

# BeaQoS: Load balancing and deadline management of queues in an OpenFlow SDN switch



L. Boero<sup>a</sup>, M. Cello<sup>b,\*</sup>, C. Garibotto<sup>a</sup>, M. Marchese<sup>a</sup>, M. Mongelli<sup>c</sup>

<sup>a</sup> University of Genoa, Via all'Opera Pia 13, 16145, Genova, Italy

<sup>b</sup> Nokia Bell Labs, Blanchardstown Business & Technology Park, Snugborough Road, Dublin 15, Ireland

<sup>c</sup> National Research Council, Via De Marini 6, 16149, Genova, Italy

## ARTICLE INFO

### Article history:

Received 17 August 2015

Revised 25 March 2016

Accepted 20 June 2016

Available online 24 June 2016

### Keywords:

SDN

OpenFlow

Packet loss

Traffic engineering

## ABSTRACT

Current OpenFlow specification is unable to set the service rate of the queues inside OpenFlow devices. This lack does not allow to apply most algorithms for the satisfaction of Quality of Service requirements to new and established flows. In this paper we propose an alternative solution implemented through some modifications of Beacon, one popular SDN controller. It acts as follows: using 'almost'-real-time statistics from OpenFlow devices, Beacon will re-route flows on different queues to guarantee the observance of deadline requirements (e.g. the flow is still useful if, and only if, is completely received by a given time) and/or an efficient queue balancing in an OpenFlow SDN switch. Differently from the literature, we do not propose any new primitive or modification of the OpenFlow standard: our mechanism, implemented in the controller, works with regular OpenFlow devices. Our changes in the SDN controller will be the base for the design of a class of new re-routing algorithms able to guarantee deadline constraints and queue balancing without any modification of the OpenFlow specification, as well as, of OpenFlow devices.

© 2016 Published by Elsevier B.V.

## 1. Introduction

Software Defined Networking (SDN) is revolutionizing the networking industry by enabling programmability, easier management and faster innovation [1,2]. These benefits are made possible by its centralized control plane architecture which allows the network to be programmed and controlled by one central entity.

The SDN architecture is composed both of SDN enabled devices (switches/routers)<sup>1</sup> and of a central controller (SDN controller). An SDN device processes and delivers packets according to the rules stored in its flow table (forwarding state), whereas the SDN controller configures the forwarding state of each SDN device by using a standard protocol called OpenFlow (OF) [2]. The SDN controller is responsible also to build the virtual topology representing the physical topology. The virtual topology is used by the application

modules that run on top of the SDN controller to implement different control logics and network functions (e.g. routing, traffic engineering, firewall actions).

Currently the Quality of Service (QoS) management in OF is quite limited: in each OF switch one or more queues can be configured for each outgoing interface and used to map flow entries on them. Flow entries mapped to a specific queue will be treated according to the queue's configuration in terms of service rate, but the queue's configuration *takes place outside the OF protocol*. For example, the queue's service rate cannot be modified by OF.

Supposing that a flow is traversing a chain of queues from the source to the destination node, and that the flow data rate increases, a possible consequence is that queues increase their occupancy, and a bottleneck may be generated with consequent network congestion. The impossibility to change the bottleneck queue's service rate through real-time OF directives can lead to a severe performance degradation for the flows traversing that queue because, without a proper rate assignment, it is very difficult to guarantee Quality of Service requirements to the flows [3].

A possible solution to mitigate the performance degradation involves the re-routing of the flows experiencing a violation of deadline constraints (e.g. the flows that are totally received beyond the fixed time constraint) [4] on less congested paths or queues. The underlying idea is that, since we cannot change the service rate of

\* Corresponding author. The work has been performed while M. Cello was employed at University of Genoa.

E-mail addresses: [luca.boero@edu.unige.it](mailto:luca.boero@edu.unige.it) (L. Boero), [marco.cello@nokia-bell-labs.com](mailto:marco.cello@nokia-bell-labs.com) (M. Cello), [chiara.garibotto@edu.unige.it](mailto:chiara.garibotto@edu.unige.it) (C. Garibotto), [mario.marchese@unige.it](mailto:mario.marchese@unige.it) (M. Marchese), [maurizio.mongelli@ieiit.cnr.it](mailto:maurizio.mongelli@ieiit.cnr.it) (M. Mongelli).

<sup>1</sup> In the following we will use the terms: SDN device, OpenFlow device, OpenFlow switch, interchangeably, even if the term "OpenFlows switch" or simply "switch" indicates an SDN enabled device in most SDN literature.

the queues, we act on the ingress traffic, moving a subset of flows on different paths or queues in case of need. In order to be 100% compatible with current OF hardware, we impose no changes to OF specifications and directives. Instead we propose to modify one popular SDN controller: Beacon [5]. The proposed solution, *BeaQoS*, applied to a single SDN switch, is an extension of our previous work presented in [6]. Our new updated controller will receive statistics about queues, flows and ports from OF switches and will compute an estimation of the flow rates and of the packet loss of the queues. Based on customizable policies, *BeaQoS* will be able to select a subset of flows experiencing congestion over the bottleneck queue and to re-route them on another and less congested queue, so improving the switch performances. The action of flow re-routing may be exploited not only for deadline management but also for efficient queue load balancing. On the other hand load balancing is often seen as an action to prevent congestion and, consequently, to limit and delay performance detriment.

The remainder of this paper is structured as follows. We describe related works on this field in Section 2. Concerning the main contributions of the paper:

- We explain the motivations that lead to consider multi-queue interfaces with variable service rate to support deadline management in Section 3;
- We describe the basic idea concerning the re-routing mechanisms introduced in this paper in Section 4.1, where we also show how it can be usefully applied in case of multi-core architectures and load balancing issues among queues;
- We describe the modifications of the Beacon controller required to implement re-routing in Section 4.2;
- We propose five effective re-routing strategies in *BeaQoS*: two of them aimed at improving deadline management and three of them aimed at balancing the load among queues in a SDN switch in Section 5.

We show the performance analysis of our proposed algorithms in Section 5. We report a discussion about the obtained results together with the conclusions in Section 7.

## 2. Related works

Despite traffic engineering (TE) approaches are often ruled by MPLS-TE [7,8], the ability of the SDN controller to receive (soft) real-time information from SDN devices and to make decisions based on a global view of the network, coupled with the ability of “custom”-grained flow aggregation inside SDN devices, makes TE one of the most interesting use cases for SDN networks.

Global load balancing algorithms are proposed in [9] that addresses load-balancing as an integral component of large cloud services and explores ways to make load-balancing scalable, dynamic, and flexible. Moreover [9] states that load-balancing should be a network primitive, not an add-on, and presents a prototype distributed load-balancer based on this principle.

[10], shows that the controller should exploit switch support for wildcard rules for a more scalable solution that directs large aggregates of client traffic to server replicas. [10] also presents algorithms that compute concise wildcard rules that achieve a target distribution of the traffic and automatically change load-balancing policies without disrupting existing connections. Furthermore, the authors implement these algorithms on top of the NOX OpenFlow controller, evaluate their effectiveness, and propose avenues for further research.

The work presented in [11] shows a system that re-configures the network’s data plane to match current traffic demands by centrally controlling the traffic that each service sends on a backbone connecting data-centres. [11] develops a novel technique that

**Table 1**

Performance metrics of the traffic for 1-queue and 3-queues configurations.

Performance metric	Queue configuration	
	1-queue	3-queue
BF - packet loss	25%	71.16%
DF1 - percentage of flows matching the deadline	11.43%	74.29%
DF2 - percentage of flows matching the deadline	17.39%	19.57%

leverages a small amount of scratch capacity on links to apply updates in a provably congestion free manner, without making any assumptions about the order and timing of updates at individual switches. Further, to scale to large networks in the face of limited forwarding table capacity, [11] greedily selects a small set of entries that can satisfy current demands and updates this set without disrupting traffic.

Reference [12] analyses a partially deployed SDN network (a mix of SDN and non-SDN devices) and shows how to exploit the centralized controller to get significant improvements in network utilization as well as to reduce packet losses and delays. [12] shows that these improvements are possible even in cases where there is only a partial deployment of SDN capability in a network. The authors formulate the SDN controller’s optimization problem for traffic engineering with partial deployment and propose a fast Fully Polynomial Time Approximation Schemes (FPTAS) to solve it.

This last problem is also tackled in [13] that introduces a traffic management method to divide, or to “slice”, network resources to match user requirements. [13] presents an alternative to resort to low-level mechanisms such as Virtual LANs, or to interpose complicated hypervisors into the control plane, by introducing an abstraction that supports programming isolated slices of the network. The semantics of slices ensures that the processing of packets on a slice is independent of all other slices. They define their slice abstraction, develop algorithms to compile slices, and illustrate their use by using examples. In addition, [13] describes a prototype implementation and a tool to automatically verify formal isolation properties.

In our previous work [6], we propose a solution based on SDN, which implements a software strategy to cope with non-conformant traffic flows inside a class-based system. This approach is therefore independent of the underlying hardware, as it is conceived to run as an algorithm inside the SDN controller. The proposed strategy will manage non-conformant flows, based on a set of statistic data gathered by a modified version of the Beacon controller, in order to mitigate the quality degradation of flows traversing the network.

In order to support traffic engineering in the SDN environment, OpenFlow Management and Configuration Protocol (OF-Config) has been proposed. OF-Config [14] is a protocol developed by the Open Networking Foundation used to manage physical and virtual switches in an OpenFlow environment. This tool gives network engineers an overall view of the network and also provides the ability to set policies and to manage traffic across devices.

## 3. Motivations

Some approaches consider a single queue for each outgoing interface. In order to support QoS mechanisms and traffic differentiation, it is common to configure multiple queues in advance [3]. The importance of traffic differentiation is highlighted by the first group of simulations (Table 1) reported in the following.

Flow entries mapped to a specific queue will be treated according to that queue’s configuration in terms of service rate. Most of the previously mentioned approaches assumes the ability of SDN/OpenFlow to set the service rate of the queues in each SDN

**Table 2**  
Performance metrics of the traffic for fixed and variable service rate.

Performance metric	Queue configuration	
	Fixed rate	Variable rate
BF - packet loss	0%	0%
DF1 - percentage of flows matching the deadline	100%	100%
DF2 - percentage of flows matching the deadline	25%	100%
DF - percentage of flows matching the deadline	34.78%	100%

**Table 3**  
Traffic classes and their deadline requirements.

Traffic class		Percentage of overall traffic	Deadline requirements
Name	Traffic descriptor		
BF	50 – 80 kbit/s × 50 s	30%	–
DF1	4.5 Mbit/s × 1 s	55%	deadline: 9 s
DF2	1.5 Mbit/s × 1 s	15%	deadline: 5 s

device. This chance would be very helpful to improve the SDN switch performance, as would be clear from the second group of simulations (Table 2) reported below.

Table 1 shows the results of simulations we ran aimed at showing how it is hard, without traffic differentiation, to guarantee deadline requirements. During 120s of simulation, an Open vSwitch<sup>2</sup> s1 receives a mix of traffic, generated with *iperf*, composed of “Background flows” (BF) and “Deadline flows” (DF)<sup>3</sup>. BF are CBR flows with a rate randomly chosen in the set {50, 60, 70, 80} kbit/s. DF are divided into two classes: DF1 and DF2. DF1 has a 5 s deadline, while DF2 a 9 s constraint. The overall traffic descriptors and requirements are defined in Table 3.

We tested two configurations by using 125 generated flows. In the first one s1 has 1 queue on the outgoing interface ( $q_0$ ) with a FIFO (First Input, First Output) service rate  $s_{q_0} = 3$  Mbit/s, whereas in the second one it has 3 queues, each of them dedicated to a specific traffic:  $q_0$  for BF,  $q_1$  for DF1 and  $q_2$  for DF2. The service rate of the queues (set in advance) are  $s_{q_0} = 300$  kbit/s,  $s_{q_1} = 1.7$  Mbit/s,  $s_{q_2} = 1$  Mbit/s. The QoS metrics considered here are the packet loss rate in percentage for BF and the percentage of flows matching the deadline for DFs as shown in Table 1.

As one can note the 3-queue configuration consistently improves the percentage of flows matching the deadline and penalizes the packet loss rate of BF. Setting the service rates of simple queues differently, the performances will change but it is clear that traffic differentiation through multi-queues interfaces gives the fundamental gears to manage deadline flows and to tune the level of performances of the network traffic.

Table 2 shows the results of the second set of simulations we ran aimed at showing how the power to change the service rate of the queues can improve the deadline management performances. As the previous simulation, s1 is receiving a mix of traffic composed of BF, DF1 and DF2. Again two configurations are tested with the same number of generated flows. In the first configuration, s1 has 3 queues with a pre-fixed service rate:  $s_{q_0} = 2$  Mbit/s,  $s_{q_1} = 4$  Mbit/s and  $s_{q_2} = 4$  Mbit/s, whereas in the second one,  $q_1$  can grab the spare capacity from the other two when it needs more bandwidth:  $s_{q_1}$  is in the range [4 – 10] Mbit/s.

The variable rate configuration consistently improves the total percentage of flows that match the deadline, leading it up to 100%,

<sup>2</sup> Open vSwitch (OVS) [15] is a production-quality open source implementation of a virtual switch in Linux.

<sup>3</sup> As said before, DF are the flows for which there is an associated deadline: the flow is useful if, and only if, is completely received at the destination within the deadline.

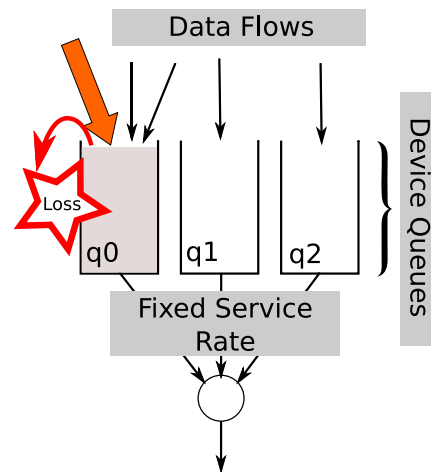


Fig. 1. Congestion at one of the queue.

without any impact on BF packet loss. In Table 2 the label DF tags the Deadline flows without distinction between DF1 and DF2.

Unfortunately, as highlighted in the introduction, current OF specification [16] is not able to configure queues’ service rate and delegates this task to an external dedicated configuration protocol: “Queue configuration takes place outside the OpenFlow protocol, either through a command line tool or through an external dedicated configuration protocol.” ([17], Section 7.3.5.10). As a consequence, this paper, even if applies multiple queues for traffic differentiations, supposes queue’s service rate set and unchangeable in a SDN switch.

## 4. Possible solutions and required Beacon modification

### 4.1. General idea

Although the design and implementation of a new OpenFlow directive able to configure the queues’ service rate would be the best solution in terms of performances, this choice would come up with a main drawback: it would be totally incompatible with current OF switches that would not take any benefit from the directive.

For this reason we propose an alternative solution *totally compatible* with current OF switches. The underlying idea is shown in Figs. 1 and 2.

Let us suppose that, during the network operation, the OF switch in Fig. 1 receives 5 flows that manages through 3 outgoing queues  $q_0$ ,  $q_1$  and  $q_2$ . Let us suppose that the orange flow (i.e. the largest arrow) increases its data rate so that  $q_0$  receives more packets than those it can handle.  $q_0$  incoming rate is higher than the pre-configured service rate. In this situation, increasing the incoming rate eventually leads to packet loss and to a severe reduction of the quality experienced by the flows in  $q_0$ . Being unable to change the service rate of the queue, a possible solution involves the re-routing of some flows arriving at  $q_0$  to another queue (e.g.  $q_1$  in Fig. 2) in order to reduce  $q_0$  incoming rate. Re-routing mechanisms attempt to use the spare bandwidth unused by other queues.

Since we want to keep simple both OF switches and OF specification, we design and implement re-routing mechanisms inside the SDN controller. Even if the idea is simple, the design of re-routing mechanisms involves functionalities of the SDN controller and, in particular, the following features/requirements:

- The compatibility with early versions of OpenFlow (which is obviously a must);

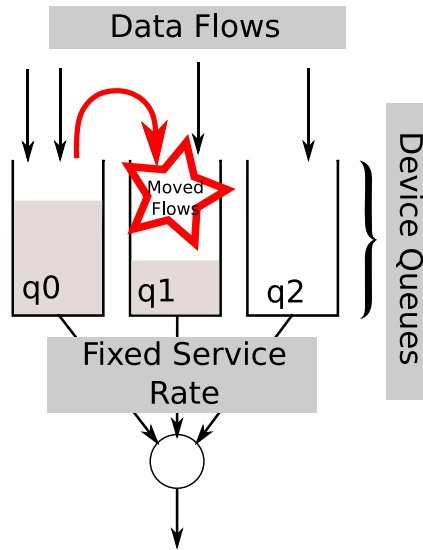


Fig. 2. Re-routing of some flows.

- The creation of a module able to handle statistics;
- The implementation of the proposed approaches;
- No primitives shall be modified with respect to the current OpenFlow standard.

The idea of re-routing and the strategies proposed in this paper can be exploited both for specific deadline management purposes, and in the context of the optimal management of hardware resources provided to common software routers. Software routers can run on off-the-shelf general-purpose CPUs and commodity hardware, rather than on expensive dedicated hardware. Commodity hardware not only maintains a high level of programmability and flexibility but is more cost-efficient than specialized hardware solutions and network components. For this reason, software routers are largely widespread [18]. On the other hand, it has been proved that the CPU is the main bottleneck in a software router. Recent advances propose to increase the packet processing performance through parallel processing based on off-the-shelf multi-core processors [19]. In more detail, current software routers implement filters for multi-queue NICs (Network Interface Controllers) used to address incoming packets to a certain queue based on specific packet attributes. By these filters, NICs are able to efficiently distribute the incoming packet processing workload across multiple CPU cores. This also ensures that each packet of a specific flow is served by the same CPU core so avoiding, for example, packet reordering [20].

Instead of using dedicated hardware filters provided by NICs we propose a flexible solution based on the OpenFlow architecture. Our approach consists in using an OF software controller which can monitor incoming flows and has the intelligence to decide the correct queueing strategy. We develop a series of control algorithms able to re-arrange flows in order to make lighter the computational burden of the CPU by equally distributing flows among the available queues.

#### 4.2. Implementation: BeaQoS

We chose Beacon [5] as SDN controller. Beacon is a multi-threaded Java-based controller that relies on OSGi and Spring frameworks and it is highly integrated into the Eclipse IDE. Anyway, independently of the specific choice of the controller, our modifications can be implemented in any controller. The structure of the controller consists of a group of functions (called bundles)

**Table 4**  
BeaQoS Statistics compared with OpenFlow 1.0 statistics.

Statistics available in OpenFlow 1.0	Statistics computed by BeaQoS
Tx Bytes per Flow	→ Estimated Rate per Flow
Tx Bytes per Port	→ Estimated Rate per Port
Tx Bytes per Queue	→ Estimated Rate per Queue
FlowMatch FlowActions QueueID	→ Flows per Queue

with dedicated functionalities. The main bundle we focused on is the *Routing* one, which takes care of finding the correct path between the source and destination to forward packets. Moreover, we created an ad-hoc bundle, called *Statistics*, to the purpose of collecting and processing the statistics of the reply messages provided by network switches. The principal proposed modifications of Beacon are:

**Statistics polling** Beacon controller has been modified in order to send statistic requests to the switches. We added a function that triggers the dispatch of statistic and feature request messages with a polling interval (*PI*) configurable through an external properties file. We also designed and implemented a class dedicated to the creation of statistic request messages, such as *ofp\_flow\_stats\_request*, *ofp\_port\_stats\_request*, *ofp\_queue\_stats\_request* [21], in order to obtain useful information about the status of flows, ports and queues.

**Statistics** This module has two main functions: one is devoted to the creation of the data structures needed to generate a database of statistics related to the network nodes, the other one is dedicated to implement the collection of data extracted from the messages about statistics. The reply messages obtained from the network switches are the introduced *ofp\_flow\_stats*, *ofp\_port\_stats*, *ofp\_queue\_stats*. In addition to the basic statistics that the OpenFlow protocol 1.0 makes available, we added specific functions to the controller, which allow BeaQoS to exploit the collected data in order to compute parameters useful to apply the chosen strategy. The additional statistics computed by BeaQoS are shown in Table 4, compared with the ones available in OpenFlow 1.0.

The main extracted feature is the Estimated Rate (*ER*) for ports, queues, and flows. We computed the Estimated Rate  $ER^t$  at a given time instant as follows:

$$ER^t = \frac{TB^t - TB^{t-1}}{PI} \quad (1)$$

in which  $t$  is the sampling instant,  $TB^t$  are the transmitted bytes at the current instant,  $TB^{t-1}$  are the transmitted bytes at the previous sampling instant and  $PI$  represents the polling interval in seconds. Obviously the quantity “transmitted bytes” and, consequently, the expression in (1), may be applied to ports, queues, and flows. Another parameter we extracted is the number of flows currently belonging to a specific queue (Flows per Queue).

**Routing** This module has been modified so as to implement the proposed algorithms. When a switch receives a new flow, it contacts the controller in order to know where to forward the traffic. When the controller has to assign each flow to a specific queue, it checks a variable that identifies the algorithm to run. BeaQoS performs a routine to select the correct queue based on the chosen strategy and then notifies the node through the installation of a flow modification.

The proposed approaches are described in detail in the following section.

## 5. Re-routing strategies analysis

In this section, we present two main scenarios in which we compare different proposed re-routing algorithms to find the most efficient solution. The first scenario deals with the problem of the priority flows that must be served within a specific deadline, as introduced in Section 3. The second one faces the issue of balancing the load among different queues in a single SDN node.

### 5.1. Deadline management scenario

In this scenario we consider both “Background flows” (BF) and “Deadline flows” (DF). As previously described, DF are flows for which there is an associated deadline: the flow is useful if, and only if, it completes within the deadline [4]. DF are of interest in datacenter applications (e.g. web search, social networking) where user requests need to be satisfied within a specified latency target and when the time expires, responses, irrespective of their completeness, are shipped out<sup>4</sup>. Moreover, online services have a partition-aggregate workflow, being user requests partitioned among (multiple) layers of servers (workers) whose results are then aggregated to form the response. The combination of latency targets and partition-aggregate workflow has implications for the traffic inside the datacenter. Specifically, for any network flow initiated by these workers, there is an associated deadline.

We propose and implement two schemes in order to provide a basic support for deadline management inside a SDN network with mixed traffic BF, DF1 (each flow with *deadline1*) and DF2 (each flow with *deadline2*). To clarify the description of these approaches we assume that all interfaces of each switch are configured with three queues:  $q_0$ ,  $q_1$ ,  $q_2$ .  $q_0$  is dedicated to BF, whereas the others are used for DF1 and DF2, respectively. The schemes are the following:

**Dedicated** This scheme assigns each traffic class to a specific queue of the considered switch port. Upon the arrival of a new flow inside the switch, the routing engine of the Beacon controller decides which queue to choose based on the traffic descriptor of the flow. BF are enqueued on  $q_0$ , DF1 are assigned to  $q_1$  and DF2 are assigned to  $q_2$ .

**Deadline** This scheme is triggered when the controller receives a request from a switch on how to manage an upcoming flow. The routing module checks the Type of Service field<sup>5</sup>: BF are enqueued on  $q_0$ , whereas for DF1 or DF2, the controller chooses the less utilized queue  $q_{i^*}$ . The utilization of the queues is computed based on the following function  $U(q_i)$ :

$$i^* = \arg \min_{i=1,2} U(q_i); \quad U(q_i) = s_{q_i} - \sum_k target_k \cdot n_{k,q_i} \quad i = 1, 2 \quad (2)$$

being:  $s_{q_i}$  the service rate of  $q_i$ , known a-priori and configurable from an external properties file;  $k$  the index that spans among the classes of service (here DF1 and DF2);  $target_k$  the rate we need to guarantee to the flow of class  $k$ <sup>6</sup>;  $n_{k,q_i}$  the number of flows belonging to the class  $k$  and assigned to  $q_i$ .

The aim is to maximize the number of DF whose deadline is matched, even at the expense of background flows, if necessary.

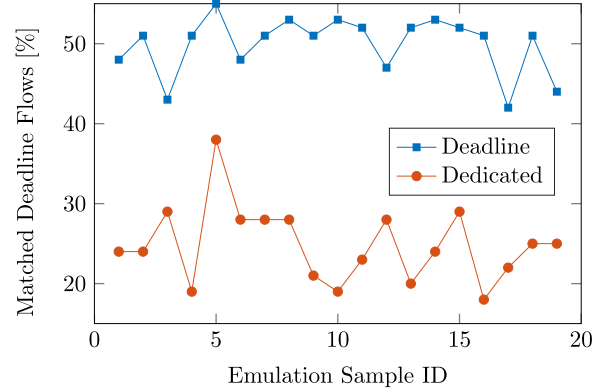
<sup>4</sup> Today's online services have service level agreements (SLAs) baked into their operation [22–24].

<sup>5</sup> We choose the ToS field to differentiate DF1 and DF2 having in mind the DSCP (Diff Serv Code Point) bits in the ToS field, but other solutions can be implemented.

<sup>6</sup> For example, a flow of class  $k$  with size of 100 kByte and a deadline of 10 s needs a  $target_k \geq 10 \text{ kByte/s}$ .

**Table 5**  
Queue configurations.

Queue ID	Service rate	Buffer size
$q_0$	0–3 Mbit/s	1000 packets
$q_1$	2 Mbit/s	1000 packets
$q_2$	1 Mbit/s	1000 packets



**Fig. 3.** Percentage of flows that satisfy the deadline, computed with  $H = 100$ .

We carried out the performance analysis on a PC running Mininet (version 2.1.0) [25]. The scenario is composed of two hosts connected to a SDN switch. The chosen implementation of the switch is Open vSwitch 2.0.2 [15], managed by an instance of BeaQoS running on the same machine. Each port of the switch is configured with 3 queues,  $q_0$ ,  $q_1$ ,  $q_2$ . The rate assigned to each buffer is shown in Table 5. The overall service rate is 3 Mbit/s. The queue dedicated to BF has a variable service rate ranging from 0 to 3 Mbit/s: this implies that  $q_0$  can be served only if the priority queues are not using the entire link bandwidth. Queue service rates are configured through the Traffic Control (*tc*) module in Linux Kernel.

The traffic used for these simulations, generated through the *iperf* tool, consists, as said above, of 3 types of flows, BF, DF1 and DF2 composed of the following percentages and features: 30% of the overall traffic is BF, which is characterized by a random rate chosen in the set {50, 60, 70, 80} kbit/s and a duration of 50s; 55% is DF1, generating data at 4.5 Mbit/s for 1s and, undergoing a deadline of 9s; and 15% is DF2 with 1.5 Mbit/s data rate for 1s and with a deadline of 5s. The summary of traffic descriptors and requirements are reported in Table 3, already used for the results in Section 3.

In this scenario we compare the performances of the two different proposed solutions: Dedicated and Deadline.

We ran an emulation of 3 hours of duration composed of 3000 flows structured into BF and DF flows as described above. We present the obtained values averaged over an *Horizon* ( $H$ ) of consecutive flows. Each averaged value is called Emulation Sample ID. The metrics used to compare the proposed approaches are the percentage of Matched Deadline Flows (e.g. the percentage of flows satisfying the deadline) and the Loss of Background Flows (i.e. the percentage of lost packets of BF flows). For what concerns Figs. 3 and 4, showing the Matched Deadline Flows, an *Horizon*  $H = 100$  and  $H = 250$ , respectively, is applied taking into account only DF flows. Figs. 5 and 6, showing the Loss of Background Flows, apply again  $H = 100$  and  $H = 250$ , respectively, but involving only BF flows.

Intuitively, large  $H$  values capture the steady state of the system and small  $H$  values present more measurement noise. Instead of choosing a specific  $H$  or trying to capture a flat steady state behaviour, we decided to track the performances over fixed time

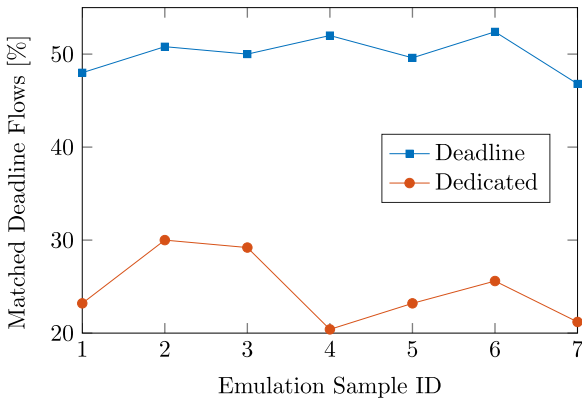


Fig. 4. Percentage of flows that satisfy the deadline, computed with  $H = 250$ .

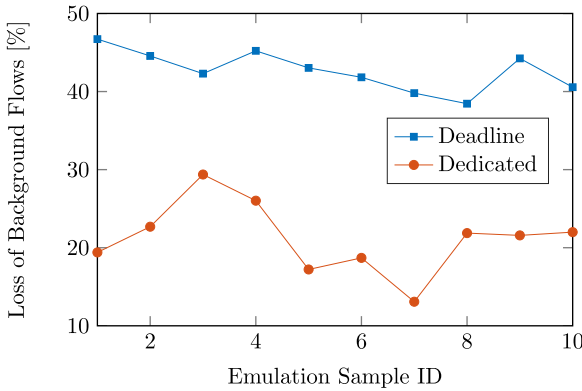


Fig. 5. Percentage of lost packets for Background Flows, computed with  $H = 100$ .

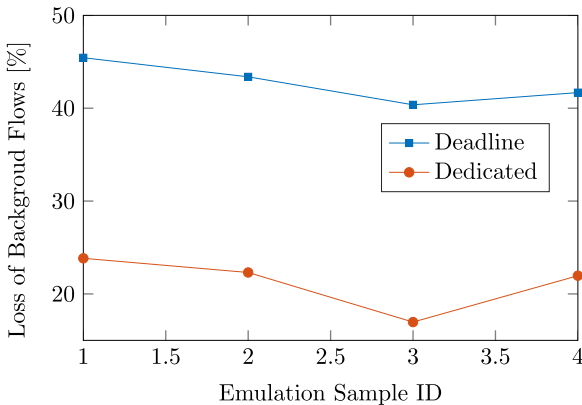


Fig. 6. Percentage of lost packets for Background Flows, computed with  $H = 250$ .

horizons in order to obtain a more realistic approach, as discussed in [26] and [27]. The results of these tests show that the Deadline scheme allows satisfying the time constraints of a much larger number of DF than the Dedicated scheme. In practice, the Deadline scheme is able to double, on average, the performance of the other approach, referring to Matched Deadline Flows (Figs. 3 and 4). The improvement of the number of DF flows matching the deadline is obtained at the expense of BF traffic, which suffers from a much higher packet loss than in the Dedicated scheme, as shown in Figs. 5 and 6.

In short independently of the  $H$  value, the Deadline technique is better than the Dedicated one with respect to the percentage of satisfied deadlines for DF flows, at the cost of increasing the loss achieved on BF flows.

## 5.2. Queue balancing scenario

As far as load balancing strategies are concerned, we propose three schemes aimed at equalizing the traffic burden in each queue. In order to better illustrate the operating principles of our solutions, we assume a network scenario in which each interface of each switch has four available queues,  $q_0$ ,  $q_1$ ,  $q_2$  and  $q_3$ . The service rate of the outgoing interface is equally divided among the different queues. The proposed schemes are the following:

**Min load** This scheme consists in assigning the upcoming flow to the least loaded queue. This task is performed by the routing module of the BeaQoS controller. When a new flow reaches a SDN switch the controller checks the estimated rate (computed as in (1)) of the queues belonging to the considered output port and selects the one which has the minimum value.

If we think to the rate of the flows as numbers, it is possible to model the load balancing problem among the available queues as a problem of partitioning a given set of numbers into a collection of subsets so that the sums of the numbers in each subset (i.e. the queues of the switches) are as close as possible [28]. This problem is already known in literature as Multi-Way Number Partitioning and it is NP-complete. For the sake of simplicity we choose to implement an algorithm, which we call Multiway, based on the greedy heuristic described below.

**Multiway** In this scheme all the flows are queued into  $q_0$  at the beginning, then the controller periodically runs a scheme that sorts the flows in decreasing order based on the computed Estimated Rates (ER) in (1) and assigns each flow, analysed by following the established ER decreasing order, to the queue with the lower utilization so far, in order to equalize the load among the queues.

**N-migrations** When the number of flows is huge, the Multiway approach tends to become computationally heavy since it has to analyse and possibly move all the flows traversing the interface. For this reason we introduced the N-Migration strategy, where the number of flow migrations is limited to  $N$ . The algorithm runs on scheduled times and iterates  $N$  times a routine which selects a flow from the most loaded queue and re-routes it in the least loaded one. The flow selected by the strategy is the one which assures the best load equalization among the queues. This selection is performed evaluating all the possible outcomes through a simple simulation of re-routing.

Although these strategies may seem similar, the performance results are different. Tests about Queue Balancing use a very similar Mininet topology as described for the Deadline scenario. The overall rate availability is 4 Mbit/s. The main difference is in the configuration of the queues inside the OpenFlow switch: each interface of the switch has four queues,  $q_0$ ,  $q_1$ ,  $q_2$ ,  $q_3$  and the rate of the outgoing interface is equally divided among the different queues such as each one has 1 Mbit/s available.

The traffic used in these simulations was generated by using the *iperf* tool and consisted of flows with a rate randomly chosen in the set {50, 60, 70, 80, 90, 100} kbit/s. Flow duration is 50 s.

The network was tested with increasing workloads: 100, 125 and 150 flows running with different seeds.

To better analyse the results, we introduce a performance index that provides a measure of accuracy of our algorithm with respect to the optimal solution, which ideally allows getting the exact amount of traffic in every queue to get load balancing. We call this parameter *index* and we compute it at each time instant  $t$  as:

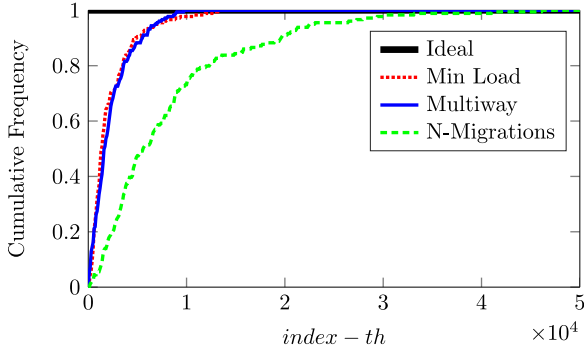


Fig. 7. Queue balancing performances with 100 flows.

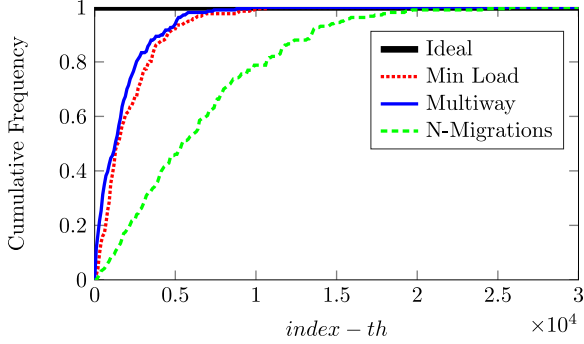


Fig. 8. Queue balancing performances with 125 flows.

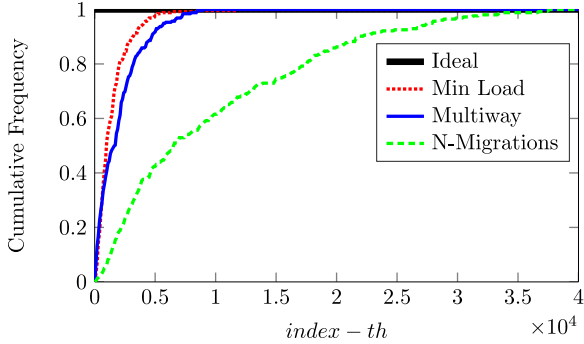


Fig. 9. Queue balancing performances with 150 flows.

$$index^t = \frac{\sum_i (r_{q_i}^t - \bar{r}^t)^2}{4}, \quad t = 0, 1, \dots \quad (3)$$

where  $r_{q_i}^t$  is the measured output rate of queue  $q_i$  and  $\bar{r}^t$  is the optimal queue rate, both evaluated at time instant  $t$ . In other words,  $index^t$  is a measure of the distance between our solution and the ideal one.

The following plots show the Cumulative Frequency (CF) of  $index^t$ . CF is defined as the number of occurrences over the total samples in which the  $index^t$  is below a certain threshold ( $index - th$ ). Figs. 7–9 show CF versus  $index - th$ , for Min Load, Multiway and N-Migrations in case of 100, 125 and 150 flows, respectively. For what concerns the N-Migrations approach, the N parameter is set to 1 for all simulations.

The results highlight that, in all examined cases, Min Load and Multiway schemes show a very satisfying behaviour and have better performances with respect to the N-Migrations approach. For example, when we consider 100 flows inside the network, as shown in Fig. 7, we can say that, in 90% of cases, the distance between our solution and the ideal one doesn't exceed 5000 for what

concerns Min Load and Multiway strategies. On the contrary, N-Migrations accuracy curve has a less steep trend than the alternative solutions: the value of  $index$  for this approach is below 5000 in 50% of cases. In particular it is important to note that Min Load and Multiway behaviours are very close to the Ideal one (CF is 1 for any  $index - th$  value, including 0) and overlap it for a relatively small  $index - th$ .

Also the simulations involving 125 and 150 flows confirm the same behaviour, as shown in Figs. 8 and 9.

Concerning N-Migrations: the results show that the N-Migrations approach cannot achieve the same performances of the Min Load and Multiway. This is due to the choice of the N parameter, which is the key of the algorithm. This parameter can be set in order to tune the performances of this approach: as the N parameter grows, the behaviour of the algorithm approaches the Multiway scheme. The choice of the N parameter leads to a trade-off between performance and computational complexity.

## 6. Considerations

### 6.1. Scaling performances

Concerning statistics (see Table 4) acquisition: the types of messages sent by the controller are flow, queue and port requests that are used to gather information about port rates, queue rates and individual flow statistics. The controller receives three statistic replies, one for ports, one for queues and one dedicated to all flows traversing the OpenFlow switch in a given instant.

Since the maximum information sent through the Ethernet frame is 1500 byte, each flow statistics reply message can report only the information about 10 flows. For this reason the number of flow statistic packets in the case of  $f$  flows is  $\lceil f/10 \rceil$ . Given  $N$  the number of switches composing the network and considering another two packets for port and queue statistics, the number of packets  $p$  that the controller must process at every polling interval is

$$p = \left( \left\lceil \frac{f}{10} \right\rceil + 2 \right) \cdot N \quad (4)$$

Considering a significant number of flows  $f$  and switches  $N$ , the number of packets  $p$  received by the controller can be large. This is the price of a fine-grained control of an SDN network at flow-level (*IntServ*). The number of  $p$  can be reduced by using the flows statistics for a small number of “aggregate” flows. This could reduce the fine-grained control but relieves the controller from the management of a large number of packets.

A performance analysis with a large scale scenario will be the next step of our work in this topic.

### 6.2. Switch coordination

Even if in this paper we show the results by using a single OpenFlow switch in the network, it is possible to extend the concept across multiple SDN devices. The routing module implemented in the BeaQoS controller can manage more than one single switch. For each switch the controller computes all the needed parameters in order to provide the best behaviour, given the chosen algorithm. In order to extend this concept to the entire network, given a specific path to the destination, it would be possible to compute the optimal queue  $q_{i^*}$  for each switch belonging to the specific path.

Alternatively, since the controller BeaQoS has the view of the entire network, another possible solution is to examine all existing paths between source and destination for the considered flow. The controller could then compute the best path for the specific

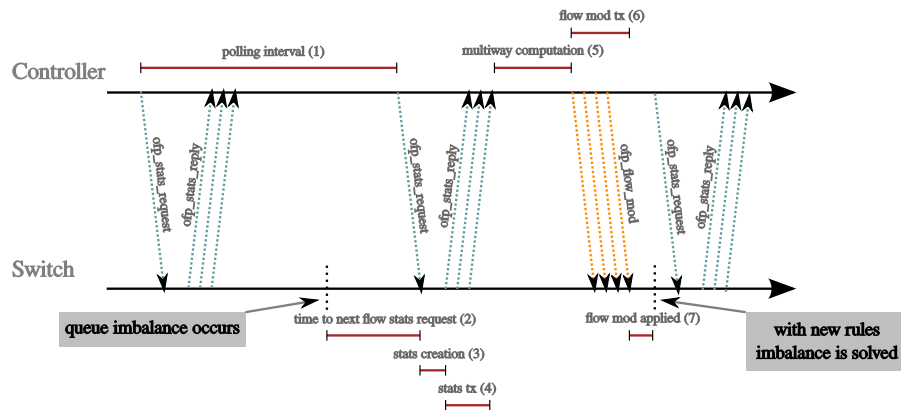


Fig. 10. Timing performances in queue balancing scenario.

flow and finally decide the optimal queue  $q_i^*$  for all the switches belonging to the selected path.

### 6.3. Timing performances and overheads in queue balancing scenario

In queue balancing scenario timing performances are essential to guarantee an “almost” instantaneous load balance among queues in each switch. The main difficulty of this approach is due to the remote nature of the actions of the SDN controller that acts as if the actions were internal switch functionalities. The time elapsing from the load imbalance event at the switch and the new queue balance (*queue balance delay*) can be expressed as the sum of several components, as depicted in Fig. 10.

All our tests are performed with a relatively small number of flows. This allows the controller to manage per flow performances. Considering the Multiway algorithm, the controller can reorder the total amount of flows traversing an SDN switch in a time of the order of milliseconds. Moreover, considering that the controller is connected with the switches using an out of band connection, the time needed to deliver the flow modifications is negligible.

In a large scale scenario with a huge number of flows, it is possible to aggregate flows, reducing the number of sent flow stats and the computation time of the Multiway algorithm.

## 7. Conclusions

The impossibility to configure the service rate of the queues in a OpenFlow switch through an OF directive is a limitation that could reduce the quality management capabilities in an SDN network but it is a fact for now.

In this paper, exploiting the re-routing mechanism, we propose a method able to provide a basic deadline management support and an efficient queue balancing without any modification of OpenFlow specifications and switches. We present BeaQoS, an updated version of the Beacon controller able to receive statistics from OpenFlow switches, compute more complex statistics and decide the best queue re-routing strategy. We show the results obtained in performance tests in which we compare alternative Deadline Management approaches and Queue Balancing solutions. Our cases of study show that the proposed solutions allow getting satisfying results when applied to the current OpenFlow environment.

Future developments will be devoted to the scalability tests of our solutions and to the study of more complex queue management schemes that could lead to further improvements in performances. We also plan to develop an extension of our internal re-routing approach for the computation of alternative paths be-

tween the source and destination, in order to reduce the network congestion.

## References

- [1] M. Casado, M.J. Freedman, J. Pettit, J. Luo, N. McKeown, S. Shenker, Ethane: Taking control of the enterprise, in: Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, in: SIGCOMM '07, 2007, pp. 11–12.
- [2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, Openflow: enabling innovation in campus networks, SIGCOMM Comput. Commun. Rev. 38 (2) (2008) 69–74.
- [3] M. Marchese, QoS Over Heterogeneous Networks, Wiley Publishing, 2007.
- [4] C. Wilson, H. Ballani, T. Karagiannis, A. Rowtron, Better never than late: meeting deadlines in datacenter networks, in: Proceedings of the ACM SIGCOMM 2011 Conference, in: SIGCOMM '11, 2011, pp. 50–61.
- [5] D. Erickson, The Beacon Openflow Controller, in: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, in: HotSDN '13, 2013, pp. 13–18.
- [6] L. Boero, M. Cello, C. Garibotto, M. Marchese, M. Mongelli, Management of non-conformant traffic in openflow environments, in: Performance Evaluation of Computer and Telecommunication Systems (SPECTS), 2015 International Symposium on, IEEE, 2015, pp. 1–6.
- [7] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, S. G. Rsvp-te: extensions to rsvp for lsp tunnels, 2001, (RFC 3209).
- [8] D. Applegate, M. Thorup, Load optimal mpls routing with  $n + m$  labels, in: INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies, 1, 2003, pp. 555–565.
- [9] N. Handigol, S. Seetharaman, M. Flajslik, R. Johari, N. McKeown, Aster\*x: load-balancing as a network primitive, in: Plenary Demo, 9th GENI Engineering Conference, in: 9th GENI, 2010.
- [10] R. Wang, D. Butnariu, J. Rexford, Openflow-based server load balancing gone wild, in: Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services, in: Hot-ICE'11, 2011, 12–12.
- [11] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, R. Wattenhofer, Achieving high utilization with software-driven wan, in: Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM, in: SIGCOMM '13, 2013, pp. 15–26.
- [12] S. Agarwal, M. Kodialam, T. Lakshman, Traffic engineering in software defined networks, in: INFOCOM, 2013 Proceedings IEEE, 2013, pp. 2211–2219.
- [13] S. Gutz, A. Story, C. Schlesinger, N. Foster, Splendid isolation: a slice abstraction for software-defined networks, in: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, in: HotSDN '12, 2012, pp. 79–84.
- [14] OpenFlow Management and Configuration Protocol, Open Networking Foundation, 2014 <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow-config/of-config-1.2.pdf>.
- [15] Open vSwitch, 2014, (<http://openvswitch.org/>).
- [16] OpenFlow Switch Specification - version 1.5.0, 2015, (<https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.5.0.noipr.pdf>).
- [17] Openflow switch specification - version 1.4.0, 2013, (<https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf>).
- [18] M. Dobrescu, N. Egi, K. Argyraki, B. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, S. Ratnasamy, Routebricks: exploiting parallelism to scale software routers, ACM Symposium on Operating Systems Principles (SOSP), 2009.
- [19] T. Meyer, D. Raumer, F. Wohlfart, B. Wolfinger, G. Carle, Validated model-based performance prediction of multi-core software routers, Praxis der Informationsverarbeitung und Kommunikation (PIK), pp. 1–12.



- [20] T. Meyer, D. Raumer, F. Wohlfart, B. Wolfinger, G. Carle, Low latency packet processing in software routers, in: Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2014), International Symposium on, 2014b, pp. 556–563.
- [21] OpenFlow Switch Specification - version 1.0.0, 2009, (<https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf>).
- [22] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, W. Vogels, Dynamo: Amazon's highly available key-value store, *SIGOPS Oper. Syst. Rev.* 41 (6) (2007).
- [23] T. Hoff, 10 ebay secrets for planet wide scaling, 2009. (<http://highscalability.com/blog/2009/11/17/10-ebay-secrets-for-planet-wide-scaling.html>)
- [24] W. Vogels, Performance and scalability, 2009 [http://www.allthingsdistributed.com/2006/04/performance\\_and\\_scalability.html](http://www.allthingsdistributed.com/2006/04/performance_and_scalability.html).
- [25] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, N. McKeown, Reproducible network experiments using container-based emulation, in: Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies, in: CoNEXT '12, ACM, New York, NY, USA, 2012, pp. 253–264.
- [26] C.G. Cassandras, Y. Wardi, B. Melamed, G. Sun, C.G. Panayiotou, Perturbation analysis for online control and optimization of stochastic fluid models, *IEEE Trans. Autom. Control* 47 (8) (2002) 1234–1248.
- [27] M. Cello, M. Marchese, M. Mongelli, On the qos estimation in an openflow network: The packet loss case, *IEEE Commun. Lett.* 20 (3) (2016) 554–557.
- [28] R.E. Korf, Multi-way number partitioning, in: International Joint Conferences on Artificial Intelligence, Citeseer, 2009, pp. 538–543.



**Luca Boero** was born in Genoa, Italy in 1989. In 2012 he got his Bachelor Degree in Telecommunication Engineering at the University of Genoa; in 2015 he achieved a Master Degree in Multimedia Signal Processing and Telecommunication Networks with a thesis on Quality of Service in Software Defined Networking. From November 2015 he is a Ph.D student at the University of Genoa. His main research activities concern networking and SDN.



**Marco Cello** was born in Savona, Italy in 1983. He got his "Laurea Magistrale" (M.Sc.) degree cum laude and his Ph.D in 2008 and 2012, respectively both at the University of Genoa. In 2012, 2014 and 2015 worked as Post-doc research fellow at University of Genoa with a fellowship funded by Fondazione Carige. In 2013 he was Post-Doc research fellow at Polytechnic Institute of New York University and Visiting Research Fellow at New York University Abu Dhabi. He is currently Post-Doc research fellow at Nokia Bell Labs in Dublin, Ireland. He is a researcher on networking with almost 8 years of experience, including managing research projects funded by national industries, the European Community and the European Space Agency (ESA). He has strong expertise in software for simulation, Linux-based emulation of telecommunication networks and Linux administration. He is a co-author of over 20 scientific works, including international journals, conferences and patents. His main research activities concern: Network Modelling/Teletraffic Engineering; Call Admission Control; Routing and Congestion Control in Delay Tolerant Networks; Software Defined Networking.



**Chiara Garibotto** was born in Chiavari, Italy in 1985. In 2012 she got her Bachelor Degree in Telecommunication Engineering at the University of Genoa; in 2015 she achieved a Master Degree in Multimedia Signal Processing and Telecommunication Networks with a thesis on Quality of Service in Software Defined Networks. From November 2015 she is a Ph.D student at the University of Genoa. Her main research activities concern networking and signal processing.



**Mario Marchese** was born in Genoa in 1967. He got his degree with honors from the University of Genoa in 1992 and the PhD in July 1997. From 1999 to 2005 he worked at the National Consortium for Telecommunications (CNIT). Associate Professor at the University of Genoa from 2005 to January 2016, since February 1, 2016 he has been Full Professor at the same University. He researched at the German Aerospace Center (DLR), as a Visiting Professor / Guest Scientist. He is the founder and head of the Laboratory "Satellite Communications and Networking". He coordinated the technical-scientific and financial management of many research projects. He got 4 patents. He has published over 280 scientific papers including 1 international book, 2 edited books, 77 articles in international journals and 13 book chapters. He has been the Coordinator of the PhD in "Science and Technology for Electronic and Telecommunication Engineering" since 2013. He was "Chair" (2006–2008), "Vice-Chair" (2004–2006) and "Secretary" (2002–2004) of the "Satellite and Space Communications Technical Committee" of "IEEE ComSoc". He is the winner of the IEEE ComSoc "Satellite Communications Distinguished Service Award" in 2008 and of numerous "Best Paper Award". His main research interests concern: space, satellite and heterogeneous telecommunications networks, quality of service over heterogeneous networks and applications for smartphones.



**Maurizio Mongelli** got his PhD degree in Electronic and Computer Engineering at the University of Genoa (UniGe) in 2004. His PhD was funded by Selex Communications SpA (Selex). He worked for both Selex and the CNIT from 2001 to 2010. During the PhD and in the subsequent years, he worked on quality of service for military networks for Selex. From 2007 to 2008, he coordinated a joint laboratory between UniGe and Selex, dedicated to Ethernet resilience. He was recently the CNIT technical coordinator of a research project concerning satellite emulator systems, funded by the European Space Agency; he spent 3 months working on the project at the German Aerospace Centre in Munich, Germany. He is now a researcher at the Institute of Electronics, Computer and Telecommunication Engineering of the Italian National Research Council. He is a co-author of over 70 scientific works, including international journals, conferences and patents. His main research activity concerns resource allocation and optimization algorithms for telecommunication, machine learning and cybersecurity.