NITOS Testbed: A Cloud based Wireless Experimentation Facility

Katerina Pechlivanidou, Kostas Katsalis, Ioannis Igoumenos, Dimitrios Katsaros,
Thanasis Korakis, Leandros Tassiulas
Dept. of Electrical & Computer Engineering, University of Thessaly
Volos, Greece
{kapehliv, kkatsalis, ioigoume, dkatsar, korakis, leandros}@uth.gr

***Abstract*—The NITOS wireless testbed, is one of the main building blocks in the wireless testbed experimentation, offered by the FIRE community, and a core Fed4Fire partner. Its main focus is on wireless accessed technologies and on all layers of the protocol stack. In the rapidly changing technological environment, a unique opportunity is provided to enhance the NITOS testbed with cloud computing experimentation capabilities and increase the heterogeneity and diversity of possible services that can be offered to experimenters/ developers. In this paper we describe the NITOS approach on adopting cloud technologies and SDN capabilities and how we upgrade the meaning of ”delivered service” in testbed experimentation using SOA extensions.**

Keywords—Testbed experimentation, Wireless Networks, Cloud Computing, Big Data

I. Introduction

The main principle of the testbed experimentation is that we must understand the technology if we want to understand the future. Currently there are some clear trends that the cutting edge technology follows like the convergence of heterogeneous wireless access technologies for seamless handooffs and seamless high bandwidth operations, cloud computing, Software Defined Networking(SDN) and Network Function Virtualization (NFV). All aforementioned technologies will provide the backhaul for the Future Internet ecosystem, where already an increasing need for converged technological solutions is caused by the explosion of mobile Internet data traffic.

In order to make a wireless testbed cloud and service enabled the following questions have to be answered: how to use the control framework of a wireless testbed like OMF[1] in parallel with the Openstack cloud OS, how can SDN be used to build virtual networks and slice network resources, which extensions are necessary in both the dataplanes and control/management planes of the testbed, how can Big Data technology be supported and which extensions are required to make software services testing part of the experimentation cycle.

In this paper we describe how this step towards cloud and service enablement is performed in both the data and the control/management planes of the Network Implementation Testbed using Open Source platforms (NITOS testbed)[2]. We describe the methodologies used, the extensions developed and provide preliminary results of our approach. A special reference is made for Big Data and the MapReduce[3] capabilities of NITOS tetsbed through the execution of a Hadoop MapReduce application scenario.

Besides presenting the new services NITOS testbed is capable to publicly ooffer to the FIRE testbed community, this paper can be used to provide the necessary guidelines to other wireless testbeds for ways of entering the cloud computing and software defined networking world. A successful marriage of wireless and cloud technologies paves the way for experimentation that is able to span all the protocol stack, bring network's sophistication to the edge, support diverge needs and validate/verify business models that could only be evaluated when they enter the deployment phase. For example, utilization of cloud infrastructure and cloud technologies at the edge network, supports for testbed experimentation in use cases like CloudNets [4], VM migrations between multi-domain (wireless, optical) environments, IoT with Big Data support, Mobile Cloud Computing and so on.

A. Motivation behind the cloud extensions

In NITOS we are interested in all aspects and services (e.g IaaS, PaaS, NaaS etc) a cloud facility can provide to an experimenter, but due to page size limitations we present the ones we consider as the most important and that are already in (or are close to) the production phase in our testbed. In more detail, we present our extensions made for using IaaS and Openstack technologies, Big Data and (Hadoop's) MapReduce, SDN networking and also our extension on the SOA world using Enterprise Service Bus (ESB) technologies.

IaaS and NaaS: Hardware, Infrastructure and Network as a Service are key enablers in order to maximize the usage efficiency of physical resources while giving the experimenter the potential to rapidly build his own private network, customize his servers, drivers and adapters or evaluate his algorithms, policies, applications and technologies. In subsections II-B and II-C we show how we approached the adoption of cloud technologies while not compromising the freedom of the experimenter's control on the testbed resources.

Big Data: An efficient data storage mechanism is required to meet the scalability constraints that arise in large-scale experimentation. Traditional data storage approaches are not responsive to manage huge data amount arriving from thousands sources of information mostly in an unpredictable and bursty way. Future Internet urges for new data infrastructures that can seamlessly and efficiently meet the requirements of demanding applications [5]. Such solutions can be sought in the realm of cloud storage technologies. For example, in IoT use cases, the generated data sets are gaining tremendous size

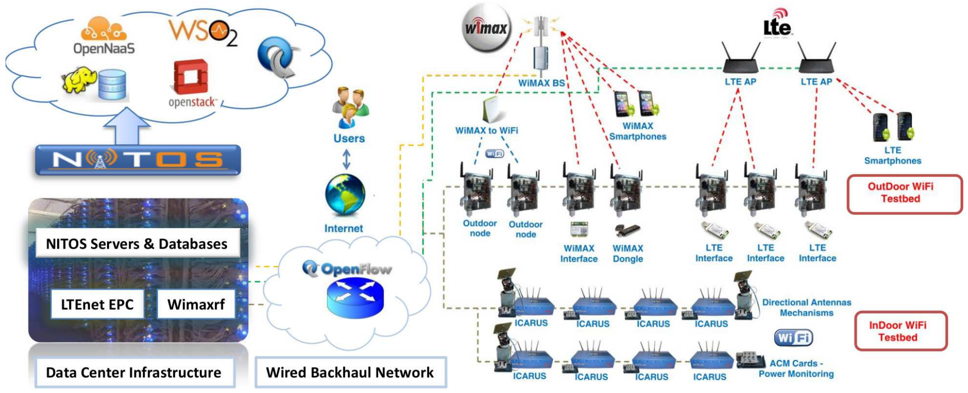


Fig. 1: NITOS testbed: a cloud based,service oriented wirelss testbed facility

and complexity and are too large to be analyzed, managed or processed using common data processing tools; these data sets are the so called Big Data. Nowadays, Cloud Computing has become a mainstream solution for Big Data processing and therefore we present in section III how we adopted the MapReduce processing paradigm to meet the Big Data challenges.

SOA: The following extension may be surprising; why extend a wireless testbed with a pure SOA mechanism? Why an ESB? ESB servers allow the configuration of tasks like message routing, mediation or transformation and load balancing. Usually they support the most commonly used Enterprise Integration Patterns(EIPs) and enable transport switching, rule-based and priority-based mediation for advanced integration requirements. Additionally, typical functions at the ESB tier may include XML/XSLT transformations, logging, and XML firewall and security/authentication services. What usually is underestimated when talking about testbed infrastructures is that at the end of the day the whole infrastructure (physical or virtualized) and all software components must cooperate harmoniously in order to provide services; the need for services drives the technology (although in some cases the opposite is possible). In section II-D we present how we extended NITOS testbed with an ESB mechanism.

The experimenter must have the total control of all protocol stack layers and all infrastructure architecture tiers, while beeing able to experiment using orchestrations of hardware and software components that use state-of-the-art technologies. Since today's testbed technology is almost exclusively suitable for wireless experimentation and was not designed initially for cloud experimentation there is hardly none adequate software mechanism to support mining and analysis of enormous large data sets in demanding applications like in IoT. By extending the capabilities of NITOS testbed in the wireless field with a cloud infrastructure, the set of use cases that can be supported is extended significantly.

II. Extensions Description

A. *The NITOS testbed*

The NITOS hardware experimental facility comprises of the following testbeds: the Wi-Fi outdoor and indoor testbed, an OpenFlow testbed, a Software Defined Radio testbed, the Wireless Sensor Network testbed (Fixed positions (building-scale), Mobile utilizing bikes (city-scale), Rural areas (agricultural deployment)), a WiMAX testbed, a LTE testbed and a 3G testbed with femtocells. In the heart of NITOS testbed resides a powerful Control and Management plane, a software that must be extended further to support the cloud concept. The basic control and management frameworks/tools used in NITOS testbed are the OMF framework[1], the OMF/OML measurement framework, the NITOS scheduler and the NITOS Broker[6]; the first was originally created in the Orbit testbed and soon became the dominant state-of-the-art framework in experimentation control. The NITOS scheduler enables resource slicing of NITOS testbed and is responsible for resource reservation, thus allowing parallel experimentation of multiple users. NITOS Broker on the other hand, is an extension of the OMF framework that integrates NITOS's Scheduler functionalities and at the same time it provides an XML-RPC interface to support federation with other testbeds through the Slice Federation Architecture (SFA). We note that OMF v6 will incorporate the functionality of various tools like the NITOS scheduler.

B. *IaaS, MaaS services and cloud management*

NITOS cloud infrastructure supports a hybrid IaaS-MaaS service model; the IaaS model provides access to computing resources in a virtualized environment and suggests offering of virtual server space, network connections, bandwidth, IP addresses and load balancers. NITOS embraced the solution of Openstack software, enabled all the above and hence it gives the user/ experimenter the ability to access a number of virtualized components across its platform and utilize them for the experimentation purposes.

In NITOS's case, the IaaS model complements the MaaS. MaaS, standing for Metal as a Service, is a cutting edge trend that allows users to provision their servers and nodes dynamically, which are in this case the whole physical machines and not just cloud instances. Thus, the existence of NITOS testbed along with the OMF v6 framework allows an on-demand design and implementation of small to larger scaled cloud infrastructures on top of which cloud services are supported.

NITOS uses the Openstack Gizzly version over Ubuntu 13.04 Operating System (UbOS) through the KVM hypervisor

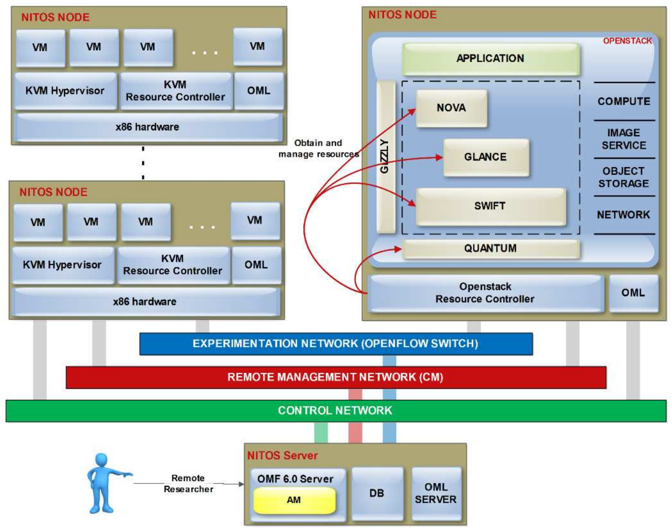


Fig. 2: NITOS testbed: Cloud Management Architecture

to provision the different resources each Testbed Node (TeN) exposes. The set up used is customized according to the needs and demands dictated by the XIFI project and community, a project of the European Public-Private-Partnership on FI-PPP[7] programme that aims to create sustainable pan-European federation FI test infrastructures; a set of activities, requirements and developments are demanded to integrate the NITOS infrastructure with the XIFI Fi-PPP system.

In order to accomplish the described architecture we created two baseline TeN images; one for the Openstack TeN and one for the KVM hypervisor hosting TeNs. Both use the same UbOS and provide a twofold advantage to the experimenter: The first is the ability of UbOS to cooperate with OMF v6 and the second is the ability to modify the wireless drivers hosted by the selected TeNs according to the experimenter's scenario and needs. Thus, the TeNs combined with the provided cloud functionalities maintain their full experimental purpose and hence the diversity of the configurations of all given resources.

The OMF v6 framework allows to load the Openstack and KVM hypervisor baseline image to any allocated TeNs and create a smaller or bigger cloud according to our needs. In advance, the Openstack Resource Controller (ORC)[8] offers the ability to create and manage virtual machines over the Openstack platform and handle any available resources. It also utilizes the Openstack API using Ruby (Fog gem), an open source project, and provides a common API for a number of cloud services including Openstack.

Regarding the network configuration among the KVM TeNs, the Openstack TeN uses a DNS server on NITOS's server side. It provides static IPs, according to the MAC address of each TeN, over a separated and isolated experimental network, establishing this way a stable network environment. For the experimentation network connections an Openflow network is used.

C. NITOS and Software Defined Networking

Following the SDN paradigm, the control plane establishes the local data set used to create the forwarding table for state exchange and topology changes updates and the data plane uses the forwarding table entries to forward traffc between ingress and egress ports. Southbound interfaces (APIs) are used for the communication between the controller in the control plane, and the switches and routers of the network (data plane). They also facilitate control over the network in order to dynamically make changes according to real-time demands. Northbound APIs can enable network functions like path computation, loop avoidance, routing and securite and enable orchestration systems like OpenStack Neutron to manage network services in a cloud. Nitos's work related to SDN is supported by many EU funded projects like CONTENT [9], FIBRE, etc. and concerns exploitation of research on both the southbound and northbound APIs.

The current landscape of SDN controllers includes openflow-based (like Floodlight, NOX/POX, Trema etc) and non-openflow-based solutions (like Junipers Contrail that is XMPP based). Our approach is to extensively use programmable dataplane technologies (like OpenVSwitch (OVS) and the Click modular router) and switches with native openflow support in both the wired backhaul network and the Linux based NITOS wirelss access systems. The usage of programmable dataplanes and Linux systems allows an extremely flexibile usage and adoption of many types of SDN controllers and southbound interfaces. We have and are still evaluating, many types of controllers (NOX/POX, Trema, FloodLight,Opendaylight etc) over the FlowVisor in various application scenarios in the access of the 802.11 network. In the backhaul of the wired NITOS network the FlowVisor is used to expose to experimenters and service builders the way to "attach" their OpenFlow controllers to the OpenFlow switches and have their own view and control of their experimental network.

In the SDN direction we made additional extentions to support compatibility with OpenNaaS[10] regarding the resource abstraction and resource reservation of the virtualized network resources. To this end, similarly with the available XML-RPC OpenSFA interface a REST interface is also available to expose the inventory of the testbed network resources to OpenNaaS. Another direction we are investigating is towards integration capabilities of NITOS's SDN networking with the Neuthron/ OpenStack services.

D. SOA and Professional ESBs

The software ESB solution we adopted is the open source WSO2 ESB[11], used by very large players like eBay, to perform billions of transactions per day. Currently one instance is functional and we plan to extend the installation in a cluster deployment. As an initial step, the experimenters have to request an account and a slice through the NITOS scheduler in order to use their own domain. This way, the experimenter will be able to use a professional ESB as an intermediate between his services, deployed in the server tier of his experimental setup, and the access tier, used to support real traffic offer (eg. by mobile users) using various wireless technologies in real conditions. Thereby, we bring the experimentation phase, the service designer and the service consumer closer to the supporting technologies.

One of our recent developments is that the OMF/OML measurements are available to the experimenter through cus-

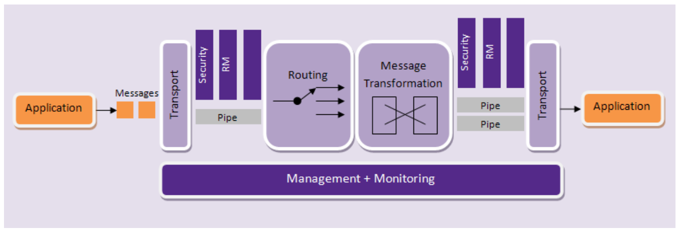


Fig. 3: NITOS testbed: a service oriented wirelss testbed facility (src: WSO2 ESB site)

tom statistic consumers we build (that implement the MediationStatisticsObserver interface of the WS02 ESB). Thus, the experimenter can have a global view in a uniform OML way of both the network/ infrastructure statistics of his experiment and the ESB services performance statistics. We note that towards being service oriented other products of the open source WSO2 family, like the Complex Event Processor (CEP) and the Business Activity Monitor (BAM), will be part of a large Service and Business Framework we envision to operate in the NITOS testbed.

III. NITOS Cloud Services: Big Data

To process and analyze Big Data, datacenters that operate clusters of machines are programmed by high-performance MapReduce-based middleware. The MapReduce programming model[3] is used to perform computations of very large data sets with a distributed and parallel algorithm over a cluster of commodity machines following the operating prototype of functional programming (eg. Lisp). The cluster comprises of a master node and several slave nodes, and the MapReduce program consists of a filtering and sorting procedure, namely Map(), and a summary operation called Reduce().

To support the MapReduce concept we adopted the Hadoop software library, a middleware that allows a cluster to scale up from a single server to thousands of machines that offer both storage and local computation. Among others, the reason behind Hadoop's selection is that it is perfectly suited for distributed computations, it is natively designed to detect and handle system failures at the application layer and it delivers high-availability without relying on the hardware level.

In order to build a Hadoop cluster we designed an OMF image that comes coupled with a script and is stored in the image pool of NITOS server. The image includes Hadoop at its simplest level (basic plugins/installations for Java, JVM and Hadoop) and the script makes the following system adjustments:

1. In order to distribute computations over the cluster, the master should be able to ssh password-less to the slaves to start the Daemons. Our extension provides an automated RSA key exchange between all workers and the master.

2. The replication factor is set initially to 3, if the cloud includes more than two nodes; should the cluster consist of less than three nodes, the replication factor is set to 2. The replication factor is the number of copies of a file that the Hadoop Distributed File System (HDFS) should maintain.

3. Since Hadoop follows strict rules about choosing the right DNS entries, we are required to setup a "local Hadoop-DNS". Our API creates such a Hadoop-DNS and also sets alias names of the nodes.

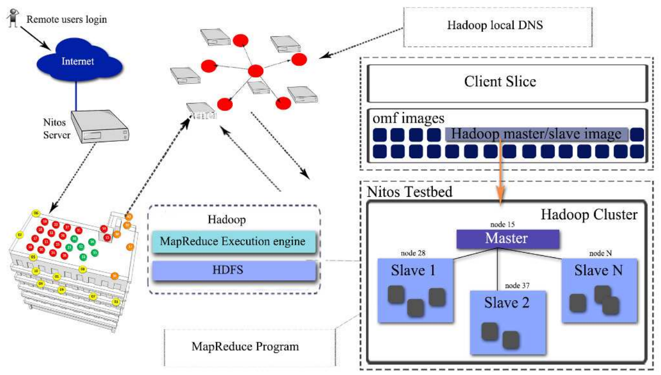


Fig. 4: NITOS testbed: a cloud based, service oriented wirelss testbed facility

4. The communication between all nodes is carried out solely over the internal ip of our internet connection because it offers speedup when nodes exchange their data during the experimentation. This is defined in the core settings of Hadoop by the API .

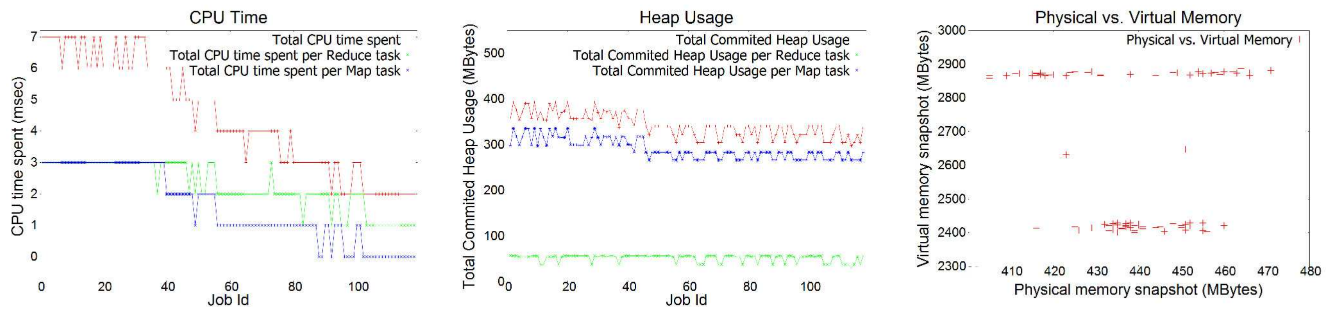
5. It may appear trivial, but we have to define which nodes will play the role of the slaves and which will work as the master node in our cluster so that it works properly. This, we also implemented to be executed automatically as part of the script.

A. *Experiment Demonstration in NITOS testbed*

Here we present a sample MapReduce experiment on NITOS testbed that is about retrieving the *k*-shells of a given Complex Network[12] starting with the cluster setup description. More speciffcally, the input file is a description of a network graph and the output file contains the corresponding *k*-shells. Here, our NITOS Hadoop-based cloud consists of six wireless nodes, one master and five slave nodes.

We loaded the designed OMF image on NITOS's nodes 20 to 25, where every single node is now a standalone slave. The script appointed node 20 automatically as the master and the rest of the nodes as slaves, established the keyless ssh and chose the internal ip for the communication. Afterwards, the master set up the local Hadoop-DNS defining the ip addresses, the hostnames and the alias names of all nodes; the same hostnames were adopted in the master and slave configuration files. The script set also the replication factor to 3. Finally, the master handed out all edited files to every slave to achieve data coherence. We also ran a separate script that starts the Hadoop Daemons (i.e. the HDFS, the Map/Reduce Daemons, the datanodes, the jobtracker and the tasktrackers). At this point we have a full functional Hadoop-based Cloud (Fig. 4).

We transferred the input file into the HDFS of our cluster and ran the program. First, the master split the data into independent chunks, distributed them and assigned Map tasks to some/all slave nodes. During a Map execution, each Mapper read pair-wise (i.e. Key and Value) the received chunk and transformed it according to specific operations; here Mappers found out all nodes of the graph with degree less or equal than



(a) CPU time spent (b) Heap memory footprint (c) Physical vs Virtual memory snapshot

Fig. 5: Results from the sample Hadoop experiment

k. The Shuffle and Sort that followed are default steps that guarantee that the input for the Reducer is sorted by Key. The Reducers were now assigned Reduce tasks; they pruned the input graph, deleted the nodes selected in previous Map tasks and retrieved the *k*-cores. After a few rounds of MapReduce jobs the final output was stored back in the HDFS.

B. Results

The experiment we performed to test the functionality of our NITOS Hadoop-based cloud is not very large, but it is suitable to grasp the concept. As the actual nodes (and not some VMs) were used as Hadoop units it is imperative to notice that the results show the actual machine resource consumption and how NITOS's cloud behaves in an illustrative experiment. We used the following quantities to visualize the obtained results:

CPU time spent (sec) stands for time spend solely by the CPU to perform the computations. Fig.5(a) shows that for our experiment it is obvious that the Reduce tasks cost in computation time more than the Map tasks. The results showed that only one NITOS node in each job performed the aggregation functions of the Reduce task and more than one the Map task.

Total committed heap usage (MB) is the memory footprint of the experiment. Fig.5(b) depicts the heap size measured for both Map and Reduce tasks. The workload of the experiment seems to have greater impact on Ten's memory in Map than in Reduce phase. The diagram shows that while CPU time decreases over time the heap demand is almost stable during the whole experiment.

Physical and Virtual memory (MB) that are shown in Fig.5(c) provide information about the variations of memory usage during each particular job. The physical memory that a task uses, as well as the corresponding virtual memory is also shown. The distributions of the obtained results show that the virtual memory that is required for the experiment is actually an order of magnitude larger than the physical measured.

IV. Related work

Under the umbrella of FIRE's initiative and the federation cover of Fed4Fire, the testbeds used for state-of-the-art research in wireless testbed experimentation are: wilab.t[13] provided by iMinds, Netmode[14] provided by NTUA and Norbit[15] provided by Nicta. All tools and developments that are developed in the NITOS testbed will be ad-hoc available in the FED4FIRE[16] federation. There is also ongoing work for integration of NITOS's cloud infrastructure with the XiFi Cloud project and Fi-PPP[7].

In our effort to provide a wileless cloud and service based experimentation infrastructure, we are closely exploring the results of the BonFIRE[17] project and testbeds like Orbit of the GENI[18] initiaitive. For example, in GENI there is signifficant work done on how to perform seamless handoffs between various wireless interfaces and (e.g WiMAX, WIFi, Ethernet etc) using programmable dataplane technology and SDN mechanisms. This is very important in our effort to provide a cloud-based wireless testbeds that target heterogeneous wireless Network (HetNet) environments.

V. Conclusions and Future Work

In the following years there will be extreme need for testbed experimentation in cloud-based wireless networks. The reason is that the technologies that support the edge network will play a central role in the way mobile services are designed, built and implemented. By making a pure wireless testbed (LTE, WiMAx, WiFi, WSN) cloud and service enabled (IaaS, NaaS, OpenStack OS, SDN and SOA extensions), we give the ability to researchers, infrastructure providers ,and service and application providers to cooperate over a common place. We are closely following the advancements in cloud computing technology, in SDN framework and in Network Functions Virtualizaton (NFV) that are related to the testbed's experimentation goals and can potentially be exploited by the proposed NITOS testbed system.

References

[1] T. Rakotoarivelo *et al.*, "Omf: a control and management framework for networking testbeds," *SIGOPS*, vol. 43, no. 4, pp. 54–59, 2010.

[2] "NITOS Wireless Testbed," http://nitlab.inf.uth.gr, 2013.

[3] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[4] T. Wood *et al.*, "Cloudnet: dynamic pooling of cloud resources by live wan migration of virtual machines," in *SIGPLAN*, vol. 46, no. 7. ACM, 2011, pp. 121–132.

[5] B. Rimal *et al.*, "Architectural requirements for cloud computing systems: An enterprise cloud approach," *Journal of Grid Computing*, vol. 9, no. 1, pp. 3–26, 2011.

[6] A.-C. Anadiotis *et al.*, "A new slicing scheme for efficient use of wireless testbeds," in *Experimental evaluation and characterization Workshop*. ACM, 2009, pp. 83–84.

[7] "XIFI — FI-PPP," http://www.fi-ppp.eu/projects/xifi/, 2013.

[8] "Openstack Resource Controller," https://github.com/maxott/omf_rc_cloud, 2014.

[9] K. Katsalis *et al.*, "Content project: Considerations towards a cloud-based internetworking paradigm," in *SDN4FNS*. IEEE, 2013, pp. 1–7.

[10] "OpenNaaS," http://opennaas.org/, 2014.

[11] "WSO2 ESB," http://wso2.com/products/enterprise-service-bus, 2013.

[12] K.-I. Goh *et al.*, "The human disease network," *Proceedings of the National Academy of Sciences*, vol. 104, no. 21, pp. 8685–8690, 2007.

[13] "iMinds," http://www.crew-project.eu/wilabt.

[14] "Netmode," http://www.netmode.ntua.gr/.

[15] "Norbit," http://www.nicta.com.au/people/mehanio/oml.

[16] "FED4FIRE," http://www.fed4fire.eu/.

[17] "BonFIRE," http://www.bonfire-project.eu/.

[18] "GENI," http://www.geni.net/.