part 5) of Figure 8.3.
Thus from this case study it's evident that our OpenFlow framework can be used in many other bandwidth-intensive applications thereby improving the application performance.
Chapter 9 Conclusion

The emergence of OpenFlow technology that is built upon software-defined networking principles and supported in several vendor switches provides new capabilities for programmable control of routing and other services (e.g., virtual machine migration) for application flows. In this paper, we leveraged these capabilities in a novel and intelligent resource placement framework that uses human-and-network awareness (i.e., user group profiles and network health measurements) for thin-client based VD delivery in VDCs. We showed how special OpenFlow header ‘marker packets’ can be handled for implementing: (a) “route setup” when handling non-IP traffic within VLANs between thin-client sites and data centers, (b) “path selection” and “load-balancing” of virtual desktop flows to improve performance of interactive applications and video playback, and to cope with faults such as link-failures or Denial-of-Service cyber-attacks. From our framework implementation and empirical results in a multi-domain GENI testbed spanning backbone and access networks, we demonstrated the importance of scheduling regulated measurements that can be used for intelligent resource placement decisions. Further, we showed the feasibility and benefits of using OpenFlow controller applications for path selection and load balancing between thin-client sites and data centers in VDCs. Our planned future work involves developing a metascheduler capability in OpenFlow controller applications to derive performance intelligence without affecting application
traffic. We also are developing more comprehensive set of VD application performance metrics that can be used in OpenFlow-based load balancing triggers in VDCs.

In this thesis, we presented a framework that leverages OpenFlow for resource allocation on a large scale VDC infrastructure and measure the utility of system under different resource schemes with varying faults, loads and cross traffic. We also observed the performance of the system on a real network by running experiments with and without load balancing on the network and observed its impact on the user QoE. Future work for the simulation tool can include integrating the tool with VLAN technologies like Virtual Private LAN service [15], Overlay Transport Virtualization [12], etc.

![Figure 9.1 Virtual Tenancy Using OpenFlow](image)

Virtual Tenancy using LAN extension technologies shown in Figure 9.1 has been popular with data center networking for connecting geographically distributed data centers in which multiple tenants share the same network core to extend their LANs. There are several proprietary protocols available in the market that extend LAN across
WANs, do traffic isolation with scheduling such that traffic from one virtual network does not affect another. However, there is no open solution available that provides services like guaranteed bandwidth for a virtual network along with traffic isolation. We are trying to extend our method of leveraging OpenFlow for Virtual Desktop Clouds to accommodate Tenant network virtualization and integrate it with the VDC Sim tool. The marker packet shall be modified with new fields TNV ID, along with a Type of Service (TOS) field bit, which sets the IP TOS bits in the packet. A data packet from a user’s machine is identified by a unique six tuple combination as described in[5]. Now this data packet containing the marker packet within the header is sent to the network controller so that the controller can classify packets belonging to several tenant networks and create a rule to set the TOS bit as per the OpenFlow 1.10 specification. In the network devices, values of maximum bandwidth according to SLA can be configured for the packets with particular TOS Value using traffic shaping. The marker packet, TOS bits, the rule to set the TOS bits and Traffic Shaping for packets with particular TOS values provides a way for effective traffic isolation and queuing.

The thesis serves as a model for network virtualization using OpenFlow and stands as a significant proof for CSPs to capitalize OpenFlow for network resource allocation. Our experiments gives results that demonstrate the need for different network resource allocation models that emphasizes proper path selection and load balancing and enable CSPs to substantially improve user QoE and Cloud scalability.
Appendix A: OpenFlow Slice Setup Instructions

1) Creating slices and slivers in GENI.
   • For information on Creating Slice, Slivers and configuring Omni the tool used to reserve resources in GENI please refer to http://groups.geni.net/geni/wiki/HelloGENI

2) Reserving the OpenFlow Controller Node.
   • In order to use the OpenFlow switches we need to reserve a Utah Emulab PC with Ubuntu Linux node to be used as OpenFlow Controller.
   • We can reserve an Emulab PC using an rspec similar to pc472.rspec which reserves pc472 in Utah Emulab.

```xml
<rspec type="request"
   xsi:schemaLocation="http://www.geni.net/resources/rspec/3 http://www.geni.net/resources/rspec/3/request.xsd"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://www.geni.net/resources/rspec/3">
   <node client_id="pc472"
        component_manager_id="urn:publicid:IDN+emulab.net+authority+cm"
        component_id="urn:publicid:IDN+emulab.net+node+pc472"
        component_name="pc472"
        exclusive="true">
      <sliver_type name="raw-pc">
        <disk_image name="urn:publicid:IDN+emulab.net+image+emulab-ops:UBUNTU12-64-STD-UPD"/>
      </sliver_type>
    </node>
</rspec>
```
• After reserving the node, we install a OpenFlow Controller of our own choice to be used with our Virtual Desktop Cloud. We use the POX controller for our experiment.

3) Configuring the controller node

• Installing POX Controller
  
  o Requirements: POX requires Python 2.7. In practice, it also mostly runs with Python 2.6, but nobody is presently trying to push or support this.

  o Python runs on Windows, Mac OS, and Linux. A lot of the development happens on Mac OS, so that almost always works. Occasionally things will break for the other OSes. Usually that's noticed on Linux fairly quickly (especially for big problems), and Windows generally slowly. This is just because nobody reports the problems.

  o POX can be used with the "standard" Python interpreter (CPython), but also supports PyPy (see below).

• Getting the Code
  
  o The best way to work with POX is as a git repository. You can also grab it as a tarball or zipball, but source control is generally a good thing.
POX is hosted on github. If you intend to make modifications to POX itself, you might consider making your own github fork of it from the POX repository page. If you just want to grab it quickly to run or play around with, you can simply create a local clone:

$ git clone http://github.com/noxrepo/pox
$ cd pox

Selecting a Branch / Version

The POX repository has multiple branches. Specifically, it has at least some release branches and at least one active branch. The default branch (what you get if you just do the above commands) will be the most recent release branch. Release branches may get minor updates, but are no longer being actively developed. On the other hand, active branches are being actively developed. In theory, release branches should be somewhat more stable (by which we mean that if you have something working, we aren't going to break it). On the other hand, active branches will contain improvements (bug fixes, new features, etc.). Whether you should base your work on one or the other depends on your needs. One thing that may factor into your decision is that you'll probably get better support on the mailing list if you're using an
active branch (lots of answers start with "upgrade to the active branch").

- As of this writing, the active branch is named betta. To use the betta branch, you simply check it out after cloning the repository:
  ```
  ~$ git clone http://github.com/noxrepo/pox
  ~$ cd pox
  ~/pox$ git checkout betta
  ```

PyPy Support

- While it's not as heavily tested as the normal Python interpreter, it's a goal of POX to run well on the PyPy Python runtime. There are two advantages of this. First, PyPy is generally quite a bit faster than CPython. Secondly, it's very easily portable – you can easily package up POX and PyPy in a single tarball and have them ready to run.

- You can, of course, download, install, and invoke PyPy in the usual way. On Mac OS and Linux, however, POX also supports a really simple method: Download the latest PyPy tarball for your OS, and decompress it into a folder named "pypy" alongside pox.py. Then just run pox.py as usual (./pox.py), and it should use PyPy instead of CPython.
• Invoking POX

POX is invoked by running pox.py. POX itself has a couple of optional commandline arguments than can be used at the start of the commandline:

<table>
<thead>
<tr>
<th>option</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>--verbose</td>
<td>Display extra information (especially useful for debugging startup problems)</td>
</tr>
<tr>
<td>--no-cli</td>
<td>Do not start an interactive shell (No longer applies as of betta)</td>
</tr>
<tr>
<td>--no-openflow</td>
<td>Do not automatically start listening for OpenFlow connections</td>
</tr>
</tbody>
</table>

But running POX by itself doesn't do much – POX functionality is provided by components (POX comes with a handful of components, but POX's target audience is really people who want to be developing their own). Components are specified on the commandline following any of the POX options above. An example of a POX component is forwarding.l2_learning. This component makes OpenFlow switches operate kind of like L2 learning switches. To run this component, you simply name it on the command line following any POX options:

```
./pox.py --no-cli forwarding.l2_learning
```

You can specify multiple components on the command line. Not all components work well together, but some do. Indeed, some components depend on other
components, so you may need to specify multiple components. For example, you can run POX’s web server component along with l2_learning:

```
./pox.py --no-cli forwarding.l2_learning web.webcore
```

Some components take arguments themselves. These follow the component name and (like POX arguments) begin with two dashes. For example, l2_learning has a "transparent" mode where switches will even forward packets that are usually dropped (such as LLDP messages), and the web server's port number can be changed from the default (8000) to an arbitrary port. For example:

```
./pox.py --no-cli forwarding.l2_learning --transparent web.webcore --port=8888
```

4) Configuring OpenFlow Slivers

- The GENI OpenFlow Topology can be found at http://groups.geni.net/geni/wiki/TangoGENI.

- For our experiment we shall need to reserve the ports in all the switches in the GENI OpenFlow Network.

- We shall use VLAN 3716 for our experiments as shown in the topology.

- We have reserved IP subnet 10.42.116.x/24 for our setup and all our nodes and VMs in Data center use this IP subnet
• We shall create three slivers, from internet2, NLR and BBN parts of the network reserving all ports from all switches for VLAN 3716.

5) Configuring a WAPG for smart Thin Client
• We use the WAPG nodes from all over the topology as thin clients to connect to our Virtual Desktop Cloud Datacenter. For this experiment we use nodes PG47 and PG46 from Clemson and PG48 and PG49 from Stanford. We can use the rspecs to reserve these nodes.
• Once these nodes are reserved these has to be configured for the OpenFlow VLANs first and an interface with IP address in the subnet that we have reserved.
• Follow these steps to configure the host
  o Add these lines in the end of /etc/network/interfaces, edit this file as sudo:
    
    auto eth3.3716
    
    iface eth3.3716 inet static
    
    address 10.42.130.211
    
    netmask 255.255.255.0
    
    mtu 1500
    
    o Install the vlan package
    sudo apt-get install vlan
    
    o Reboot the machine
    sudo init 6 && exit
• Once VLAN is configured we install the VMware smart thin client along with its requisites such as the rdesktop software. Once the thin client is ready we can connect to our data center with the user credentials and connect to the data center.

5) Configuring the Cross traffic nodes.
• We used nodes from Rutgers and Stanford to generate cross traffic. We reserve WAPG in these places and install the Iperf Client and server at stand and Rutgers respective and generate cross traffic.
Bibliography


PlanetLab Aggregate in GENI - http://groups.geni.net/geni/wiki/PlanetLab.

TangoGENI Operations - http://groups.geni.net/geni/wiki/TangoGENI.


OTV IETF Submission - https://datatracker.ietf.org/doc/draft-hasmit-otv


MiniNet Software-defined Network Emulation Platform -
http://yuba.stanford.edu/foswiki/bin/view/OpenFlow/Mininet


Network Simulator 2 http://www.isi.edu/nsnam/ns/

OPNET IT Guru http://www.opnet.com/

CLOUDS Lab, University of Melbourne “CloudSim Toolkit 2.1.1”, -
http://www.cloudbus.org/cloudsim


