Performability Evaluation of Software Defined Networking Infrastructures

Mario Di Mauro
University of Salerno
Via Giovanni Paolo II, 132,
I-84084, Fisciano (SA), Italy
mdimauro@unisa.it

Maurizio Longo
University of Salerno
Via Giovanni Paolo II, 132,
I-84084, Fisciano (SA), Italy
longo@unisa.it

Fabio Postiglione
University of Salerno
Via Giovanni Paolo II, 132,
I-84084, Fisciano (SA), Italy
fpostiglione@unisa.it

ABSTRACT

An innovative model in traffic engineering, named Software Defined Networking (SDN), has been recently proposed to simplify network management and control by means of programmability concepts. This emerging strategy addresses the recent network challenges by decoupling the packet forwarding features, namely the data plane, from the decision system, namely the control plane, via OpenFlow, a specific standardized protocol. The controller element of an SDN infrastructure represents the core entity in charge of managing the whole service logic and, being this module failure-prone, its performance and its availability are crucial issues for an accurate plan of an SDN-based network. The approaches considering both performance and availability assessment in data and telecommunication networks are frequently referred to as the performability evaluations. Thus, a performability evaluation is presented in this work to the aim of selecting the most convenient redundancy scheme of the SDN controller, where the controller has been modeled by a finite number of virtual operator instances serving different network zones. By assuming that the SDN controlling unit is described by a Continuous-Time Markov Chain with a vector state, the availability in long runs of the SDN controller in parallel redundancy configuration is computed by an approach based on the Universal Generating Function tailored for the vector case, and the minimal cost redundant configuration for the SDN controller is found out.

Keywords

Software Defined Networking, Performability Evaluation, Multivariate Universal Generating Function.

1. INTRODUCTION

Recently, a ground-breaking networking model named Software Defined Networking (SDN) has been proposed in order to decouple control protocols from network forwarding mechanisms and thus to simplify network management and to reinforce the provisioning and configuration of telecommunication services. This separation addresses a new vision of the network concepts where the switches are now assuming the basic role of packet forwarding devices containing the flow tables, a set of rules imposed by a central element called controller acting as depository of the network intelligence. In the SDN environment, a crucial role is played by OpenFlow [14], a novel protocol aiming to enable the communication between the control entity, named the *SDN controller*, and the devices (*switches* in the SDN jargon) at the data level.

Such a new vision, where control and forwarding entities are strongly decoupled, allows for an extensive set of flexible network solutions. A logically centralized controller in fact provides a unified programmable interface for software and higher level applications deployment, by offering an abstraction level similar to the Operating System (OS) where, *mutatis mutandis*, the controller acts as the OS kernel [1].

In this paper, the authors address a performance evaluation model in an SDN environment, with focus on the Controller designed as an appliance hosting some (virtual) operator instances that manage a set of network devices through the OpenFlow protocol when random failures occur.

An overview about the performance evaluation methods for multi-state systems and the availability issues and models is offered in [12, 13], whereas an application in the context of innovative network and telecommunication scenarios has been presented in [8, 9].

This paper is organized as follows: in Section 2, an overview of the SDN paradigm by describing the main features of the proposed infrastructure is offered. Section 3 provides a vector performance model of the controller in the presence of random failures. Section 4 introduces the Multivariate Universal Generating Function (MUGF) concept, expressly designed for a multivariate environment, aiming to finding out the minimal cost SDN configuration in terms of redundant elements. Section 5 provides an evaluation of an exemplary SDN scenario by applying the proposed MUGF approach and by using realistic telecommunication data. Finally, Section 6 concludes the paper.

2. AN OVERVIEW OF THE SOFTWARE DEFINED NETWORKING APPROACH

The SDN paradigm has been systematized starting from two seminal works: the SANE Ethene project [4], and the Routing Control Platform [3]. Basically, the SDN architecture includes a set of network entities with switching functionalities, managed and supervised by a critical entity

VALUETOOLS 2016, October 25-28, Taormina, Italy

DOI 10.4108/eai.25-10-2016.2266605

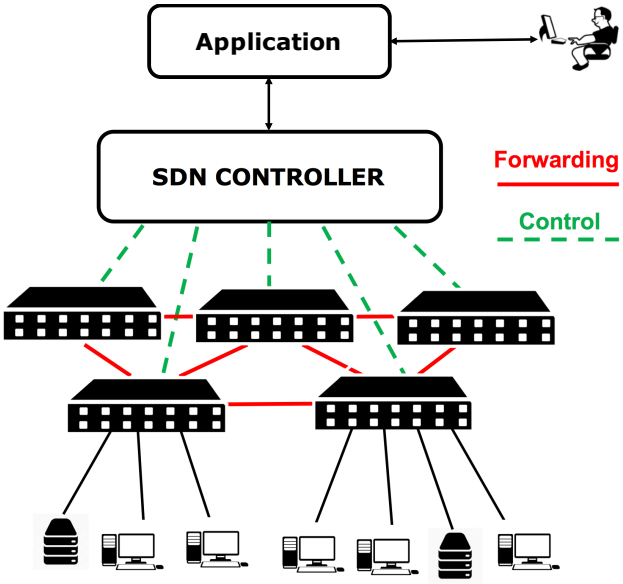

Figure 1: An SDN architecture, Forwarding (red solid lines) and Control (green dashed lines) planes.

named *Controller* through the OpenFlow protocol. The key idea of the SDN approach relies on a network view based on some centralized control agents devoted to elaborate routing service logic, such as access control agent, routing agent, and traffic management agent, and light-weighted network appliances (i.e. switches, firewalls, routers) designated to communicate with the Controller and execute commands.

2.1 OpenFlow

The OpenFlow protocol, proposed and maintained by Open Networking Forum (ONF), describes a set of specifications that represent a standard communication interface between data and control layers on a OpenFlow capable device.

Such a protocol allows the communication between controller and SDN devices (usually named switches) by implementing some messages and interoperability formats. In particular, the standard proposes three types of messages: *Asynchronous*, *Controller-to-Switch* and *Symmetric*, with various sub-types. *Asynchronous* messages are sent by switches to warn the SDN controller about critical events (node failures, network issues etc.). *Controller-to-Switch* messages come from SDN controller and are exploited to govern or audit the state of a single switch. *Symmetric* messages are sent by switches or controller and do not need to be triggered by specific events. Ultimately, OpenFlow defines the behaviour that SDN switches should have when solicited by the controller element; it is based on TCP and, if required, it supports Transport Layer Security (TLS) as an asymmetrical encryption standard.

A sketch of an SDN architecture is depicted in Figure 1, where Forwarding (or Data) and Control Planes are marked by continuous red lines and dashed green lines, respectively. The OpenFlow protocol acts through che control messages exchanged between the SDN controller and the SDN devices. On top of the controller lies an application layer offering the possibility to extremely customize the control logic on behalf of dedicated dashboards and command line interfaces.

Specific Application Program Interfaces (APIs) allow to

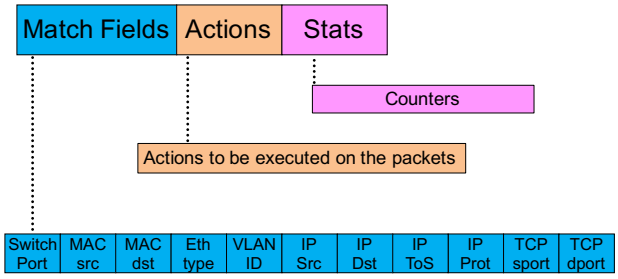

Figure 2: Example of OpenFlow table with a specific flow entry.

deploy some functionalities on board the SDN devices such as: abstraction layers, communication interfaces to guarantee an interaction with the controller, and packet-processing engines [6]. When dealing with physical elements, the latter feature is embodied in the hardware design logic while in case of virtual appliances it is deployed as a software-based agent. The abstraction layer is in charge of managing the flow tables that, after a packet inspection operation, instruct the SDN device to perform specific actions including packet forwarding, packet dropping and so forth. A representation of a flow table is offered in Figure 2, where two elements emerge: *match fields* devoted to packet comparison and *actions* denoting the operations to perform on packets.

2.2 The core element of the architecture: SDN Controller

As before said, the controller is responsible for remotely managing the switch rules playing the same role of a router that, on behalf of specific routing algorithms, is in charge of programming and filling (or deleting) the forwarding tables. More in details, once the controller loads a specific flow table in an SDN switch, the latter is able to fastly manage every packet flow that results in an exact match. On the contrary, when a table entry is missed, the following sequence is activated:

1. The first un-matched packet of the flow is sent from the switch to the controller;
2. The forwarding path for the flow is computed by the controller;
3. The controller sends the appropriate forwarding entries to the device by filling the corresponding flow table;
4. All ensuing packets that belong to the pertaining flow are forwarded with no further intervention of the control plane.

OpenFlow is able to support three different configuration schemes of connection between controller and switches: *Master*, *Slave*, and *Equal*. These operating modes can be appropriately combined in order to build certain redundant configurations to satisfy high availability constraints and/or implement load balancing schemes. Master and Equal schemes allow the SDN controller to actively instruct the switch. The main differences is that only one controller can play the Master role, while multiple (eventually synchronized) SDN controllers can be configured in Equal mode. In Slave mode, SDN controller can just collect data for statistics but no changes in switches configurations are permitted.

2.3 Related Works

An interesting work focused on a complete refactoring of network management functionalities in a distributed environment appears in [7]. More in details, a novel architecture is proposed in which the decision logic and the protocols governing the interactions among network elements are completely separated. Moreover, main challenges of the SDN paradigm are identified:

- *Security.* The centralized SDN controller should be adequately protected being the most sensitive element of the entire architecture.
- *Scale.* SDN controller is in charge of managing the whole topological infrastructure of the network and the computation of best routes, resulting in a scalability problem that has to be necessarily considered.
- *Latency.* The presence of SDN controller entails that some (eventually critical) decisions will be affected by non negligible round-trip delay.
- *High availability.* Some redundancy policies have to be taken into account with regard to the SDN controller, so to avoid a single-point-of-failure issue.

A wireless-based SDN infrastructure has been pointed in [2], where the authors discuss about the Control and Provisioning of Wireless Access Points (CAPWAP), a protocol that exploits a generic encapsulation method, making it independent of a specific radio technology. On behalf of such a protocol, control frames are delivered to a central network element responsible for MAC layer control in a way similar to the mechanisms operated by the OpenFlow protocol when it delivers to the controller messages about new incoming flows.

In [16], the authors propose *HyperFlow*, a logically centralized and physically distributed SDN-based control plane. The decision making logic is transferred via *HyperFlow*, to individual controllers thus minimizing the response time from the control plane.

Still based on a distributed architecture, the framework *DIFANE*, presented in [20], introduces a dedicated routing protocol aimed at fostering the switch interactions with no need of controller supervision.

Another branch of works, concerns the reliability evaluation of SDN infrastructures by taking into account the controller placement issue. Accordingly, in [10], the authors introduce some placement algorithms based on a novel reliability metric (expected percentage of CPL - control path loss) and find out that the simulated annealing algorithm provides an almost optimal solution. More specifically, the problem of placing k controllers among $|V|$ locations is faced; the proposed algorithm first produces a list of the possible locations, say L, increasingly ranked according to switches fault probabilities, and then picks one location at time from the first $w|V|$ $(0 < w \leq 1)$ in L, indicated as entrant locations for hosting controllers.

A further track, instead, concerns the interaction between SDN infrastructures and the virtualization concepts introduced by recent NFV paradigm. In line with this latter perspective is the Google SDN Wireless Area Network project, described in [11]. The key idea behind such project relates with setting up a WAN network connecting multiple data centers with significant bandwidth requirements governed by an SDN-based infrastructure.

Again, a module named *FlowVisor* that acts as an hypervisor in a virtualized environment, is introduced in [15]. Basically, such a module plays the role of a transparent proxy between OpenFlow switches and multiple OpenFlow controllers.

Furthermore, the authors in [19] propose an analytical performance model of OpenFlow networks based on queueing theory. In particular, they model the packet forwarding mechanism of SDN switches and the packet-in message processing of the SDN controller as the queueing systems MX/M/1 and M/G/1, respectively. A queueing model of the whole SDN networks in terms of packet forwarding performance is then presented by solving its closed-form expression.

3. SDN CONTROLLER PERFORMABILITY EVALUATION

By considering benefits and advantages a virtualized SDN-based solution can offer (as discussed in a recent work of the same authors [5]), we consider an architecture where a single SDN controller hosts and manages a number of virtual software instances as depicted in Figure 3. Each software instance S acts for the Master controller related to a specific provider; it is named Virtual Provider Instance (VPI) in the SDN jargon and is in charge of managing a bunch of OpenFlow-enabled devices. In the following, we use S_i *in lieu* of VPI i. Such an approach provides a lot of benefits in terms of managing in a ductile way the whole network infrastructure that can be effectively rearranged according to the quality of service requirements of the providers, after negotiating opportune Service Level Agreements (SLAs). On the contrary, the principal flaw concerns the possibility that the SDN controller might become a unique point-of-failure so that some redundancy procedures have to be established.

We model an SDN controller as formed by:

- a *core part*, comprising every kind of hardware equipment (e.g. power supply, blades, processors, memories etc.) and generic software (e.g. hypervisor, operating system etc.);
- a *software part*, corresponding to the VPIs, able to handle a given number of sessions providing instructions to the switches.

It is worth noting that the proposed SDN controller model is generic, but it can be adapted to describe peculiar implementations. Starting from it, we build a performance model for the controller based on the number of coexisting OpenFlow sessions that each (virtual) operator is able to control. Firstly, we suppose that a single SDN controller is able to govern k VPIs and every VPI is in charge of managing n OpenFlow concurrent sessions. We assume also that: *i)* the main elements (VPIs and core) are modelled as a two-state system (up/down), *ii)* VPI i and core failures are statistically independent Homogeneous Poisson processes (HPPs), characterized by independent and exponentially-distributed inter failure arrivals and constant hazard rates λ_i and λ_c, respectively, *iii)* repair times are independent and exponentially-distributed with rates μ_s and μ_c, respectively.

By conveying in each state the information on the VPIs working conditions (up/down), the model of the considered

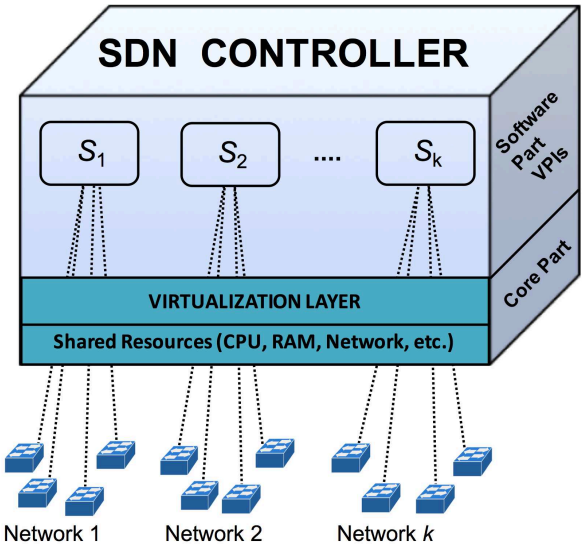

Figure 3: A set of k VPIs supervised by an SDN Controller.

Table 1: Correspondence map between states, VPIs condition and performance triples.

State number	VPIs condition	Performance
7	(S_1, S_2, S_3)	(n, n, n)
6	$(S_1, S_2, \overline{S_3})$	$(n, n, 0)$
5	$(S_1, \overline{S_2}, S_3)$	$(n, 0, n)$
4	$(\overline{S_1}, S_2, S_3)$	$(0, n, n)$
3	$(S_1, \overline{S_2}, \overline{S_3})$	$(n, 0, 0)$
2	$(\overline{S_1}, S_2, \overline{S_3})$	$(0, n, 0)$
1	$(\overline{S_1}, \overline{S_2}, S_3)$	$(0, 0, n)$
0	$(\overline{S_1}, \overline{S_2}, \overline{S_3})$	$(0, 0, 0)$
-1 (core fault)	$(\overline{S_1}, \overline{S_2}, \overline{S_3})$	$(0, 0, 0)$

controller results in a multi-state Continuous-Time Markov Chain (CTMC) where:

- $2^k + 1$ represents the total number of states. Table 1 contains information about the mapping among states and VPIs condition/performance for the case $k = 3$, namely from the state -1 to the state $2^k - 1 = 7$. S_i and $\overline{S_i}$ indicate up and down conditions (see second column), whereas the corresponding serving capacity is indicated by n or 0 (see third column), respectively.

- state -1 takes into account the not-working condition of the core component (*core fault*) implying that no VPI can be up, and corresponding to the k-tuple $(\overline{S_1}, \ldots, \overline{S_k})$. From this state, only one transition towards a completely repaired controller is presumed.

A exemplary CTMC model of an SDN controller with $k = 3$ VPIs is shown in Figure 4, where the state probabilities $p_j(t)$, $j = -1, 0, 1, .., 7$ are derived by solving the system (1), with the initial conditions $p_7(0) = 1$ and $p_i(0) = 0$, $\forall i = -1, 0, ..., 6$, representing a fully working system at the starting time $t = 0$.

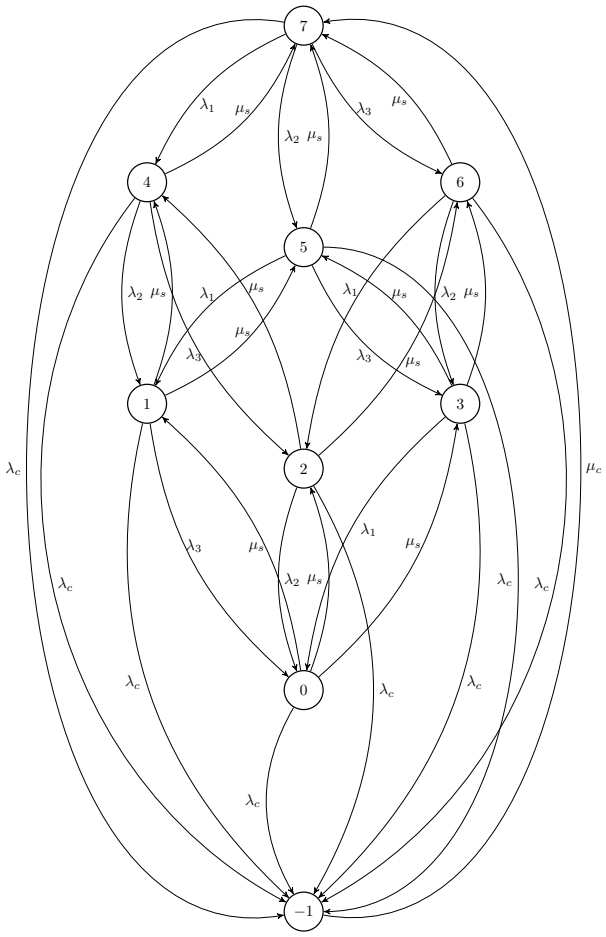

Figure 4: CTMC representing an SDN controller supervising 3 VPIs.

$$
\begin{cases}
\frac{dp_7(t)}{dt} = \mu_s[p_4(t) + p_5(t) + p_6(t)] + \mu_c p_{-1}(t) + \\
\qquad -(\lambda_1 + \lambda_2 + \lambda_3 + \lambda_c)p_7(t) \\
\frac{dp_6(t)}{dt} = \mu_s[p_2(t) + p_3(t)] + \lambda_3 p_7(t) + \\
\qquad -(\lambda_1 + \lambda_2 + \lambda_c + \mu_s)p_6(t) \\
\frac{dp_5(t)}{dt} = \mu_s[p_1(t) + p_3(t)] + \lambda_2 p_7(t) + \\
\qquad -(\lambda_1 + \lambda_3 + \lambda_c + \mu_s)p_5(t) \\
\frac{dp_4(t)}{dt} = \mu_s[p_1(t) + p_2(t)] + \lambda_1 p_7(t) + \\
\qquad -(\lambda_2 + \lambda_3 + \lambda_c + \mu_s)p_4(t) \\
\frac{dp_3(t)}{dt} = \mu_s p_0(t) + \lambda_3 p_5(t) + \lambda_2 p_6(t) + \\
\qquad -(\lambda_1 + \lambda_c + 2\mu_s)p_3(t) \\
\frac{dp_2(t)}{dt} = \mu_s p_0(t) + \lambda_3 p_4(t) + \lambda_1 p_6(t) + \\
\qquad -(\lambda_2 + \lambda_c + 2\mu_s)p_2(t) \\
\frac{dp_1(t)}{dt} = \mu_s p_0(t) + \lambda_2 p_4(t) + \lambda_1 p_5(t) + \\
\qquad -(\lambda_3 + \lambda_c + 2\mu_s)p_3(t) \\
\frac{dp_0(t)}{dt} = -(\lambda_c + 3\mu_s)p_0(t) + \\
\qquad +\lambda_3 p_1(t) + \lambda_2 p_2(t) + \lambda_1 p_3(t) \\
\frac{dp_{-1}(t)}{dt} = -\mu_c p_{-1}(t) + \lambda_c \sum_{i=0}^{7} p_{(i)}(t)
\end{cases}
\tag{1}
$$

It is worth noting that the performance levels pertaining to states of the proposed model of the SDN controller are described by a vector enclosing the number of coexisting sessions that each VPI is able to handle (see Table 1).

Since some redundancy is needed for the SDN controller in real-case applications, the vectors of node r are in the set

$$\mathbf{g}^{(r)} = \{\mathbf{g}_{-1}^{(r)}, \mathbf{g}_{0}^{(r)}, ..., \mathbf{g}_{2^k-1}^{(r)}\}, \quad (2)$$

where the k-dimensional vector $\mathbf{g}_j^{(r)} = \left(g_{1,j}^{(r)}, ..., g_{k,j}^{(r)}\right)$ contains the serving capacities $g_{i,j}^{(r)}$ offered by VPI i, $\forall i = 1, \ldots, k$, when SDN controller node r is in the state $j = -1, 0, ..., 2^k - 1$. Accordingly, the vector stochastic process $\mathbf{G}^{(r)}(t) \in \mathbf{g}^{(r)}$ describes the performance level of node r, $\forall t \geq 0$, whose probability $p_j^{(r)}(t) = \Pr\{\mathbf{G}^{(r)}(t) = \mathbf{g}_j^{(r)}\}$ is derived by solving the system (1).

Therefore, the steady-state probabilities of the CTMC describing controller node r are:

$$p_j^{(r)} = \lim_{t \longrightarrow \infty} \Pr\{\mathbf{G}^{(r)}(t) = \mathbf{g}_j^{(r)}\}, \quad (3)$$

that can be computed by (1), with all the derivatives equal to 0 and $p_j^{(r)}(t)$ replaced by $p_j^{(r)}$, along with the condition:

$$\sum_{j=-1}^{2^k-1} p_j^{(r)} = 1. \quad (4)$$

4. MULTIVARIATE UNIVERSAL GENERATING FUNCTION FOR AVAILABILITY EVALUATION

We start by considering the SDN controller as fully working when a needed *demand* (a minimal performance level for a correct service delivery) is met, thus a demand vector $\mathbf{W}(t) = (W_1(t), ..., W_k(t))$ is advanced.

In many realistic deployments, for instance in order to satisfy SLAs, a certain redundancy level for the SDN controller has to be planned. In our work, we consider a Master-Slave scheme, where replicas of a single controller (composed by core and VPIs) are allowed: in such scenario, all VPIs associated to each domain are aligned and have all the information about the flows traversing the switches they control. Thus, it becomes irrelevant which is the specific VPI replica to have managed a certain flow entry.

Besides, the SDN controller is modeled as a network node having h parallel units without *flow dispersion* [12], and the stochastic process accounting for the performance level (coexisting OpenFlow sessions) provided to network i is the maximum performance level offered by all the VPIs replicas responsible of managing network domain i, viz.

$$G_i(t) = \max_{r=1,\ldots,h} G_i^{(r)}(t), \quad (5)$$

where $G_i^{(r)}(t)$ represents the element i of the vector random process $\mathbf{G}^{(r)}(t)$.

For long runs ($t \to \infty$), the values of random processes $G_i(t)$, for $i = 1, ..., k$, can be expressed by a random vector $\mathbf{G} = (G_1, ..., G_k)$ characterized by a multivariate probability function $p_{\mathbf{G}}(\cdot)$, represented by the steady-state distribution pertaining to the CTMC describing the dynamic behavior of the parallel redundancy configuration of the SDN controller.

Conforming to [12], the controller *instantaneous availability* $A_{SDN}(t)$ represents the probability that, at $t > 0$, the controller is in one of the possible states characterized by a performance level not less than demand $W_i(t)$ for each network domain $i = 1, ..., k$, namely,

$$A_{SDN}(t) = \Pr\{G_i(t) - W_i(t) \geq 0, \forall i = 1, ..., k\}. \quad (6)$$

As $t \to \infty$, the initial state of the SDN controller has no sensible effect on its availability. If we consider one and the same constant demand level $W_i(t) = w, i = 1, ..., k$, the *steady-state availability* $A_{SDN}(w)$ of the controller is expressed by:

$$A_{SDN}(w) = \sum_{j=1}^{m} p_{\mathbf{G}}\left(\mathbf{g}_j^{SDN}\right) \cdot \mathbf{1}\left(g_{i,j}^{SDN} \geq w, \forall i = 1, ..., k\right), \quad (7)$$

where $\mathbf{g}_j^{SDN}$ identifies the state j of the controller with parallel units, whose overall model is a CTMC composed by m states, and where $\mathbf{1}(\mathcal{A}) = 1$ when the condition $\mathcal{A}$ is true, 0 otherwise.

In order to perform the availability evaluation of the SDN controller in a parallel redundancy configuration, we exploit the Universal Generating Function (UGF) methodology that appeared for the first time in [17].

The UGF can be considered a kind of hierarchical approach that avoids to solve the overall CTMC model describing the performance of the whole system under analysis, whose solution is typically unfeasible due to the high dimension of the state space. The UGF allows to combine the performance distribution of the subsystems (much simpler to be solved) composing the complex series-parallel system, by means of some suitable operators for both parallel and series configurations of subsystems. Further details are available in [12].

A (discrete) random variable Y admits the following UGF representation:

$$u(z) = \sum_{i=1}^{A} \alpha_i z^{y_i}, \quad (8)$$

where $\alpha_i = \Pr\{Y = y_i\}$ and Y has A values y_i. The UGF of a random variable represents the performance levels of a system with multiple states thus allowing availability evaluation. In addition, the UGF of a complex system can be proficiently calculated by applying series and parallel operators to the UGF functions of the subsystems, according to their logical connections.

In this work we propose an extension of classical UGF to a multivariate case to manage performance vectors $\mathbf{G}$ and $\mathbf{G}^{(r)}$. Such an extension is called *Multivariate* UGF (MUGF) and is defined for a vector $\mathbf{G}$, having k dimensions and values in $\{\mathbf{g}_1, ..., \mathbf{g}_m\}$, by the following expression:

$$u(\mathbf{z}) = \sum_{j=1}^{m} p_{\mathbf{G}}(\mathbf{g}_j) \prod_{i=1}^{k} z_i^{g_{i,j}}, \quad (9)$$

where $p_{\mathbf{G}}(\cdot)$ is the multivariate probability function of $\mathbf{G}$, and $\mathbf{z} = (z_1, ..., z_k)$.

Denoting by h the number of parallel units (without flow dispersion) that constitute the controller with performance levels (in terms of concurrent OpenFlow sessions) governed by (5), it is possible to express the MUGF of the SDN con-

troller via the π operator defined as follows:

$$u_{SDN}(\mathbf{z}) = \sum_{r} p_r \prod_{i=1}^{k} z_i^{g_{i,r}^{SDN}} \tag{10}$$
$$= \pi\left(u_1(\mathbf{z}), \ldots, u_h(\mathbf{z})\right)$$
$$= \sum_{j_1=-1}^{2^k-1} \cdots \sum_{j_h=-1}^{2^k-1} \prod_{r=1}^{h} p_{j_r}^{(r)} \prod_{i=1}^{k} z_i^{\max_{r=1,\ldots,h} g_{i,j_r}^{(r)}},$$

where the steady-state probabilities $p_{j_r}^{(r)}$ are derived by (3), being related to the performance levels vector $\mathbf{g}_{j_r}^{(r)}$ in (2). The SDN controller steady-state availability is obtained by (7), by using p_r and $g_{i,r}^{SDN}$ derived from the MUGF (10).

The final objective is to compute the minimal number of SDN controller units h^* in parallel redundancy configuration so that a given steady-state availability level A_0 is reached, that is provided by:

$$h^* = \arg\min_{h\in\mathbb{N}} \left(A_{SDN}(w,h) \geq A_0\right). \tag{11}$$

The problem in (11) is a simple version of "redundancy optimization problem" [18].

5. A NUMERICAL EXPERIMENT

In the present section, we provide a numerical example of the proposed methodological approach. We assume one and the same serving capacity of $n = 5000$ sessions per time unit (stu) for all the three VPIs ($k = 3$). We assume also the same SLAs for every Service Provider, so the same number of coexisting sessions are to be handled by the SDN controller for each VPI, and we require a pretty high service level to consider available a VPI: we choose $w = 4800$ stu.

To the aim of accounting for a plausible differentiation in terms of capabilities allocated for every single service operator, we consider diverse failure rates for every VPI in our model. On the other hand, we consider the same value μ_s by guessing common repair actions for all software instances and then for all VPI. Thus, the following failure and repair rates are adopted: $\lambda_1 = 3.858 \times 10^{-7}$ s^{-1} (equivalent to 1 fault per month for S_1), $\lambda_2 = 7.716 \times 10^{-7}$ s^{-1} (equivalent to 2 fault per month for S_2), $\lambda_3 = 1.157 \times 10^{-6}$ s^{-1} (equivalent to 3 fault per month for S_3), and $\mu_s = 1.388 \times 10^{-4}$ s^{-1} (equivalent to a mean repair time of 2 hours for every VPI).

According to the SDN controller model presented in Figure 3, we recall that all the VPI software instances run on the top of a core part (virtualization layer, operating system and shared hardware resources). The core part failure rate is assumed to be $\lambda_c = 1.268 \times 10^{-7}$ s^{-1} (equivalent to 4 core faults per year), while the rate of the repair activity on a failed core part is $\mu_c = 3.472 \times 10^{-5}$ s^{-1} (equivalent to a mean repair time of 8 hours). This activity is supposed to completely restore the node functionalities by eventually reactivating also VPI instances in down conditions, as modeled in Figure 4 and remarked in Section 3.

In Table 2, all the parameters values adopted in the numerical experiment are listed. It is worth noting that all the selected values, although arbitrarily chosen, are in keeping with the experience of system engineers.

By solving (1) with all the derivatives equal to 0 and by using the condition (4), the steady-state probabilities $p_j^{(r)}$ in (3) are computed for a single node r, whose 3-dimensional performance vectors $\mathbf{g}_j^{(r)}$ are reported in Table 1, for each state $j = -1, 0, \ldots, 7$. Subsequently, the MUGF of the vector performance distribution of node r is derived after (9).

Table 2: Parameters values in the numerical experiment

Parameter	Value
k	3
n	5000 stu
λ_1	3.858×10^{-7} s^{-1}
λ_2	7.716×10^{-7} s^{-1}
λ_3	1.157×10^{-6} s^{-1}
μ_s	1.388×10^{-4} s^{-1}
λ_c	1.268×10^{-7} s^{-1}
μ_c	3.472×10^{-5} s^{-1}
w	4800 stu
A_0	0.999999

The redundancy optimization problem (11) is solved by implementing the MUGF π operator in (10) and by computing the steady-state availability (7) directly from the MUGF of the system composed by the nodes connected in parallel. Given $A_0 = 0.999999$, the "six 9s" availability condition (increasingly desirable in telecommunication systems) of the SDN controller is reached (and even exceeded) with at least $h^* = 4$ parallel elements; indeed, the steady-state availability of this redundant configuration of the SDN controller is:

$$A_{SDN}(w,h^*)\big|_{\substack{w=4800\\h^*=4}} = p_7 = 0.999999971.$$

Table 3 reports the complete list of the performance vectors and the corresponding steady-state probabilities for an SDN controller in the same redundant configuration.

The numerical experiment has been performed by a Mathematica routine implementing the MUGF approach. The execution time of the said routine, running on a notebook based on an Intel Core i7-4960 HQ CPU@2.6GHz, is about 0.0327 s, which shows that the proposed MUGF approach is very fast to apply.

In order to evaluate and appreciate the differences in terms of availability by varying the number h of redundant elements, we refer to Figure 5 where, for sake of simplicity, we consider the steady-state unavailability of the system $1-A_{SDN}(w,h)$. The horizontal dashed line in the aforementioned figure represents the required steady-state unavailability $1-A_0 = 10^{-6}$. It is worth noting that in case of $h = 3$ redundant elements, the resulting A_{SDN} value amounts to 0.999997403 that is considered by now not fully compliant

Table 3: Performance vectors and steady-state probabilities of an SDN controller composed by 4 parallel units

Probability	Performance vectors
1.755×10^{-10}	$(0,0,0)$
8.903×10^{-12}	$(5000,0,0)$
4.410×10^{-12}	$(0,5000,0)$
1.964×10^{-8}	$(5000,5000,0)$
2.931×10^{-12}	$(0,0,5000)$
6.789×10^{-9}	$(5000,0,5000)$
1.491×10^{-9}	$(0,5000,5000)$
0.999999971	$(5000,5000,5000)$

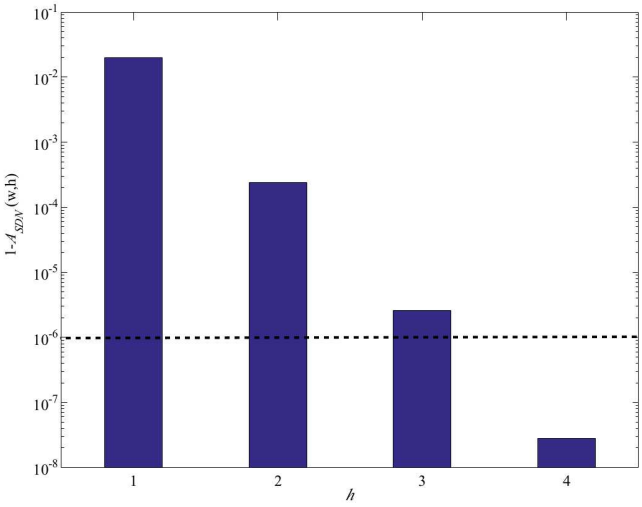

Figure 5: Unavailability $1 - A_{SDN}(w, h)$ of the SDN controller architecture for $h = 1, 2, 3, 4$ parallel elements. The horizontal dashed line represents the required steady-state unavailability $1 - A_0 = 10^{-6}$ of the SDN controller architecture.

with the modern standard requirements of very high availability. Besides, some changes in the values of the repair rates μ_c and μ_s does not weaken the "six 9s" availability condition, as shown in Figures 6 and 7, respectively.

6. CONCLUSIONS

In an SDN environment, the network control and forwarding functions are decoupled, and the network intelligence is centralized and managed by the controller, the most critical element on the entire SDN infrastructure. Being the controller a failure-prone network element, we propose a performability analysis aiming to obtain the so called "six 9s" availability condition, increasingly required by the telecommunication world. In our modeling, some virtualized software instances (representing different virtual telecom operators) are managed by the SDN controller. Such instances, referred to as VPIs, are supposed to manage a set of SDN switches via OpenFlow protocol. Therefore, the number of coexistent OpenFlow sessions has been selected as performance metric and the minimal cost redundant configuration of the SDN controller was found. The performability analysis of the controller has been faced by the Multivariate UGF, a novel extension of UGF introduced to deal with performance vectors. Such an approach results advantageous in a multi-operator environment where different VPIs can share the same information. In a future work, the authors will try to consider a more challenging environment that accounts for Network Function Virtualization (NFV) paradigm and its interaction with the considered SDN infrastructure.

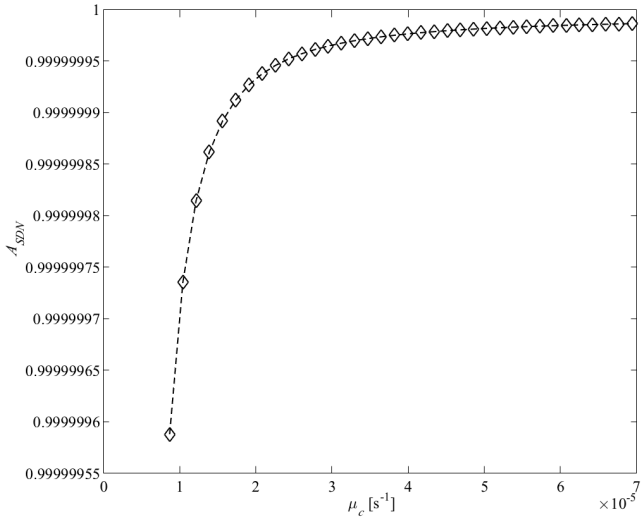

Figure 6: Influence of core repair rate on an SDN controller composed by 4 parallel units.

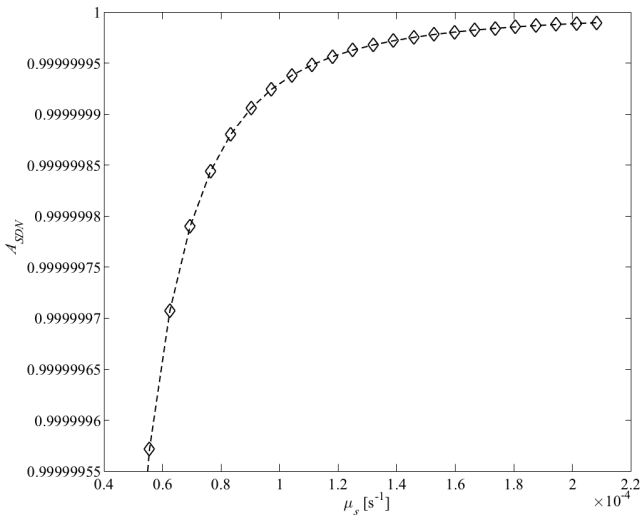

Figure 7: Influence of software instances (VPIs) repair rate on an SDN controller composed by 4 parallel units.

7. REFERENCES

[1] S. Ali, V. Sivaraman, A. Radford, and S. Jha. A survey of securing networks using software defined networking. *IEEE Transactions on Reliability*, 64(3):1086–1097, 2015.

[2] C. Bernardos, A. De La Oliva, P. Serrano, A. Banchs, L. Contreras, H. Jin, and J. Ziga. An architecture for software defined wireless networking. *IEEE Wireless Communications*, 21(3):52–61, 2014.

[3] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe. Design and implementation of a routing control platform. In *Proceedings of 2nd Symposium on Networked Systems Design & Implementation - Volume 2*, pages 15–28, 2005.

[4] M. Casado, T. Garfinkel, A. Akella, M. J. Freedman, D. Boneh, N. McKeown, and S. Shenker. Sane: A protection architecture for enterprise networks. In *Proceedings of 15th Conference on USENIX Security Symposium - Volume 15*, 2006.

[5] M. Di Mauro, F. Postiglione, and M. Longo. Reliability analysis of the controller architecture in Software Defined Networks. *Safety and Reliability of Complex Engineered Systems: ESREL2015*, pages 1503–1510, 2015.

[6] P. Goransson and C. Black. *Software Defined Networks: A Comprehensive Approach*. Morgan Kaufmann, Burlington, 2014.

[7] A. Greenberg, G. Hjalmtysson, D. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. A clean slate 4d approach to network control and management. *Computer Communication Review*, 35(5):41–54, 2005.

[8] M. Guida, M. Longo, and F. Postiglione. Reliability analysis of next generation mobile networks. In Briš, G. Soares, and Martorell, editors, *Reliability, Risk and Safety, three volume set:Theory and Applications*, volume 3, pages 1999–2006. Taylor & Francis Group, London, 2010.

[9] M. Guida, M. Longo, F. Postiglione, K. Trivedi, and X. Yin. Semi-Markov models for performance evaluation of failure-prone IP multimedia subsystem core networks. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, 227(3):290–301, 2013.

[10] Y. Hu, W. Wendong, X. Gong, X. Que, and C. Shiduan. Reliability-aware controller placement for software-defined networks. In *Proceedings of IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, pages 672–675, 2013.

[11] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat. B4: Experience with a globally-deployed Software Defined Wan. *ACM SIGCOMM Comput. Commun. Rev.*, 43(4):3–14, 2013.

[12] G. Levitin and A. Lisnianski. *Multi-state system reliability: assessment, optimization and applications*. World Scientific, Singapore, 2003.

[13] Y. Liu and K. S. Trivedi. Survivability quantification: The analytical modeling approach. *International Journal on Performability Engineering*, 2(1):29–44, 2006.

[14] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling innovation in campus networks. *Computer Commununication Review*, 38(2):69–74, 2008.

[15] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. FlowVisor: A Network Virtualization Layer. Technical report, Deutsche Telekom Inc. R&D Lab, Stanford, Nicira Networks, 2009.

[16] A. Tootoonchian and Y. Ganjali. Hyperflow: A distributed control plane for OpenFlow. In *Proceedings of Internet Network Management Conf. on Research on Enterprise Networking*, pages 3–3, 2010.

[17] I. A. Ushakov. A universal generating function. *Soviet Journal of Computing System Science*, 24(5):37–49, 1986.

[18] I. A. Ushakov. Optimal standby problems and a universal generating function. *Soviet Journal of Computing System Science*, 25(4):79–82, 1987.

[19] B. Xiong, K. Yang, J. Zhao, W. Li, and K. Li. Performance evaluation of openflow-based software-defined networks based on queueing model. *Computer Networks*, 102:172–185, 2016.

[20] M. Yu, J. Rexford, M. J. Freedman, and J. Wang. Scalable flow-based networking with difane. *Computer Commununication Review*, 41(4):351–362, 2010.