Cloud and Network Orchestration in SDN Data Centers: Design Principles and Performance Evaluation

[2]M. Gharbaoui, [1]B. Martini, [1]D. Adami, [3]S. Giordano and [2]P. Castoldi

[1]CNIT, Pisa, Italy - ({davide.adami, barbara.martini}@cnit.it)
[2]Scuola Superiore Sant'Anna, Pisa, Italy - ({m.gharbaoui, castoldi}@sssup.it)
[3]Dept of Information Engineering, University of Pisa, Pisa, Italy – (stefano.giordano@iet.unipi.it)

Abstract - The oversubscription of Data Center network links and the high volatility of Virtual Machine (VM) deployments call for a flexible and agile control of Data Center networks,coordinated with computing resource control (i.e., cloud resource management). The Software-Defined Network (SDN) paradigm opens up new opportunities to design convergent resource management systems able to address the provisioning of cloud services while meeting dynamicallychanging traffic demands of running VMs.

This paper presents the architectural design of an SDN-based orchestration system which is able to coordinate the provision of composite cloud and network services while assuring computational requirements as well as a better than best effort VM data delivery. The proposed orchestration system is able to perform VM allocations also based on estimations of switch/link and server loads as result of the synergistic interwork of the following functions: (i) resource selection and composition functions, (ii) coordinated resource configuration and management functions, (iii) monitoring and registration functions of resource status. A set of resource selection and composition strategies and estimation schemes have been also specified.

The orchestration process has been thoroughly evaluated through a comprehensive set of simulations that clearly showan increasing acceptance rate of service requests, an improved utilization of network capabilities while effectively preventing significant degradations of the user experience despite the oversubscription of data center network links.

I. INTRODUCTION

Nowadays, Data Centers (DCs) are the primary infrastructures for the delivery of Cloud Computing services. Thanks to advanced software virtualization techniques (e.g., Hypervisor) and agile Virtual Machine (VM) operations (e.g., start-up, cloning, and migration), Cloud DCs are able to deploy a huge number of workloads characterized by a high level of volatility. The large amount and the high volatility of VM deployments give rise to unpredictable traffic patterns that might challenge the proper operation of the DC network due to the oversubscription of switches and possible bandwidth contentions. To meet the traffic demands, cloud DCs need more advanced network control capabilities, i.e., increased provisioning automation, rapid deployments and more granular configurations [1]. In particular, cloud DCs require elastic and agile network control functions orchestrated with computing resource control in order to guarantee proper VM operations also from the point of view of traffic performance [2][3].

The Software Defined Networking (SDN) and the OpenFlow (OF) protocol offer innovative network control capabilities able to cope with the aforementioned challenges. The separation of the control plane from the hardware-based data plane allows flow-based forwarding rules in network switches to be directly configured, i.e., programmed, at different levels of granularity by a centralized software controller through a South-Bound (SB-I) interface. Moreover, the controller exposes a Northbound Interface (NB-I) that can be exploited for a straightforward interaction with application delivery platforms (i.e., cloud management platforms) aiming at coordinating cloud service deployments with the establishment of data delivery paths. Ultimately, SDN paradigm allows network resource provisioning functions (e.g., data delivery) to be effectively orchestrated with cloud management functions and dynamically optimized to address diversified requirements and to avoid VM data delivery degradations while following rapid cloud dynamics [4].

In this work, we present an SDN-based orchestration process for cloud DCs enabling the automated and coordinated management of both computing and network resources in virtualized DC environments. More specifically, the orchestration process is in charge of dynamically (re-)arranging VMs across servers and, accordingly, (re-)establishing a delivery path throughout switches, while addressing specified bandwidth and computational demands. The orchestration process decides the most appropriate combination of servers and switches where to deploy servicesusing specified selection and composition strategies. It also relies on estimations of the load of systems/nodes, thus assuring adequate computing resources to VMs and better than best effort data delivery among VMs. The estimation of load has been conceived that consolidates measurements of load data (e.g., OF traffic statistics) periodically collected from switches to highlight resource usage trends and to make more reliable predictions about expected available capacities to host services.

The contribution of this work is twofold. Firstly, we present the architectural design of an SDN-based orchestration system which is fundamentally based on the synergistic interworking between three core functions, i.e., selection and composition function, monitoring, estimation and registration function, and coordinated configuration and provisioning function. Overall, we provide an holistic view of the orchestration process in cloud DCs by relating the orchestration building blocks, i.e., on one side, to cloud application functions and, on the other side, to cloud and network infrastructure managers. We also analyzed how virtualization and abstraction play a role in separating resource- and service-related components and how this decoupling eases the coordination and the orchestration of different set of resources. Finally, based on this architectural design, we present a set of orchestration strategies that combines specified server/switch selection and composition algorithms with a measurement-based estimation mechanism of the load based on the *Exponential weighted moving average* scheme.
Secondly, we assess the effectiveness of the proposed SDN-based orchestration process with a thorough set of simulations. We provide performance evaluations in terms of different metrics (e.g., acceptance rate of service requests, resource utilization, probability of degradations, global efficiency of orchestration process) while considering the two most common DC topologies (i.e., Fat Tree and VL2) and different traffic patterns (i.e., elephant and mice flows). We also compare the proposed orchestration process with current management practices in DCs and with similar approaches in literature through a proper selection of baselines. It is worth pointing out that this work does not aim at individually addressing the optimized operation or performance of any of the three above mentioned core functions of the orchestrator. In fact, the optimal operation of each of those functions has already been thoroughly investigated in the literature in the three corresponding research fields, i.e., scheduling of computing tasks (e.g., VMs) [5], converged infrastructures [6], decentralized and probabilistic network management [7]. Instead, we aim at evaluating the overall orchestration process as result of the synergistic interworking of the above three functions even when a non-optimized operation is considered for each of them. Results show that even when not optimized schemes are used for selection, composition and load estimation, the proposed orchestration process further improves the selection of the available resources, achieves a higher effectiveness in the resource utilization compared to existing solutions while preventing congestions, and, ultimately, degradations of the VM data delivery.
This work follows previous contributions from the authors on experimental tests carried out running a prototype of the proposed orchestration processin a laboratory testbed [8, 9],on preliminary simulations to evaluate impacts of orchestration strategies in terms of blocking probability [10,11], or on both of them [12, 13]. More specifically, this paper extends [12] and [13] with a more comprehensive set of simulation results considering two different kinds of DC topologies and a different and richer set of metrics (i.e., degradation index, saturation index, waste index) as well as studying the impacts of monitoring and estimation parameters on the effectiveness of the orchestration process.
The paper is organized as follows. Section II provides an overview of the state-of-art of the research and the position of this work with respect to the related works. Section III highlights the architectural guidelines and design principles of the cloud SDN orchestrator in terms of building blocks and functional layering. It also describes the proposed resource selection and composition strategies for the joint selection of computing and network resources and the adopted monitoring operation. Section IV describes the simulation settings, the adopted metrics and the baselines. Section V evaluates the performance of the proposed strategies through a set of exhaustive simulations. Finally, Section VI concludes the paper with final remarks.

II. Related works

Due to the increasing demand for communication intensive applications running in virtualized DCs, novel resource management solutions are urgent that effectively use DC resources while addressing more stringent requirements in terms of dynamicity [14].
A number of initiatives in this area has been devoted to the optimal placement of VMs across DC servers, including the development of both commercial tools for capacity planning (e.g., VMware Capacity Planner [15], IBM WebSphere [16]) and research works [5][17][18]. The common target of these initiatives is to decide the optimal VM placements considering constraints imposed by server capacity only, i.e., CPU and amount of memory. The optimal VM placement process, also taking into account network traffic load across DCs, is investigated in [19]-[23] where a mathematical formulation of the VM placement optimization problem is provided that takes into account network state information in terms of either topology constraints (e.g., interconnection network infrastructure) or VM communication patterns. Due to the NP-hard complexity of the problem, the authors end up proposing also heuristics (i.e., greedy algorithms) to generate a viable solution considering the DC scale. Moreover, such works aim at fulfilling a set of VM placement requests that are known in advance, thus addressing a planning problem. Instead, in cloud environments featured by high volatility, placement decisions are preferably to be made incrementally and dynamically as this work does. This approach is used in [24] where authors address the placement of VMs onto virtualized DC servers as request arrives and considering both application-level (i.e., redundancy requirements) and network-level (e.g., VM communication patterns)constraints. Moreover, in [25] authors tackle the Virtual DC embedding problem also addressing bandwidth constraints. They claim that the problem is NP-hard and then design a heuristic based on three embedding phases: VM mapping, link mapping and switch mapping. Such three-phase heuristic takes as

parameters the server defragmentation and the residual bandwidth for embedding VMs and presents higher acceptance rates and better resources utilization with respect to the works that consider the whole set of VM placement requests. [26] also evaluates the possibility of VM allocations request by request while addressing bandwidth constraints. The proposed algorithm is based on the choice of the most unloaded server, then the shortest distance path is selected. In the most common DC topologies, the shortest-path algorithm for path selection practically means a random choice since the number of hops from one server to every core switch is very likely to be the same, which decreases significantly the performance of the algorithm from the traffic performance point of view. However, in all these works, the actual traffic generated by the whole set of VMs is not measured and not accounted while deciding for the admittance and the arrangement of incoming VMs as this work actually does as part of the orchestration process. The actual network load for dynamic resource provisioning is considered in [27], where authors propose to dynamically allocate the resources to virtual networks while taking advantage of unused virtual nodes and links to ensure that requests are not rejected when resources reserved to already allocated requests are idle. They use a demand-driven approach based on reinforcement learning which is characterized by a continuous monitoring of the nodes and the links. Based on the actual resources utilization, the un-used resources are made available for other requests. This solution improves the performance with respect to the static approaches. However, it does not consider computing resource allocations. Moreover, the effectiveness of the proposed solution requires a high number of learning episodes to finally converge and takes time to learn an optimal policy, unlike the orchestration strategies proposed in this work which depend only on the traffic measurements, converge immediately and are tailored for cloud DCs.

The design of measurement-based admission control functions is another related research area, where admission decisions are made based on measured properties of traffic rather than a priori user-specified traffic models, that are difficult to specify in advance especially in DC environments where a huge amount of flows run with many different time-scale behaviors. A number of research works exist in this area (e.g., [28][29]) where methods and algorithms have been proposed to allow the dynamic characterization of traffic flows, while adapting the admission of new flows to avoid possible congestions. The main goal of such an approach is to keep the overload probability of the network or packet delays as low as possible while exploiting statistical multiplexing gains. However, because of high measurements errors, their performance is poor in case of small links capacities. Instead, the proposed SDN orchestration process effectively exploits link capacities and is independent from the network load. As far as our knowledge, no research works have been proposed that use the measurement-based admission control in cloud DCs as this work does. Moreover, the approach we use in this work is in line with an emerging network management paradigm, i.e., probabilistic management, recently conceived to address new requirements that come from the increasing complexity and dynamicity of communication networks. In the probabilistic management, decisions and policies are not based on deterministic and guaranteed data or objectives, but use probabilistic models to represent the network state (e.g., average link load over the network expressed as expected mean value and variance), to specify objectives (e.g., probability of service degradation occurrence), to make decisions using elements of randomness (e.g., changing network conditions) [7]. This approach is highly promising, especially for DC networks that are characterized by the high dynamicity of service workflows and, consequently, of data traffic flows, while assuring efficient resource usage, scalability, robustness, and adaptability. This work is aligned with such an approach in making decisions based on estimations of the network load using the *Exponential weighted moving average* scheme that consolidates consecutive traffic load measurements to derive trends of link utilizations. The effectiveness of the decisions are then evaluated in terms of probability to have data delivery degradations, i.e., degradation index. As far as our knowledge, no research works adopt probabilistic-based approaches in the context of DC network management.

The use of SDN-based solutions and OF in DC networks is widely discussed and in some cases also deployed successfully [30][4]. Specifically, in [31] the authors propose a solution to reduce the network congestion events through fine grained traffic engineering strategies implemented by OF controllers that leverage short term predictability of traffic demands. In [32] the authors evaluate the performance of OF in a DC environment using DC traffic models gathered from real DC measurements. However, none of the above research works consider the use of OF in the context of an orchestration process as this work actually does. In particular we argue that the statistics collected on the switches can be used to estimate the state of the network, and that the programmability of forwarding rules of data flows with different granularities can be exploited to have a more dynamic and pervasive control on the traffic while following cloud application dynamics. The use of SDN-based approach to orchestrate cloud and network resources has been mentioned in [33]. The authors claim that a unified control of both cloud- and network-layer resources can be addressed exploiting an SDN approach to provide adaptive service data delivery and adequate user service experiences, although their solution does not rely on OpenFlow to configure delivery paths. In this work, an orchestration process for VM deployments is devised that exploits SDN/OF capabilities increasingly available in DC networks. The proposed orchestration process is able to direct VM allocations also based on estimations on switch/link and server loads as result of the synergistic interwork of the following functions: (i) resource selection and composition functions, (ii) coordinated resource configuration and management functions, (iii) monitoring and registration functions of resource status. A set of resource selection and composition strategies and estimation schemes have been also specified and evaluated through a thorough set of simulations.

III. SDN ORCHESTRATION FOR CLOUD DCS

The usage of orchestration is often discussed in the context of service computing, virtualization, converged (i.e., horizontally-integrated) infrastructure and dynamic datacenter topics. In general terms, the orchestration is referred to as the automation of tasks involved with arranging, managing and coordinating resource capabilities deployed across different resource/administrative domains, with the purpose of exposing them as a single service instance while addressing user requirements [34]. Moreover, orchestration also relies on monitoring to handle exceptions or deviations from normal workflows and to adjust provisioned resources to recover from service degradations or outages [35]. In cloud DCs, two different domains of resources are involved in the orchestration process, i.e., computing and network resources, that are provided by a pool of servers interconnected through switches in a typical tree-like 3-layer network topology. The following three core functions can be envisioned that are involved in an orchestration process:

- *selection* and *composition* of a set of heterogeneous, i.e., cross-domains, resource capabilities thereby fulfilling user service requests (e.g., IaaS with network QoS assurance) while addressing specified management objectives (e.g., server utilization balancing/consolidation) or user requirements (e.g., specified bandwidth requirements). This implies an admission control and a decision making process that is carried out according to specified policies or strategies.
- *coordinated configuration and provisioning* of selected resources across different domains through the enforcement of consistent configuration directives and the activation of corresponding services (e.g., VMs and VM-to-VM delivery paths with CPU and bandwidth requirements).
- *monitoring and registration of the operational status* of activated services to assess current and long-term resource utilization rate to adaptively affect selections or to trigger (re-)arrangement actions of provisioned services.

The synergistic interworking among the above functions is crucial for an effective orchestration process able to adaptively provision services while recovering from service degradations, service outages or system failure events. For the orchestration purpose in highly dynamic cloud DC networks, the SDN can play a crucial role thanks to distinguished features, i.e., flow-based abstraction combined with high programmability of forwarding functions that allow data forwarding rules to be dynamically established with different granularities while following cloud service provisioning dynamics. Moreover, the rich set of statistics offered by OF allows significant network status information (e.g., traffic load) to be collected and elaborated thereby (re-)directing provisioning and recovery actions, accordingly. In the currently deployed DC management platforms, the network status is monitored to detect failures or severe congestions and, in case, to recover proper network operations. On the other hand, network monitoring data does not affect cloud service provisioning workflows nor drive VM allocations across servers, accordingly. Indeed, latency issues are experienced in cloud DCs mainly due to the oversubscription of network links [36]. Instead, the proposed orchestration system is conceived to dynamically deploy composite services (including computing and data delivery services) while driving VM allocations based on estimations of trends in the load of switch/link and server obtained from elaborations and consolidations of OF traffic statistics. Indeed, by continuously monitoring the traffic generated by the allocated VMs and by estimating trends in resource utilization, the proposed orchestration process can exploit the capacity of resources in a more effective way. Moreover, despite the high variability of traffic in cloud DCs, this approach allows more service requests to be fulfilled without significant impacts on user service experience.

1. Orchestrator design and functional layering

In this subsection we present the architectural design principles of the proposed SDN orchestration system. Firstly, we describe the overall design in terms of functional layering as well as building blocks (see Fig. 1). Then, we show a sequence diagram describing the interactions between the functional blocks involved in the orchestration process.

The *Application Layer* includes service instantiation and lifecycle management functions operated in DC management platforms to enable the provisioning of applications and services to users. It includes either cloud-related management applications (e.g., VM provisioning/IaaS, Service Function Chaining, Authentication/Authorization/Accounting) or network appliances (e.g., Virtual Tenant Network Management). Also the provisioning of a combination of cloud and network services, i.e., composite services, can be envisioned in this scenario, e.g., provisioning of VMs along with a Virtual Tenant Network to connect VMs with specified delay constrains along the VM-to-VM paths.

The *Orchestration Layer* relates to the automated arrangement, coordination and composition of cross-domain resources including both computing resources and communication (i.e., data delivery) capabilities provided by servers and network switches/links, respectively. The orchestration layer also provides decoupling functions between the *Application Layer* and the *Control and Management Layer*.

The *Control and Management Layer* includes the set of resource- and technology-specific functions used to provision the required resources, to collect operational status information and to ensure the correct operation of a specific resource pool, i.e., device configuration, monitoring, failure recovery. Different control and management systems are foreseen for each resource set at the *Infrastructure Layer*, namely, the OF controller (e.g., OpenDayLight [37]) for the DC network and the VM Manager (e.g., XEN Cloud Platform [38]) for the virtualized servers. In particular, the OF controller provides basic control functions, such as topology discovery, statistics management, forwarding rules set-up and delete.

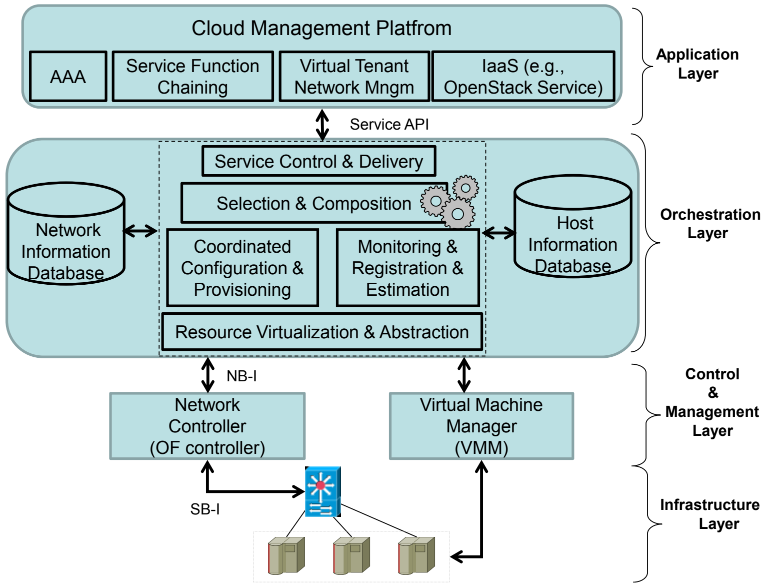
Cloud Management Platfrom
AAA
Service Function Chaining
Virtual Tenant Network Mngm
IaaS (e.g., OpenStack Service)
Application Layer
Service API
Service Control & Delivery
Selection & Composition
Network Information Database
Coordinated Configuration & Provisioning
Monitoring & Registration & Estimation
Host Information Database
Resource Virtualization & Abstraction
Orchestration Layer
NB-I
Network Controller (OF controller)
Virtual Machine Manager (VMM)
Control & Management Layer
SB-I
Infrastructure Layer

Fig. 1: SDN-based Orchestration system for cloud DCs: functional layering, building blocks and interactions

For the purpose of decoupling the resource-specific functions from the upper layer functions, the *Orchestration Layer* includes the *Resource Abstraction and Virtualization* function to hide underlying technology-specific resource details while generalizing resource capabilities in terms of independent and self-contained service components. More specifically, resource capabilities are sliced and, through abstraction, each slice is provided as different service components. Basically, servers and switches act as containers of service components that can be invoked and combined to meet specified user service requirements. Thus, servers featured by CPU capabilities are exposed as pools of virtual computing service components (i.e., VMs with assigned CPU cores), while switches featured by bandwidth capacity are exposed as a pool of data delivery service components (i.e., virtual networks with assigned bandwidth) connecting VMs.

On the other hand, the *Service Control & Delivery* function supports a Service API for the applications to issue (composite) service requests while specifying requirements using application-level semantics (e.g., computational demand in terms of number of CPU cores for VMs, bandwidth demand or maximum delay tolerated for VM-to-VM data delivery) instead of technology-specific parameters (i.e., packet queue specification with priority value). While elaborating service requests, the *Service Control & Delivery* function may also perform policing functions (e.g., access control) and trigger (re-)provisioning actions.

The processing of the service requests triggers the *Selection & Composition* function in charge of picking up the proper combination of resources designed to host computing and data delivery service components throughout the DC while meeting requirements specified in the request. The selection and the composition are performed according to a specified strategy while taking into account the operational data of resource capabilities.

The *Monitoring and Registration* function coordinates the background collection of operational (i.e., monitoring) data related to DC resources (i.e., servers and switches) as well as to service components to drive selections while preventing service degradations due to resource contentions. Collected monitoring data can be also elaborated to derive estimations or longer-term cycles about utilization of resources. The overall set of collected information is stored in the *Network and Host Information Databases* both serving as registry of resource status including the CPU utilization of computing service components and throughput at data delivery service components, respectively. Such data are formatted according to a service-oriented data model enabled by the underlying abstraction and virtualization function.

The *Coordinated Configuration and Provisioning* function coordinates the provisioning of each service component in the respective domain, e.g., VM set-up with specified CPU cores at servers, VM-to-VM data delivery path set-up with specified bandwidth along a specified sequence of switches/links. The provisioning phase finally triggers resource-specific configuration directives to the related control and management systems (i.e., OF controller, VM Manager) through the corresponding Southbound interface (e.g., OpenFlow, SSH).

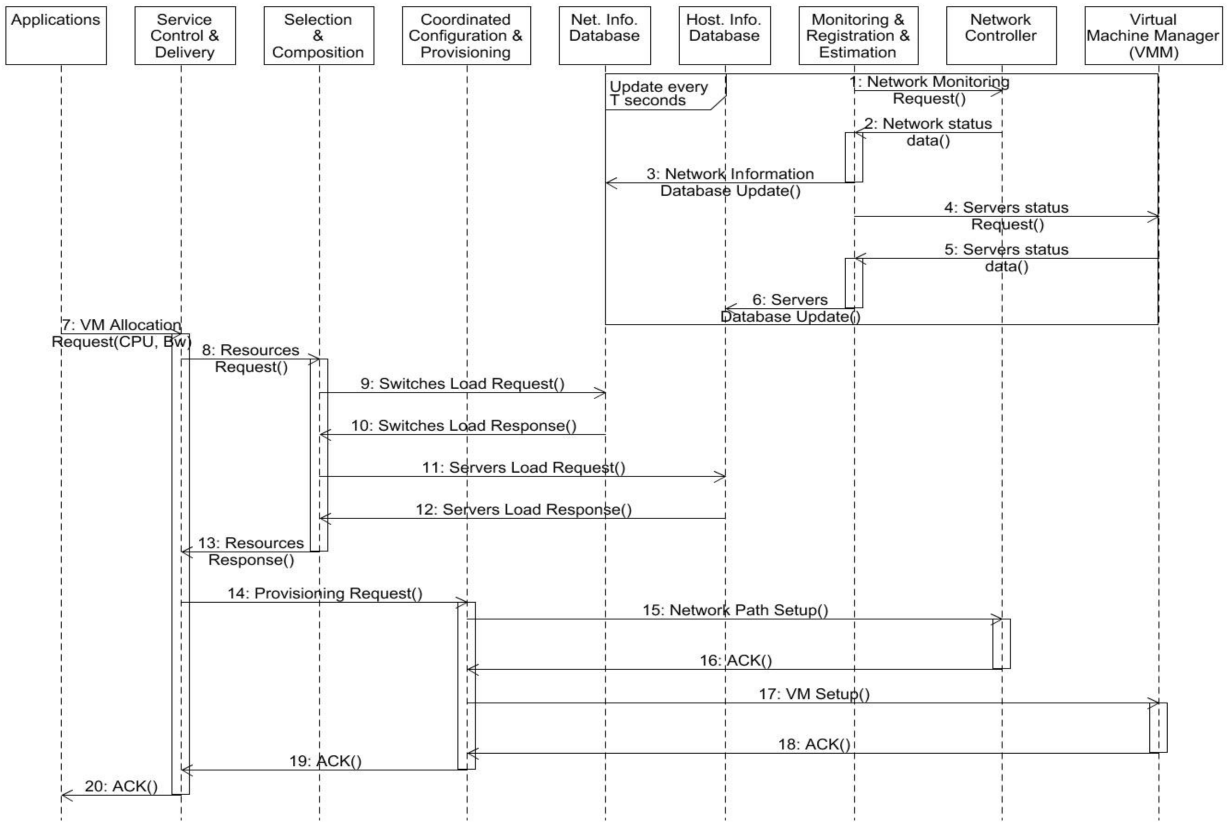
Applications
Service Control & Delivery
Selection & Composition
Coordinated Configuration & Provisioning
Net. Info. Database
Host. Info. Database
Monitoring & Registration & Estimation
Network Controller
Virtual Machine Manager (VMM)
Update every T seconds
1: Network Monitoring Request()
2: Network status data()
3: Network Information Database Update()
4: Servers status Request()
5: Servers status data()
6: Servers Database Update()
7: VM Allocation Request(CPU, Bw)
8: Resources Request()
9: Switches Load Request()
10: Switches Load Response()
11: Servers Load Request()
12: Servers Load Response()
13: Resources Response()
14: Provisioning Request()
15: Network Path Setup()
16: ACK()
17: VM Setup()
18: ACK()
19: ACK()
20: ACK()

Fig. 2: SDN-based Orchestration system for cloud DCs: sequence diagram

In Fig. 2 the sequence diagram of orchestration workflows is reported. The box on the top depicts the workflow related to the background collection of monitoring information coordinated by the *Monitoring and Registration* function. Every T seconds, the *Monitoring and Registration* function asks the OF Controller and the VM Manager for the monitoring data related to the switches and servers, respectively. Then, it updates the databases after data elaborations to derive long-term trends on resource utilization. Below, the workflow shows the main interactions for processing a VM allocation request as result of the orchestration process. The request is delivered to the *Service Control & Delivery* functionand includes the VM requirements specified in terms of CPU as well as bandwidth demands (step 7). As part of the processing of the request, the *Service Control &Delivery* function asks the *Selection & Composition* function to return the combination of resources, i.e., servers and switches, able to host the required service (step 8). While selecting the proper combination of resources, the *Selection & Composition* function asks the Network and Host Information Databases to return operational data related to resource utilization (step 9-12). When the decision is returned back (step 13), the *Service Control & Delivery* function asks the *Coordinated Configuration and Provisioning* function to coordinate the provision of the computing and data delivery service components (step 14). In turn, the *Coordinated Configuration and Provisioning* function relies on the OF controller (step 15-16) and to the VM Manager (step 17-18) to set-up the VM and the network path to route the traffic exchanged by the VM according to the specified requirements. Finally, acknowledgements are sent back with the operation outcomes (step 19-21).

In this work, the selection and composition of service components is carried out based on estimations and predictions about the load of switches derived from collected OF statistics. In the following two subsections, we describe the selection and composition strategies along with the technique used to estimate the network switch/link load.

2. Resource Selection and Composition Strategies

In cloud DCs, two sets of physical resources are deployed (i.e., servers and network switches) and are typically interconnected through a tree-like 3-layer network topology. As shown in Fig. 3, the edge layer provides physical connectivity to the servers in the data centers, while the aggregation layer connects together edge layer switches. Similarly, aggregation layer switches are connected at the core level of networking. Basically, at all layers multiple redundant links connect together pairs of switches at all layers thus, enabling high availability at the risk of forwarding loops.

In this work, a couple of heuristics are conceived which firstly select a server to host the VM and then a network forwarding path throughout DC switches for data delivery to/from the selected server and from/to possible destinations of the VM data, i.e., a server or a user at its premise. In the following, we refer to such heuristics as Server-Driven (SD) strategies. In this work, without lack of generality, we will consider the selection of a path that connects one selected server to the core switch, throughout the aggregation and edge switches and we will refer to such heuristics as Server-Driven (SD) strategies. This corresponds to the case of data delivery from the selected server, where the VMs are placed, to/from users or servers located in other DCs, i.e., North-South traffic.

A case of selection can be described with reference to Fig. 3. Firstly, among the whole set of servers, SD selects one server (red dashed-line circle) with enough available computational resources (i.e., CPU cores) to host the VM. Then, at each level of the network, one switch and one link are selected to provide data delivery (blue dashed-line circle). More specifically, the selection of the server constrains the selection of the edge switch/link since only one edge switch is generally connected to each server through one link (i.e., edge link). Then, a candidate aggregation switch and, correspondently, a link are selected among two possible options (blue dashed-line circle) out of overall 8 switches/16 links at the aggregation level. The final decision on the candidate aggregation link is done after checking the available bandwidth at the link and available throughput at the switch the link is connected to. Similarly, the selection of the core links is constrained by the selection of the aggregation switch/link. Within each set of resources (i.e., server and switches/links at each network layer), the actual selection among the possible options depends on the criteria that is adopted. In this regard, two bin packing heuristics have been considered. The First Fit (FF) criterion just goes for the first-indexed available resource provided that it has enough availability to meet the requirements of the request. The Worst Fit (WF) criterion chooses the less loaded resource, i.e., the resource that has most remaining availability. As result, two SD strategies are considered when coupled with FF and WF selection criteria, i.e., SD-FF and SD-WF.

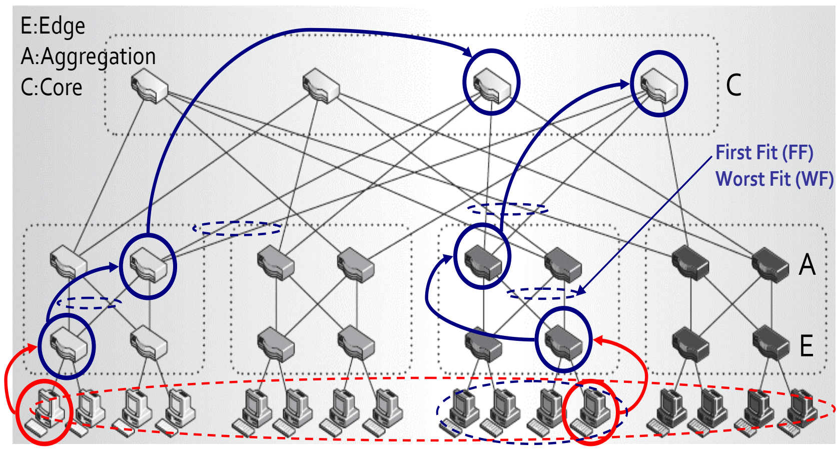

Fig. 3: Example of Server-Driven (SD) resource selection and composition in cloud DCs

Algorithm 1 describes the steps performed by the orchestration process to allocate a VM while a Server-Driven algorithm is considered. Through a VM placement strategy (e.g., FF, BF, WF), a candidate physical server is firstly selected for possibly placing the VM (line 2-7). Then, the estimated traffic load of the edge switch directly connected to the candidate server is retrieved (line 8). In case of overload, the candidate physical server is discarded (line 12) and another candidate server is searched excluding the ones directly connected to the overloaded edge switch (line 13-18). Otherwise, the edge switch is added to the candidate node list (line 10). Then, the switch selection strategy and the estimation of the traffic load are repeated to select the aggregation switch (line 20-31) and the core switch (line 32-43).

Algorithm 1 Server-Driven Algorithm

```
Input:
  H={H_n | n=1,2,…N} //set of N physical servers
  ES={ES_k| k=1,2,…K} // set of K edge switches
  AS={AS_l| l=1,2,…L} // set of L aggregation switches
  CS={CS_m| m=1,2,…M} // set of M core switches
  Th // Traffic load threshold
  VM_PP // VM placement policy (e.g., FF, BF, WF)
  SWITCH_SP //Switch selection policy (e.g., FF, BF, WF)
  VM_Req(C), C={C_x| x=1,2,…X} // VM request with X resource demands
Output:
  CN = {ES(H_n), AS_l(ES(H_n)), CS_m(AS_l(ES(H_n))) H_n} // List of the candidate nodes at the end of selection and composition
Begin procedure
 1: CN = {Ø} // Initialize set of candidate nodes
 2: run VM_PP(VM_Req(C)) // find H_n for the VM with resource demands C
 3: if (∃H_n) then
 4:    CN = CN ∪ H_n  // add H_n to the candidate nodes list
 5: else
 6:    go to End Procedure//No solution found
 7: end if
 8: get_load_data(ES(H_n)) // retrieves the estimation of traffic load of the edge switch connected to H_n.
 9: if ((load(ES(H_n))/AggregateThroughput(ES(H_n))) <T_h) then  //compare traffic load of (ES(H_n)) with the threshold
10:    CN = CN ∪ ES(H_n) // add ES(H_n) to the candidate nodes list
11: else
12:    H = H\H(ES(H_n)) //delete from H all servers connected to ES(H_n)
13:    CN = {Ø} //empty the candidate node list
14:    if H = {Ø} then
15:       go to End Procedure//No solution found
16:    else
17:       go to Line 2
18:    end if
19: end if
20: run SWITCH_SP (AS(ES(H_n))) // select aggregate switch connected to ES(H_n)
21: if ( ∃ AS_l(ES(H_n)) then
22:    CN = CN ∪ AS_l(ES(H_n)) // add AS_l(ES(H_n))to the candidate nodes list
23: else
24:    H = H\H(ES(H_n)) //delete from H all servers connected to ES(H_n)
25:    CN = {Ø} //empty the candidate node list
26:    if H = {Ø} then
27:       go to End Procedure//No solution found
28:    else
29:       go to Line 2
30:    end if
31: end if
32: run SWITCH_SP (CS(AS_l(ES(H_n))) //select core switch connected to AS_l(ES(H_n))
33: if( ∃ CS_m(AS_l(ES(H_n))) then
34:    CN = CN ∪ CS_m(AS_l(ES(H_n)) // add CS_m(AS_l(ES(H_n))to the candidate nodes list
35: else
36:    H = H\H(ES(H_n)) //delete all servers connected to ES(H_n)
37:    CN = {Ø} //empty the candidate node list
38:    if H = {Ø} then
39:       go to End Procedure//No solution found
40:    else
41:       go to Line 2
42:    end if
43: end if
End procedure
```

3. *SDN-based network monitoring and load estimation in cloud DCs*

For the purpose of orchestration, a background monitoring is necessary to obtain operational data related to the utilization of the servers and of network switches/links to support decisions on the admission of service requests as well as on the selection and composition of resources. Within SDN, the OF protocol allows for the live-manipulation of the flow tables of OF-enabled switches as well as the retrieval of real-time data statistics for monitoring purpose [39]. As traffic data flows throughout a switch, a set of counters is updated to collect the following statistics:

- per-flow received/transmitted packets
- per flow received/transmitted bytes
- per-flow duration (with nanosecond granularity)
- per-port received/transmitted packets
- per-port received/transmitted bytes
- per-port received/transmitted errors

In cloud DC networking, a reliable assessment of traffic load (i.e., throughput) is needed to prevent concurrency issues and avoid significant impacts not only on the acceptance rate of VM allocations but also on the Quality of Service perceived by the user (i.e., network latency). In this regard, two performance indexes, i.e., the overall traffic load and the per-port traffic load (i.e., link load) of the switch are considered. Since OF statistics do not provide the per-port throughput, this can be obtained by periodically retrieving the amount of bytes received/transmitted at a specified port and, then, by dividing by the duration of the polling interval. Similarly, the average throughput of each flow can be computed by dividing the per-flow amount of bytes received/transmitted by the duration of each flow [40].

The assessment of traffic load in cloud DCs is a pretty challenging task due to the high dynamicity of traffic flows generated by VMs as they elastically start, expand, reduce or migrate. Moreover, a plathora of applications run in DCs that generate different traffic patterns and use network resources concurrently. Traditionally, the majority of network traffic flows are short lived, require little bandwidth (less than 5 Mbps) and last few seconds or minutes. Those are related to bursty applications, e.g., web browsing, search applications, interactive services, and are referred to as *mice* flows. On the contrary, the majority of the packets and consequently of the consumed bandwidth is related to a small number of long-lived flows that require high amounts of bandwidth (tens to hundreds of Mbps). These highly persistent flows have a duration ranging from several minutes to a number of hours and they address transfers of large blocks of data, in case of backups, data migrations and Big Data applications (e.g., MapReduce/Hadoop). In the following, they are referred to as *elephant* flows [21][22]. Moreover, traffic analyses have also revealed the presence of a *hybrid* traffic pattern identified by a significant elasticity in bandwidth demands and moderate flow lifetime (i.e., tens of minutes) [41].

Due to the high variability of traffic patterns, current values of throughput are also highly variable over time. In such a case, an estimation of the throughput would be desirable. Such estimation should be obtained not only from the instantaneous value of the traffic load generated by the VMs but also from the historical values generated over time. Moreover, the balancing between the instantaneous and historical values is also fundamental to obtain a reliable load estimation. In fact, outweighing the instantaneous values allows for responding to fluctuations in a quicker way, whereas giving more weight to the historical values allows for smoothing out short-term fluctuations and highlighting longer-term trends or cycles. The threshold to be set between short-term and long-term trends depends on the actual composition of the overall DC traffic, generally given by a mixture of mice, elephant and, in case, also hybrid flows. Moreover, such balance needs to be also correlated with the granularity of the polling interval, i.e., the time elapsed between two consecutive measurements. In fact, having a long polling interval might miss the burst of traffic flows whereas a short polling interval could increase the processing load at the SDN orchestrator and worsen the performance [40].

For traffic load estimations we use the same formula adopted for the TCP Round Trip Time estimations [42] where an average is made using a number of historical measurements according to *Exponential weighted moving average* scheme where influence of past sample decreases exponentially fast. The estimated traffic load (i.e., *ETL*) at time T over a bidirectional link l connecting two switches (s_i and s_j) is computed according to the following formula:

$$\begin{aligned} ETL(l_{ij},T) &= (1-\alpha)*ETL(l_{ij},T-\Delta T)+\alpha*ITL(l_{ij},T) \\ ETL(l_{ji},T) &= (1-\alpha)*ETL(l_{ji},T-\Delta T)+\alpha*ITL(l_{ji},T) \end{aligned} \tag{1}$$

where *ITL* is the instantaneous traffic load at the time *T*, ΔT is the polling interval, *ETL(T - ΔT)* is the traffic load estimated one step before, l_{ij} and l_{ji} are "unidirectional links" from s_i to s_j and vice versa, respectively. The term α is used to assign a different weight to a

component over the other, thereby giving rise to different reactivity in following traffic load variations generated by the VMs over the link. The higher is α, the quicker is the estimation in following the variations.
The combination of the selection and composition strategies with the *Exponential weighted moving average* scheme for load estimation is hereafter referred to as orchestration strategies.

IV. Simulation Environment

1. Simulation settings

We have implemented a custom-built, event-driven Java simulator to evaluate the performance of the proposed orchestration process. The simulator supports the construction of any type of data center network topology thus allowing for the creation of a realistic DC environment capable of running under different network loads, traffic conditions and VM placement requirements. More specifically, the simulator reproduces two different kinds of Clos-based DC network topologies, i.e., Fat Tree [43] and VL2 [44], shown in Fig. 4 and Fig. 5, respectively, which are the most commonly used in DCs and in the related works in the literature [25][48][49]. The Fat Tree topology is composed of 128 servers. Each 4 servers are connected to 1 edge switch at 100 Mbps speed. Moreover, the 32 Edge switches are further interconnected to 32 aggregation switches through 1 Gbps links. Finally, these 32 aggregation switches are connected to 16 core switches through 1 Gbps links. The VL2 topology is composed of 320 servers, 16 edge switches, 8 aggregation switches and 4 core switches. The links capacity between the servers and the edge switches is equal to 100 MBps whereas the links capacity between the edge switches and aggregation switches is equal to 1Gbps and the links capacities between aggregation switches and core switches is equal to 1 Gbps. In both topologies, each server is featured by a computing capacity given by 4 cores. The specific number of nodes and the link capacities have been decided to have the same number of links in the two topologies and, thus, to make them comparable.

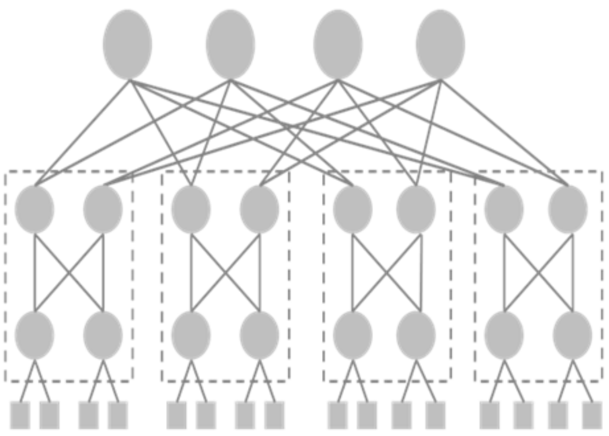
Fig. 4 Fat Tree Topology

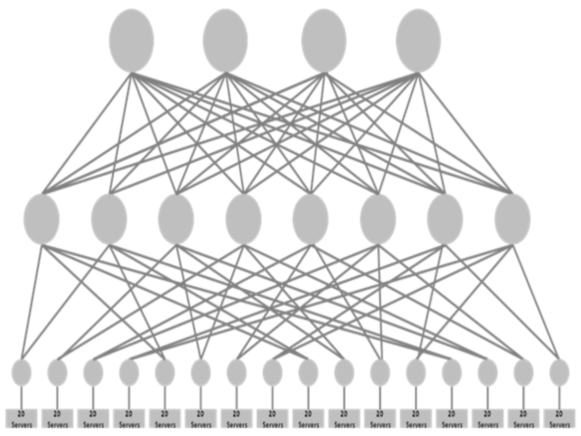

Fig. 5 VL2 Topology

Regarding the traffic generated by the allocated VMs, we distinguished three types of traffic, i.e., mice, elephant and hybrid. The characteristics of such traffic flows were found in literature obtained from measurements performed in real data centers [21].
The VM allocation requests arrive following a Poisson distribution characterized by an inter-arrival and holding times exponentially distributed with an average of 1/λ and 1/μ, respectively. 1/μ is fixed according to the specific traffic pattern while 1/λ varies in the range [1.3sec, 7600sec] to achieve the desired load, which is computed as λ/μ and expressed in Erlang. In particular, the simulations considered traffic in the range [50-3000] Erlang with a step of 500 Erlang. As described in Section. III.4, according to the traffic generated, we distinguish three types of VMs each of which characterized by a different service time. The long lifetime VM instances, generating elephant traffic have an average service time 1/μ equal to 42 hours, the short-lived ones corresponding to the mice traffic are characterized by 1/μ equal to 13 minutes. Finally, the average instances generate hybrid traffic with1/μ equal to 85 minutes.

Each VM allocation request specifies the computational power of the VM (*VM-cpu*), expressed in number of cores, and the required data rate (*VM-traff)* of the traffic generated to/from the VM, expressed in Mbps. Such values are used for selecting resources, i.e., a server hosting the VM and a network path throughout the edge, aggregation and core switches. Based on the availability of resource capabilities, i.e., computational power at switches and available throughput at switches, the admission of the VM allocation is decided. *VM-cpu* values are uniformly distributed in the range [0, 4] cores with a step of 0.1. *VM-traff* values are uniformly distributed in the range [1, 5] Mbps with a step of 1 for *mice* traffic, [80, 100] Mbps with a step of 5 for *elephant* traffic and [10, 80] Mbps with a step of 5 for the *hybrid* traffic pattern.

VM-traff value is also used for reproducing the actual load of interconnection links during the simulation. Specifically, *VM-traff* is used for generating the VM traffic profiles so as to simulate the traffic load along the network forwarding path selected for a specific VM allocated in the selected server. In this work, we assume that a VM generates a variable traffic profile and that the traffic variations occur with a granularity of ΔT seconds. Then, at the beginning of every ΔT interval, for each VM, a traffic value is randomly generated with a uniform distribution in the range of [0**VM-traff*, *VM-traff*] for *mice* traffic, [0.8**VM-traff*, *VM-traff*] for *elephant* traffic and [0.5**VM-traff*, *VM-traff*] for *hybrid* traffic. The *ITL* at each network link is given by the sum of all VM traffic loads admitted to that link, also changing with a granularity of ΔT seconds. The available bandwidth of a network link is obtained from an estimation of the overall traffic load according to (1) with $\alpha=0.5$ which means a pretty quick reactivity in following traffic load variations. The *ETL* value is updated every ΔT seconds and polled when the VM admission needs to be decided. In fact, the admission criteria uses such *ETL* for evaluating if the switch port serving the network link would be able to correctly handle the data flows generated by the VM on that link. If the sum of the *ETL* on the link, given by (1), and the incoming *VM-traff* value exceeds a given threshold, i.e. T_h, of link capacity, such link is discarded as candidate component of the network forwarding path. Considering the whole *VM-traff* value in such an admission criteria is a sort of worst case evaluation of the impact of VM traffic load on the link.

It is worth highlighting that the overall resource environment and simulation settings have been set-up such that no blocking occurs at the servers' level and the unique component of the blocking probability is due to the lack of network resources. Indeed, in this paper we focus on the evaluation of the orchestration performance from the network point of view. Moreover, this choice reflects the actual cloud DCs set-up, where network resources are typically more constrained than computational resource capabilities at servers [45].

The numeric values that have been assumed for all the considered parameters in Section.V are summarized in Table I.

Parameters	**Values**
Total number of requests	10000
Edge link speed	100 Mbps
Aggregation link speed	1 Gbps
Core link speed	1 Gbps
1/μ elephant traffic	42 hours
1/μ mice traffic	13 minutes
1/μ hybrid traffic	85 minutes
VM-traffic elephant [Mbps]	[1, 5]
VM-traffic mice [Mbps]	[80, 100]
VM-traffic hybrid [Mbps]	[10, 80]
VM-cpu range [cores]	[0, 4]
α	0.5
T_h	1.0
ΔT	180 sec

Table I. Simulation parameters

2. *Performance metrics*

We defined a number of performance metrics to evaluate the effectiveness of the orchestration process from both the user and the service provider (i.e., DC operator) perspectives.

Firstly, we consider the rejection rate experienced by the VM allocation requests, referred to as *Blocking Probability (BP)*. The rate of blocked request gives a measure on the inability to provision the required service to users, thus causing dissatisfaction to users and lack of revenues to service providers.

The *BP* is important in the performance assessment of the orchestration strategies but it does not give a complete idea on the effectiveness of those heuristics in terms of network resource utilization. For this purpose we evaluate the *Saturation Index (SI)* of the links which is expressed as the percentage of links having the utilization greater than 80% of the overall link capacity. Moreover, we evaluate the *Waste Index (WI)* which is defined as the percentage of links that are utilized under 50% (i.e., underutilized links). Finally, we define the *Degradation Index (DI)* as the percentage of times the links are overloaded which introduces delays in the delivery of the packets. As result, the *DI* accounts of the likelihood of degradations in the data delivery that might be experienced by

users. Such degradation events likely happen due to possible estimation errors derived from the high variability of overall traffic flows.
To provide a quantitative measure of the efficiency of the orchestration process, we also evaluate the economic impact of the heuristics expressed as the potential total gain generated by accepting a certain number of VM allocation requests against the actual bandwidth consumption observed after the allocation of the VMs. The total gain is calculated, as follows, as the sum of the total allocated bandwidth [26]:

$$Gain = \sum_i u * bw(VM_i) \qquad (2)$$

whereas the actual consumption of the bandwidth is expressed as:

$$Consumption = \sum_i u * effect\_bw(VM_i) \; (3)$$

where:

$$effect\_bw = \frac{\sum_i ITL(VM_i)}{i} \qquad (4)$$

and u is fixed and equal to 1 monetary unit per Mb/s.

In such a configuration, the efficiency of the resources utilization will be expressed as:

$$Efficiency = \frac{Consumption}{Gain} * 100 \quad (5)$$

Moreover, in order to evaluate the impact of our monitoring scheme on the performance of the heuristics, we carry out most of the simulations using reasonable values (e.g., 180 s for the polling interval, 0.5 for α), while completing the simulations with a study on how the metrics behave varying both the polling interval and α.

3. Baselines

We used a couple of baselines as a benchmark for evaluating the effectiveness of the proposed orchestration process. More specifically, we implemented two baselines: the first is the *Greedy Node Mapping* (GNM) algorithm presented in [26] and proposed for the purpose of effective virtual network embedding. It is defined as follows: first the most unloaded server is chosen, then a shortest path algorithm is applied to choose the appropriate network forwarding path. The admission control depends on the occupation of the servers and switches that is based on the nominal bandwidth values.
The second baseline, called *No Admission Control* (*No-AC*), reproduces the current practice in DC resources management and is implemented according to the following criteria. Firstly, an available server is selected in a random way, then a path is selected across edge, aggregation and core switches using a preconfigured flow table that specifies for every switch a forwarding path towards the upper/lower layer switch. From the implementation point of view, adopting the preconfigured paths leads to not considering the redundant links for routing data at both the aggregation and core levels which corresponds to ignoring 50% of the available links in the Fat Tree topology and 10% of the available links in the VL2 topology.
The *No-AC* baseline aims at reproducing the current management practices of legacy DCs where load balancing approaches for routing are used, e.g., Equal-Cost Multi-Path (ECMP) [46][47]. Since ECMP wastes on average 61% of bisection bandwidth [18], this baseline, that does not use part of the available network links, can be considered an acceptable approximation of current routing practices in legacy DCs. Moreover, in current practices, in the admission of VM requests the current network status is not considered. Thus, the bandwidth requirements are ignored and requests are admitted in any case irrespective of the availability of network capabilities (e.g., bandwidth). However, the admission without network constraints leads to unfair comparison with proposed strategies, especially in terms of *BP*. For this reason, in the *BP* computations the *No-AC* criteria is slightly revised to include an artificial admission criteria. The request of VM allocation is admitted if one server is available and if the set of switches along the pre-configured network forwarding path are able to assure required data delivery without degradations. Thus, in the admission criteria at the level of network resources the threshold has been set to 1 (i.e., T_h=1) so as to include in the computation of the BP the requests that, if admitted, would cause the overload of preconfigured network paths (i.e., severe service degradation) thereby making this baseline comparable with the proposed orchestration strategies.

V. Performance Evaluation

In this section we present a thorough set of simulation results to evaluate the performance of our proposed orchestration process. We compare the orchestration strategies with the baselines considering both the Fat Tree and VL2 topologies and different scenarios of traffic patterns (i.e., hybrid traffic, mixed traffic). Finally, we present the effect of the monitoring parameters on the orchestration process. For each specific scenario, the simulations have been carried out independently for multiple iterations and the results are averaged over these iterations. All the results are reported with the confidence interval at the 95% confidence level. The error bars are so small that they are imperceptible on the graphs while being a synonym of the high accuracy of the results.

1. Traffic Pattern: Hybrid

In this subsection we show the performance of our SDN orchestrator while considering only the hybrid traffic.

Fig. 6 plots the *BP* of the orchestration strategies with respect to the baselines while considering the VL2 topology as a function of the actual network load expressed in Erlang. Results show that the proposed heuristics outperform the baselines with *No-AC* presenting the worst results and *GNM* offering a *BP* that is slighly higher (about 10%) than SD-FF and SD-WF. Specifically, No-AC presents the worst results since a reduced set of resources are actually utilized compared to the others heuristics. In fact, VM requests are admitted in any case and, consequently, degradation events are highly frequent. In the *GNM* case, while a smart criterion is applied for the selection of the server (i.e., the most unloaded) similarly to the proposed SD-WF, a random selection is done for the network path. In fact, *GNM* chooses the paths randomly among a set of possible paths with the same number of hops, while SD-FF and SD-WF select links, and thus paths, by applying clever policies as for the servers selection leading to a most efficient selection. Most importantly, SD-FF and SD-WF base the selection on the current load of switches which is estimated from the monitoring data collected in real-time. Instead, GNM derives the load of switches from the nominal values of bandwidth granted in the previous selections. Finally, no significant differences are noted between SD-FF and SD-WF since the number of edge switches in the VL2 topology is high which restricts the effect of the policy on the blocking rate.

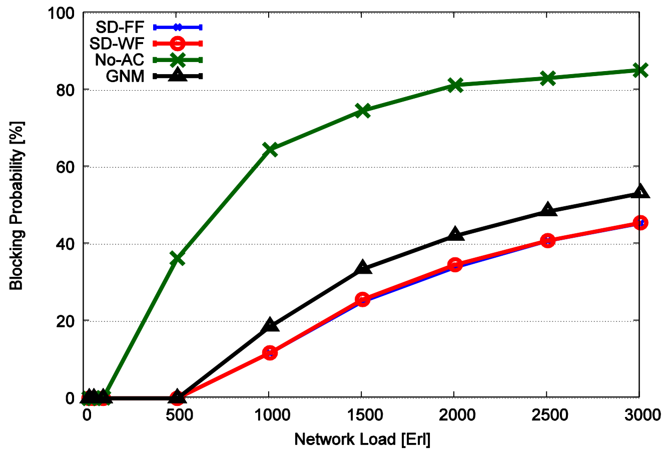

Fig. 6: Blocking Probability -VL2 Topology

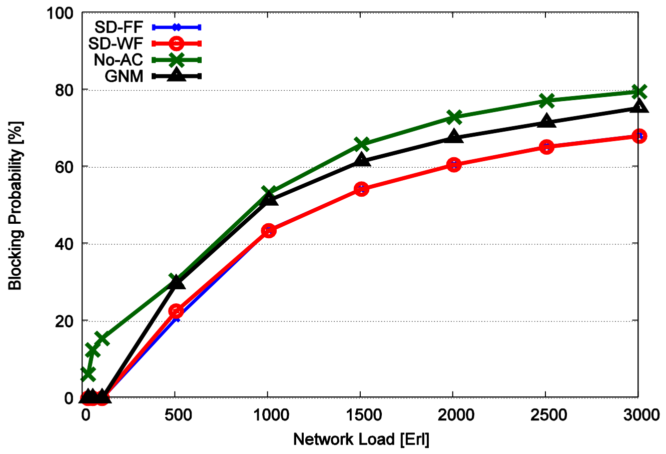

Fig. 7 Blocking Probability – Fat Tree Topology

Fig. 7 shows the *BP* of the SD-FF and SD-WF heuristics with respect to the baselines in case of Fat Tree topology. The same trends are experienced as in the case of VL2 topology with the proposed algorithms still outperforming the baselines. However, a reduced improvement emerge, i.e., from 2% to 10% in the best cases. In fact, while the *BP* for *No-AC* is almost unchanged, the BP for the GNM, SD-FF and SD-WF is much higher with respect to the VL2 topology (around 20% more). This is mainly due to the fact that the number of edge links is higher in the VL2 topology than in the Fat Tree (320 against 128). Since the CPU blocking is nullified and the main blocking is encountred at the edge links level, having more edge links increases the probability to find more available resources and then decreases the blocking. Finally, between SD-FF and SD-WF, the SD-WF presents slightly a lower rejection rate especially at low loads. This is mainly due to the fact that the scattered utilization of the first selected resource (i.e., servers) reduces the load at the network level and improves the overall blocking probability.

Fig. 8 presents the Saturation Index (SI) in the VL2 topology as a function of the network load after the reception of 7000 requests. We chose this value in order to plot the utilization of the links while a certain dinamicity is still observed and other VMs are still to be allocated and released. In SD-FF and SD-WF cases, we observe that the percentage of links utilized above 80% of their capacity increases by increasing the network load and stabilizes after 1000 Erlang. In fact, receiving more requests in a small interval of time rapidly saturates the links. Such behaviour is confirmed by the increase in the blocking probability as previously shown in Fig. 6. We notice also that, at low loads, the SD-WF heuristic has a lower *SI* with respect to SD-FF thanks to a balanced distribution of the load among all the available links.

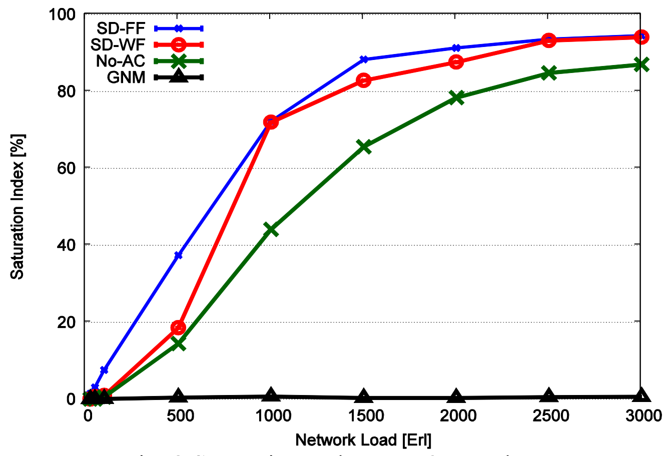

Fig. 8 Saturation Index - VL2 Topology

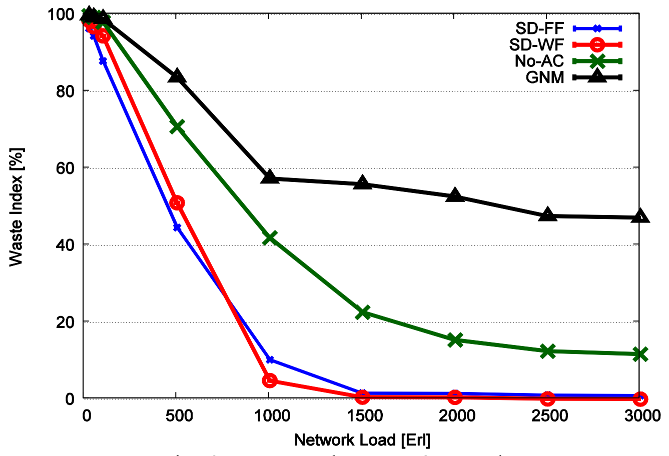

Fig. 9 Waste Index - VL2 Topology

At high loads, almost all the links are saturated and no difference between the two heuristics is noticed. Moreover, we can notice that SI is higher in the SD-FF and SD-WF cases with respect to the baselines. This trend can be explained as follows. SD-FF and SD-WF operate based on a real-time monitoring of the traffic and estimation of load of network links/switches, which fosters the allocation of new VMs and then a higher utilization of the links. Such monitoring is not exploited in the baselines where the assessement of the links utilization is not carried out in the admission control. Specifically, in case of *GNM* the admission control is based only on the requested bandwidth against an assumptive load derived from the nominal bandwidth of previously accepted requests. While in the *No-AC* case a percentage of links are saturated due to lack of network availability check in the admission control, in the case of *GNM* the *SI* is even nullified. In fact, the assumption on network load induces an important waste of the resources.

Fig. 9 plots *WI* as a function of the network load in the VL2 topology. According to *SI* trends for SD-FF and SD-WF, results show that for network loads from 0 to 1000 Erlang the links are increasingly loaded with *WI* decreasing from 90% to 10% while decreasing below 2% at higher loads (above 1500 Erlang). Even though there are no significant differences among SD-FF and SD-WF, the WF case presents the lowest *WI*. In fact, WF tends to make the utilization more evenly distributed across all the links which slightly decreases both its saturation and waste indexes. The *No-AC* baseline shows a *WI* almost 20% higher than SD-FF and SD-WF despite it allocates VMs on preconfigured paths that are unconditionally selected even when they are saturated. In the VL2 topology, almost 10% of the links are not used which is confirmed by the value of WI at high loads. Finally, *GNM* baseline shows a *WI* higher than SD-FF and SD-WF since the admission is based on the network load that is assumed from the nominal bandwidth of previously accepted requests and not on the actual generated traffic, which increases the waste and makes it around 50% for high loads.

Fig. 10 presents the *SI* in the Fat Tree topology as a function of the network load after the reception of 7000 requests. In SD-FF and SD-WF cases, the trend is similar to the VL2 topology: SI increases when the network load increases then it stabilizes after 500 Erlang because of the overload of the links. However, we notice that *SI* is much lower in the Fat Tree topology with respect to the VL2 (35% against 90% respectively, for high loads) as result of a lower acceptance rate, i.e., higher *BP*, as shown in Fig. 6 and Fig. 7. The *SI* for SD-FF and SD-WF is still higher than the GNM baseline for the same reasons explained for the VL2 topology, while we can notice that *SI* becomes lower with respect to the No-AC baseline. In fact, in the *No-AC* baseline 50% of the links are not used (due to the removal of redundant links at the aggregation and core levels) while for the remianing links no admission control is performed and all the requests are accepted independently from the links status. By increasing the load, almost all the links are overfilled, and accepting new requests results in increasing the percentage of links occupied at more than their capacity (as we see in Fig. 13) which worsens the performance.

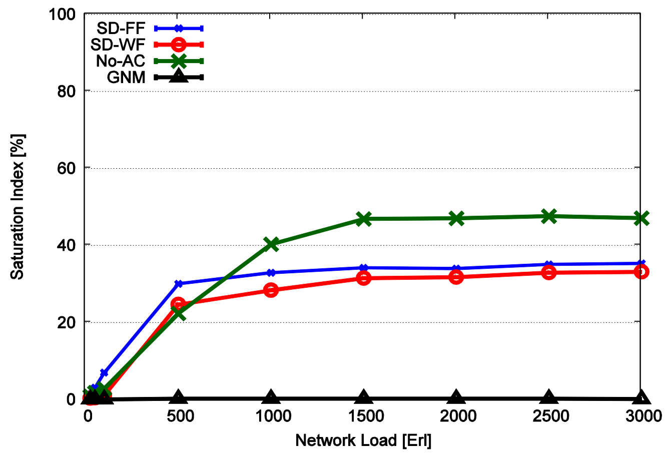

Fig. 10 Saturation Index - Fat Tree Topology

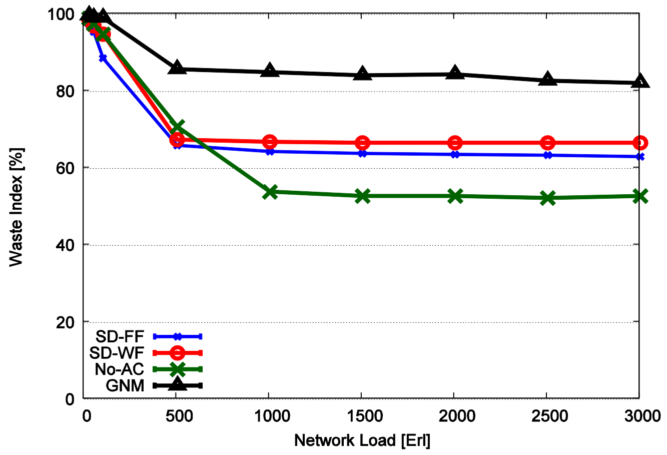

Fig. 11 Waste Index - Fat Tree Topology

Fig. 11 plots the *WI* in the Fat Tree topology after the reception of 7000 requests as in the SI case. Results show that as expected, by increasing the network load, WI decreases, then above 500 Erlang it stabilizes. In case of SD-FF and SD-WF, for loads higher than 500 Erlang, the *SI*, as shown in Fig. 10, is stable around 30% which mainly corresponds to the saturation of the edge links. On the contrary, the aggregation and core links that have a higher capacity are not fully utilized which is confirmed by a relatively high *WI* (around 65%). In the *GNM* baseline, *WI* is still higher (around 20% more) due to the same reasons related to the admission criteria based on assumptive load derived from the nominal bandwidth of previously accepted requests. Finally, in the *No-AC* baseline, the choice of the aggregation and core switches is performed according to a preconfigured flow table that specifies the forwarding paths throughout the switches. Since in case of Fat Tree topology, this corresponds to considering the 50% of the available links at the aggregation and core levels, the *WI* settles to 50%, accordingly. Finally, the different trend with respect to VL2 can be explained as follows. The main blocking component in the VL2 topology is devoted to the saturation of the edge links. Since the number of aggregation and core links is higher in the Fat Tree topology (256 links against 64 links in the VL2 topology), *WI* is also higher in Fat Tree.

The effectiveness of the proposed orchestration process is demonstrated by a slightly higher acceptance rate of service requests against a significant improvement in terms of resource utilization both in VL2 and Fat Tree topologies. However, this can be obtained at the cost of an increased risk of service degradation as a consequence of admission decisions based on load estimations. The following plots give a measure of such a risk which, overall, is kept restrained.

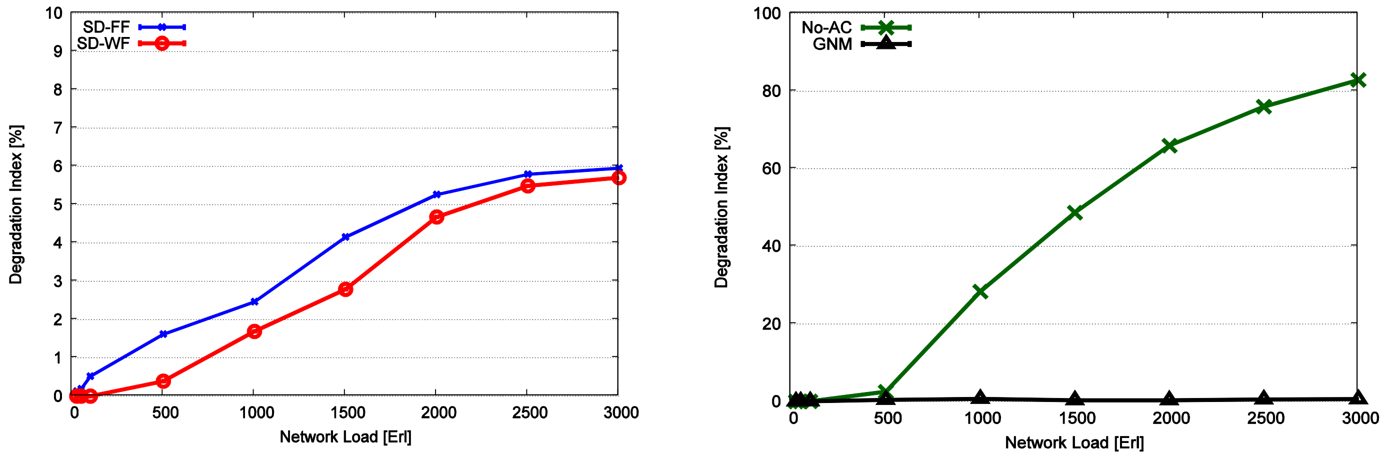

Fig. 12 Degradation Index - VL2 Topology

Fig. 12 plots the *DI* in the VL2 topology while applying SD-FF and SD-WF heuristics (Fig. 12.a) and the baselines (Fig. 12.b). Results show that DI increases by increasing the network load which is expected since more links are overloaded. Moreover, while the SD-FF heuristic tends to saturate some links and then has worse performance, the SD-WF scatters the traffic load on all the available links and thus presents a lower degradation index. However, in both cases, the DI reaches at maximum for higher loads the value of 6%, which is definitely acceptable, considering the significantly lower *BP* and *WI* values. The degradation events are not experienced in the GNM baseline where the admission control is based only on the requested bandwidth against an assumptive load derived from the nominal bandwidth of previously accepted requests and not on the actual traffic measured on the links generated by the allocated VMs. Such assumption guarantees that the requested bandwidth never exceeds the capacity of the links, which nullifies *DI* at the cost of a higher *BP* and *WI* as shown in Fig. 6 and Fig. 9, respectively. In the *No-AC* baseline, DI exceeds 60% for high loads. In fact in this case, no admission control is performed and all the VMs are allocated using preconfigured links, even if the links are saturated. Moreover, we notice that at high loads, the percentage of saturated links coincides with the percentage of degraded links which confirms the deterioration of the performance in the No-AC case.

Fig. 13 plots the *DI* while applying the SD-FF and SD-WF heuristics (Fig. 13.a) and the baselines (Fig. 13.b) in Fat Tree topology. The same trend is observed as in the VL2 topology, where by increasing the network load, *DI* increases due to a higher number of saturated links. Moreover, *DI* is null while applying the GNM baseline whereas it increases exponentially in the No-AC case and at high loads, all the saturated links are degraded. Finally, in the Fat Tree topology *DI* is lower than in the VL2 case (5% lower for SD-FF and SD-WF). In fact, in the Fat Tree more links are present at the aggregation and core levels. Since these links have a higher capacity than the edge links (1Gbps with respect to 100Mbps), more resources are available and then *DI* is lower.

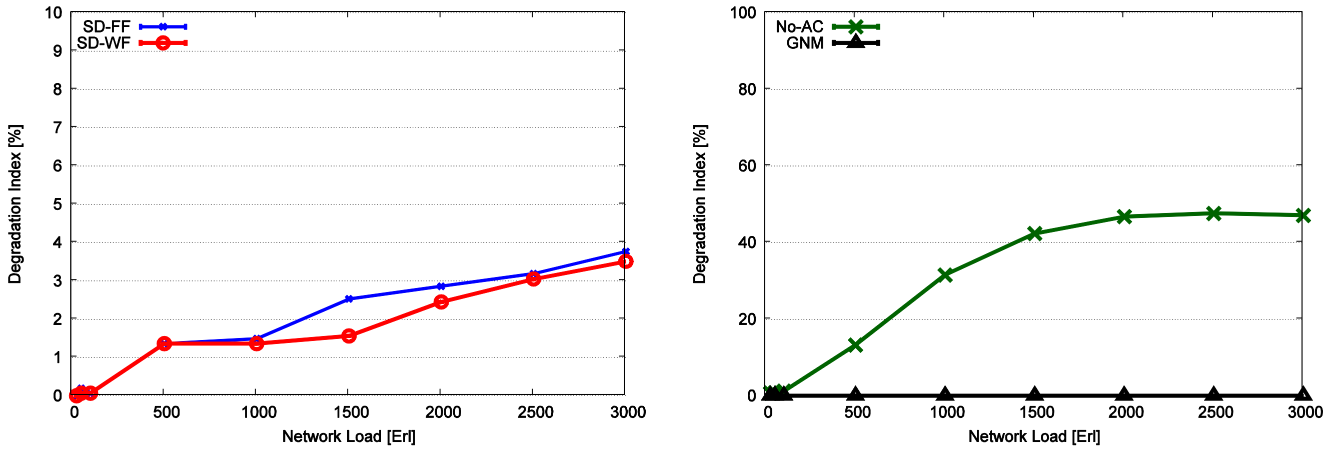

Fig. 13 Degradation Index - Fat Tree Topology

The previous remarks and comparisons can be summarized by assessing the efficiency of the overall orchestration process according to (5) and Table II and Table III for Fat Tree and VL2 topologies, respectively. The efficiency in both cases is reported for low loads (i.e., below 50 Erl) and high loads (i.e., above 2500 Erl). The efficiency is expressed as the ratio of the actual consumption calculated using the real time monitoring of the network over the gain generated by the allocation of the virtual machines. For the Fat Tree topology, results show that SD-FF and SD-WF heuristics present a higher efficiency than baselines thanks to the orchestration process that estimates the actual utilization of the links. Since the acceptance rate is almost the same in the two heuristics, almost no difference is noticed between them also in the efficiency at comparable loads. Switching from low to high loads decreases the efficiency since more links are saturated and less allocation requests are accepted. Such loss is slightly higher in the SD-FF heuristics which tends to saturate more the links. Finally, the baselines have a lower efficiency (around 20% less) which shows the importance of the load estimations and throughput monitoring in taking advantage of the difference between the resources allocated and the resources actually used to accept more requests and then generate more profit.

The same trend as in the Fat Tree topology is observed for the VL2 topology. However, as the gain is higher in the VL2 topology, the efficiency is also higher.

Heuristic	Efficiency - Low Loads	Efficiency - High Loads
SD-FF	75.63	72.77
SD-WF	75.31	73.08
GNM	55.73	51.87
No-AC	55.47	55.51

Table II. Efficiency - Fat Tree

Heuristic	Efficiency - Low Loads	Efficiency - High Loads
SD-FF	77.2	74.48
SD-WF	75.18	74.22
GNM	55.78	53.28
No-AC	54.9	54.6

Table III. Efficiency - VL2

2. *Traffic Pattern: Mixed*

In this subsection we focus the simulations on the VL2 topology and we show the impact of the flows patterns on the performance of the proposed heuristics by considering a spectrum of elephant and mice flows mixtures. We fix the network load to 1000 Erlang which corresponds to a relatively stressed scenario. We also fix the hybrid traffic to 40% and then we vary the percentage of elephant and mice traffics in the remaining 60% of traffic flows [41].
Fig. 15 plots the *SI* after the reception of 7000 requests. As expected, *SI* increases by increasing the percentage of *elephant* flows. In fact, these flows are long lived and require very high amounts of bandwidth which implies the saturation of the available links in the network. At low percentage of elephant flows, SD-WF presents a lower *SI* than SD-FF due to the nature of the SD-WF heuristic that selects the most unused links for the allocation of the new VMs and ends up distributing the load on all the available network resources. Conversely, SD-FF tends to consolidate allocations and, as result, presents a higher number of saturated links. Below 60% of elephant flows, No-AC and *GNM* baselines have also a lower *SI* since they do not consider the actual traffic in the admission control policy and then accept less requests. Finally, when the percentage of elephant flows is high, the No-AC baseline presents the lowest SI values. In fact, as we show in Fig. 16, almost 35% of the links are underutilized. However, out of the remaining 65%, almost 40% of the links are overloaded (see Fig. 17) which results in performance degradations.

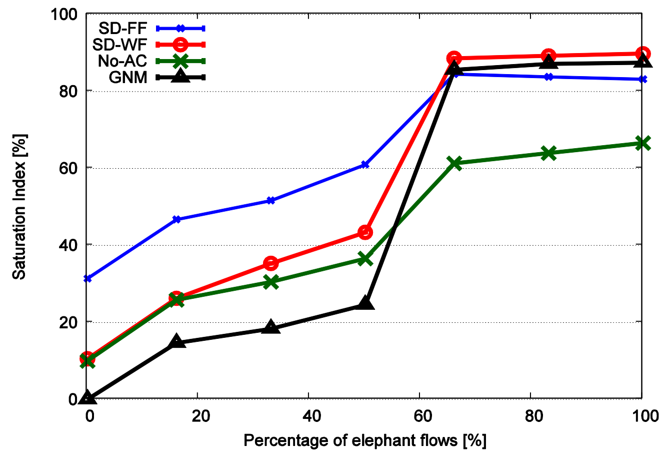

Fig. 15 Saturation Index as a function of the percentage of elephant flows

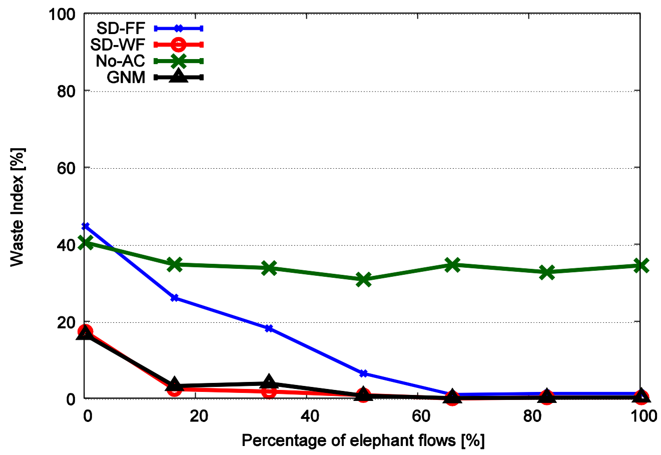

Fig. 16 Waste Index as a function of the percentage of elephant flows

Fig. 16 presents the *WI* as a function of the percentage of *elephant* flows. Results show that by increasing the *elephant* flows the links become more and more saturated and then WI tends to zero as expected. In this case, SD-WF presents a significantly lower *WI* value than SD-FF. As previously explained, this is due to the fact that SD-WF tends to use all the links looking for the most unutilized ones, while the SD-FF policy tends to consolidate the allocated VMs in a limited number of links. As result, SD-WF uses more links in a lightly way, while SD-FF uses less links in more intensive way. Depending on the management purpose, i.e., minimization of the power consumption, load balancing, either of two should be consequently selected, i.e., SD-FF, SD-WF, respectively. While it presents lower *SI* values than the proposed heuristics, the *GNM* baseline has almost the same *WI* values as SD-WF. In fact, *GNM* also applies the WF policy for choosing the server, but unlike the SD-WF heuristic, which still uses the WF policy to choose the network resources, *GNM* selects them randomly. However, since the *elephant* flows are persistent and consume high bandwidth, the benfit from applying the WF policy on the network resources is limited compared to the random selection in *GNM*. Finally, the waste is high and practically constant in the No-AC baseline due to the persistency of *elephant* flows throughout a reduced set of links.

Finally, Fig. 17 plots the *DI* as a function of the percentage of *elephant* flows for SD-FF and SD-WF heuristics. Results show that when only *mice* traffic is present, *DI* is relatively high which is due to the bursty characteristic of the traffic and the frequent change of the bandwidth demand. By increasing the elephant flows, DI starts decreasing and then stabilizes. This is because of the persistency of *elephant* flows that reduces estimations errors (at α and ΔT values considered for simulations) which prevents the orchestrator from accepting other requests to fill the holes generated by the unstable traffic flows. For SD-WF, *DI* is almost null, independently from the percentage of *elephant* flows due to the beneficial effect of the allocations dispersion leading to a lower risk of link overload. While the consolidation of allocations in SD-FF leads to higher values of *DI* especially in case of prevalence of *mice* flows due to the higher estimation errors. As previously explained, DI is also null in case of GNM baseline whereas it keeps increasing in the case of No-AC baseline where no admission control is performed and all the requests are accepted.

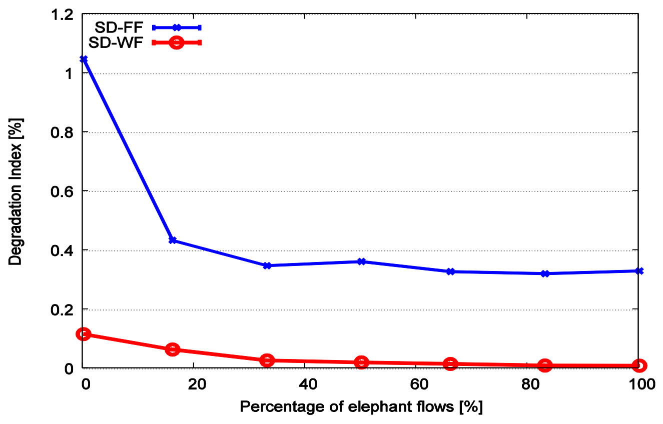

Fig. 17 Degradation Index as a function of the percentage of elephant flows

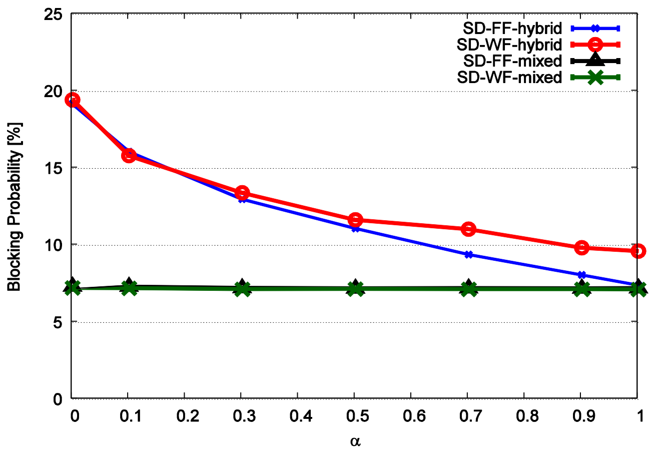

Fig. 18 Impact of α on the Blocking probability - ΔT = 180sec

3. *Effect of the monitoring parameters on the orchestration process*

In this subsection we focus on the effect of the monitoring parameters (i.e., α and ΔT) on the performance of the orchestration process in order to give some design guidelines for the estimation process. For this purpose, we use the VL2 topology, we fix the network load to 1000 Erlang and we consider two traffic patterns: only the *hybrid traffic* and a *mixed traffic*. By mixed traffic we denote a configuration of traffic flows characterized by 20% of *mice* traffic, 40% of *elephant* traffic and 40% of *hybrid* traffic. Such scenario is representative of a realistic traffic pattern in the DCs as reported in [41].

Fig. 18 plots the blocking probability as a function of α. By increasing α more weight is given to the instantaneous component of the traffic. In case of *hybrid traffic*, since the traffic generated by the allocated VMs is within the interval [50%, 100%] of the nominal requested value, by increasing α and giving more weight to the instantaneous values, it is more likely that more bandwidth is considered as available and more requests are accepted, which decreases the *BP*. Moreover, *BP* is slightly lower in the SD-FF policy when α is higher than 0.5. In fact, since the SD-FF policy tends to consolidate the allocated VMs, the bandwidth freed by the difference between the instantaneous generated traffic and the allocated amount is more likely to be higher than in the WF case which improves the acceptance rate. On the contrary, the trend in the mixed traffic is different. In fact, the introduction of the *elephant* flows which give more persistency feature to the aggregated traffic makes lower estimations errors even with low α values.

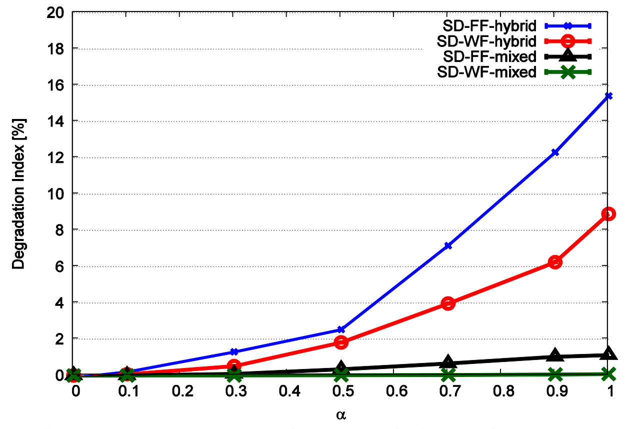

Fig. 19: Impact of α on the Degradation Index - ΔT = 180sec

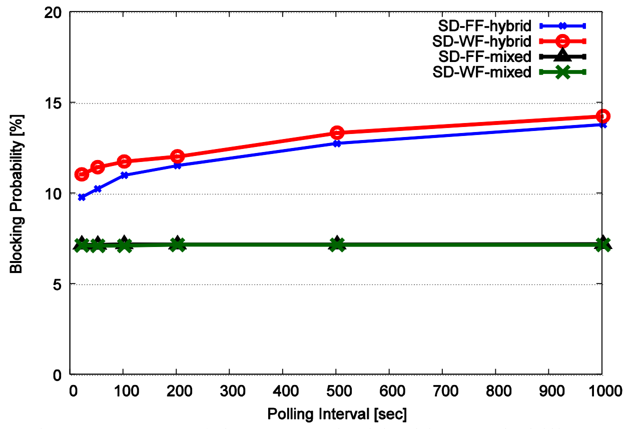

Fig. 20 Impact of the ΔT on the blocking probability - α = 0.5

Fig. 19 shows the impact of the parameter α on the *DI*. When α is lower than 0.5 the previous measurements outweigh the instantaneous value which corresponds to highlighting long-term trends by smoothing out short-term variations. In such a case, the estimations are more smooth, especially for low traffic variability. This explains low *DI* values for both traffic scenarios with higher DI values in case of *hybrid traffic* featured by more variability than in the mixed case. When α gets higher, more weight is given to the instantaneous traffic which corresponds to taking more into account the traffic variations at the cost of the robustness of estimations. As a consequence of a decrease of the *BP* as shown in Fig. 18, *DI* significantly increases in the *hybrid* case. Instead, the increase is limited in the *mixed* case due to the stable nature given by the *elephant* flows. Finally, thanks to its nature to scatter the load on all the available links, the SD-WF policy presents the best results with a *DI* lower than 10% in the hybrid case and almost equal to zero for the mixed pattern.

Fig. 20 shows the effect of the polling interval ΔT on the overall blocking probability. Results show that by increasing ΔT the rejection rate slightly increases in the hybrid case (around 5% more). In fact, a large polling interval corresponds to a large difference between the allocation requests arrival rate (average IAT equal to 5 seconds) and the resources monitoring rate which gives to the orchestrator a less precise view on the actual utilization of the network resources and makes it generally accept less requests. On the contrary, in the *mixed traffic*case, the polling interval does not affect the blocking probability mainly because of the *elephant* flows that give a significant level of persistency to the aggregated traffic.

In Fig. 21 we plot *DI* as a function of the polling interval. When ΔT is null, no estimations of the links load are performed and the allocations are made on the basis of the nominal traffic which nullifies DI as occurred for *GNM*. By slightly increasing ΔT (between 10 and 100sec), the monitoring of the resources is still executed very often and the orchestration process practically follows the variability of the traffic (in this case we intend the decrease in the bandwidth consumption) for allocating new VMs, which makes DI increase. Above 100 seconds and up to 500 seconds, the polling time corresponds to a quite precise estimation of the actual traffic which decreases the degradation due to the overload of the links. However, increasing ΔT too much (i.e., above 500 seconds) corresponds to an *ETL* value updated few times thus leading to admission decision taken on the basis of an inaccurate network status. Mainly, the variable behavior of the traffic will be neglected and *DI* stabilizes around 2%. The traffic variations are also infrequent in the *mixed traffic* due to the *elephant* flows which almost nullifies the *DI*.Finally, in the *hybrid* case, at short polling intervals, we notice a lower DI in the SD-WF heuristic. In fact, for low ΔT values, the orchestrator tends to allocate more VMs (as seen in Fig. 20 where *BP* decreases as ΔT decreases). Since the WF policy scatters the load on all the available links, accepting more requests has a lower probability to overload the links with respect to the FF policy which is reflected by a lower DI.

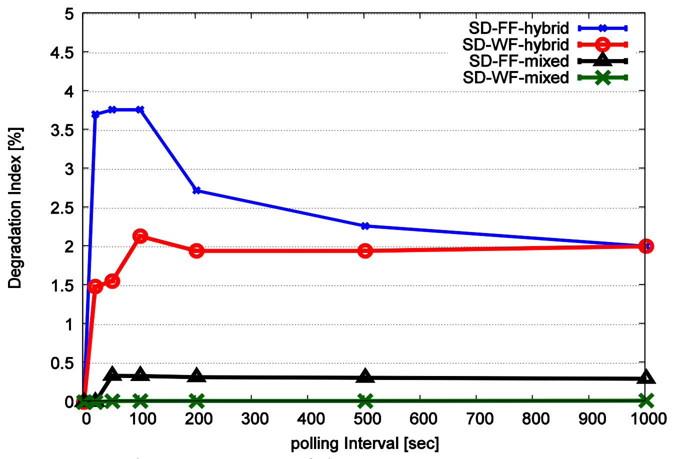

Fig. 21: Impact of the ΔT on DI - α = 0.5

VI. Conclusion

This paper presented an SDN-based orchestration system that allows for a coordinated selection and composition of both network and computing DC services dictated by cloud services requirements. It leverages flow-based OF statistics to provide optimal allocations of VMs across DC servers and of VM delivery paths based on the actual traffic load estimated over DC interconnections. The adaptive orchestration of computing and network services is the result of the synergistic interwork of the following functions: (i) resource selection and composition functions, (ii) coordinated resource configuration and management functions, (iii) monitoring and registration functions of resource status. A set of selection and composition strategies and a traffic estimation scheme have been also proposed and described. The evaluation of the proposed SDN-based orchestration process has been carried out through the definition of a set of heuristics and an estimation scheme. The orchestration strategies have been compared through an exhaustive set of simulations against their capability to address service demands and effective resource utilization. Moreover, they were compared to current management practices in DCs and to similar approaches in literature. Finally, different kinds of topologies and different sets of traffic patterns have been considered.

Results demonstrate the effectiveness and cost efficiency of our proposed strategies that outperform the existing solutions in terms of acceptance ratio, efficiency and resource waste minimization. Thanks to the continuous monitoring and online estimation of the traffic flows, the proposed SDN orchestration process ensures higher rate of VM allocations while guaranteeing a higher resource utilization of both servers and network capacity. Moreover, a lower waste of the resources leads to a higher efficiency of the overall resource capacities resulting in higher gain collected by the DC operators with respect to the approaches currently adopted in DCs. However, all these benefits are possible at the cost of possible degradations of data delivery due to load estimation errors in the orchestration process. From simulations, both in case of VL2 and Fat Tree topologies, the orchestration process presents a degradation rate that is at maximum at 6% (worst case) which can be definitely an acceptable risk given the large set of benefits that are obtained. Moreover, results show that the VL2 topology presents a slightly lower efficiency than in the Fat Tree case with a DI 2% higher at the cost a lower SI and a lower acceptance rate.

Simulations also revealed that in case a majority of the traffic is composed of persistent flows (characterized by a low variability), the SD-FF appears to be more suitable. In fact, despite it tends to concentrate the resources utilization in a limited number of servers/links, there is a lower probability for estimation errors that are likely to result in degradation events. On the other hand, the SD-WF strategy fosters a more distributed usage of the resources which improves the overall acceptance rate and diminishes the degradation and waste indexes. Such heuristic is well adapted for both traffic patterns although it presents a lower saturation rate with respect to SD-FF in case of bursty flows.

Finally, regarding the monitoring scheme, thanks to an adequate polling interval and the consideration of historical traffic, the proposed orchestration process is capable of smoothing out short-term fluctuations, highlighting longer-term trends in flow behaviors and adapting the admission decision according to the estimations of the resulting network load. Results show that a tradeoff between accuracy and computational load is necessary to tune the estimation and monitoring parameters, taking into consideration the traffic patterns and the computational load of the orchestration process. The accuracy resides in a precise estimation of the actual traffic in the network that could be obtained at the cost of a considerable monitoring data to be handled and processed. Depending on the variability of traffic patterns in the DC, a balance can be found to achieve accurate and reliable estimations for that kind of traffic patterns without unreasonably increase the computational load of the orchestration process.

REFERENCES

[1] F. Baroncelli, B. Martini, P. Castoldi, "Network Virtualization for Cloud Computing", In Annales des Télécommunications, 2010.

[2] Y. Zhang, A.J. Su, G. Jiang, "Evaluating the impact of data center network architectures on application performance in virtualized environments," Quality of Service (IWQoS), 2010 18th International Workshop on, vol., no., pp.1-5, 16-18 June 2010.

[3] Web Site, http://www.6wind.com/solutions/data-center-networking

[4] A. Lara, A. Kolasani, B. Ramamurthy, "Network Innovation using OpenFlow: A Survey," IEEE Communications Surveys & Tutorials, vol. 16, Issue: 1, 2014.

[5] J. Shahabuddin, A. Chrungoo, V. Gupta, S. Juneja, S. Kapoor, and A. Kumar, "Stream-packing: Resource allocation in web server farms with a QoS guarantee", HiPC, 2001.

[6] A. Tzanakaki, M.P. Anastasopoulos, S. Peng, B. Rofoee, Y. Yan, D. Simeonidou, G. Landi, G. Bernini, N. Ciulli, J.F. Riera, E. Escalona, J.A. Garcia-Espin, K. Katsalis, T. Korakis, "A converged network architecture for energy efficient mobile cloud computing," in Optical Network Design and Modeling, 2014 International Conference on , vol., no., pp.120-125, 19-22 May 2014.

[7] A.G. Prieto, D. Gillblad, R. Steinert, A. Miron, "Toward decentralized probabilistic management," in Communications Magazine, IEEE , vol.49, no.7, pp.80-86, July 2011.

[8] D. Adami, B. Martini, A. Sgambelluri, M. Gharbaoui, P. Castoldi, A. Del Chiaro, L. Donatini, S. Giordano, "An OpenFlow controller for cloud data centers: Experimental setup and validation", Communications (ICC), 2014 IEEE International Conference on, 1344-1349.

[9] D. Adami, B. Martini, A. Sgambelluri, M. Gharbaoui, P. Castoldi, C. Callegari, L. Donatini, S. Giordano, "Cloud and Network Service Orchestration in Software Defined Data Centers," International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS), 2015.

[10] D. Adami, B. Martini, M. Gharbaoui, P. Castoldi, G. Antichi, S. Giordano, "Effective resource control strategies using OpenFlow in cloud data center," IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), 2013.

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

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

[13] B. Martini, D. Adami, M. Gharbaoui, P. Castoldi, L. Donatini, S. Giordano, "Design and Evaluation of SDN-based Orchestration System for Cloud Data Centers", Communications (ICC), 2016 IEEE International Conference on.

[14] B. Jennings and R. Stadler, "Resource management in clouds: Survey andresearch challenges," J. Netw. Syst. Manage., vol. 23, no. 3, pp. 567–619,2015.

[15] VMware Capacity Planner, http://www.vmware.com/products/capacity-planner/

[16] IBM Workoload Deployer Home Page http://www-01.ibm.com/software/webservers/cloudburst/

[17] S. Metha and A. NEogi, "Recon: a tool to recommend dynamic server consolidation in multi-cluster data centers,", NOMS 2008.

[18] Y. Ajiro and A. Tanaka, "Improving packing algorithms for server consolidation", in Proceedings of the International Conference for the Computer Measurement Group (CMG), 2007.

[19] L. Sanghwan, S. Sahu, "Efficient Server Consolidation Considering Intra-Cluster Traffic," Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE , vol., no., pp.1-6, 5-9 Dec. 2011.

[20] M. Xiaoqiao, V. Pappas, L. Zhang, "Improving the Scalability of Data Center Networks with Traffic-aware Virtual Machine Placement," INFOCOM, 2010 Proceedings IEEE , vol., no., pp.1-9, 14-19 March 2010.

[21] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The Nature of Datacenter Traffic: Measurements & Analysis," " in Proc. IFIP/IEEE International Symposium on Integrated Network Management (IM), 2009.

[22] "SDN Analytics for Elephant Flow marking", Alcatel-Lucent Enterprise Application Note.

[23] Meng Wang; Xiaoqiao Meng; Li Zhang; , "Consolidating virtual machines with dynamic bandwidth demand in data centers," INFOCOM, 2011 Proceedings IEEE , vol., no., pp.71-75, 10-15 April 2011.

[24] M. Mihailescu, A. Rodriguez, C. Amza, "Enhancing application robustness in Infrastructure-as-a-Service clouds," Dependable Systems and Networks Workshops (DSN-W), 2011 IEEE/IFIP 41st International Conference on, vol., no., pp.146-151, 27-30 June 2011.

[25] M.G. Rabbani, R. Pereira Esteves, M. Podlesny, G. Simon, L. Zambenedetti Granville, R. Boutaba, "On Tackling Virtual Data Center Embedding Problem," in Proc. IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), pp. 177–184.

[26] M. Yu, Y. Yi, J. Rexford, M. Chiang, "Rethinking Virtual Network Embedding: Substrate Support for Path Splitting and Migration," ACM SIGCOMM Computer Communication Review, vol. 38, no. 2, pp. 17–29, April 2008.

[27] R. Mijumbi, J.L. Gorricho, J. Serrat, M. Claeys, F. De Turck, S. Latre, "Design and evaluation of learning algorithms for dynamic resource management in virtual networks," Proceedings of. IEEE Network Operations and Management Symposium (NOMS) 2014.

[28] D. Tse, M. Grossglauser, "Measurement-based call admission control: analysis and simulation," in Proc. IEEE INFOCOM 1997, Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution, vol. 3, pp. 981–989.

[29] H. el Allali, G. Heijenk, A. Lo, I. Niemegeers, "A Measurement-Based Admission Control Algorithm for Resource Management in Diffserv IP Networks," in Personal, Indoor and Mobile Radio Communications, 2006 IEEE 17th International Symposium on , vol., no., pp.1-5, 11-14 Sept. 2006.

[30] M. Davy, and et al, "A Case for Expanding OpenFlow/SDN Deployments On University Campuses," GENI Workshop Whitepaper, Jul, 2011.

[31] Theophilus Benson, Ashok Anand, Aditya Akella and Ming Zhang “MicroTE: Fine grained Traffic Engineering for Data Centers”, CoNEXT 2011, Japan.

[32] Rastin Pries, Michael Jarschel, Sebastian Goll, “On the Usability of OpenFlow in Data Center Environments”, Workshop on Clouds, Networks and Data Centers collocated with the IEEE International Conference on Communications (ICC 2012), Ottawa, Canada, June 2012.

[33] Walter Cerroni, Molka Gharbaoui, Barbara Martini, Aldo Campi, Piero Castoldi, Franco Callegati, Cross-layer resource orchestration for cloud service delivery: A seamless SDN approach, Computer Networks, Volume 87, 20 July 2015, Pages 16-32.

[34] B. Martini, M. Gharbaoui, P. Castoldi, "Cross-Functional resource orchestration in optical telco clouds," ” in Proc. International Conference on Transparent Optical Networks (ICTON), 2015.

[35] Onisick, Joe, "Private Cloud Automation, Orchestration, And Measured Service," Network Computing, June 23, 2011.

[36] "Effects of virtualization and cloud computing on data center networks," HP, October 2011.

[37] https://www.opendaylight.org/

[38] http://www.xenproject.org/

[39] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “Openflow: enabling innovation in campus networks,” SIGCOMM Comput. Commun. Rev., vol. 38, pp. 69–74, March 2008.

[40] van Adrichem, N.L.M.; Doerr, C.; Kuipers, F.A., "OpenNetMon: Network monitoring in OpenFlow Software-Defined Networks," in Network Operations and Management Symposium (NOMS), 2014 IEEE , vol., no., pp.1-8, 5-9 May 2014 doi: 10.1109/NOMS.2014.6838228.

[41] C. Peng, M. Kim, Z. Zhang, H. Lei, “VDN: Virtual Machine Image Distribution Network for Cloud Data Centers,” in Proc. IEEE INFOCOM 2012, pp. 181–189.

[42] V. Jacobson. Congestion avoidance and control. In Proceedings of ACM SIGCOMM ’88, pages 314–329, August 1988.

[43] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, “PortLand: a scalable fault-tolerant layer 2 DC network fabric” in Proc. ACM SIGCOMM 2009 conference on Data communication, ser. SIGCOMM ’09, 2009, pp. 39-50.

[44] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, “VL2: a scalable and flexible DC network,” in Proc. ACM SIGCOMM 2009 conference on Data communication, ser. SIGCOMM ’09, 2009, pp. 51–62.

[45] K. Bilal, S.U.R Malik, S.U. Khan, A.Y. Zomaya, "Trends and challenges in cloud datacenters," in IEEE Cloud Computing, Year: 2014, Volume: 1, Issue: 1.

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

[47] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, A. Vahdat. “Hedera: Dynamic flow scheduling for DC networks”, in Proc. of NSDI, CA, 2010.

[48] W. Fang, X. Liang, S. Li, L. Chiaraviglio, N. Xiong, "VMPlanner: Optimizing virtual machine placement and traffic flow routing to reduce network power costs in cloud data centers," Computer Networks, 2012.

[49] J.W. Jiang, T. Lan, S. Ha, M. Chen, M. Chiang, "Joint VM Placement and Routing for Data Center Traffic Engineering," IEEE INFOCOM, 2012.