Experiments on SDN-based Network and Cloud Resource Orchestration in FED4FIRE

M. Gharbaoui*, B. Martini†, D. Adami†, P. Castoldi*, S. Giordano‡
*Scuola Superiore Sant'Anna, Pisa, Italy
†CNIT, Pisa, Italy
‡University of Pisa, Italy

***Abstract*—This paper presents the experimental evaluation of an SDN-based orchestration system enabling the automated and coordinated arrangement of both computing and network resources in cloud Data Centers. The testbed has been set-up in the Fed4FIRE virtual experimental environment that allows for extensive, large-scale and cross-functional tests to be carried out. Results show that the selection of servers and paths that are driven by the estimated trends in their usage allows for resources to be better exploited at the cost of a limited risk of degradations.**

***Index Terms*—SDN, cloud computing, data center, orchestration, Fed4Fire.**

I. Introduction

In the last years, cloud computing gained increasing popularity in the Information Technology (IT) field as an "on-demand" service provisioning model for storage and computational resources or even applications [1]. Data Centers (DCs) are the primary infrastructures for the delivery of cloud computing services, where advanced software virtualization techniques (e.g., Hypervisor) enable agile Virtual Machine (VM) operations (start-up, cloning, migrations) resulting in high dynamicity of Cloud DC traffic flows. Definitely, for efficient operations, Cloud DCs require elastic and agile network control functions, coordinated (i.e., orchestrated) with computing resource control, in order to guarantee both proper VM operations and traffic performance [2].

In such a context, the Software Defined Networking (SDN) paradigm, being able to cope with the aforementioned challenges, is driving prominent innovations in DC networks. Indeed, SDN uses a centralized and highly programmable model as a result of the separation of the control plane from the hardware-based data plane, thus allowing for more granular and automated network control. Moreover, the controller exposes Northbound Interface (NB-I) that can be exploited for a straightforward interaction with application service delivery platforms (i.e., cloud management platforms) aiming at coordinating service deployments with the establishment of data delivery paths while following rapid cloud dynamics [3]. Finally, the rich set of statistics offered by the OpenFlow (OF) protocol at both per-port and per-flow level, allows for a substantial network status information to be collected and elaborated thereby (re-)directing provisioning and recovery actions, accordingly [4].

978-1-4673-9486-4/16/$31.00

In this work, we present the experimental evaluation and performance analysis of an SDN-based orchestration system (i.e., SDN-DC orchestrator) that enables an automated and coordinated provisioning and management of both computing and network resources in cloud DCs. By providing an abstraction of computing and network resources as well as composition and selection functions, the SDN-DC orchestrator allows for DC administrator to deploy innovative management applications. For example, with reference to IasS, and focusing on the VMs provisioning service, the orchestrator enables the dynamic placement of VMs on physical servers using heuristics that attempt to satisfy their processing, storage and communication requirements against the current status (e.g., load) of DC resources while assuring better than best effort VM data delivery. The SDN-DC orchestrator has been designed, developed and preliminary tested in the framework of EU PF7 OFELIA project [5]. In the design phase, we proposed several heuristics for the coordinate selection of computational and network resources [6] [7]. Since reproducing a real Cloud DC would require a very large number of physical devices and a great effort in the initial and subsequent deployment phases, we set-up a small-scale testbed to carry out functional tests only [8]. Similarly, we used a Java-based simulator to evaluate the performance of heuristics on a large-scale basis, without the possibility to assess other orchestration capabilities (e.g., actual evaluation of server and link utilization). As far as our knowledge, there are no other work in the literature related to the experimental evaluation of an SDN-based orchestrator for Cloud DCs.

The goal and motivation of this work lie in extending our previous activities by assessing the performance of the SDN-DC orchestrator in a virtual testbed environment, called Virtual Wall, supported by Fed4FIRE [9] [10]. Funded by the EU in the FP7 topic "Future Internet Research and Experimentation", Fed4FIRE establishes a heterogeneous, scalable and federated experimental framework for researchers in which a large number of European facilities are integrated. Using a virtual testbed including slices of computing and network capabilities, it is possible to overcome the limitations of both physical testbeds and simulators. Indeed, virtual environments are flexible, easily and quickly reconfigurable, and allow the deployment of medium or large size testbeds. Moreover, unlike simulators, virtual testbeds can be built using the same software modules as real testbeds, thus enabling the validation

of all components and functions.

II. SDN-based Resource Orchestration

In this section, we describe the architectural components and their interactions of the SDN-DC orchestrator realizing the coordinated and adaptive arrangement of both computing and network resources. In Cloud DCs, those set of resources are provided by a pool of servers interconnected through switches and links, arranged in a typical tree-like 3-layer network topology.

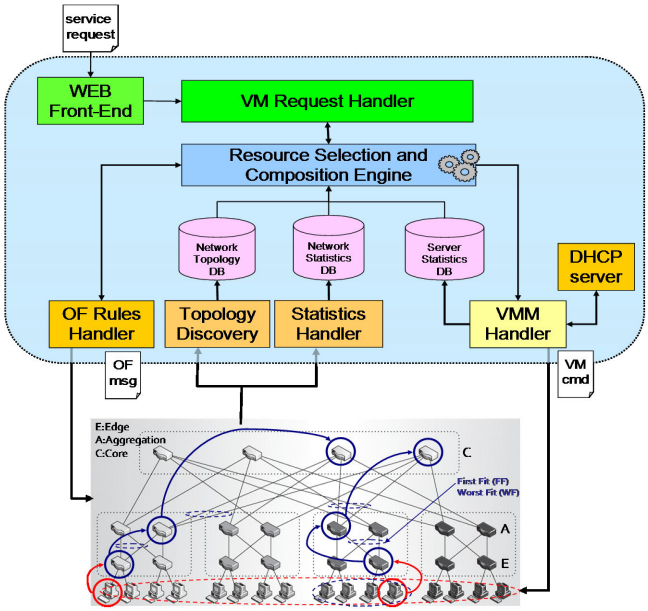

Fig. 1. Design of the SDN-DC orchestrator operating over a DC topology

The SDN-DC orchestrator basically consists in an OF controller enhanced with cloud management capabilities to deliver composite cloud-network services while addressing both computational and communication requirements. The Web-based Front-End block collects service requests (i.e., VM set-up request) from users and forwards them to the VM Request Handler. The VM Request Handler extracts the requirements for the requested VM (i.e., computational power and bandwidth) and triggers the Resource Selection and Composition Engine for selecting a proper server and data delivery path throughout DC switches, thus guaranteeing both set of requirements. In case a server or a path, able to meet the request requirements, is not found, the service request is rejected. The selection of servers and paths is made taking into account the traffic load of interconnection links (i.e., packet forwarding capability available at switch ports serving the interconnection links) and the processing load of servers (i.e., computational power available at server CPU). To this purpose, statistics are periodically collected by the Statistic Handler for the switches/links and by the VMM Handler for the servers and then stored in the Network Statistics Database (DB) and Server Statistics DB, respectively. Moreover, up-to-date topological information concerning links, switches and servers status are retrieved by the Topology Discovery block by leveraging services for network discovery and host tracking and then stored in the Network Topology DB. Once the server and the DC switches have been selected, the VMM Handler is triggered to actually deploy the VM in the selected server whereas the OF Rules Handler is used to accordingly update the Flow Table of the selected switches thus enforcing the selected delivery path. A DHCP server is deployed to assign an IP address to the VM upon its creation. A detailed description of the SDN-DC orchestrator can be found in [6] [11].

In this work the selection and composition of both resources is carried out using different kinds of algorithm, that leverage estimations about the load of servers and links/switches that are derived from collected data statistics. In the following, the estimation process and the heuristics are detailed.

As for the link/switch load estimations, the Statistics Handler firstly calculates the average traffic rate, i.e., data throughput, at each interface while transmitting and receiving data. Specifically, the traffic rate is obtained by collecting two consecutive counter values for transmitted and received bytes (i.e., Tx_Bytes or Rx_Bytes) at t_{i-1} and t_i, from the switch s at the interface connected to the link l. Then, the average input and output rates at l in the time interval $(t_i - t_{i-1})$ are computed as follows:

$$Traffic\_Rate\_In_i(l) = \frac{Rx\_Bytes(l,t_i) - Rx\_Bytes(l,t_{i-1})}{(t_i - t_{i-1})} \quad i = 1,2,...n \tag{1}$$

$$Traffic\_Rate\_Out_i(l) = \frac{Tx\_Bytes(l,t_i) - Tx\_Bytes(l,t_{i-1})}{(t_i - t_{i-1})} \quad i = 1,2,...n \tag{2}$$

Due to the high variability of traffic patterns, current values of traffic rates are also highly variable over time. In order to obtain consolidated data on trends of resource utilization over time, the instantaneous value of the traffic rates should be correlated with historical values sampled over time. To this purpose, the estimated traffic load at the interface connected to the link l, with bandwidth B, in each direction (i.e., In and Out with respect to the switch interface) is computed as follows:

$$Estimated\_Link\_Load\_Out_i(l) = \frac{(1-\alpha)Traffic\_Rate\_Out_i(l) + \alpha \sum_{k=i-M}^{i-1} \frac{Traffic\_Rate\_Out_k(l)}{M}}{B} \tag{3}$$

$$Estimated\_Link\_Load\_In_i(l) = \frac{(1-\alpha)Traffic\_Rate\_In_i(l) + \alpha \sum_{k=i-M}^{i-1} \frac{Traffic\_Rate\_In_k(l)}{M}}{B} \tag{4}$$

where α is the history weight, a parameter that allows to assign a weight to the average of the past M samples (collected at time $i-1$ down to $i-M$) against the newest one (at time i) with samples collected every T seconds. The balancing between the instantaneous and historical values is fundamental to obtain a reliable load estimation.

As for selection and composition strategies, we conceived four different on-line Resource Selection and Composition algorithms to be adopted upon the arrival of a VM set-up request. They can be classified in two categories, based on

the order used to select computational and communication resources:

- *Server-Driven* (SD): first attempts to find a server and then the delivery path throughout a sequence of switches/links.
- *Network-Driven* (ND): first attempts to find a delivery path and then a server where to deploy the VM.

A case of selection is described for each algorithm category in the DC example topology shown in the bottom of the Fig. 1, i.e., SD case on the left, ND case on the right. The fat-tree topology represents a multi-rooted tree which offers the possibility of taking advantage of several paths from the servers to the core switches. Without lack of generality, we show in the bottom of Fig. 1 the selection of a path that connects a server to the core switch, throughout the aggregation and edge switches, i.e., North-South delivery path. At each step of the switches/links selection, and for the selection of the server, two bin-packing heuristics have been considered, i.e., the First Fit (FF) policy and the Worst Fit (WF) policy. The FF policy allocates the VM (traffic load) in the first available server (switch), where the resources are ordered according to an identifier. On the contrary, the WF policy searches for the most unloaded server (switch) to allocate the requested VM (traffic load) where the largest free space is available. If not enough availability exists at server (switch) the request is blocked. Further details about the algorithms can be found in [6].

III. Fed4Fire Set-up for Experiments

The Fed4Fire platform offers a set of tools that allow to reserve resources, get access to them and then configure them, in order to run the experiments. In particular, the JFed framework [12] is the graphical interface that makes it easy for the experimenters to check the availability of the resources in the different testbeds and to reserve them. As shown in Fig. 2, in our experiments we used nodes exclusively from the iMinds testbed (Virtual Wall 2) which is an emulation environment hosting 159 physical nodes that might be used directly as bare metal hardware or might be virtualized [10]. We created two independent slices, i.e., virtual experiment containers: one including our SDN-based controller (one physical node) and one including the fat-tree topology representing the DC and one outside host that emulates a sink for the traffic generated from the VMs. The fat-tree topology is composed of 20 OVS [13] switches that communicate with the controller through the OpenFlow protocol. Moreover, on the 16 hosts we installed the Xen Cloud Platform (XCP) [14], which is a virtualization solution including the XEN API tool stack responsible for the VMs lifecycle (clone, run, shutdown, remove). Once a VM is created, we generate a variable bit rate (VBR) traffic from the outside host to each VM using iperf in order to load the DC network and assess the effectiveness of the SDN-based orchestration process.

During the experiments, we apply the selection and composition algorithms described in Section. II, i.e., SD-FF, SD-WF, ND-FF and ND-WF. The statistics from the switches are collected every 35 seconds, whereas the estimation scheme is carried out using an history weight α equal to 0.2 and a monitoring window M including 6 samples. Such algorithms and the estimation mechanism are defined as two possible schemes for carrying out the orchestration process, and are used in the experiments for assessing the SDN-DC orchestrator and for demonstrating improvements with respect to current practices in DCs where the selection of paths are not engineered because static approaches for routing are used, e.g., Equal-Cost Multi-Path (ECMP), that eventually underutilizes the network links [15]. For this reason, we use as a baseline the case of a random selection of the server and the application of the spanning tree algorithm to forward the VM traffic that finally does not use the over redundant links [16]. It is worth pointing out that, for a sake of fairness in the comparison with orchestration algorithms, the estimation of the load on network links is maintained to decide for the admission of VM allocations in the baseline.

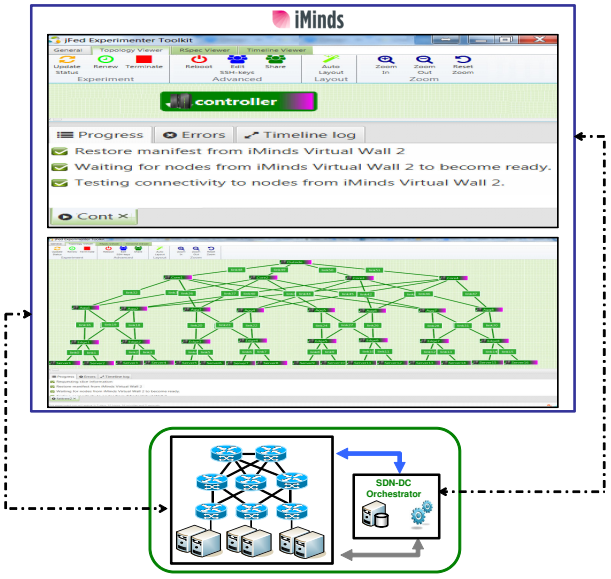

Fig. 2. Experiment components in the Fed4Fire testbed infrastructure

The VM allocation requests are generated according to a Poisson distribution characterized by inter-arrival and holding times exponentially distributed with an average of $1/\lambda$ and $1/\mu$, respectively. $1/\lambda$ varies within the range $[80sec, 200sec]$ whereas $1/\mu$ is fixed to 10.000 seconds. In such configuration the network is relatively loaded which might cause the rejection of some requests. For the traffic we have considered a script that generates a variable traffic from the allocated VM towards the gateway following the selected path. For the requests characteristics, the computational power is uniformly distributed within the interval $[2, 4]$ whereas the bandwidth is uniformly distributed within the range $[60Mb/s, 80Mb/s]$, the storage capacity is equal to $1GB$ and the requested RAM is also equal to $1GB$. Regarding the characteristics of the topology, the initial computational power of the servers is fixed to 20 CPU whereas the links capacity is fixed to 80 Mb/s.

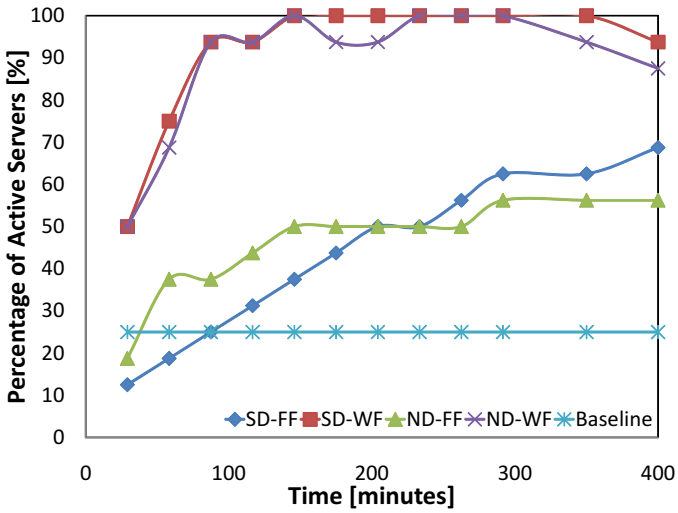

Fig. 3. Number of used servers in time

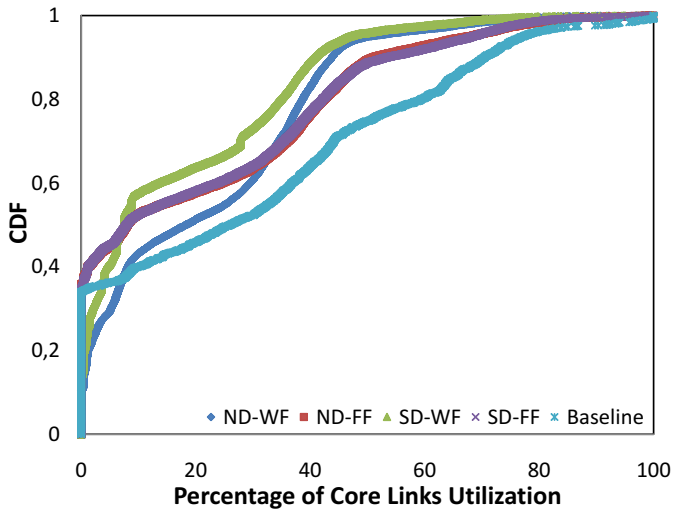

Fig. 4. CDF of core links utilization

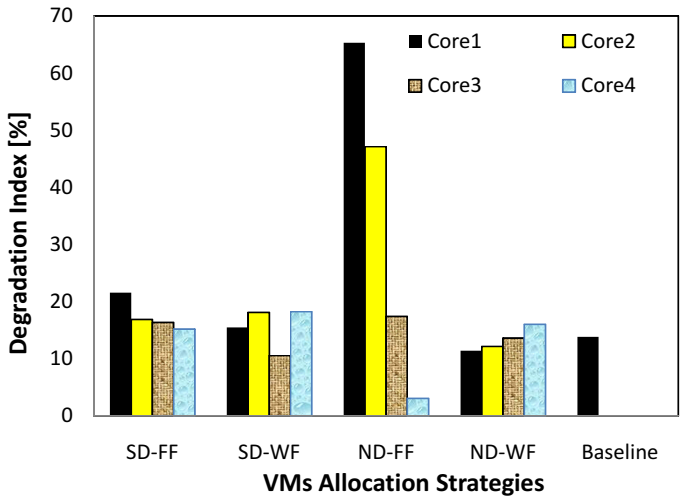

Fig. 5. DI a function of the core switch

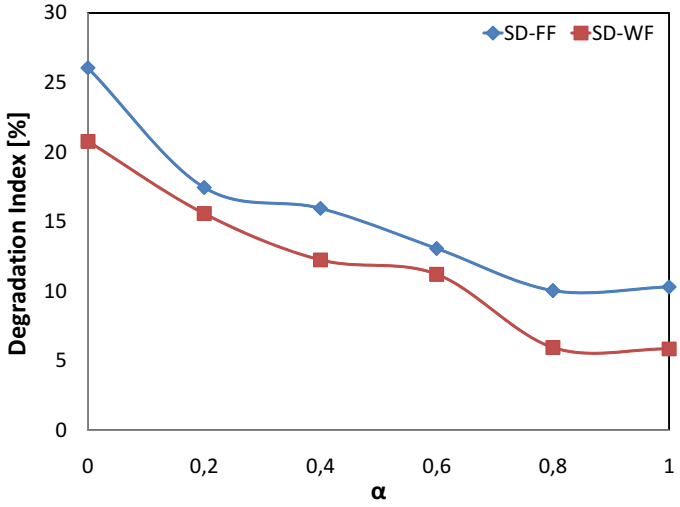

Fig. 6. Effect of α on DI

IV. Performance Evaluation

This section describes some experimental results we achieved during the tests. Fig. 3 plots the percentage of active servers over time as an index of computational resources utilization. Results show that the orchestration process (i.e., selection of the server and of the path according to the estimated load) allows for an increased utilization of servers. Indeed, in the baseline less servers are active (i.e., 4 during all the experiment). This is mainly due to the saturation of the aggregation and core links as result of the spanning tree algorithm that avoids using a number of links (redundant ones). The saturation of aggregation and core links cause them to be able to practically handle the traffic generated by only one server in each pod. Among the orchestration algorithms, as expected the FF policy minimizes the percentage of active servers, independently from the adopted algorithm, either SD or ND. In fact, FF policy tries to place all the allocated VMs in the same server, and selects another server only if the current one is full and the resources are unavailable (CPU on the server or bandwidth on the selected path). On the other hand, we can observe that the WF policy utilizes all the servers which has the advantage of balancing the load between the available resources. Fig. 4 illustrates the CDF of the core links utilization for the different algorithms compared with the baseline. The CDF for the baseline shows that the core links are much more saturated than in the orchestration algorithm (30% of the core links are utilized at more than 50% of their capacities against only 10% in the best orchestration case) which is due to the exploitation of a lower number of core links in the spanning tree topology under the same load. As expected, the ND-WF and SD-WF algorithms present the best results since they tend to minimize the percentage of empty core links. ND-FF and SD-FF present almost the same results with a high percentage of links utilized under 5%. However,

the FF policy tends to saturate more the core links which is confirmed by a 10% more of links utilized at more than 50% with respect to the WF policy.

While resource utilization is better in the orchestration case, this could come at the risk of a degraded data delivery due to possible estimation errors as result of variability of overall traffic. For this reason, we evaluate the Degradation Index (DI) defined as the percentage of time at least one link is saturated (as observed by monitoring and estimation system), which introduces some delay in the delivery of the packets. In Fig. 5 we plot DI considering only the core links that we regroup by core switch. Results show that the ND-FF algorithm presents the worst degradation performance. In fact, 2 core switches out of 4 present a degradation index higher than 40%. Moreover, an important difference is noticed between the 4 core switches where core switch 1 presents a DI higher than 60% while DI is lower than 5% in core 4. This trend supports the idea that this algorithm saturates a part of the topology while it keeps the other part almost empty. In the other algorithms, DI is almost equivalent in all the core switches. In SD-WF and ND-WF this is due to their load balancing characteristic. In the SD-FF case, DI is slightly higher (around 5%). In fact, SD-FF selects first the server which also implies the selection of the edge switch, following a consolidation trend that is then attenuated at the aggregation and core level due to multiple choices of links and to the dynamicity of traffic. This results in a non significant difference between the core switches, although a DI that remains slightly higher for core1.

Fig. 6 plots DI as a function of α. Due to space constraints we limit the results to the SD-FF and SD-WF heuristics. We notice that by increasing α DI significantly decreases, thus improving the performance of the running virtual machines. In fact, higher values of α represent the case where the traffic variations are attenuated and the estimated traffic corresponds to the sum of average traffic loads of the historical samples. Once a request arrives, the status of the network is stable and the allocation of a new VM does not deteriorate the performance. Moreover, DI is lower in the SD-WF case which is the result of its load balancing characteristic that prevents from saturating the links.

V. Conclusion

This paper has presented the experimental evaluation of an SDN-based orchestration system enabling the automated and coordinated arrangement of both computing and network resources in cloud DC environments. The testbed has been set-up in the Fed4FIRE virtual experimental environment that allows for large-scale and cross-functional tests to be carried out. Results shows that the selection of servers and paths that are driven by the estimated trends in their usage allows for the resources to be better exploited at the cost of possible degradations. The comparison among orchestration strategies highlights that the spreading of selections and the proper settings of the monitoring parameters allows for limiting the risk of degradations at less than 10%.

Acknowledgment

This work was funded in part by the Scuola Superiore Sant'Anna on the NET-DRIVE project and supported by the EVIDENCE project approved within the 2 Call of the FED4FIRE project (n. 318389).

References

[1] C. Hofer *et al.*, "Cloud computing services: taxonomy and comparison," *Journal of Internet Services and Applications, Springer*, vol. 2, no. 2, pp. 69–79, 2011.

[2] Y. Zhang *et al.*, "Evaluating the impact of data center network architectures on application performance in virtualized environments," in *Quality of Service (IWQoS), 2010 18th International Workshop on*, June 2010, pp. 1–5.

[3] C. Chappel, "Unlocking network value: Service innovation in the era of SDN," HeavyReading, White paper, June 2013.

[4] A. Lara *et al.*, "Network Innovation using OpenFlow: A Survey," *IEEE Communications Surveys and Tutorials*, vol. 16, 2014.

[5] B. Martini, D. Adami, A. Sgambelluri, M. Gharbaoui, L. Donatini, S. Giordano, and P. Castoldi, "An SDN orchestrator for resources chaining in cloud data centers," in *European Conference on Networks and Communications (EuCNC)*, 2014.

[6] D. Adami, B. Martini, M. Gharbaoui, P. Castoldi, G. Antichi, and S. Giordano, "Effective resource control strategies using openflow in cloud data center," in *IFIP/IEEE International Sympsium on Integrated Network Management (IM2013)*, 2013.

[7] M. Gharbaoui, B. Martini, D. Adami, G. Antichi, S. Giordano, and P. Castoldi, "On virtualization-aware traffic engineering in OpenFlow Data Centers networks," in *IEEE Network Operations and Management Symposium (NOMS)*, 2014.

[8] D. Adami, B. Martini, A. Sgambelluri, M. Gharbaoui, P. Castoldi, A. Del Chiaro, L. Donatini, and S. Giordano, "An OpenFlow controller for cloud data centers: Experimental setup and validation," in *IEEE International Conference on Communications (ICC)*, 2014.

[9] http://www.fed4fire.eu/.

[10] http://ilabt.iminds.be/iminds-virtualwall-overview.

[11] "EMOTICON requirements and architecture design," OFELIA Deliverable D13.1, http://www.fp7-ofelia.eu/.

[12] http://jfed.iminds.be/.

[13] http://openvswitch.org/.

[14] http://wiki.xen.org/wiki/XCP_Overview.

[15] C. Hopps, "Analysis of an Equal-cost Multi-Path algorithm," RFC 2992. [Online] Available: http://tools.ietf.org/html/rfc2992.

[16] D. Kerger, "Approximating, verifying and constructing minimum spanning tree problem," *Annals of the history of computing*, 1985.