Hindawi Publishing Corporation
Journal of Electrical and Computer Engineering
Volume 2016, Article ID 3583962, 10 pages
http://dx.doi.org/10.1155/2016/3583962

Research Article

A Processor-Sharing Scheduling Strategy for NFV Nodes

Giuseppe Faraci, Alfio Lombardo, and Giovanni Schembra

Dipartimento di Ingegneria Elettrica, Elettronica e Informatica (DIEEI), University of Catania, 95123 Catania, Italy

Correspondence should be addressed to Giovanni Schembra; schembra@dieei.unict.it

Received 2 November 2015; Accepted 12 January 2016

Academic Editor: Xavier Hesselbach

The introduction of the two paradigms SDN and NFV to "softwarize" the current Internet is making management and resource allocation two key challenges in the evolution towards the Future Internet. In this context, this paper proposes Network-Aware Round Robin (NARR), a processor-sharing strategy, to reduce delays in traversing SDN/NFV nodes. The application of NARR alleviates the job of the Orchestrator by automatically working at the intranode level, dynamically assigning the processor slices to the virtual network functions (VNFs) according to the state of the queues associated with the output links of the network interface cards (NICs). An extensive simulation set is presented to show the improvements achieved with respect to two more processor-sharing strategies chosen as reference.

1. Introduction

In the last few years the diffusion of new complex and efficient distributed services in the Internet is becoming increasingly difficult because of the ossification of the Internet protocols, the proprietary nature of existing hardware appliances, the costs, and the lack of skilled professionals for maintaining and upgrading them.

In order to alleviate these problems, two new network paradigms, Software Defined Networks (SDN) [1–5] and Network Functions Virtualization (NFV) [6, 7], have been recently proposed with the specific target of improving the flexibility of network service provisioning and reducing the time to market of new services.

SDN is an emerging architecture that aims at making the network dynamic, manageable, and cost-effective, by decoupling the system that makes decisions about where traffic is sent (the control plane) from the underlying system that forwards traffic to the selected destination (the data plane). In this way the network control becomes directly programmable and the underlying infrastructure is abstracted for applications and network services.

NFV is a core structural change in the way telecommunication infrastructure is deployed. The NFV initiative started in late 2012 by some of the biggest telecommunications service providers, which formed an Industry Specification Group (ISG) within the European Telecommunications Standards Institute (ETSI). The interest has grown, involving today more than 28 network operators and over 150 technology providers from across the telecommunications industry [7]. The NFV paradigm leverages on virtualization technologies and commercial off-the-shelf programmable hardware, such as general-purpose servers, storage, and switches, with the final target of decoupling the software implementation of network functions from the underlying hardware.

The coexistence and the interaction of both NFV and SDN paradigms is giving to the network operators the possibility of achieving greater agility and acceleration in new service deployments, with a consequent considerable reduction of both Capital Expenditure (CAPEX) and Operational Expenditure (OPEX) [8].

One of the main challenging problems in deploying an SDN/NFV network is an efficient design of resource allocation and management, functions that are in charge of the network Orchestrator. Although this task is well covered in data center and cloud scenarios [9, 10], it is currently a challenging problem in a geographic network where transmission delays cannot be neglected, and transmission capacities of the interconnection links are not comparable with the case of above scenarios. This is the reason why the problem of orchestrating an SDN/NFV network is still open and attracts a lot of research interest from both academia and industry.

More specifically, the whole problem of orchestrating an SDN/NFV network is very complex because it involves a design work at both internode and intranode levels [11, 12]. At the internode level, and in all the cases where the time between each execution is significantly greater than the time to collect, compute, and disseminate results, the application of a centralized approach is practicable [13, 14]. In these cases, in fact, taking into consideration both traffic characterization and required level of quality of service (QoS), the Orchestrator is able to decide in a centralized manner how many instances of the same function have to be run simultaneously, the network nodes that have to execute them, and the routing strategies that allow traffic flows to cross the nodes where the requested network functions are running.

Instead, centralizing a strategy dealing with operations that require dynamic reconfiguration of network resources is actually unfeasible to be executed by the Orchestrator for problems of resilience and scalability. Therefore, adaptive management operations with short timescales require a distributed approach. The authors of [15] propose a framework to support adaptive resource management operations, which involve short timescale reconfiguration of network resources, showing how requirements in terms of load-balancing and energy management [16–18] can be satisfied. Another work in the same direction is [19] that proposes a solution for the consolidation of VMs on local computing resources, exclusively based on local information. The work in [20] aims at alleviating the inter-VM network latency defining a hypervisor scheduler algorithm that is able to take into consideration the allocation of the resources within a consolidated environment, scheduling VMs to reduce their waiting latency in the run queue. Another approach is applied in [21], which introduces a policy to manage the internal on/off switching of virtual network functions (VNFs) in NFV-compliant Customer Premises Equipment (CPE) devices.

The focus of this paper is on another fundamental problem that is inherent to resource allocation within an SDN/NFV node, that is, the decision of the percentage of CPU to be assigned to each VNF. If at a first glance this could show a classical problem of processor sharing that has been widely explored in the past literature [15, 22, 23], actually it is much more complex because performance can be strongly improved by leveraging on the correlation with the output queues associated with the network interface cards (NICs).

With all this in mind, the main contribution of this paper is the definition of a processor-sharing policy, in the following referred to as *Network-Aware Round Robin* (NARR), which is specific for SDN/NFV nodes. Starting from the consideration that packets that have received the service of a network function from a virtual machine (VM) running on a given node are enqueued to wait for transmission through a given NIC, the proposed strategy dynamically changes the slices of the CPU assigned to each VNF according to the state of the output NIC queues. More specifically, the NARR strategy gives a larger CPU slice to serve packets that will leave the node through the NIC that is currently less loaded, in such a way as to minimize wastes of the NIC output link capacities, also minimizing the overall delay experienced by packets traversing nodes that implement NARR.

As a side contribution, the paper calculates an on-off model for the traffic leaving the SDN/NFV node on each NIC output link. This model can be used as a building block for the design and performance evaluation of an entire network.

The paper is structured as follows. Section 2 describes the node architecture. Section 3 introduces the NARR strategy. Section 4 presents a case study and shows some numerical results. Finally, Section 5 draws some conclusions and discusses some future work.

2. System Description

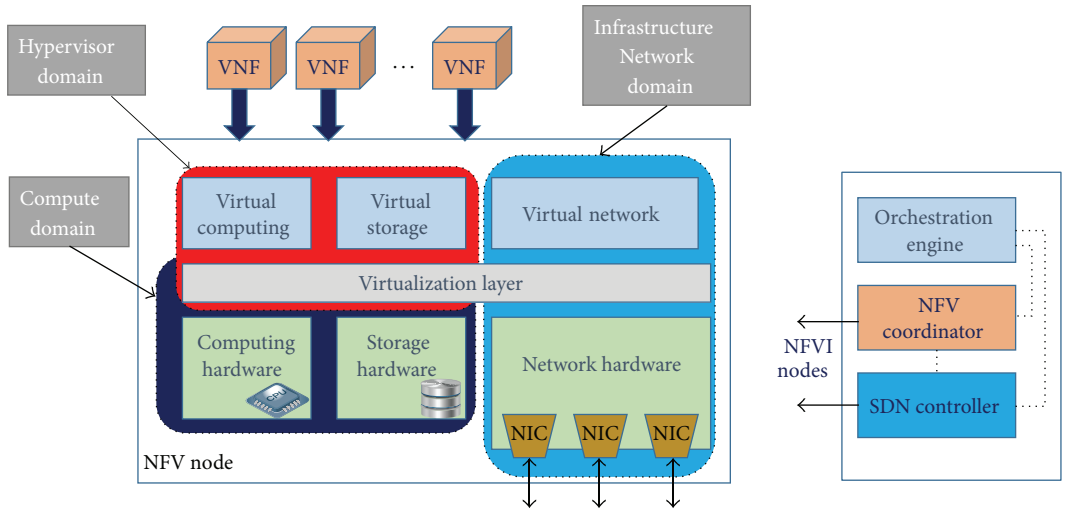
The target of this section is the description of the system we consider in the rest of the paper. It is an SDN/NFV node as the one considered in [11, 12], where we will apply the NARR strategy. Its architecture, shown in Figure 1(a), is compliant with the ETSI Specifications [24]. It is composed of three different domains, namely, the *Compute* domain, the *Hypervisor* domain, and the *Infrastructure Network* domain. The *Compute* domain provides the computational and storage hardware resources that allow the node to host the VNFs. Thanks to the computing and storage virtualization provided by the *Hypervisor* domain, a VM can be created, migrated from one node to another one, and halted, in order to optimize the deployment according to specific performance parameters. Communications among the VMs, and between the VMs and the external environment, are provided by the *Infrastructure Network* domain.

The SDN/NFV node is remotely controlled by the *Orchestrator*, whose architecture is shown in Figure 1(b). It is constituted by three main blocks. The *Orchestration Engine* executes all the algorithms and the strategies to manage and orchestrate the whole network. After each decision, the Orchestration Engine requests that the *NFV Coordinator* instantiates, migrates, or halts VMs and consequently requests that the SDN Controller modifies the flow tables of the SDN switches in the network in such a way that traffic flows can traverse VMs hosting the requested VNFs.

With this in mind, a functional architecture of the NFV node is represented in Figure 2. Its main components are the *Processor*, which manages the Compute domain and hosts the Hypervisor domain, and the Network Card Interfaces (NICs) with their queues, which constitute the "Network Hardware" block in Figure 1.

Let M be the number of virtual network functions (VNFs) that are running in the node, and let L be the number of output NICs. In order to simplify notation, in the following we will assume that all the NICs have the same characteristics in terms of buffer capacity and output rate. So, let $K^{(\mathrm{NIC})}$ be the size of the queue associated with each NIC, that is, the maximum number of packets that each queue can contain, and let $\mu^{(\mathrm{NIC})}$ be the transmission rate of the output link associated with each NIC, expressed in bit/s.

The *Flow Distributor* block has the task of routing each entering flow towards the function required by the flow. It is a software SDN switch that can be implemented, for example, with OpenvSwitch [25]. It routes the flows to the VMs running the requested VNFs according to the control messages received by the SDN Controller residing in the



(a) NFV node (b) Orchestrator

Figure 1: Network function virtualization infrastructure.

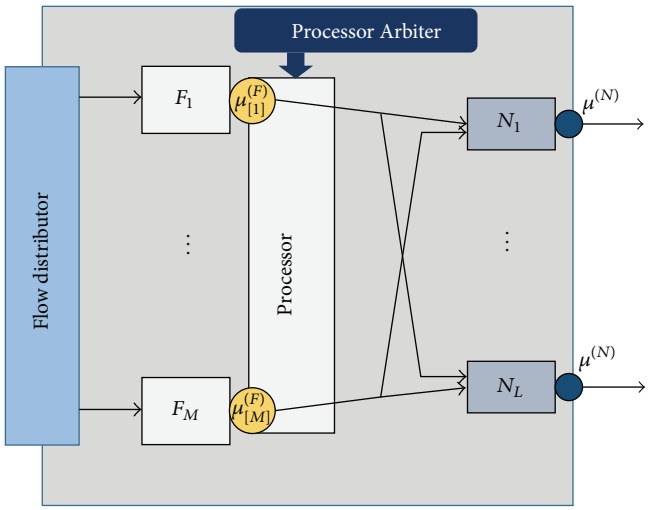


Figure 2: NFV node functional architecture.

Orchestrator. The most common protocol that can be used for the communications between the SDN Controller and the Orchestrator is OpenFlow [26].

Let $\mu^{(P)}$ be the total processing rate of the processor, expressed in packets/s. This rate is shared among all the active functions according to a processor rate scheduling strategy. Let $\underline{\mu}^{(F)}$ be the array whose generic element, $\mu^{(F)}_{[m]}$, with $m \in \{1, \ldots, M\}$, is the portion of the processor rate assigned to the VM implementing the function F_m. Of course we have

$$\sum_{m=1}^{M} \mu^{(F)}_{[m]} = \mu^{(P)}. \tag{1}$$

Once a packet has been served by the required function, it is sent to one of the NICs to exit from the node. If the NIC is transmitting another packet, the arriving packets are enqueued in the NIC queue. We will indicate the queue associated with the generic NIC l as $Q^{(\mathrm{NIC})}_l$.

In order to implement the NARR strategy proposed in this paper, we realize the block relative to each function with a set of L parallel queues, in the following referred to as *inside-function queues*, as shown in Figure 3 for the generic function F_m. The generic lth inside-function queue of the function F_m, indicated as $Q_{m,l}$ in Figure 3, is used to enqueue packets that, after receiving the service of the function F_m, will leave the node through the NIC l. Let $K^{(F)}_{\mathrm{Ins}}$ be the size of each inside-function queue. Each inside-function queue of the generic function F_m receives a portion of the processor rate assigned to that function. Let $\mu^{(F_{\mathrm{Ins}})}_{[m,l]}$ be the portion of the processor rate assigned to the queue $Q_{m,j}$ of the function F_m. Of course, we have

$$\sum_{l=1}^{L} \mu^{(F_{\mathrm{Ins}})}_{[m,l]} = \mu^{(F)}_{[m]}. \tag{2}$$

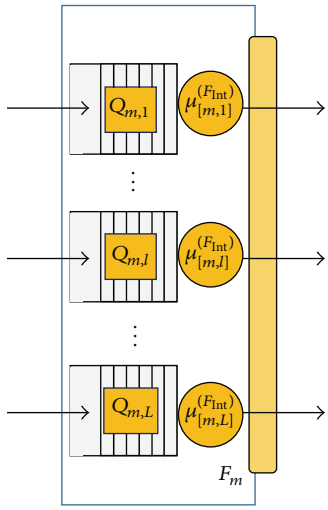


Figure 3: Block diagram of the generic function F_m.

The portion of processor rate associated with each inside-function queue is dynamically changed by the *Processor Arbiter* according to the NARR strategy described in Section 3.

3. NARR Processor-Sharing Strategy

The NARR (Network-Aware Round Robin) processor-sharing strategy observes the state of both the inside-function queues and the NIC queues, with the goal of reducing as far as possible the inactivity periods of the NIC output links. As already introduced in the Introduction, its definition starts from the fact that packets that have received the service of a VNF are enqueued to wait for transmission through a given NIC. So, in order to avoid output link capacity waste, NARR dynamically changes the slices of the CPU assigned to each VNF, and in particular to each inside-function queue, according to the state of the output NIC queues, assigning larger CPU slices to serve packets that will leave the node through less-loaded NICs.

More specifically, the Processor Arbiter decides the processor rate portions according to two different steps.

Step 1 (assignment of the processor rate portion to the aggregation of queues whose output is a specific NIC). This step meets the target of the proposed strategy that is to reduce, as much as possible, underutilization of the NIC output links and, as a consequence, delays in the relative queues. To this purpose, let us consider a virtual queue that contains all the packets that are stored in all the inside-function queues $Q_{m,l}$, for each $m \in [1, M]$, that is, all the packets that will leave the node through the NIC l. Let us indicate this virtual queue as $Q^{(\rightarrow \text{NIC}_l)}_{\text{Aggr}}$, and its service rate as $\mu^{(\rightarrow \text{NIC}_l)}_{\text{Aggr}}$. Of course, we have

$$\mu^{(\rightarrow \text{NIC}_l)}_{\text{Aggr}} = \sum_{m=1}^{M} \mu^{(F_{\text{Ins}})}_{[m,l]}. \tag{3}$$

With this in mind, the idea is to give a higher processor slice to the inside-function queues whose flows are directed to the NICs that are emptying.

Taking into account the goal of privileging the flows that will leave the node through underloaded NICs, the Processor Arbiter calculates $\mu^{(\rightarrow \text{NIC}_l)}_{\text{Aggr}}$ as follows:

$$\mu^{(\rightarrow \text{NIC}_l)}_{\text{Aggr}} = \frac{q_{\text{ref}} - S^{(\text{NIC})}_l}{\sum_{j=1}^{L} \left\{ q_{\text{ref}} - S^{(\text{NIC})}_j \right\}} \mu^{(P)}, \tag{4}$$

where $S^{(\text{NIC})}_l$ represents the state of the queue associated with the NIC l, while q_{ref} is defined as follows:

$$q_{\text{ref}} = \min \left\{ \alpha \cdot \max_j \left(S^{(\text{NIC})}_j \right), K^{(\text{NIC})} \right\}. \tag{5}$$

The term q_{ref} is a reference target value calculated from the state of the NIC queue that has the highest length, amplified with a coefficient α, and truncated to the maximum queue size $K^{(\text{NIC})}$. It is determined in such a way that, if we consider $\alpha = 1$, the NIC queue that has the highest length does not receive packets from the inside-function queues because the service rate of them is set to zero; the other queues receive packets with a rate that is proportional to the distance between their length and the length of the most overloaded NIC queue. However, through an extensive set of simulations, we deduced that setting $\alpha = 1$ causes bad performance because there is always a group of inside-function queues that are not served. Instead, all the α values in the interval $]1, 2]$ give almost equivalent performance. For this reason, in the numerical analysis presented in Section 4, we have set $\alpha = 1.2$.

Step 2 (assignment of the processor rate portion to each inside-function queue). Let us consider the generic lth inside-function queue of the function F_m, that is, the queue $Q_{m,l}$. Its service rate is calculated as being proportional to the current state of this queue in comparison with the other lth queues of the other functions. To this purpose, let us indicate the state of the virtual queue $Q^{(\rightarrow \text{NIC}_l)}_{\text{Aggr}}$ as $S^{(\rightarrow \text{NIC}_l)}_{\text{Aggr}}$. Of course, it can be calculated as the sum of the states of all the inside-function queues $Q_{m,l}$, for each $m \in [1, M]$: that is,

$$S^{(\rightarrow \text{NIC}_l)}_{\text{Aggr}} = \sum_{k=1}^{M} S^{(F_{\text{Ins}})}_{k,l}. \tag{6}$$

So, the service rate of the inside-function queue $Q_{m,l}$ is determined as a fraction of the service rate assigned at the first step to the virtual queue $Q^{(\rightarrow \text{NIC}_l)}_{\text{Aggr}}$, $\mu^{(\rightarrow \text{NIC}_l)}_{\text{Aggr}}$, as follows:

$$\mu^{(F_{\text{Ins}})}_{[m,l]} = \frac{S^{(F_{\text{Ins}})}_{m,l}}{S^{(\rightarrow \text{NIC}_l)}_{\text{Aggr}}} \mu^{(\rightarrow \text{NIC}_l)}_{\text{Aggr}}. \tag{7}$$

Of course, if at any time an inside-function queue remains empty, the processor rate portion assigned to it will be shared among the other queues proportionally to the processor portions previously assigned. Likewise, if at some instant an empty queue receives a new packet, the previous processor rate portion is reassigned to that queue.

4. Case Study

In this section we present a numerical analysis of the behavior of an SDN/NFV node that applies the proposed NARR processor-sharing strategy, with the target of evaluating the achieved performance. To this purpose, we will consider two other processor-sharing strategies as reference, in the following referred to as *round robin* (RR) and *queue-length weighted round robin* (QLWRR). In both the reference cases, the node has the same L NIC queues, but it has only M processor queues, one for each function. Each of the M queues has a size of $K^{(F)} = L \cdot K_{\text{Ins}}^{(F)}$, where $K_{\text{Ins}}^{(F)}$ represents the size of each internal function queue, already defined so far for the proposed strategy.

The RR strategy applies the classical round robin scheduling policy to serve the M function queues; that is, it serves each function queue with a rate $\mu_{\text{RR}}^{(F)} = \mu^{(P)}/M$.

The QLWRR strategy, on the other hand, serves each function queue with a rate that is proportional to the queue length; that is,

$$\mu_{\text{QLWRR}}^{(F_m)} = \frac{S^{(F_m)}}{\sum_{k=1}^{M} S^{(F_k)}} \mu^{(P)}, \tag{8}$$

where $S^{(F_m)}$ is the state of the queue associated with the function F_m.

4.1. Parameter Settings. In this numerical analysis, we consider a node with $M = 4$ VNFs, and $L = 3$ output NICs. We loaded the node with a balanced traffic constituted by $N_F = 12$ flows, each characterized by a different 2-uple {*requested NFV*, *output NIC*}. Each flow has been generated with an on-off model characterized by exponentially distributed on and off periods. When a flow is in off state, no packets arrive to the node from it; instead, when it is in on state, packets arrive with an average rate λ_{ON}. Each packet is assumed with an exponential distributed size with a mean of 1 kbyte. In all the simulations we have considered the same on-off cycle duration, $\delta = \overline{T}_{\text{OFF}} + \overline{T}_{\text{ON}} = 5$ msec, while we have varied the burstiness. Burstiness of on-off sources is defined as

$$b = \frac{\lambda_{\text{ON}}}{\lambda_{\text{Mean}}}, \tag{9}$$

where λ_{Mean} is the mean emission rate. Now, indicating the probability of the ON state as π_{ON}, and taking into account that $\lambda_{\text{Mean}} = \lambda_{\text{ON}} \cdot \pi_{\text{ON}}$ and $\pi_{\text{ON}} = \overline{T}_{\text{ON}}/(\overline{T}_{\text{OFF}} + \overline{T}_{\text{ON}})$, we have

$$b = \frac{\overline{T}_{\text{OFF}} + \overline{T}_{\text{ON}}}{\overline{T}_{\text{ON}}}. \tag{10}$$

In our analysis, the burstiness has been varied in the range [2, 22]. Consequently, we have derived the mean durations of the off and on periods, as follows:

$$\begin{aligned} \overline{T}_{\text{ON}} &= \frac{\delta}{b}, \\ \overline{T}_{\text{OFF}} &= \delta - \overline{T}_{\text{ON}}. \end{aligned} \tag{11}$$

Finally, in order to maintain the same mean packet rate for different values of $\overline{T}_{\text{OFF}}$ and $\overline{T}_{\text{ON}}$, we have assumed that 75 packets are transmitted, on average: that is, $\lambda_{\text{ON}} = 75/\overline{T}_{\text{ON}}$. The resulting mean emission rate is $\lambda_{\text{Mean}} = 122.9$ Mbit/s.

As far as the NICs are concerned, we have considered an output rate of $\mu^{(\text{NIC})} = 980$ Mbit/s, so having a utilization coefficient on each NIC of $\rho^{(\text{NIC})} = 0.5$, and a queue size $K^{(\text{NIC})} = 3000$ packets. Instead, regarding the processor, we considered a size of each inside-function queue of $K_{\text{Ins}}^{(F)} = 3000$ packets. Finally, we have analyzed two different processor cases. In the first case we considered a processor that is able to process $\mu^{(P)} = 306$ kpackets/s, while in the second case we assumed a processor rate of $\mu^{(P)} = 204$ kpackets/s. Therefore, defining the processor utilization coefficient as follows:

$$\rho^{(P)} = \frac{N_F \cdot \lambda_{\text{Mean}}}{\mu^{(P)}} \tag{12}$$

with $N_F \cdot \lambda_{\text{Mean}}$ being the total mean arrival rate to the node, the two considered cases are characterized by a processor utilization coefficient of $\rho_{\text{Low}}^{(P)} = 0.6$ and $\rho_{\text{High}}^{(P)} = 0.9$, respectively.

4.2. Numerical Results. In this section we present some results achieved by discrete-event simulations. The simulation tool used in the paper is publicly available in [27]. We first present a temporal analysis of the main variables characterizing the SDN/NFV node, and then we show a performance comparison of NARR with the two strategies RR and QLWRR, taken as a reference.

For the temporal analysis we focus on a short time interval of 720 μsec, in order to be able to clearly highlight the evolution of the considered processes. We have loaded the node with an on-off traffic like the one described in Section 4.1, with a burstiness $b = 7$.

Figures 4, 5, and 6 show the time evolution of the length of the queues associated with the NICs, the processor slice assigned to each virtual queue loading each NIC, and the length of the same virtual queues. We can subdivide the considered time interval into three different periods.

In the first period, ranging from the instants 0.1063 and 0.1067, from Figure 4 we can notice that the NIC queue $Q_1^{(\text{NIC})}$ has a greater length than the queue $Q_3^{(\text{NIC})}$, while $Q_2^{(\text{NIC})}$ is empty. For this reason in this period, as shown in Figure 5, the processor is shared between $Q_{\text{Aggr}}^{(\rightarrow \text{NIC}_1)}$ and $Q_{\text{Aggr}}^{(\rightarrow \text{NIC}_3)}$, and the slice assigned to serve $Q_{\text{Aggr}}^{(\rightarrow \text{NIC}_3)}$ is higher, in such a way that the two queue lengths $Q_1^{(\text{NIC})}$ and $Q_3^{(\text{NIC})}$ reach the same value, situation that happens at the end of the first period, around the instant 0.1067. During this period, the behavior of both the virtual queues $Q_{\text{Aggr}}^{(\rightarrow \text{NIC}_1)}$ and $Q_{\text{Aggr}}^{(\rightarrow \text{NIC}_3)}$ in Figure 6 remains flat, showing the fact that the received processor rates are able to balance the arrival rates.

At the beginning of the second period that ranges between the instants 0.1067 and 0.10693, the processor rate assigned to the virtual queue $Q_{\text{Aggr}}^{(\rightarrow \text{NIC}_3)}$ has become not sufficient to serve the amount of arriving traffic, and so the virtual queue $Q_{\text{Aggr}}^{(\rightarrow \text{NIC}_3)}$ increases its length, as shown in Figure 6.

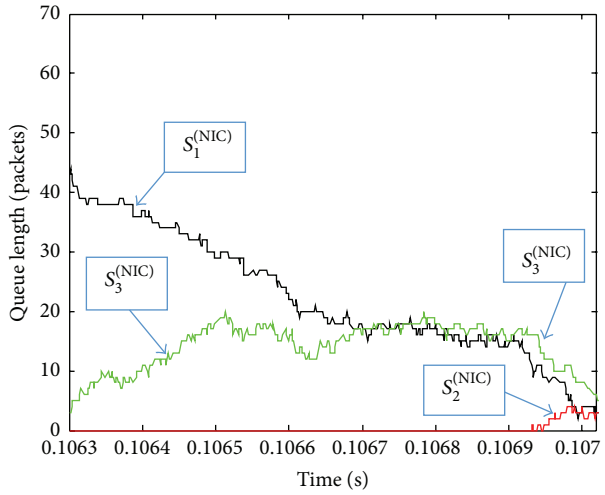


Figure 4: Lenght evolution of the NIC queues.

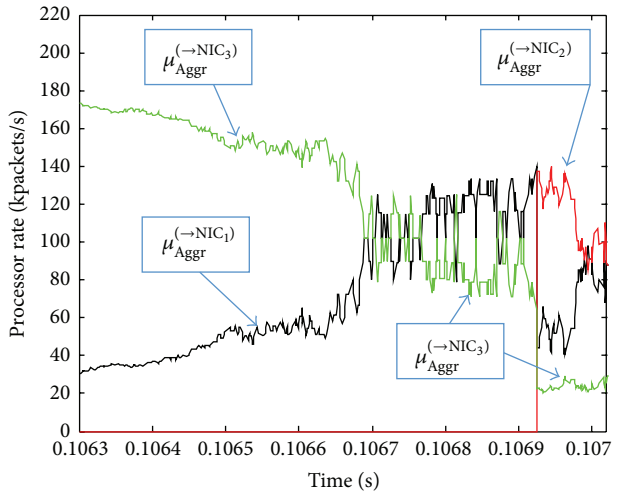


Figure 5: Packet rate assigned to the virtual queues.

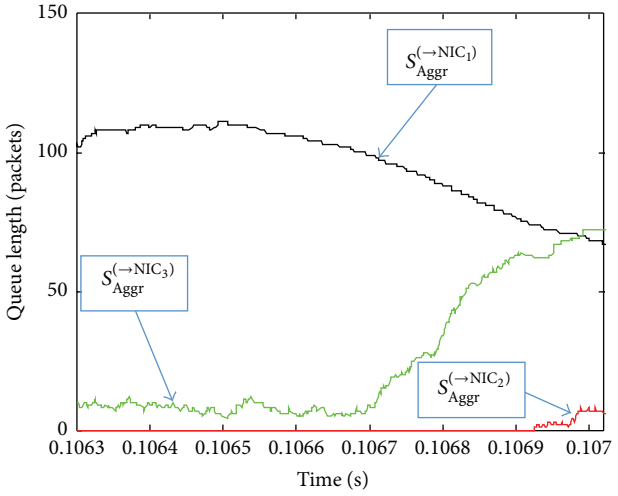


Figure 6: Lenght evolution of the virtual queues.

During this second period, the processor slices assigned to the two queues $Q_{Aggr}^{(\rightarrow NIC_1)}$ and $Q_{Aggr}^{(\rightarrow NIC_3)}$ are adjusted in such a way that $Q_1^{(NIC)}$ and $Q_3^{(NIC)}$ remain with comparable lengths.

The last period starts at the instant 0.10693, characterized by the fact that the aggregated queue $Q_{Aggr}^{(\rightarrow NIC_2)}$ leaves the empty state, and therefore participates in the processor-sharing process. Since the NIC queue $Q_2^{(NIC)}$ is low-loaded, as shown in Figure 4, the largest slice is assigned to $Q_{Aggr}^{(\rightarrow NIC_2)}$ in such a way that $Q_2^{(NIC)}$ can reach the same length of the other two NIC queues as soon as possible.

Now, in order to show how the second step of the proposed strategy works, we present the behavior of the inside-function queues whose output is sent to the NIC queue $Q_3^{(NIC)}$ during the same short time interval considered so far. More specifically, Figures 7 and 8 show the length of the considered inside-function queues, and the processor slices assigned to them, respectively. The behavior of the queue $Q_{2,3}$ is not shown because it is empty in the considered period. As we can observe from the above figures, we can subdivide the interval into four periods:

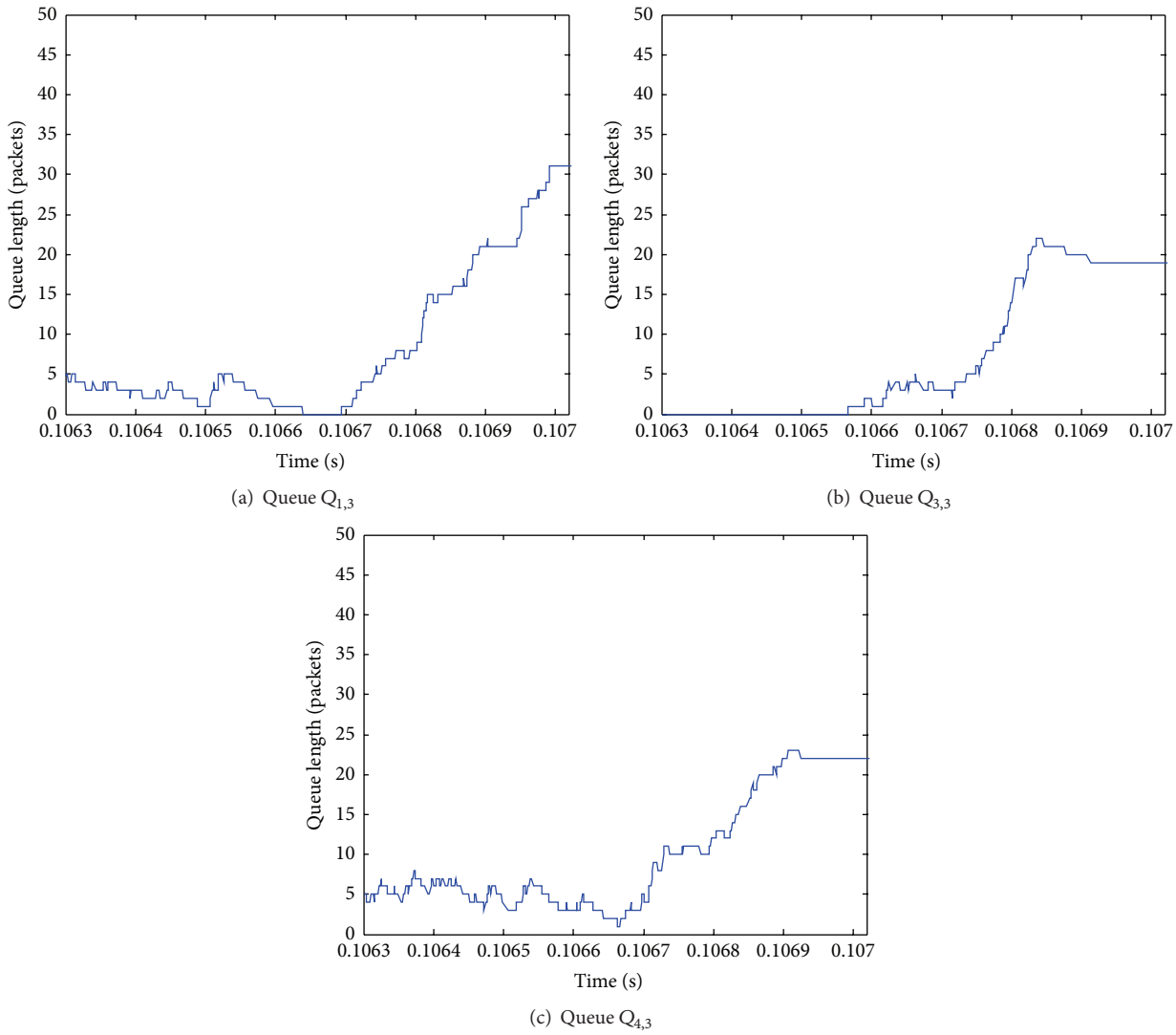
(i) The first period, ranging in the interval [0.1063, 0.10657] is characterized by an empty state of the queue $Q_{3,3}$. Thus, in this period, the processor slice assigned to the aggregated queue $Q_{Aggr}^{(\rightarrow NIC_3)}$ is shared by $Q_{1,3}$ and $Q_{4,3}$, only.

(ii) During the second period, ranging in the interval [0.10657, 0.1067], $Q_{1,3}$ is scarcely loaded (in particular it is empty in the second part of this period), and so the processor slice assigned to $Q_{3,3}$ is increased.

(iii) In the third period, ranging between 0.1067 and 0.10693, all the queues increase and equally share the processor.

(iv) Finally, in the last period, starting at the instant 0.10693, as already observed in Figure 5, the processor slice assigned to the aggregated queue $Q_{Aggr}^{(\rightarrow NIC_3)}$ is suddenly decreased, and consequently the slices assigned to the queues $Q_{1,3}$, $Q_{3,3}$, and $Q_{4,3}$ are decreased as well.

The steady-state analysis is presented in Figures 9, 10, and 11, which show the mean delay in the inside-function queues, in the NIC queues, and the durations of the off- and on-states on the output links. The values reported in all the figures have been derived as the mean values of the results of many simulation experiments, using Student's t-distribution and with a 95% confidence interval. The number of experiments carried out to evaluate each numerical result has been automatically decided by the simulation tool with the requirement of achieving a confidence interval less than 0.001 of the estimated mean value. To this purpose, the confidence interval is calculated at the end of each run, and simulation is stopped only when the confidence interval matches the maximum error requirement. The duration of each run has been chosen in such a way that the sample standard deviation is so low that less than 30 runs are enough to match the requirement.



(a) Queue $Q_{1,3}$

(b) Queue $Q_{3,3}$

(c) Queue $Q_{4,3}$

FIGURE 7: Length evolution of the inside-function queues $Q_{m,3}$, for each $m \in [1, M]$.

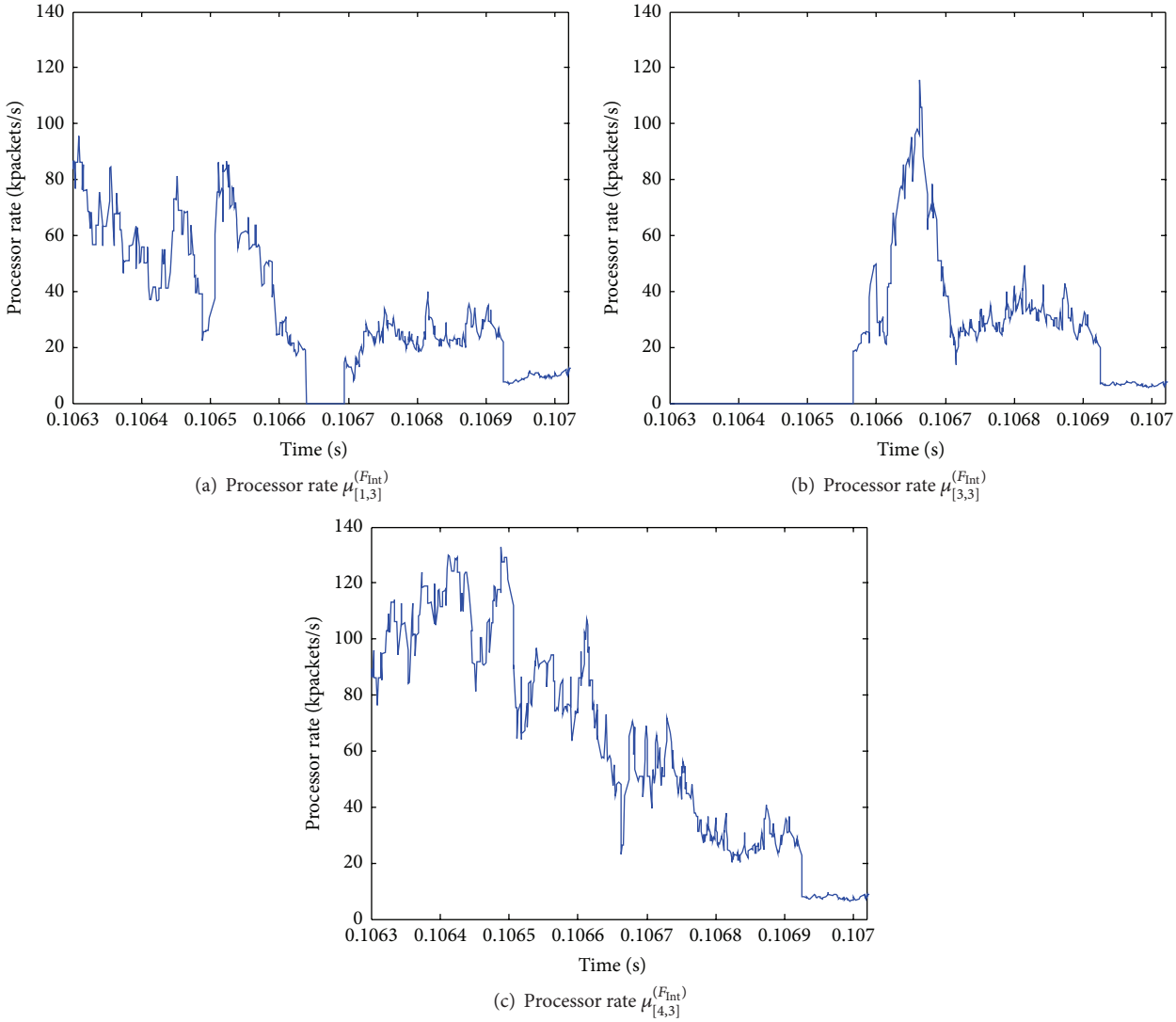
The figures compare the results achieved with the proposed strategy with the ones obtained with the two strategies RR and QLWRR. As said so far, results have been obtained against the burstiness, and for two different values of the utilization coefficient: that is, $\rho_{\text{Low}}^{(P)} = 0.6$ and $\rho_{\text{High}}^{(P)} = 0.9$.

As expected, the mean lengths of the processor queues and the NIC queues increase with both the burstiness and the utilization coefficient.

Instead, we can note that the mean length of the processor queues is not affected by the applied policy. In fact, packets requiring the same function are enqueued in a unique queue and served with a rate $\mu^{(P)}/4$, when RR or QLWRR strategies are applied, while, when the NARR strategy is used, they are split into 12 different queues and served with a rate $\mu^{(P)}/12$. If in a classical queueing theory [28] the second case is worse than the first one because of the presence of service rate wastes during the more probable periods of empty queues, this is not the case here because the processor capacity of an empty queue is dynamically reassigned to the other queues.

The advantage of the NARR strategy is evident in Figure 10, where the mean delay in the NIC queues is represented. In fact, we can observe that, with only giving more processor rate to the most loaded processor queues (with the QLWRR strategy), performance improvements are negligible, while applying the NARR strategy we are able to obtain a delay reduction of about 12% in the case of a more performant processor ($\rho_{\text{Low}}^{(P)} = 0.6$), reaching the 50% when the processor works with a $\rho_{\text{High}}^{(P)} = 0.9$. The performance gain achieved with the NARR strategy increases with burstiness and the processor load, conditions that are both likely. In fact, the first condition is due to the high burstiness of the Internet traffic; the second one is true as well because the processor should be not overdimensioned for economic purposes; otherwise, if overdimensioned, it can be controlled with a processor rate management policy like the one presented in [29] in order to save energy.

Finally, Figure 11 shows the mean durations of the on- and off-periods on the node output links. Only one curve is



(a) Processor rate $\mu_{[1,3]}^{(F_{\text{Int}})}$

(b) Processor rate $\mu_{[3,3]}^{(F_{\text{Int}})}$

(c) Processor rate $\mu_{[4,3]}^{(F_{\text{Int}})}$

FIGURE 8: Processor rate assigned to the inside-function queues $Q_{m,3}$, for each $m \in [1, M]$.

shown for each case because we have considered a utilization coefficient $\rho^{(\text{NIC})} = 0.5$ on each NIC queue, and therefore in the considered case we have $\overline{T}_{\text{ON}} = \overline{T}_{\text{OFF}}$. As expected, the mean on-off durations are higher when the processor rate is higher (i.e., lower utilization coefficient). This is because, in this case, the output processor rate is lower, and therefore batches of packets in the NIC queues are served quickly. These results can be used to model the output traffic of each SDN/NFV node as an Interrupted Poisson Process (IPP) [30]. This model can be iteratively used to represent the input traffic of other nodes, with the final target of realizing the model of a whole SDN/NFV network.

5. Conclusions and Future Work

This paper addresses the problem of intranode resource allocation, by introducing NARR, a processor-sharing strategy that leverages on the consideration that, in any SDN/NFV node, packets that have received the service of a VNF are enqueued to wait for transmission through one of the output NICs. Therefore, the idea at the base of NARR is to dynamically change the slices of the CPU assigned to each VNF according to the state of the output NIC queues, giving more CPU to serve packets that will leave the node through the less-loaded NICs. In this way, wastes of the NIC output link capacities are minimized, and consequently the overall delay experienced by packets traversing the nodes that implement NARR is reduced.

SDN/NFV nodes that implement NARR can coexist in the same network with nodes that use other strategies, so facilitating a gradual introduction in the Future Internet.

As a side contribution, the simulator tool, which is public available on the web, gives the on-off model of the output links associated with each NIC as one of the results. This model can be used as a building block to realize a model for

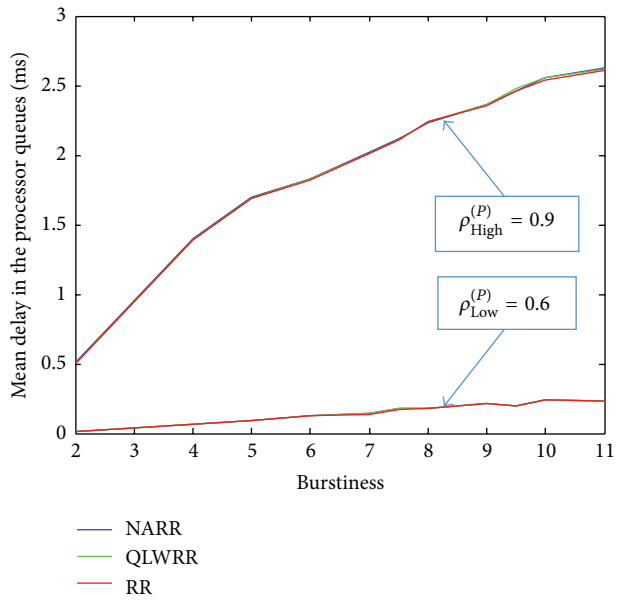


Figure 9: Mean delay in the function internal queues.

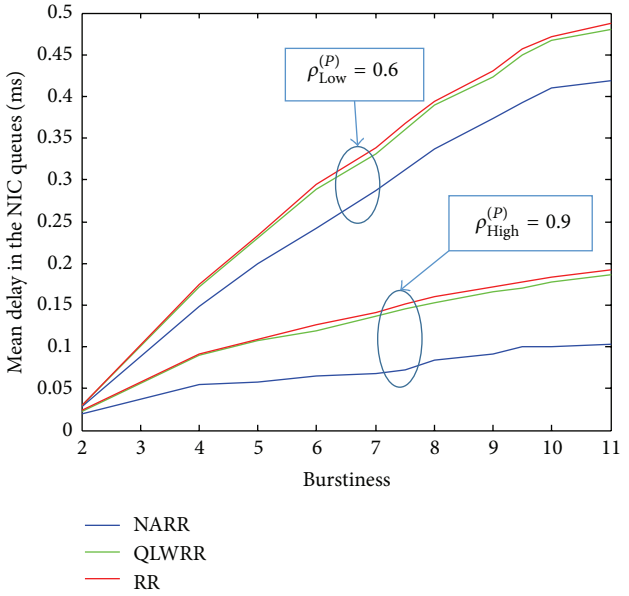


Figure 10: Mean delay in the NIC queues.

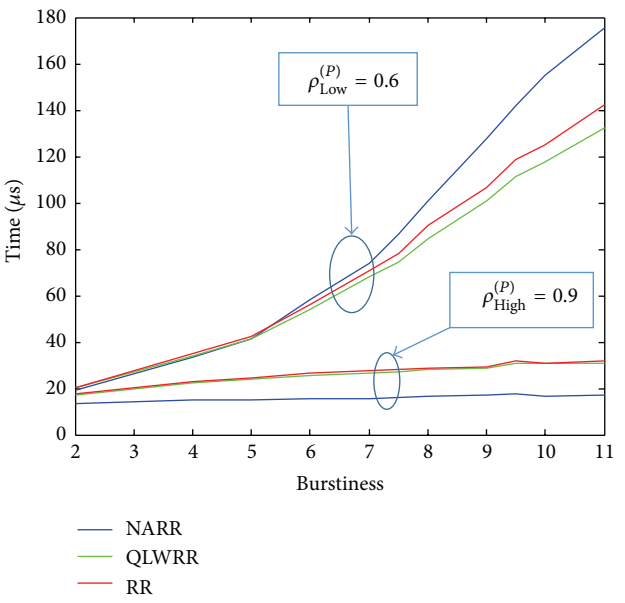


Figure 11: Mean off- and on-states duration of the traffic on the output links.

the design and performance evaluation of a whole SDN/NFV network.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

This work has been partially supported by the INPUT (In-Network Programmability for Next-Generation Personal cloUd Service Support) project funded by the European Commission under the Horizon 2020 Programme (Call H2020-ICT-2014-1, Grant no. 644672).

References

[1] White paper on "Software-Defined Networking: The New Norm for Networks", https://www.opennetworking.org/.

[2] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with OpenSketch," in *Proceedings of the Symposium on Network Systems Design and Implementation (NSDI '13)*, vol. 13, pp. 29–42, Lombard, Ill, USA, April 2013.

[3] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.

[4] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 136–141, 2013.

[5] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: past, present, and future of programmable networks," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[6] White paper on "Network Functions Virtualisation", http://portal.etsi.org/NFV/NFV_White_Paper.pdf.

[7] Network Functions Virtualisation (NFV): Network Operator Perspectives on Industry Progress, ETSI, October 2013, http://portal.etsi.org/NFV/NFV_White_Paper2.pdf.

[8] A. Manzalini, R. Saracco, E. Zerbini et al., "Software-Defined Networks for Future Networks and Services," White Paper based on the IEEE Workshop SDN4FNS, http://sites.ieee.org/sdn4fns/whitepaper/.

[9] R. Z. Frantz, R. Corchuelo, and J. L. Arjona, "An efficient orchestration engine for the cloud," in *Proceedings of the IEEE 3rd International Conference on Cloud Computing Technology*

and Science (CloudCom '11), vol. 2, pp. 711–716, Athens, Greece, December 2011.

[10] K. Bousselmi, Z. Brahmi, and M. M. Gammoudi, "Cloud services orchestration: a comparative study of existing approaches," in *Proceedings of the 28th IEEE International Conference on Advanced Information Networking and Applications Workshops (WAINA '14)*, pp. 410–416, Victoria, Canada, May 2014.

[11] A. Lombardo, A. Manzalini, V. Riccobene, and G. Schembra, "An analytical tool for performance evaluation of software defined networking services," in *Proceedings of the IEEE Network Operations and Management Symposium (NMOS '14)*, pp. 1–7, Krakow, Poland, May 2014.

[12] A. Lombardo, A. Manzalini, G. Schembra, G. Faraci, C. Rametta, and V. Riccobene, "An open framework to enable NetFATE (network functions at the edge)," in *Proceedings of the IEEE Workshop on Management Issues in SDN, SDI and NFV (Mission '15)*, pp. 1–6, London, UK, April 2015.

[13] A. Manzalini, R. Minerva, F. Callegati, W. Cerroni, and A. Campi, "Clouds of virtual machines in edge networks," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 63–70, 2013.

[14] H. Moens and F. De Turck, "VNF-P: a model for efficient placement of virtualized network functions," in *Proceedings of the 10th International Conference on Network and Service Management (CNSM '14)*, pp. 418–423, Rio de Janeiro, Brazil, November 2014.

[15] K. Katsalis, G. S. Paschos, Y. Viniotis, and L. Tassiulas, "CPU provisioning algorithms for service differentiation in cloud-based environments," *IEEE Transactions on Network and Service Management*, vol. 12, no. 1, pp. 61–74, 2015.

[16] D. Tuncer, M. Charalambides, G. Pavlou, and N. Wang, "Towards decentralized and adaptive network resource management," in *Proceedings of the 7th International Conference on Network and Service Management (CNSM '11)*, pp. 1–6, Paris, France, October 2011.

[17] D. Tuncer, M. Charalambides, G. Pavlou, and N. Wang, "DACoRM: a coordinated, decentralized and adaptive network resource management scheme," in *Proceedings of the IEEE Network Operations and Management Symposium (NOMS '12)*, pp. 417–425, IEEE, Maui, Hawaii, USA, April 2012.

[18] M. Charalambides, D. Tuncer, L. Mamatas, and G. Pavlou, "Energy-aware adaptive network resource management," in *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM '13)*, pp. 369–377, Ghent, Belgium, May 2013.

[19] C. Mastroianni, M. Meo, and G. Papuzzo, "Probabilistic consolidation of virtual machines in self-organizing cloud data centers," *IEEE Transactions on Cloud Computing*, vol. 1, no. 2, pp. 215–228, 2013.

[20] B. Guan, J. Wu, Y. Wang, and S. U. Khan, "CIVSched: a communication-aware inter-VM scheduling technique for decreased network latency between co-located VMs," *IEEE Transactions on Cloud Computing*, vol. 2, no. 3, pp. 320–332, 2014.

[21] G. Faraci and G. Schembra, "An analytical model to design and manage a green SDN/NFV CPE node," *IEEE Transactions on Network and Service Management*, vol. 12, no. 3, pp. 435–450, 2015.

[22] L. Kleinrock, "Time-shared systems: a theoretical treatment," *Journal of the ACM*, vol. 14, no. 2, pp. 242–261, 1967.

[23] S. F. Yashkov and A. S. Yashkova, "Processor sharing: a survey of the mathematical theory," *Automation and Remote Control*, vol. 68, no. 9, pp. 1662–1731, 2007.

[24] ETSI GS NFV—INF 001, v1.1.1, "Network Functions Virtualization (NFV): Infrastructure Overview", 2015, https://www.etsi.org/deliver/etsi_gs/NFV-INF/001_099/001/01.01.01_60/gs_NFV-INF001v010101p.pdf.

[25] T. N. Subedi, K. K. Nguyen, and M. Cheriet, "OpenFlow-based in-network Layer-2 adaptive multipath aggregation in data centers," *Computer Communications*, vol. 61, pp. 58–69, 2015.

[26] N. McKeown, T. Anderson, H. Balakrishnan et al., "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[27] NARR simulator, v.0.1, http://www.diit.unict.it/arti/Tools/NARR_Simulator_v01.zip.

[28] L. Kleinrock, *Queueing Systems. Volume 1: Theory*, Wiley-Interscience, 1st edition, 1975.

[29] R. Bruschi, P. Lago, A. Lombardo, and G. Schembra, "Modeling power management in networked devices," *Computer Communications*, vol. 50, pp. 95–109, 2014.

[30] W. Fischer and K. S. Meier-Hellstern, "The Markov-Modulated Poisson process (MMPP) cookbook," *Performance Evaluation*, vol. 18, no. 2, pp. 149–171, 1993.