Management of Non-Conformant Traffic in OpenFlow Environments

Luca Boero †, Marco Cello †, Chiara Garibotto †, Mario Marchese †, Maurizio Mongelli §

† University of Genoa, Genoa, Italy
3257607@studenti.unige.it, marco.cello@unige.it
2798600@studenti.unige.it, mario.marchese@unige.it

§ National Research Council, Genoa, Italy
maurizio.mongelli@ieiit.cnr.it

***Abstract*—Nowadays many systems and applications require the identification of traffic classes, which must be compliant to specific pre-defined constraints. Traditional networking approaches are based on traffic shaping, which is used to force the traffic to comply to the service agreements. This task is usually demanded to the underlying devices and this choice causes limited flexibility since the network functionalities are strictly dependant on the specific hardware. In this paper 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 limit the quality degradation of flows traversing the network.**

***Index Terms*—SDN, OpenFlow, Packet Loss, Traffic Engineering**

I. Introduction

Many applications nowadays rely on *Quality of Service* (QoS) guarantees. Some of them are telemedicine, tele-control (remote control of robots in hazardous environments, remote sensors and systems for tele-manipulation), tele-learning, telephony, video-conferences, online gaming, multimedia streaming and applications for emergencies and security. Each application, having very different characteristics, needs a specific degree of service, defined at the application layer.

Concerning the network viewpoint, QoS is the ability of a network element to have some level of assurance that its traffic and service requirements can be satisfied. QoS manages bandwidth according to application demands and network management settings [14]. The term QoS is used in different meanings, ranging from the users' perception of the service to a set of connection parameters necessary to achieve particular service quality. The QoS meaning changes depending on the application field and on the scientific scope. Three types of Quality of Service have been defined, based on the main aspect we focus on: *Intrinsic*, *Perceived* and *Assessed QoS*. Currently, most of the QoS provision is offered in terms of intrinsic QoS (objective parameters) by using a *Service Level Specification* (SLS) which is a set of parameters and their values which together define the service offered to a traffic [9].

QoS management techniques are required in order to offer the necessary tools to guarantee specific QoS requirements. Traditional approaches to QoS management range from the simple Over Provisioning, consisting of purchasing an oversupply of bandwidth to solve the challenges, to the Traffic Control, in which shaping policies limit flows to their committed rates in order for the flows to be conform with their traffic descriptor. Other techniques involve Call Admission Control [6], [7], Scheduling, and Flow Control. Shaping policies, for example, limit flows to their committed rates in order for the flows to be conform with their traffic descriptors. If connections exceed their bandwidth consumption specifications, the network, which has dimensioned resources in strict dependence on the declarations, cannot guarantee any specified QoS requirement. Two common methods used in literature to shape traffic are Leaky Bucket and Token Bucket [16], but they are strictly related to the hardware implementation.

In the traditional approach to networking, most network functionality is implemented in a dedicated appliance such as switch, router, application delivery controller. In addition, within the dedicated appliance, most of the functionality is implemented in dedicated hardware such as an Application Specific Integrated Circuit (ASIC). This kind of approach is characterized by slow evolution of network functionalities, which are by the way under the control of the provider of the device. The widespread adoption of server virtualization and the consequent need to move virtual machines dynamically between servers lead to increasing pressure for network organizations to be more efficient and agile in network management.

As the traditional networking paradigm has a rather static nature, the necessity for a new software-oriented approach started to arise. One of the main solutions that fulfil the aforementioned need, is *Software Defined Networking* (SDN). The main opportunities that SDN can address are the support of dynamic movement and allocation of network resources, the scalability of network functionalities and the reduction of network complexity. Software Defined Networking allows also

SummerSim-SPECTS, 2015, July 26-29, 2015, Chicago, IL, USA

to perform traffic engineering with an end-to-end view of the network and to apply more effective security functionalities [15]. The SDN architecture has the aim of leaving the intelligence outside the data plane, in order to make the mechanisms simpler and quicker. The control program can therefore be implemented inside a supervising SDN controller, which can interact with the network switches using the forwarding model, whose main implementation is the *OpenFlow Protocol* [2].

Current OpenFlow (OF) specification does not provide direct support to Quality of Service: in fact there is no possibility to change queue rates through specific OpenFlow directives and this task is therefore demanded to an external dedicated tool [2]. The necessity to be able to modify the service rate of the queues arises in case of flows whose rate grows unpredictably. This effect may cause a severe congestion inside a SDN node, thus affecting the overall quality experienced by flows traversing the network.

It is possible to overcome this problem by re-routing the flows which are not compliant with the a-priori constraints. Since we cannot change the service rate of the queues, our solution is based on the efficient management of the ingress flows in a queue. Since we want to devise a solution which is compatible with any underlying hardware, we set ourself the goal of keeping the OpenFlow directives unchanged and therefore we design and implement the strategy inside the SDN controller. Thus we propose a specific solution implemented in Beacon [8], a widely used SDN controller. Our new updated controller, *BeaQoS*, will receive queues, flows ans ports statistics from OF switches and will compute an estimation of the rates of the flows traversing the network and the packet losses of the queues. Based on customizable policies, the controller will be able to select a subset of flows experiencing congestion and re-route them on another and less congested queue, improving the QoS.

The remainder of this paper is structured as follow: Section II describes related works on this field. Then, concerning the main contributions of this paper:

- we explain the motivations that lead to the use of multiple queues to support Quality of Service and describe the underlying idea of our proposal (Section III);
- we describe the modifications of BeaQoS controller and we propose specific strategies in order to avoid quality degradation (Section V);

Section V shows a performance analysis of our proposed solution. Finally a discussion about the obtained results and the conclusions are drawn in Section VI.

II. Related works

Despite traffic engineering (TE) approaches are often ruled by MPLS-TE [4], [5], 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 [11], which addresses load-balancing as an integral component of large cloud services. Authors explore ways to make load-balancing scalable, dynamic, and flexible. Moreover they state that load-balancing should be a network primitive, not an add-on. Therefore, they present a prototype distributed load-balancer they built based on this principle.

In [17], authors show 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. They also present algorithms that compute concise wildcard rules that achieve a target distribution of the traffic, and automatically adjust to changes in load-balancing policies without disrupting existing connections. Furthermore, authors implement these algorithms on top of the NOX OpenFlow controller, evaluate their effectiveness, and propose several avenues for further research.

The work presented in [12] shows a system that reconfigures the network's data plane to match current traffic demand by centrally controlling when and how much traffic each service sends on a backbone connecting data-centres. They develop a novel technique that 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, their system greedily selects a small set of entries that can satisfy current demand. It updates this set without disrupting traffic by leveraging a small amount of scratch capacity in forwarding tables.
Reference [3] analyses a partially deployed SDN network (a mix of SDN and no-SDN devices) and shows how to leverage the centralized controller to get significant improvements in network utilization as well as to reduce packet losses and delays. They show 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 develop fast Fully Polynomial Time Approximation Schemes (FPTAS) for solving these problems.

This last problem is also tackled in [10] that introduces a traffic management method to divide, or "slice", network resources to match user requirements. The authors present an alternative to traditional resorting to low-level mechanisms such as Virtual LANs, or interposing complicated hypervisors into the control plane: they introduce 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 for compiling slices, and illustrate their use on examples. In addition, the authors describe a prototype implementation and a tool for automatically verifying formal isolation properties.

III. Motivations

Most of the QoS approaches cited above suppose the capability of OpenFlow to set the service rate of the queues inside a network node. In some cases, instead, only one queue per interface is considered.

Unfortunately, current OpenFlow specification does not include the ability for the protocol to configure the service rate of the queues and thus an external dedicated tool must take care of this task. The impossibility to set the suitable rate for the queues can lead to severe congestion, causing performance degradation of the quality experienced by the flows. The optimal solution would certainly be the introduction of a custom OpenFlow directive, allowing the user to dynamically change the service rate of the queues. This approach would be, on the other side, totally incompatible with the current OpenFlow hardware.

In order to describe the main idea behind our proposal, suppose that there are two queues inside the system, each dedicated to a specific type of traffic. The first queue is assigned to low rate flows, while the second one is dedicated to high rate traffic. If a flow traversing q_0, previously identified as a low rate type, suddenly increases its rate, the queue will start to grow and it could end up losing packets. The performance of the other flows traversing the queue will be affected by this event, both in terms of delay and packet loss.

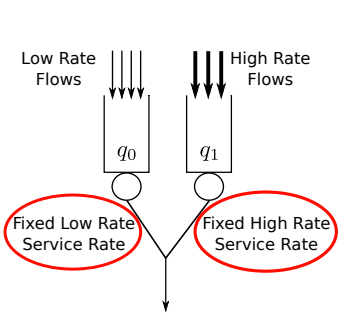

Figure 1. All traffic is conformant.

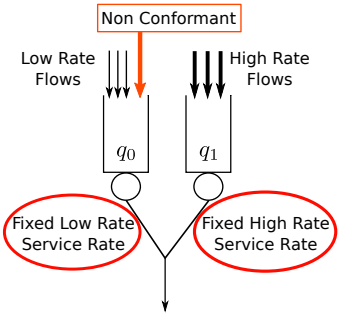

Figure 2. Some of low-rate flows become non conformant.

We therefore propose a strategy that limits this effect, ensuring that flows that prove to be compliant with the QoS constraints can exploit the needed bandwidth without suffering from any performance degradation. This solution has the task to identify the traffic which is not conformant to the rate constraints and re-route it (or drop it, if needed), in order to avoid the degradation of the quality experienced by other flows traversing the network.

Since we want to devise a solution which is compatible with any underlying hardware, we design and implement the strategy inside the SDN controller. Although the simple idea, the design of the re-routing mechanism involves several functionalities of the SDN controller. In particular, the design involves the following features:

- no primitives shall be modified with respect to the current OpenFlow standard;
- the compatibility with early versions of OpenFlow is obviously a must;
- a creation of a module able to handle statistics;
- the implementation of the proposed solution.

IV. Solutions and Beacon Modification

We chose Beacon [8] 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. In spite of a specific choice of the controller, our modifications can be implemented in any controller.

The structure of the controller consists of several bundles with dedicated functionalities. The main bundle we focused on for modifications is the *Routing* one, which takes care of finding the correct path between source and destination for packet forwarding. Moreover, we created an ad-hoc bundle, called *Statistics*, to the purpose of collecting and processing the statistics reply messages provided by the network switches. We called this new version of the Beacon software *BeaQoS*.

The scenario in which we present our solution involves a class-based system in which flows are identified by traffic descriptors. The issue we want to investigate deals with a flow characterized by a specific rate limit, which, for some reason, violates this constraint. At this point our system recognizes the problem and then it re-routes the flow in a more suitable queue, in order to avoid traffic congestion and quality degradation.
We suppose two main types of traffic:

- *Low-Rate* (LR) - characterized by a rate not exceeding 100 $kbit/s$
- *High-Rate* (HR) - displaying a rate greater than 100 $kbit/s$, most of the time

Our strategy routes the flows using their traffic descriptors. Two queues are configured inside the switch interface: one is dedicated to LR traffic, whereas the other is devoted to HR flows.
If a flow labelled as LR suddenly increases its rate so as it overcomes a specific threshold (Rate $>$ Limit 0), the queue could become more and more congested and it would start losing packets. Our system is able to recognize the constraint violation and take actions in order to avoid possible congestion.
The implemented strategy can select the flow which is not compliant to the traffic descriptor and re-route it to the suitable queue (Assign to queue Q1). Moreover, if the queue related to HR flows cannot handle the newly re-routed flow (if LR && Rate $>$ Limit 1), the controller will understand this situation through a simple computation and it will decide to drop the flow.

We introduce and implement a solution, that will be called **Conformant**, illustrated in Figure 3, to the purpose of re-establishing the correct routing of flows, based on their rates. This scheme assigns incoming flows to the queue associated to a specific traffic descriptor. q_0 is dedicated to process low-rate flows, whereas q_1 is devoted to serve high-rate traffic. The

Routing module of BeaQoS decides the correct queue based on the Type of Service field [1].

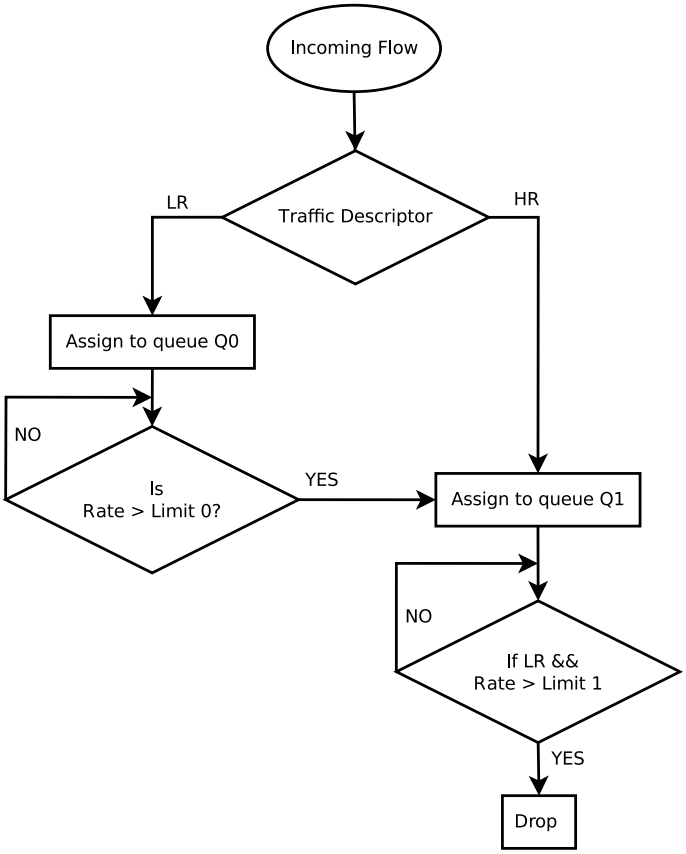

Figure 3. Strategy Flowchart

In order to implement this strategy, we modified the routing module of the Beacon controller, adding the following snippet of code:

```
if (stats.getRouting().equals("CONFORM") && (stats.
    getSwitchList().get(switchIndex).getPorts().get(
    portIndex).getQueues().size()) != 0)
{
  switch(match.getNetworkTypeOfService())
  {
    case 0x04:
      queue = 1;
      break;
    default:
      queue = 0;
      break;
  }
}
```

Furthermore, the controller periodically checks the statistics related to the flows belonging to q_0 in order to figure out if a flow is not compliant to its constraint. When BeaQoS finds a flow which is violating its traffic descriptor, it re-routes it to the HR queue in order to be able to serve the traffic without causing congestion. If the newly re-routed flow overcomes a pre-defined threshold while traversing q_1, this traffic will be dropped by the BeaQoS controller. In the present simulation we set this threshold to 700 $kbit/s$. We implemented this part of the strategy inside a specific BeaQoS module aimed at collecting statistic data. The code used to generate the desired behaviour is the following:

```
if (MODE.equals("CONFORM"))
{

    for (int g=0; g < switchList.get(switchIndex).
        getPorts().get(portIndex).getFlows().size();
         g++)
    {
        if (switchList.get(switchIndex).getPorts().
            get(portIndex).getFlows().get(g).getQID
            () == 0)
        {
            if (switchList.get(switchIndex).getPorts
                ().get(portIndex).getFlows().get(g).
                getLastRate()
                    > LIMIT0)
            {

                OFMatch match = buildMatch(
                    switchList.get(switchIndex).
                    getPorts().get(portIndex).
                    getFlows().get(g));

                switchList.get(switchIndex).getPorts
                    ().get(portIndex).getFlows().get
                    (g).setQID(1);

                sendFlowMod(sw, match,(short) (
                    switchList.get(switchIndex).
                    getPorts().get(portIndex).getPId
                    ()), 1);
            }
         }

        else if (switchList.get(switchIndex).
            getPorts().get(portIndex).getFlows().get
            (g).getQID() == 1)
        {
            if ((switchList.get(switchIndex).
                getPorts().get(portIndex).getFlows()
                .get(g).getLastRate()
                    > LIMIT1) && (switchList.get(
                        switchIndex).getPorts().get(
                        portIndex).getFlows().get(g)
                        .getTos() == (byte) 0))
            {
                OFMatch match = buildMatch(
                    switchList.get(switchIndex).
                    getPorts().get(portIndex).
                    getFlows().get(g));
                dropFlow(sw, match,(short) (
                    switchList.get(switchIndex).
                    getPorts().get(portIndex).getPId
                    ()), 1);
            }
        }
    }
}
```

V. Performance Analysis

We ran the performance analysis on a PC with Mininet (version 2.1.0) [13]. The scenario is composed of two hosts connected to a SDN switch. The chosen implementation of the switch is Open vSwitch 2.0.2 [1], managed by an instance of BeaQoS running on the same machine. Each port of the switch is configured with two queues, q_0 and q_1. We tested our strategy with two sets of simulations, one involving 50 flows and the other characterized by 100 flows inside the network. The rate assigned to each queue is shown in Table I for the 50 flows scenario (Simulation 1), and in Table II for what concerns the 100 flows simulation (Simulation 2). Queue service rates are set through the Traffic Control (*tc*) module in Linux Kernel.

The traffic used for this simulations was generated through the *iperf* tool and it consisted of 40% of LR flows and 40% of HR flows. We also introduce 20% of LR flows not respecting

[1]We choose the ToS field to differentiate LR and HR flows, but other solutions can be implemented.

Queue ID	Service Rate	Buffer Size	Traffic Descriptor
0	4 $Mbit/s$	1000 $packets$	LR
1	16 $Mbit/s$	1000 $packets$	HR

Table I
QUEUE CONFIGURATION FOR SIMULATION 1

Queue ID	Service Rate	Buffer Size	Traffic Descriptor
0	8 $Mbit/s$	1000 $packets$	LR
1	32 $Mbit/s$	1000 $packets$	HR

Table II
QUEUE CONFIGURATION FOR SIMULATION 2

their traffic descriptors, called Non-Conformant (*NC*), in order to test our strategy. Flow types and their rates are shown in Table III.

Traffic Descriptor	Rate	Percentage
Low Rate	$40 - 60$ $kbit/s$	40%
High Rate	$200 - 800$ $kbit/s$	40%
Non-Conformant	$200 - 800$ $kbit/s$	20%

Table III
FLOW DESCRIPTION FOR THE SIMULATIONS

This test aim at comparing the performances of our algorithm **Conformant** (as described in Section IV) with the **Dedicated** strategy. This last scheme consists in assigning each traffic class to the corresponding queue based on the traffic descriptor upon flow arrival and then take no further actions independently of the flow behaviour.

The results of our first set of simulations show that, while the dedicated scheme produces a 6.8% packet loss, the proposed strategy allows to completely avoid the packet loss of Low Rate flows (Figure 4). This benefit is obtained together with the fact that the quality experienced by High Rate flows is not affected. It is worth to notice that the loss of Non-Conformant flows increases, but this is acceptable since these flows are not compliant with the constraints. This effect is shown in Figure 5.

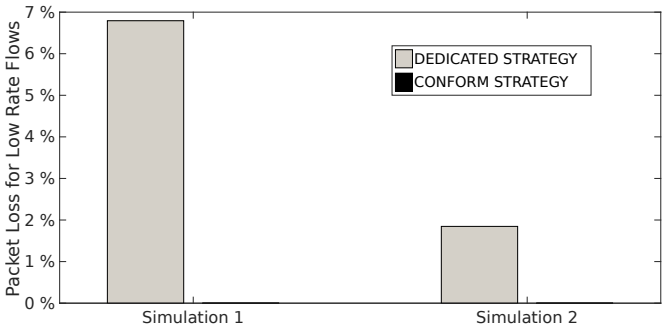

Figure 4. Packet Loss for Low Rate Flows

The second set of simulations confirm the same behaviour as the previous scenario. Packet loss for the dedicated approach shows an average value of 1.8%, whereas the Conform strategy is able to bring this value down to zero (Figure 4). Also in this case the packet loss of Non-Conformant flows increases (Figure 5).

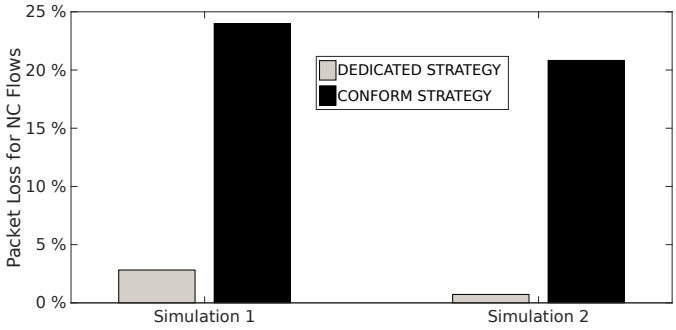

Figure 5. Packet Loss for Non Conformant Flows

Being our proposal a programmable solution, it is however possible to tune the threshold that defines the behaviour of the strategy in order to cope with different needs and situations. This parameter can be set through an external properties file, making the customization of the scheme even more flexible.

VI. CONCLUSION

In this work we analysed and explained the most important benefits and improvements that SDN architecture provided to traditional networking approaches. The main innovation brought by SDN is the decoupling of control plane and data plane.

Due to the lack of flexibility in QoS management of OpenFlow, we decided to propose new approaches to manage the non-conformant flows inside a SDN network exploiting the re-routing principle. We explained the topical modifications applied to the Beacon controller in order to implement specific strategies.

In conclusion, we showed the results obtained in performance tests in which we compared the alternative QoS approaches. Our cases of study show that the proposed QoS solutions allow to gain good results when applied to the current OpenFlow environment.

Future developments could consist in testing the network environment with a larger amount of traffic in order to test the scalability of our solutions. We also plan to devise alternative approaches such as exploiting the low rate queue in order to improve the quality perceived by high rate flows. Furthermore we plan to run our algorithms in other scenarios setted with different queue configurations. Finally we hope to be able to conduct testbed measurements with commodity hardware routers in order to avoid the problems related to the software emulation of this type of devices.

REFERENCES

[1] Open vSwitch. http://openvswitch.org/. Accessed: 2015-03-04.
[2] OpenFlow Switch Specification - version 1.5.0. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf, January 9, 2015.

[3] S. Agarwal, M. Kodialam, and T. Lakshman. Traffic engineering in software defined networks. In *INFOCOM, 2013 Proceedings IEEE*, pages 2211–2219, April 2013.

[4] D. Applegate and 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*, volume 1, pages 555–565, March 2003.

[5] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and S. G. Rsvp-te: Extensions to rsvp for lsp tunnels. RFC 3209, 2001.

[6] M. Cello, G. Gnecco, M. Marchese, and M. Sanguineti. Structural properties of optimal coordinate-convex policies for cac with nonlinearly-constrained feasibility regions. In *INFOCOM, 2011 Proceedings IEEE*, pages 466–470, April 2011.

[7] M. Cello, G. Gnecco, M. Marchese, and M. Sanguineti. Optimality conditions for coordinate-convex policies in cac with nonlinear feasibility boundaries. *IEEE/ACM Trans. Netw.*, 21(5):1363–1377, Oct. 2013.

[8] D. Erickson. The Beacon Openflow Controller. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 13–18, 2013.

[9] R. Gellens. The SYS and AUTH POP Response Codes. RFC 3206 (Proposed Standard), 2002.

[10] S. Gutz, A. Story, C. Schlesinger, and N. Foster. Splendid isolation: A slice abstraction for software-defined networks. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 79–84, 2012.

[11] N. Handigol, S. Seetharaman, M. Flajslik, R. Johari, and N. McKeown. Aster*x: Load-balancing as a network primitive. In *Plenary Demo, 9th GENI Engineering Conference*, 9th GENI, November 2010.

[12] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. Achieving high utilization with software-driven wan. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 15–26, 2013.

[13] B. Lantz, B. Heller, and N. McKeown. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. In *9th ACM Workshop on Hot Topics in Networks*.

[14] M. Marchese. *QoS Over Heterogeneous Networks*. Wiley Publishing, 2007.

[15] T. Nadeau and K. Gray. *SDN - Software Defined Networks*. O'Reilly Media Inc., 2013.

[16] A. Tanenbaum. *Computer Networks*. Prentice Hall Professional Technical Reference, 4th edition, 2002.

[17] R. Wang, D. Butnariu, and 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*, Hot-ICE'11, pages 12–12, 2011.