Datacenter in a box: test your SDN cloud-datacenter controller at home

José Teixeira[‡], Gianni Antichi[*], Davide Adami[†],
Alessio Del Chiaro[†] Stefano Giordano[†], Alexandre Santos[‡]
[*]*Computer Lab, University of Cambridge*
[†]*Dept. of Information Engineering, University of Pisa*
[‡]*Engineering school, University of Minho*

***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 paper presents a new framework that allows to develop and test new OpenFlow–based controllers for Cloud Datacenters. More specifically, our framework enhances both Mininet (a well–known SDN emulator) and POX (a network controller written in python), with all the extensions necessary to experiment novel control and management strategies of IT and network resources.**

***Keywords*-Datacenter; SDN; OpenFlow; Cloud;**

I. Introduction

A Cloud Datacenter (DC) basically 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 (VMs) 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[5]). Therefore, Cloud service providers (*e.g.*,[3]) ask for a next generation of intra-DC networks meeting the following requirements: 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[13]. A recent approach to programmable networks (*i.e.*, Software-Defined Networking – SDN) seems to be a promising way to satisfy DC network requirements[14]. SDN–based architecture decouples control and data planes: the most deployed SDN protocol is OpenFlow (OF)[16][15], which allows to set into OF–compliant switches forwarding rules established by a centralized intelligence called controller. Since SDN allows to re-define and re-configure network functionalities (possibly up to the physical layer), 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, we introduce a novel framework that enhances both Mininet[11] and POX[19] with all the software modules necessary to emulate an SDN-based intra-DC network, such as DC topology discovery, network traffic generation, etc. Specifically designed for DC environments, our framework 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[2]. It is worth highlighting that the developed software modules may be ported in a real controller without changes, as our framework inherits such basic feature from Mininet. The rest of the paper is organized as follows: section II provides a short survey of related works, whereas section III details the architecture and the functionalities of our framework. Section IV presents an use case while section V evaluates the performance of the framework. Finally, section VI concludes the paper with some final remarks.

II. Related Work

A number of research efforts recently focused on novel solutions for emulation/simulation of Cloud DCs. Calheiros et al.[6] proposed a Java-based platform, called Cloudsim, that allows to estimate cloud servers performance using a workflow model to simulate applications behaviour. Then, Garg et al.[8] 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 important parameters (*e.g.*, Jitter). Other well–known open–source cloud simulators are[12][10] and[18], but in none of them (even in Cloudsim) SDN features are available.

Ellithorpe et al.[7] proposed, an FPGA emulation platform that allows to emulate up to 256 network nodes on a single chip. However, the cost of a single board is approximately $2,000$ dollars making this solution less attractive than one based on open–source software.

Following the new shiny SDN paradigm, Banikazemi et al.[4] proposed Meridian, an SDN–based controller framework for cloud services in real environments: such a platform allows to create and manage different kind of logical network topologies, but it works on top of a cloud Iaas platform (*i.e.*, Openstack[17], IBM Smart Cloud Provisioning[9]) while our solution is a flexible, standalone software that could even run in a virtualized environment.

III. Datacenter in a box: our framework

Providing the user with a full package for the development and test of DC SDN controllers is one of the main purposes of the framework. In order to achieve such goal, we designed and developed a new software environment consisting of two main components that allow the emulation of DC topologies and DC oriented controller, respectively. As regards the first component, the starting point was Mininet, a network emulator for SDN systems, which provides an API to reproduce any kind of topology without the need of hardware resources. Therefore, Mininet allows to validate the operation of an OF controller before its deployment in a real environment. However, 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?

On the other hand, concerning the second component, the starting point was POX, a full featured python controller for OF switches, with ready-to-use modules. These modules are helpful when it comes to make a controller, as they provide useful abstractions. However, POX API is too low level for a user that aims to implement a new DC controller, which prevents the rapid development of the logic thought by the user. To fill this gap, the controller available in the framework includes all the abstraction levels needed for building a DC oriented controller while still being dynamic.

A. Framework overview

Figure 1 shows an overview of our framework. All the main modules are independent, allowing not only to directly use the controller in a real DC, but also to change/add modules in order to fulfil all the user's requirements.

Within the controller, the User Defined Logic can be easily implemented to obtain the desired DC behaviour through the API provided by the framework controller modules.

Using Topology Generator the desired Mininet DC topology can be obtained. As for the traffic generated the user can either use one of the provided generators or create the support module for the favourite one.

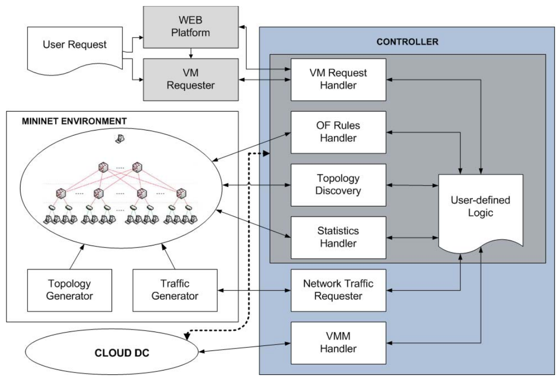

Figure 1. Our framework

The Web Server Platform has two main functionalities: 1) to be used as a monitoring tool for the whole DC; 2) to provide the end-user with a GUI for requesting VMs.

The Virtual Machines Manager (VMM) handler allows to interface the controller with the VMM of a real-world Cloud DC infrastructure (i.e., physical servers, VMs). For now this module only supports XEN hypervisor, but other VMMs (*e.g.*, VMware hypervisor) can be easily supported and does not require heavy development work by the user.

All the modules are directly portable to the real environment with the obvious exception of Mininet (and all the controller modules designed to interact with it), since it is the one emulating the DC topology. This means that after developing and testing the desired DC behaviour through our framework, the user can deploy the controller in his own DC without making any changes.

B. Discovering the DC topology

Conventional intra-DC networks are multi-layer hierarchical infrastructures with thousands of low cost commodity switches as network nodes[5]. Understanding the actual DC skeleton is the first task that every controller must perform regardless of the implemented functionalities. The standard OF protocol provides the controller with the capability of finding out the switches that actually are in the network, but does not give any information regarding their actual position in the hierarchy. For this reason, we enhanced the POX *host tracker* and *discovery* modules to better understand the actual DC topology. Such a feature allows the framework user to focus only on the development of the controller functionalities (*i.e.*, VM allocation policies, DC routing) without taking into account all the preliminary operations that do not add anything new to the DC management. The automatic discovery of link capacity is a challenge yet to be faced and it will be a matter of future works.

C. VM requester

Cloud computing provides the end user with an abstraction of hardware and software resources according to different service delivery models (IaaS, PaaS, SaaS). For this reason, emulating a Cloud DC needs an agent able to act as an external user that requires resources (*i.e.*, VMs allocation). Such a feature is provided by the VM requester, a software daemon able to interact with the SDN Cloud DC controller through a network–socket. Fully configurable and flexible, the requester asks to the SDN Cloud DC controller the allocation of VMs with given requirements (*i.e.* CPU, RAM, hard disk size and bandwidth) and lifetime. The lifetime represents the amount of time the VM will remain allocated. When the lifetime expires, the framework automatically takes care of both removing the corresponding rules in the DC switches and stopping the traffic generation to/from such VM. We point out that the VM allocation does not really take place, but it is simulated in the selected Mininet–instantiated virtual host. Choosing the VM allocation request rate allows to study the behaviour of the system in different scenarios. At the time of writing, we modeled such a inter–request time interval as a Poisson random variable with a given α parameter, but we will add the possibility to choice other distributions.

D. Traffic generation

Emulating traffic sources is a key point. Reproducing both VM-to-VM and VM to out-of-DC data exchange is necessary to create an environment as close as possible to real scenarios. Out-of-DC traffic can enter in the DC and reach a host (*i.e.*, a virtual host instantiated within Mininet) and vice–versa. Some hosts represent the world outside the DC (from now on we will call them **outside hosts**), enabling in this way the emulation of data exchange from DC to Internet while others are actually the DC hosts. We point out that such traffic emulation must be fully customizable in order to allow the user's experiments: while traffic modeling is out of the scope of this work, giving the user tools that allow to easily create different traffic profiles is a main issue. For this reason we opted for D-ITG[1], a distributed traffic generator that allows to generate a large spectrum of network traffic profiles(*e.g.*, Poisson distribution, DNS, VoIP, etc..). Application-specific traffic profiles can be defined, inserting their statistical parameters in a configuration file (*i.e.*, traffic shape, transport protocol, transmission rate, traffic duration, etc..). Moreover, during the configuration phase, the user can specify how frequently these applications run into the DC. Every time a new VM is successfully allocated (*i.e.*, the SDN Cloud DC controller chooses the host to allocate the VM and sets up the rules on the OF switches) at least a new bidirectional traffic instance starts between one outside host and the one that hosts the new VM. We point out that the number of instances and the type of traffic strictly depend on the application chosen in the configuration phase. As for the internal DC communications, when a inter-VMs communication request arrives, the proper rules are installed and traffic between them is generated.

E. Mininet DC Configuration

Flexibility is one of the key features in emulation systems. For this reason, we made our framework fully customizable through a configuration file that is used in the initialization phase. Such a file enables the user to define the DC topology as well as the traffic characteristics. Choosing the DC architecture (*i.e.*, 2–tier fat tree, 3–tier fat tree, etc..), the number of switches (*i.e.*, core, aggregation and edge) and the number of hosts per edge switch leads the user to define the DC skeleton. Providing a simple interface to select even the number of links that connect each switch to the others allows to create different topologies (*i.e.*, simple tree, fat-tree, etc..) while setting the number of outside hosts gives the possibility to connect the DC to the outside world in more than one point. Providing an interface to choose the links bandwidth allows the user to scale such value depending on the computational power of the physical machine where the framework actually runs. In this way, links can be saturated regardless of the CPU power. Choosing the hosts physical resources (*i.e.*, CPU, RAM and disk size) as well as modeling both the inter–VMs allocation request time and VM lifetime random variables give the possibility to create very dynamic environments. Finally, using D-ITG the user can set the per-VM behaviour. It is only required that the user choose a pool of traffic profiles. Once a VM is allocated, one of the them will be randomly selected and will be used to emulate the VM data exchange.

Limitations due to the computational power will be discussed in Performance Evaluation section (§ V).

F. User Defined Logic

The user can insert his own code inside the User Defined Logic module. While all the other modules provide an abstraction level for the DC itself (*i.e.*, VM allocation request, traffic intra DC, etc..), in this one the user can define his own controller functionalities (*i.e.*, VM allocation policies, smart DC routing) by just implementing the algorithms. No limitations in terms of management functionalities are present. Everything that is OF compliant could even be used in the logic that it is able to interact with the others framework modules through some provided APIs.

IV. Use Case: understanding VM allocation policies dynamics

Understanding the impact on the intra-DC network of well–known VM allocation policies represents the first step for finding more and more optimized solutions. Our main concern was to validate our framework analyzing its behaviour under common situations, in order to compare the obtained results with the theoretical ones. For this reason in the *User-Defined Logic* module, we firstly implemented Best Fit (BF) and Worst Fit (WF). The BF algorithm chooses the server with the smallest available resources that suits the requirements. On the other hand WF chooses the one with the most available resources. Therefore, we expected that, as each request comes, using a BF policy, all the VMs were allocated in one single host until it was able to fulfill the requirements. Then a new host was selected, and so on until all the hosts had no more free space. In the second case (*i.e.*, WF policy), the VMs firstly had to be equally spread through all the hosts. We configured the DC topology with 1 outside host, 2 core switches, 4 aggregation switches, 8 edge switches, and 16 hosts (*i.e.*, 2 per edge). We set each host in order to be able to allocate up to 3 VM, for sake of simplicity (and to easily understand the results), and all the requests had the same requirements (*i.e.*, CPU, RAM, disk size and bandwidth). We defined the host link ratio as the amount of per-host traffic received against the link speed set on the DC initialization phase. We also set the DC in order to saturate the host link when three different VMs had been allocated.

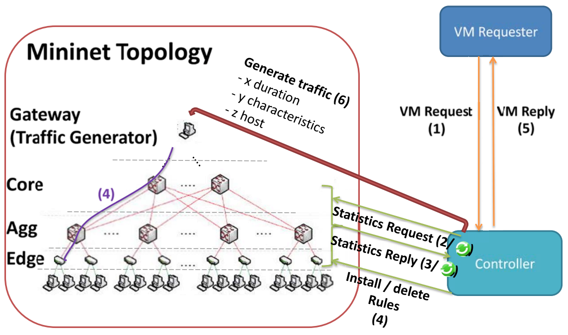

Figure 2. The environment

Figure 2 shows an high–level vision of the proposed environment. Starting from our framework, we only added few lines of code to implement the allocation policy, since the framework provides all the necessary APIs that enable the controller to interact with the VM Requester, Traffic Generator and the DC switches. Every time the controller receives a new VM allocation request (*i.e.*, generated by the VM requester according to the DC configuration), it installs the proper rules in the switches (optionally it can ask for switches statistics – even periodically). Once this process is completed, the controller informs the VM requester about the result of the allocation process and the traffic generation starts.

Figure 3 shows the first host link ratio over the time. Using the BF allocation policy, once a VM has been allocated in a host, all the following VMs are allocated in the same host until no more can be allocated (*e.g.*, useful for energy saving). Having a new VM allocation request per second, after three seconds the first host link reaches the saturation. Using the WF policy instead, VMs firstly are equally spread through all the hosts. In fact, being 16 the DC hosts, and having just 1 request per second, the first host link saturates at the 33–th second.

V. Performance Evaluation

We evaluated the actual performance of the proposed framework through a variety of experiments using a PC equipped with an Intel i5 3 GHz and 8 GB of DD3 RAM

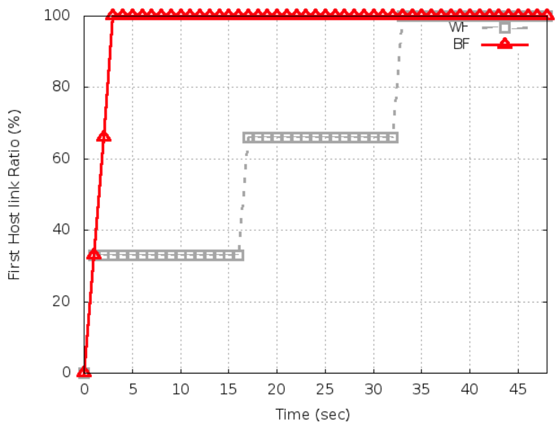

Figure 3. WF vs BF

(*i.e.*, from now on we will call it Host-PC). The first set of tests has been carried out to investigate 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 generated 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 per-host generated traffic . As shown in figure

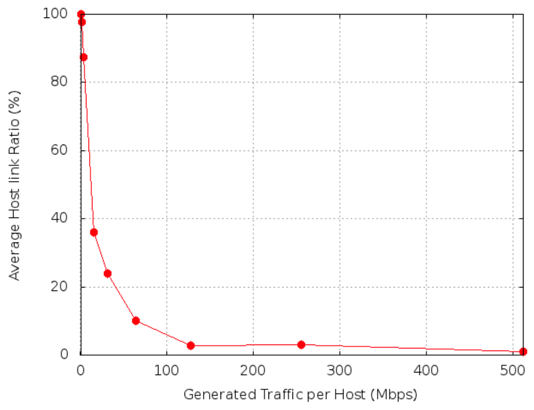

Figure 4. Average Host link Ratio vs per Host Generated Traffic

4, 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 the 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 the number of switches and hosts was exponentially increased. We started from the previous test topology.

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

The results in figure 5 show that regardless of the hosts

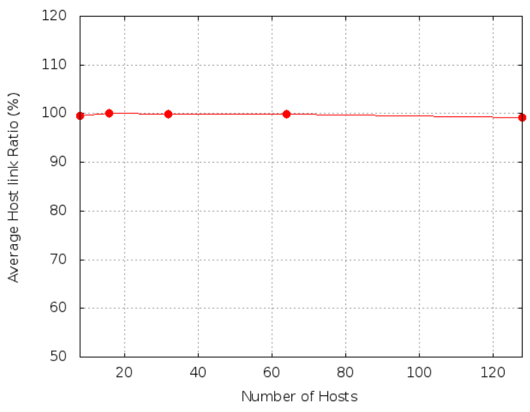

Figure 5. Average Host Link Ratio vs number of Hosts

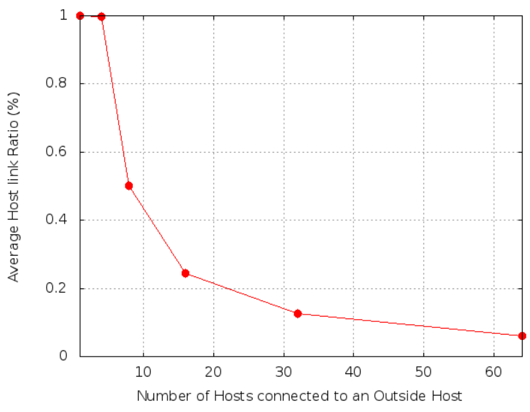

Figure 6. Average Host Link Ratio vs number of Hosts per Outside Host

number, the host link ratio remains constant. This means that as long as the total amount of per-host generated traffic 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 6 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 provides the user with an important constraint to 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 set of tests was carried out to investigate the impact of both the amount of generated traffic and the DC topology size on the amount of memory the Host-PC needs. Figure 7 shows that memory utilization does not depend on the amount of generated traffic for each host. On the other hand, as shown in figure 8, as the topology size grows, the memory usage also grows in the same proportion, which allows to conclude that it scales linearly.

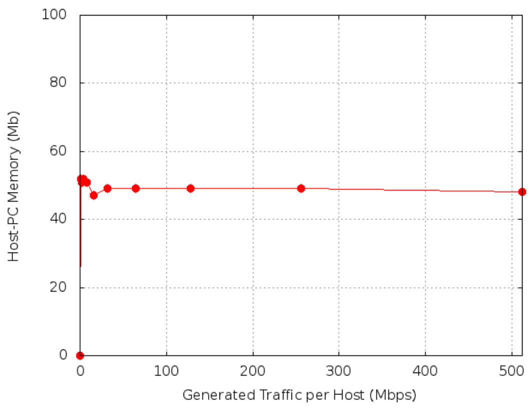

Figure 7. Host-PC Memory Utilization vs per Host Traffic Generated

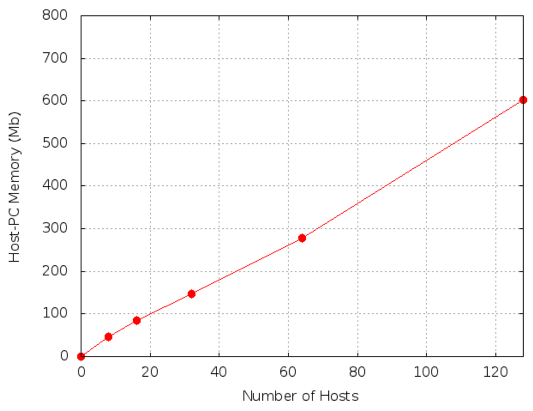

Figure 8. Host-PC Memory Utilization vs number of Hosts

VI. Conclusions

In this paper, we presented a novel SDN Cloud DC framework, built on top of Mininet and POX, that allows the user to evaluate the performance of their SDN Cloud DC controllers. Our framework addresses several issues in testing such controllers by providing some useful APIs (i.e., topology discovery, traffic generation, DC configuration and VM request). This work has been validated showing one use–case where two different well–known VMs scheduling algorithms were implemented. Framework scalability and stability has been also evaluated increasing both the number of emulated hosts and the DC links load. Work is still ongoing. We are planning to insert new features in order to consider VM migration and storage.

Acknowledgment

The authors would like to thank the FP7 EU project OFELIA (n. 258365) for the support of this research activity. This work was partially supported by the Italian Ministry of University and Research funded project GreenNet (contract n. RBFR100QHJ) and by FEDER Funds through the Programa Operacional Fatores de Competitividade - COMPETE and by National Funds through the FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project FCOMP-01-0124-FEDER-022674.

References

[1] A. P. 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.

[2] 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.

[3] AWS Home Page. *http://aws.amazon.com. Last visited: 23-09-2013.*

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

[5] K. Bilal, S. Khan, J. Kolodziej, L. Zhang, K. Hayat, S. 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.

[6] N. Calheiros, R. Ranjan, A. Beloglazov, C. 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.

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

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

[9] IBM Smart Cloud Provisioning Home Page. *http://www-01.ibm.com/software/tivoli/products/smartcloud-provisioning. Last visited: 23-09-2013.*

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

[11] Mininet Home Page. *https://mininet.github.com. Last visited: 23-09-2013.*

[12] A. Nunez, J. Vzquez-Poletti, A. Caminero, G. Casta, J. Carretero, and I. Llorente. icancloud: A flexible and scalable cloud infrastructure simulator. *Journal of Grid Computing*, 2012.

[13] 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. Last visited: 23-09-2013.*

[14] J. Oltsik and B. L. v. .-.-. Laliberte. Ibm and nec bring sdn/openflow to enterprise data center networks.

[15] Open Networking Foundation Home Page. *https://www.opennetworking.org. Last visited: 23-09-2013.*

[16] OpenFlow Home Page. *http://www.openflow.org. Last visited: 23-09-2013.*

[17] Openstack Home Page. *http://www.openstack.org. Last visited: 23-09-2013.*

[18] 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.

[19] POX Home Page. *http://ww.noxrepo.org. Last visited: 23-09-2013.*