A new Framework to enable rapid innovation in Cloud Datacenter through a SDN approach.

José Teixeira

A thesis submitted to the University of Minho in the subject of Informatics, for the degree of Master of Science, under scientific supervision of Prof. Stefano Giordano and Prof. Alexandre Santos

University of Minho

School of Engineering

Department of Informatics

September, 2013

Acknowledgments

I would like...

I also...

Abstract

In the last years, the widespread of Cloud computing as the main paradigm to deliver a large plethora of virtualized services significantly increased the complexity of Datacenters management and raised new performance issues for the intra-Datacenter network. Providing heterogeneous services and satisfying users' experience is really challenging for Cloud service providers, since system (IT resources) and network administration functions are definitely separated.

As the Software Defined Networking (SDN) approach seems to be a promising way to address innovation in Datacenters, the thesis presents a new framework that allows to develop and test new OpenFlow–based controllers for Cloud Datacenters. More specifically, the framework enhances both Mininet (a well–known SDN emulator) and POX (a Openflow controller written in python), with all the extensions necessary to experiment novel control and management strategies of IT and network resources.

... talk about obtained results and conclusions(not finished yet, complete when you finish everything)

Keywords: Datacenter, Cloud, SDN, OpenFlow.

Contents

List of Acronyms

DC	Datacenter
IP	Internet Protocol
IT	Information Technology
OF	Openflow
QoS	Quality of Service
SDN	Software Defined Networking
VM	Virtual Machine
VMM	Virtual Machine Manager
	Add as needed...

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

A Cloud DC consists of virtualized resources that are dynamically allocated, in a seamless and automatic way, to a plethora of heterogeneous applications. In Cloud DCs, services are no more tightly bounded to physical servers, as occurred in traditional DCs, but are provided by Virtual Machines that can migrate from a physical server to another increasing both scalability and reliability. Software virtualization technologies allow a better usage of DC resources; DC management, however, becomes much more difficult, due to the strict separation between systems (*i.e.*, server, VMs and virtual switches) and network (*i.e.*, physical switches) administration.

Moreover, new issues arise, such as isolation and connectivity of VMs. Services performance may suffer from the fragmentation of resources as well as the rigidity and the constraints imposed by the intra-DC network architecture (usually a multilayer 2-tier or 3-tier fat-tree composed of Edge, Aggregation and Core switches [2]). Therefore, Cloud service providers (*e.g.*, [3]) ask for a next generation of intra-DC networks meeting the following features: 1) efficiency, *i.e.*, high server utilization; 2) agility, *i.e.*, fast network response to server/VMs provisioning; 3) scalability, *i.e.*, consolidation and migration of VMs based on applications' requirements; 4) simplicity, *i.e.*, performing all those tasks easily [4].

In this scenario, a recent approach to programmable networks (*i.e.*, Software-Defined Networking) seems to be a promising way to satisfy DC network requirements [5]. Unlike the classic approach where network devices forward traffic according to the adjacent devices, SDN is a new

network paradigm that decouples routing decisions (control plane) from the traffic forwarding (data plane). This routing decisions are made by a programmable centralized intelligence called controller that helps make this architecture more dynamic, automated and manageable.

Following the SDN–based architecture the most deployed SDN protocol is OpenFlow [6] [7], and it is the open standard protocol to communicate and control OF-compliant network devices. Openflow allows a controller to install into OF–compliant network devices forwarding rules which are defined by the administrator/network engineer and match specific traffic flows.

Since SDN allows to re-define and re-configure network functionalities, the basic idea is to introduce an SDN-cloud-DC controller that enables a more efficient, agile, scalable and simple use of both VMs and network resources. Nevertheless, before deploying the novel architectural solutions, huge test campaigns must be performed in experimental environments reproducing a real DC. To this aim, a novel framework is introduced that allows to develop and assess novel SDN-Cloud-DC controllers, and to compare the performance of control and management strategies jointly considering both IT and network resources [8].

TODO:should describe better openflow and SDN

1.2 Motivation and objectives

Although SDN came as a solution to fulfill the network requirements of the DCs, the only point of interaction with the IT resources is the generated traffic. By definition SDN does not go further, but if there could be a controller that manages both IT and network resources, all the information could be shared easily and both of them could greatly benefit: the network could start to anticipate IT actions and adapt itself to have higher performance, more redundancy, etc; the IT because the resources could be better managed so that the network, not only stops being the bottleneck, but actually helps the IT complete the tasks faster and without affecting adjacent resources.

When developing an Openflow controller, the administrator/network engineer goals are to implement the desired behaviour and to test it (making sure it suits the requirements). The currently available controllers already provide some abstraction, which varies according to the type of programming language, but they are still too low level to allow rapid innovation. Following the implementation, tests campaigns must be performed and for it a controlled environment should

be set. Although Openflow allows the use of slices of the real network for testing purposes, it is more convenient to use an emulator since the DC size can be dynamic, different scenarios can be easily produced and it only needs a single computer – Mininet is such an emulator. Despite its flexible API, Mininet does not provide any type of traffic generator and is not DC–oriented: poor topology generation regarding DCs; no support for VMs;

A whole framework composed by a modified OF controller that allows the access to both IT and network resources through an easy-to-use but full featured API, and a testing environment that communicates with it to provide a real DC emulation is the the main objective. With this is is expected to endue the administrator/network engineer with all the tools needed to quickly develop, test and deploy VM and network management strategies into a DC.

1.3 Thesis layout

This thesis is structured into five chapters: the present Chapter 1 is a brief introduction of the proposed work, its motivation and objectives; the second is the state of art, it addresses the currently available solutions relating innovation in DCs, OF controllers and VM allocation and migration algorithms; the third one fully describes the framework, its evolution, extensions and how it can be used; in the forth chapter is presented the framework validation and performance tests; and in the last chapter are made conclusions about the developed work, as well as suggestions for future work.

Chapter 2

State of art

2.1 Available solutions

A number of research efforts have focused on novel solutions for emulation/simulation of Cloud DCs. The available solutions provide a reference and material to analise and explore the concepts addressed along this thesis. This section presents and overview of them, highlighting their architecture, features and limitations.

2.1.1 CloudSim

Calheiros et al. [9] proposed a Java-based platform, called Cloudsim, that allows to estimate cloud servers performance using a workflow model to simulate applications behaviour. By providing a framework for managing most key aspect of a Cloud infrastructure (DC hardware and software, VM placement algorithm, Applications for VM, Storage access, Bandwidth provisioning) and by taking into consideration factors as energy-aware computational resources and costs, it helps to identify possible bottlenecks and improve overall efficiency.

Regarding the network aspect of Clousim, Garg et al. [10] extended such a system with both a new intra–DC network topology generator and a flow–based approach for collecting the value of network latency. However, in such a simulator, networks are considered only to introduce delay, therefore it is not possible to calculate other parameters (*e.g.*, Jitter). A SDN extension for Cloudsim as already been thought, Kumar et al. [11], but it still just an architecture design,

meaning it has not been implemented yet.

Although it allows to predict how the management strategies will behave, as a simulator, it does not allow to run real applications and deploying the tested management logic in a real environment still requires everything to be developed.

2.1.2 FPGA Emulation

Ellithorpe et al. [12] proposed, a FPGA emulation platform that allows to emulate up-to 256 network nodes on a single chip.

> ”Our basic approach to emulation involves constructing a model of the target architecture by composing simplified hardware models of key datacenter building blocks, including switches, routers, links, and servers. Since models in our system are implemented in programmable hardware, designers have full control over emulated buffer sizes, line rates, topologies, and many other network properties.”
>
> Ellithorpe et al. [12]

This platform also allows the emulation of full SPARC v8 ISA compatible processor, which along with full system control provides a greater system visibility. However, hardware programming skills might be a requirement and the cost of a single board is approximately 2, 000 dollars making this solution less attractive than ones based on just open–source software.

2.1.3 Meridian

Following the new shiny SDN paradigm, Banikazemi et al. [1] proposed Meridian, an SDN–based controller framework for cloud services in real environments.

As shown in figure 2.1, the architecture is divided into three main layers: Network abstractions and API, where the network information can be accessed and manipulated (*e.g.* access controlling policies, prioritizing traffic); Network Orchestration, translates the command provided by the API into physical network commands and orchestrates them for more complex operations.

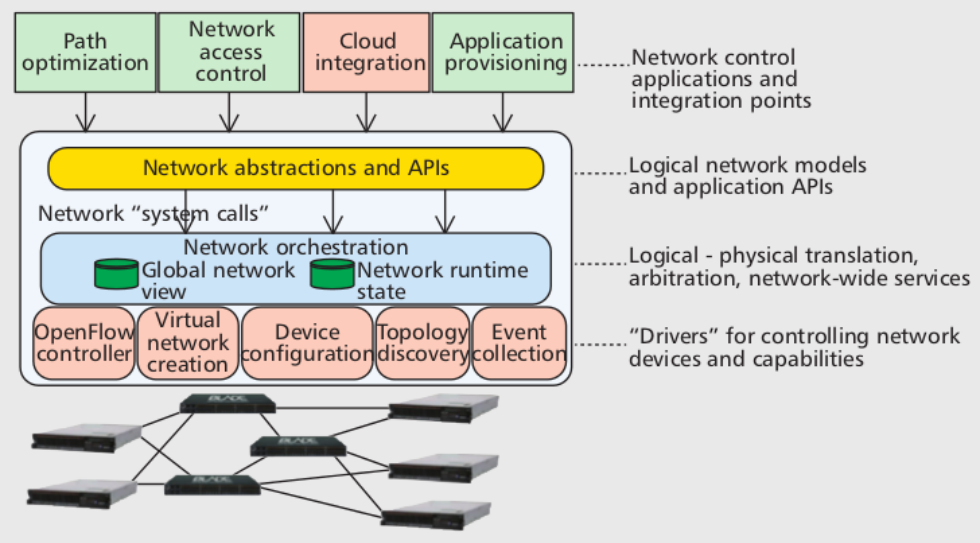

Figure 2.1: Meridian SDN cloud networking platform architecture (Banikazemi et al. [1])

it also reveals the network topology and its variations; finally the ”drivers” layer is an interface for underlying the network devices so several network devices and tools can be used.

Generally, this platform allows to create and manage different kind of logical network topologies and use their information for providing a greater control of the DC. But as it works on top of a cloud Iaas platform (i.e., Openstack [13], IBM SmartCloud Provisioning [14]), it is limited to their management strategies and is only useful if one already has this type of infrastructure. Not having a testing environment is also a downside since the normal operation can be compromised and also alter the testing results.

2.1.4 ICanCloud, GreenCloud and GroudSim

Other well–known open–source cloud simulators are ICancloud [15], GreenCloud [16] and GroudSim [17], but in none of them SDN features are available.

2.1.5 Mininet

> "Mininet is a network emulator which creates a network of virtual hosts, switches, controllers, and links. Mininet hosts run standard Linux network software, and its switches support OpenFlow for highly flexible custom routing and Software-Defined Networking."
>
> Mininet [18]

As a network emulator for SDN systems, mininet can generate OF compliant networks that connect to real controllers without the need of hardware resources. Such features derives from the use of Open vSwitch and enables the assessment of the operation of an OF controller before its deployment in a real environment.

It also provides tools for automatically generating topologies, however, as they can be basic, an API is available to reproduce any type of topology and experiments. Mininet hosts behave just like real hosts, can run any program as long as it does not depend on non linux kernels, and can send packets through emulated interfaces. But as they share the same host file system and PID space, a special attention is required when killing/running programs.

Despite its flexibility, Mininet lacks of a complete set of tools that easily allow to emulate the behaviour of a cloud DC, thus raising the following questions:

- How to easily generate and configure typical DC topologies?
- How to simulate VMs allocation requests?
- How to emulate the inter and in/out DC traffic?

2.2 Openflow Controllers

2.3 Virtualization Platforms

Chapter 3

The Framework

3.1 Requirements

Provide the user with a full package for the development and test of DC SDN Controller was one of the main purposes of the framework. Aiming for such goal, but without discarding the deployment in a real DC, a single software platform was designed and developed. Because the requirements change according to the controller being in the development or the deployment phase, so should the platform by creating and environment that best suits each of them.

Development & Testing Phase

Encourage a rapid development is one of the main requirements since it promotes innovation in the cloud DC. It must be simple and fast to develop the desired logic, which can be achieved by providing easy access to information and management of the network and servers. More specifically, automatic topology detection (and changes in it) associated with a high level API for accessing and managing switch's and server's information and statistics.

When testing, the framework should provide an automatic way of generating the VM requests and the traffic associated to each request (for testing the VM allocation and the network behaviour). The traffic generator should also correctly represent the DC traffic profiles. Allowing an easy access outside the controller for the statistics is also important, so it is possible to analyze the logic effects on the DC.

Deployment Phase

For the deployment, the framework should be easy to configure and monitor, and no extra effort should be made for the framework to run on the real DC (it should adapt automatically). There should also be an intuitive way to make manual VM requests, so clients can generate and manage their own VMs.

3.2 Chosen technologies

Openflow Controller: POX

Being POX a python derivative of the NOX controller, which was developed by the same people who developed the Openflow protocol, adopting it would be a surplus since there is a higher chance it will continue to support Openflow, and that the new versions/features are available as soon as possible. Besides, being a high level (comparing to C and C++), object and event oriented programming language, helps to create the abstraction level required for agile development and turn the controller more interactive.

Datacenter Emulator: Mininet

Mininet comes recommended in the Openflow tutorials as the platform for testing the OF compliant controllers. It also provides an API in python for the development of custom made topologies and specific experiments, which along with the capacity that the virtualized hosts have of running almost any program, makes it a powerful platform.

Virtualization platform: XCP 1.6 (Xen Cloud Platform)

As a free and opensource platform though for the cloud, XCP bring all the features belonging to Xen, plus it comes with ready-to-install images, making it simpler to install and configure. Having multiple interaction option is also an attractive feature, but having a Xen python API was

decisive since having the possibility to write all the code in one programming language helps keeping the platform consistent.

3.3 Framework architecture

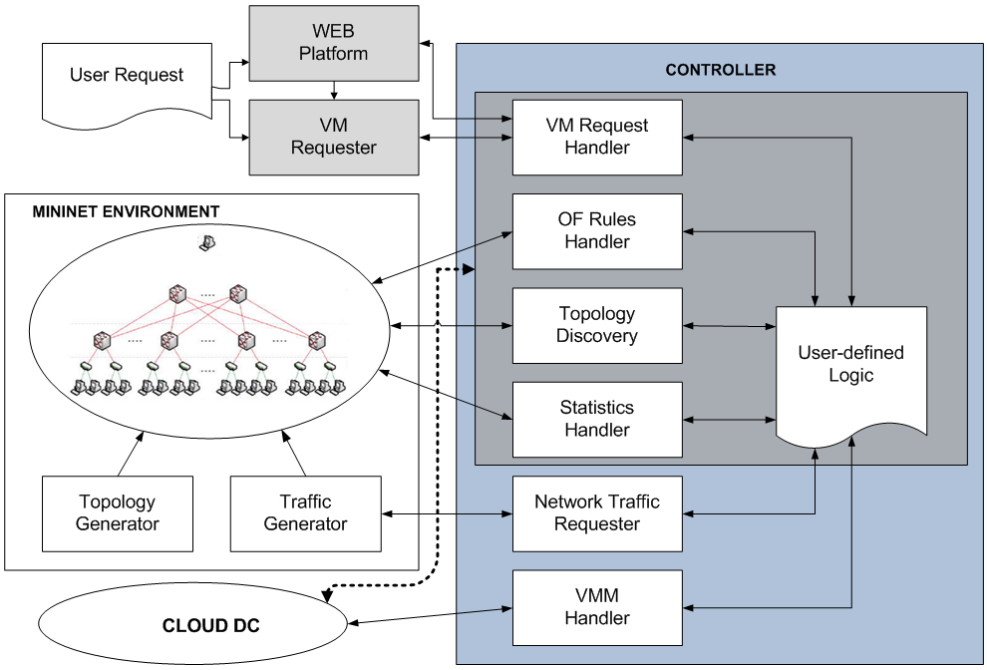

Figure 3.1: Framework Architecture

The framework architecture, shown in figure 3.1, gives an overview of the its modules and their interaction. The framework is divided into two main parts: the mininet environment - an extended version of mininet; and the controller - a modified, improved version of POX;

The mininet environment is made to be only used when testing the controller. It is composed by the mininet platform with two extra modules that explore its API. One of them is the *Topology Generator*, which allows to easily create a fat tree or tree DC topologies. The other one is the *Traffic Generator* that allows to correctly simulate the allocation of VM into each server by generating traffic from that server to the exterior of the DC. It also allows to simulate inter VM communication.

As for the controller, it automatically manages the modules in order to only use the ones that are needed for each phase (testing or deployment). Depending on it, the controller will interact with the mininet environment or the Cloud DC, which represent the real Cloud DC infrastructure. In the figure 3.1, in the controller part, it can be seen a darker area which corresponds to the modules that are used in both phases. These modules are:

- VM Request Handler – Connects with the Web platform and/or the VM requester, and processes the VM requests;
- OF Rules Handler – Manages and keeps track of the installation/removal of the OF rules from the switches;
- Topology Discover – Manages all the information regarding the switches, links, servers and its detection;
- Statistics Handler – Collects statistics about the switches and links, can be periodical or manual;
- User-defined Logic – Space for the administrator/network engineer to develop the desired DC management logic;

Regarding the other controller modules: the *Network Traffic Requester* which is only used when performing tests, tells the mininet environment how much, when, from where and where to, the traffic should be generated; and the *VMM Handler* which is only active when the controller is in a real environment, communicates with the hypervisor to perform all the operation regarding the VMs (allocate, deallocate, upgrade, etc).

The *WEB platform* and the *VM Requester* where created for making VM requests. While the first one is a platform where DC clients can request VMs that will be processed by the controller and later allocated by the hypervisor (oriented for the deployment phase), the *VM Requester* is an automatic full configurable VM request generator powered by random variables (oriented for the testing phase).

An important feature that was taken into consideration when designing the framework's architecture is that all the modules are independent from each other, and they can be changed, removed or added in order to fulfill all the user requirements.

3.4 Framework modules: Mininet Environment

3.4.1 Topology Generator

3.4.2 Traffic Generator

Describe each module, it's functionalities, limitations, how it can be used/improved (improved if the user wants to add new features)

- Talk generally about the traffic generator
- Talk about the one's we tried (pros and cons)

3.5 Framework modules: Controller

Describe each module, it's functionalities, limitations, how it can be used/improved (improved if the user wants to add new features)

3.5.1 Topology Discovery

3.5.2 OF Rules Handler

3.5.3 Statistics Handler

3.5.4 VM Request Handler

3.5.5 VMM - Virtual Machines Manager

3.5.6 Network Traffic Requester

3.5.7 POX Modules

3.5.8 User Defined Logic

3.6 Framework modules: Web Platform

Describe each module, it's functionalities, limitations, how it can be used/improved (improved if the user wants to add new features)

3.7 Framework modules: VM Requester

Describe each module, it's functionalities, limitations, how it can be used/improved (improved if the user wants to add new features)

3.8 Using the framework

3.8.1 Emulator

Describe how to use the framework (emulation part) and how to access the API..

3.8.2 Real Environment

-Figura com o esquema da configuraÃ§Ã£o Describe what changes in the real environment (the modules that are disabled and the ones that need to be enabled)

Real environment tests

- Talk about the environment which was setup
  - Chosen hypervisor
  - Talk about Xen api and the alternative solution (ssh each server and run a script to clone the vm)
  - OpenVswitches VS NetFPGA problems
  -

3.9 Framework extensions

3.9.1 Enabling QoS

State of art: QoS in SDN

QoS in the framework

3.9.2 Enabling Virtual Machine migration

State of art: Virtual Machine migration

Virtual Machine migration in the framework

Usecase

Chapter 4

Validation and tests

Usually test and validation of the proposed solution ...

4.1 Framework Validation

- Show how Bf goes against WF with server driven algorithm (show server occupation)
- Show how Bf goes against WF with network driven algorithm (show network occupation) (although the behaviour is similar is allow to say that net algorithm may use switch statistics)

4.2 Performance Evaluation

TODO: CHANGE THIS FOR NOT BEING IN THE SECOND PERSON OF THE PLURAL
TODO: PUT IMAGES IN THE RIGHT PLACE (ALONG WITH THE TEXT)

We evaluated the actual performance of the proposed framework through a variety of experiments using a PC equipped with an Intel i5 3GHz and 8GB of DD3 RAM (*i.e.*, from now on we will call it Host-PC). The first tests have been carried out to inspect the impact of the amount of generated traffic, the DC topology size and the number of outside hosts on the host link ratio. Firstly, we generate a static topology (*i.e*, 2 outside hosts, 2 core switches, 4 aggregation switches, 8 edge switches, 8 hosts), then we started measuring the host link ratio increasing the generated traffic per host.

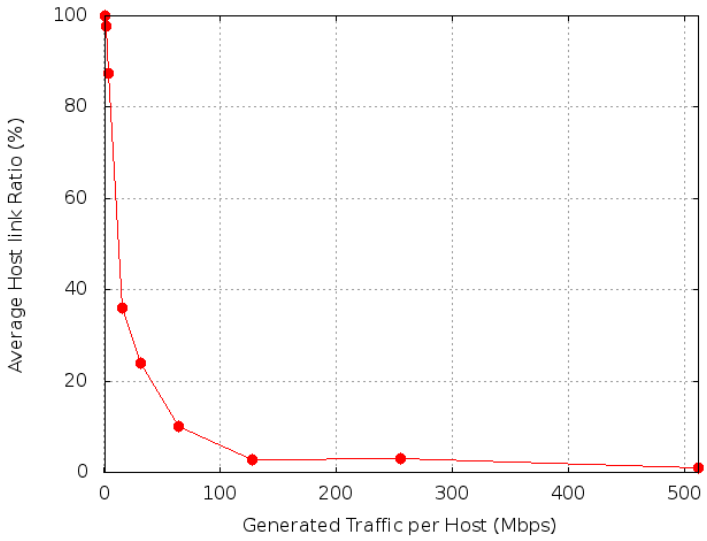

Figure 4.1: Average Host link Ratio vs per Host Generated Traffic

As shown in figure 4.1 we were able to generate up to few Mbps of traffic per host. Then the host link ratio decreases as the generated traffic grows. We point out that such limitation does not affect any kind of DC performance tests made with our framework, because we can scale the link speed as much as we want during the DC initialization phase, reaching every time 100% of host link ratio.

In order to test the impact of the DC topology size on the host link ratio we kept the amount of the generated aggregated traffic constant while exponentially increasing the number of switches

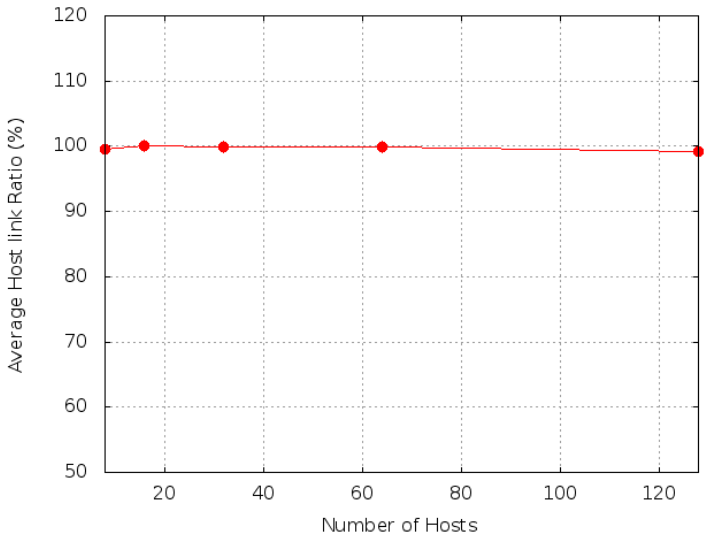

Figure 4.2: Average Host Link Ratio vs number of Hosts

and hosts. We started from the previous test topology.

On DC initialization phase, we set the link speed in order to fully saturate the host links.

The results in figure 4.2 show that regardless of the hosts number, the host link ratio remains constant. This means that as long as the total amount of generated traffic per host and the links speed can guarantee the link saturation, the system can scale indefinitely, being the only limits the Mininet itself, or the controller. Finally we investigated the relationship between the number of hosts connected to just one outside host and the average link ratio. Figure 4.3 shows that a maximum of 8 hosts can be managed by just one outside host (*i.e.*, the host link speed is set in order to have a link saturation).

Such a result gives to the user an important constraint that should be used during the DC configuration phase. We point out that this limitation is native of the Mininet environment and it is not due to our framework. The second tests have been carried out to inspect the impact of both the amount of generated traffic and the DC topology size on the amount of memory the Host-PC needs. Figure 4.4 shows that the memory utilization does not depend on the amount of generated traffic for each host. On the other hand, as shown in figure 4.5, as the topology size grows, the memory usage also grows in the same proportion, which allows to conclude that it scales linearly.

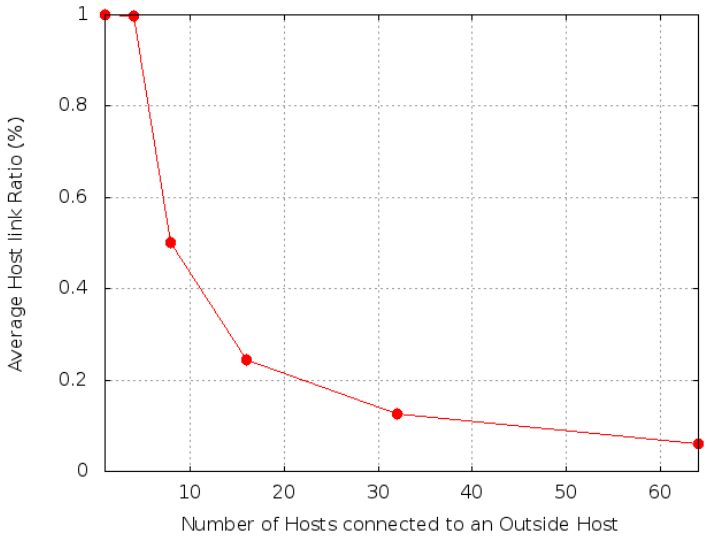

Figure 4.3: Average Host Link Ratio vs number of Hosts per Outside Host

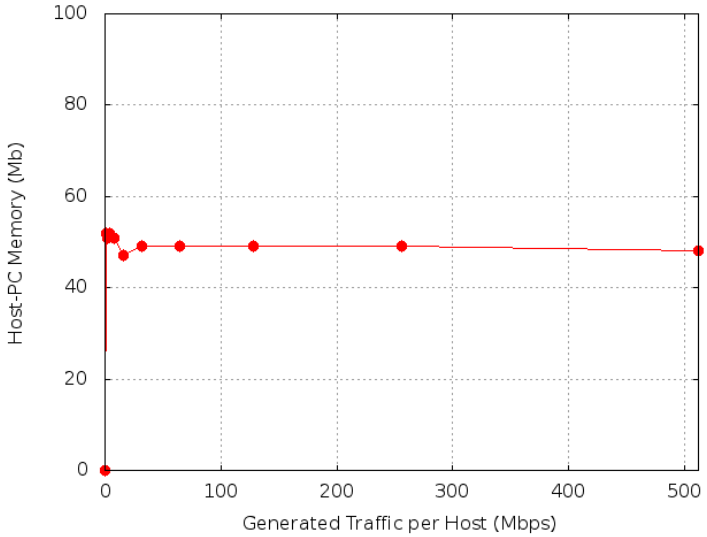

Figure 4.4: Host-PC Memory Utilization vs per Host Traffic Generated

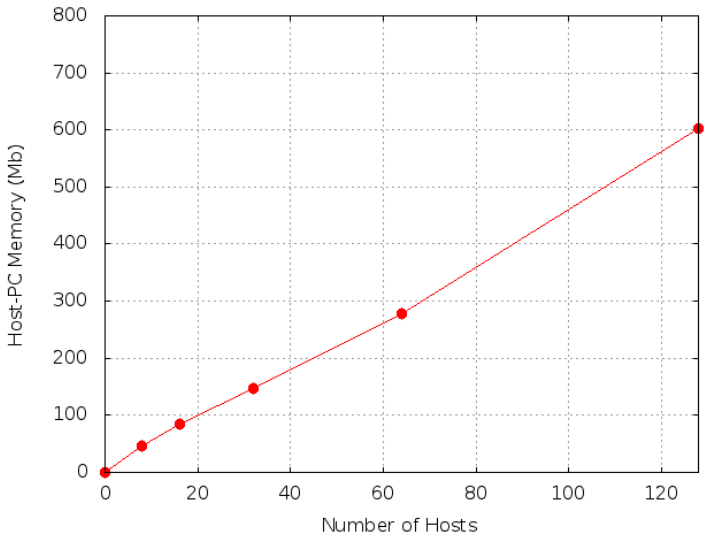

Figure 4.5: Host-PC Memory Utilization vs number of Hosts

Chapter 5

Conclusions

This chapter provides ...

5.1 Main contributions

5.2 Future work

Appendix A

Name of the Appendix

Bibliography

[1] M. Banikazemi, D. Olshefski, A. Shaikh, J. Tracey, and G. Wang. Meridian: An sdn platform for cloud network services. *Communications Magazine*, 2013.

[2] K. Bilal, S.U. Khan, J. Kolodziej, L. Zhang, K. Hayat, S.A. Madani, N. Min-Allah, L. Wang, and D. Chen. A comparative study of data center network architectures. In *European Conference on Modelling and Simulation*, 2012.

[3] AWS Home Page. *http://aws.amazon.com.*

[4] O. Baldonado, SDN, OpenFlow, and next-generation data center networks. *http://www.eetimes.com/design/embedded/43715 43/SDN–OpenFlow–and-next-generation-data-center-networks.*

[5] J. Oltsik and B. Laliberte. Ibm and nec bring sdn/openflow to enterprise data center networks.

[6] OpenFlow Home Page. *http://www.openflow.org.*

[7] Open Networking Foundation Home Page. *https://www.opennetworking.org.*

[8] D. Adami, B. Martini, G. Antichi, S. Giordano, M. Gharbaoui, and P. Castoldi. Effective resource control strategies using openflow in cloud data center. In *International Symposium on Integrated Network Management*. IEEE/IFIP, 2013.

[9] N. Calheiros, R. Ranjan, A. Beloglazov, C.A.F. De Rose, and R. Buyya. Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, 2011.

[10] K. Garg and R. Buyya. Networkcloudsim: Modelling parallel applications in cloud simulations. In *Internation Conference on Utility and Cloud Computing*. IEEE, 2011.

[11] A. Kumar, N. Siddhartha, A. Soni, and K. Dubey. http://search.iiit.ac.in/uploads/cloudsim.pdf.

[12] J.D. Ellithorpe, Z. Tan, and R.H. Katz. Internet-in-a-box: Emulating datacenter network architectures using fpga's. In *Design Automation Conference*. ACM/IEEE, 2009.

[13] Openstack Home Page. *http://www.openstack.org.*

[14] IBM Smart Cloud Provisioning Home Page. *http://www-01.ibm.com/software/tivoli/products /smartcloud-provisioning.*

[15] A. Nunez, J.L. Vázquez-Poletti, A.C. Caminero, G.G. Castañé, J. Carretero, and I.M. Llorente. icancloud: A flexible and scalable cloud infrastructure simulator. *Journal of Grid Computing*, 2012.

[16] D. Kliazovich, P. Bouvry, and S.U. Khan. Greencloud: A packet-level simulator of energy-aware cloud computing data centers. In *Globecom*. IEEE, 2010.

[17] S. Ostermann, K. Plankensteiner, R. Prodan, and T. Fahringer. Groudsim: An event-based simulation framework for computational grids and clouds. *Euro-Par 2010 Parallel Processing Workshops*, 2011.

[18] Mininet Home Page. *https://mininet.github.com.*

[19] NS3. *http://www.nsnam.org.*

[20] POX Home Page. *http://ww.noxrepo.org.*

[21] J. Matia, E. Jacob, D. Sanchez, and Y. Demchenko. An openflow based network virtualization framework for the cloud. In *International Conference on Coud Computing Technology and Science*. IEEE, 2011.

[22] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *NSDI*. ACM, 2010.

[23] R. Raghavendra, J. Lobo, and K.W. Lee. Dynamic graph query primitives for sdn-based cloud network management. In *HotSDN*. ACM, 2012.

[24] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker. Applying nox to the datacenter. In *HotNets*. ACM, 2009.

[25] A. Pescapè A. Dainotti, A. Botta. A tool for the generation of realistic network workload for emerging networking scenarios. *Computer Networks (Elsevier), 2012, Volume 56, Issue 15, pp 3531-3547*, 2012.