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

Software Defined Networking (SDN) has been introduced as a novel paradigm in traffic engineering, aiming to simplify network management and control through 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, through a dedicated protocol named OpenFlow. 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. 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 steady-state availability 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 a redundancy optimization problem for the architecture of the SDN controller is solved.

Keywords

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

1. INTRODUCTION

Software Defined Networking (SDN) is a rising networking paradigm based on the idea of decoupling network control plane from data forwarding plane in order to optimize network utilization, facilitate management and enhance the provisioning and configuration of the network and 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 [15], a novel protocol aiming to enable the communication between the control plane (implemented through the SDN controller) and the data plane (represented by a set of SDN network devices called switches).

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 application developers with a unified programmable interface exploited to deploy software and higher level applications, by offering an abstraction level similar to the operating system where, *mutatis mutandis*, the controller acts as the OS kernel [1].

It's worth remembering that, in data and telecommunication networks, the ability to deal with random failures is measured in three ways [14]: *Reliability, Availability* and *Survivability. Reliability* refers to the probability that a network entity performs a designated set of functions under some conditions for a time greater than a definite operational time. *Availability* represents the probability that a network entity performs its functions at any given instant under established conditions, and *Survivability* is a measure of the ability to work in the presence of a partial service outage. For further details, we refer to [12, 13], while an application of the said concepts to novel telecommunication systems has been shown in [8, 9], where a performability analysis of an IP Multimedia Subsystem (IMS) environment has been addressed.

In the present work, the authors address a performance evaluation model for the SDN controller, equipped with a set of virtual operator instances managing a bundle of network devices through the OpenFlow protocol, in the presence of random failures.

The paper is organized as follows: Section 2 offers an overview of the SDN paradigm by describing the main features of the architecture. In Section 3 the authors define a performance metric for the SDN controller hosting a set of virtualized software instances and describe a multi-state performance model, by mapping each state with a specific performance level. In Section 4 the authors introduce the Multivariate Universal Generating Function (MUGF) concept, expressly designed for a multivariate environment, aiming to

VALUETOOLS 2016, October 25-28, Taormina, Italy

DOI 10.4108/eai.25-10-2016.2266605

face and solve the redundancy optimization problem for an SDN controller. Section 5 proposes a numerical experiment by using realistic telecommunications data and in Section 6 some concluding remarks have been drawn.

2. AN OVERVIEW OF THE SOFTWARE DEFINED NETWORKING APPROACH

The SDN paradigm has been systematized starting from two seminal works: the Routing Control Platform (RCP40) developed at Princeton and Carnegie Mellon Universities [3] and the SANE Ethene project proposed at Stanford and Berkeley Universities [4]. Basically, the SDN architecture includes a set of network entities with switching functionalities, managed and supervised by a central element named Controller through the OpenFlow protocol. The key idea of the SDN approach is based on having an exhaustive network view on behalf of centralized control agents: access control agent, routing agent, traffic engineering agent and so forth. This control logic lies on top of the network infrastructure with no impact on network appliances (e.g. switches, routers) resulting discharged from routing algorithm computation.

2.1 The OpenFlow Protocol

The OpenFlow protocol, proposed and maintained by Open Networking Forum (ONF), describes a set of specifications representing 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: *Controller-to-Switch*, *Asynchronous* and *Symmetric*, each with multiple sub-types. *Controller-to-Switch* messages are initiated by the controller and used to directly govern or audit the state of a switch. *Asynchronous* messages are initiated by the switch and exploited to warn the controller about network events (node faults, network problems etc.). *Symmetric* messages are initiated by either the switch or the controller and sent without solicitation. Ultimately, OpenFlow specifies 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.

An overview of SDN architecture with a centralized controller is shown in Figure 1, where the separation between Control and Data planes of switches is highlighted. The OpenFlow protocol acts through che control messages (green dashed lines) 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.

Every SDN device is designed with an Application Programming Interface (API) for communication with controller, an abstraction layer, and a packet-processing function [6]. In case of a physical switch, the above mentioned functionality is incorporated in the hardware for packet processing logic whereas in a virtual switch it is implemented as a software agent. The abstraction layer includes one or more flow tables allowing the device to evaluate incoming packets and to take the appropriate actions based on the contents of the

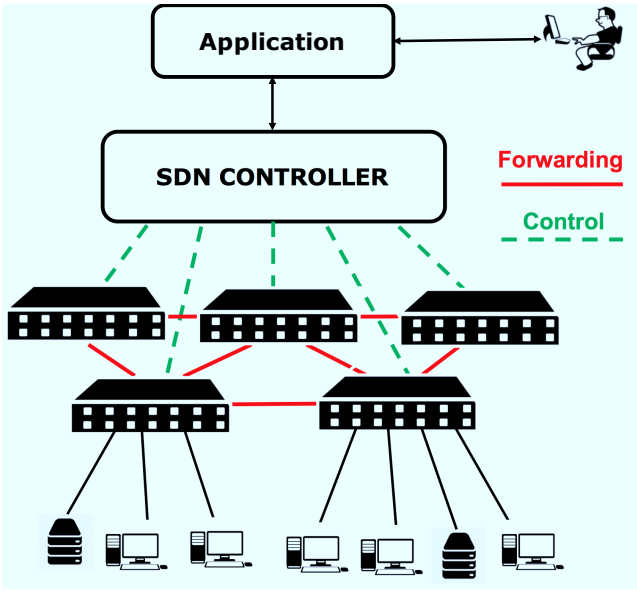

Figure 1: Overview of SDN architecture with Forwarding Plane (red solid lines) and Control Plane (green dashed lines).

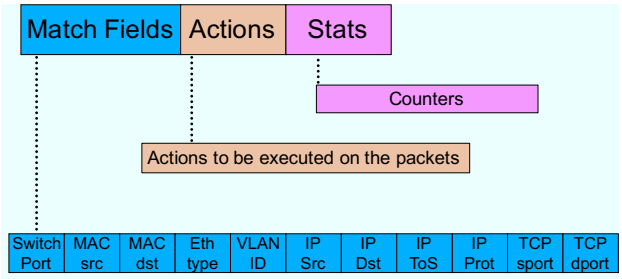

Figure 2: Example of a flow entry in an OpenFlow table.

packet that has just been received. Examples of these actions could be: forwarding a packet to a specific port, dropping a packet, flooding a packet on all ports and so on. From a logical point of view, a flow table consists of a number of prioritized flow entries with two main components: match fields and actions as shown in Figure 2. The former are used to compare packet fields with a set of specified rules; the latter represent the instructions that the network device must perform if an incoming packet satisfies one or more match fields.

2.2 SDN Controller: the core element of the architecture

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 subsequent packets in the flow are forwarded inside the data plane and do not need any control plan action.

The OpenFlow protocol supports three different modes for controllers connected to a switch, named *Equal*, *Slave* and *Master*. These operating modes can be conveniently combined to perform some redundancy schemes for high availability and load balancing purposes. A switch may be concurrently connected to multiple controllers in Equal state, as well as in Slave state, but to a single controller in Master state. Both Equal and Master modes grant the controller with the full ability to program the switch, but with the difference that just one SDN controller may act as Master, whereas multiple synchronized controllers may assume the Equal mode. As Slave instead, the controller may only gather data (such as statistics) from the switch but no modifications are allowed.

2.3 Related Works

An interesting work focused on the transferring networking technology from distributed elements into a centralized controller appeared in [7]. In their paper the authors descant on a *4D* architecture (Decision, Dissemination, Discovery and Data) where a massive refactoring of networking is designed by congregating control plane operations in a separate and independent system. The challenges of the SDN paradigm are:

- *Latency.* The presence of a centralized element entails that a certain number of decisions will be affected by non negligible round-trip latency as the networking element requests policy directions from the controller.

- *Scale.* A centralized controller is in charge of managing the whole topological infrastructure of the network and the computation of optimal paths, resulting in a scalability issue that must be necessarily taken into account.

- *High availability.* The centralized controller must not represent a single point of failure for the network so a redundancy configuration has to be addressed.

- *Security.* The centralized element could be a sitting duck for attackers so the most important countermeasures have to be taken.

In [2] an architecture for Software Defined Networking in a wireless environment has been addressed. The authors consider the Control and Provisioning of Wireless Access Points (CAPWAP) protocol standardized by the Internet Engineering Task force (IETF) in 2009 aiming to centralize the control in a wireless network. On behalf of such a protocol, control frames are delivered to a central element responsible for MAC layer control in a way that can be easily related to the way OpenFlow delivers to the controller information about new incoming flows.

HyperFlow [17] is a distributed event-based control plane for the SDN network that implements a logically centralized network control on behalf of a physically distributed control plane, with the aim of addressing the scalability problem while keeping the benefit of centralized control, so as to minimize the control plane response time.

An alternative distributed flow architecture named *DIFANE* has been proposed in [21], where a link-state routing scheme enables the switches to learn about the topology changes without invoking the controller intervention.

Another branch of works, has been devoted to reliability analysis of SDN infrastructures by taking into account the controller placement problem. In [10], for example, some placement algorithms have been designed after defining the expected percentage of control path loss as a possible reliability metric. In particular, the authors propose a *l-w-greedy* algorithm where the problem is to place k controllers among $|V|$ potential locations; the algorithm first generates a list of the potential locations, denoted as S_r increasingly ranked on the basis of switches failure probabilities, and then chooses one location at time, from the first $w|V|$ $(0 < w \leq 1)$ elements of S_r, named candidate locations for hosting controllers.

Another emerging need in the SDN context is related to advantages that the network virtualization is able to bring. A representative example is the Google SDN Wireless Area network (WAN) project [11] devoted to the design, implementation and evaluation of *B4*, a private WAN connecting Google's data centers across the world relying on a Software Defined Architecture with OpenFlow protocol, aiming to enable a way to globally optimize network usage. The advantage of virtualization mechanisms applied to SDN concept is also investigated in [16] by introducing *FlowVisor*, a module playing the same role as the hypervisor for virtual machines. Such a module allows a network operator to divide its physical infrastructure into virtual and mutually independent "slices" to be assigned to various service providers.

Furthermore, the authors in [20], 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 respectively as the queueing systems MX/M/1 and M/G/1. Subsequently, they propose a queueing model of the whole SDN networks in terms of packet forwarding performance by solving its closed-form expression.

3. A PERFORMABILITY ANALYSIS OF AN SDN CONTROLLER

By considering all the benefits and advantages a virtualized SDN solution can offer (as discussed in a recent work of the same authors [5]), we focus on a scenario with a single controller element hosting and supervising some virtual instances as shown in Figure 3, where every instance S_i represents the Master controller of the i-th service provider and is responsible of managing a set of OpenFlow-capable devices. In the following, we refer to S_i as the i-th Virtual Provider Instance (VPI). Such an approach provides a lot of benefits in terms of managing in a ductile way the whole network infrastructure that can be easily updated by negotiating opportune Service Level Agreements (SLAs) with

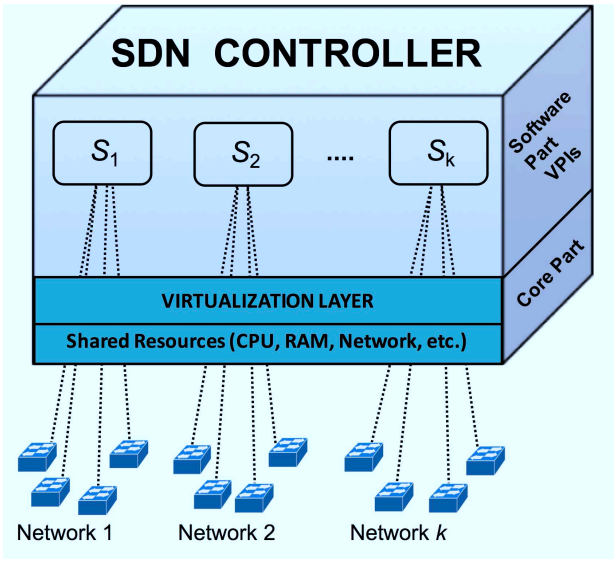

Figure 3: An SDN Controller architecture supervising k Virtual Provider Instances (VPIs).

providers resulting in better quality of committed services. On the contrary, the main flaw concerns the possibility that the physical controller might become a critical point of failure so that some redundancy procedures have to be established. Although a suitable model for an SDN controller is mainly influenced by peculiar hardware and software implementations, it is plausible to consider the controller element as composed by:

- a *core part*, comprising every kind of hardware equipment (e.g. power supply, processors, memories, blades etc.) and basic software (e.g. operating system, hypervisor etc.);
- a *software part*, corresponding to the virtual instances of controller (VPIs), able to govern a certain number of OpenFlow connections towards a bundle of SDN devices.

By considering that each VPI can manage a specific number of coexisting OpenFlow sessions within its data network, we choose as a representative performance metric the number of the actual coexisting OpenFlow sessions that each virtual operator is able to control, namely the controller serving capacity.

3.1 A multi-state performance model

In our model, 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. At this point, we introduce a multi-state performance model for SDN controller under the following hypotheses: each VPI and the core part of controller may be described by a two-state model: an "up" state and a "down" state; VPI i and core failures are statistically independent Homogeneous Poisson processes (HPPs), resulting in independent and exponential inter-failure arrivals and constant hazard rates λ_i and λ_c, respectively; VPI and core repair times are independent exponentially-distributed with rates μ_s and μ_c, respectively.

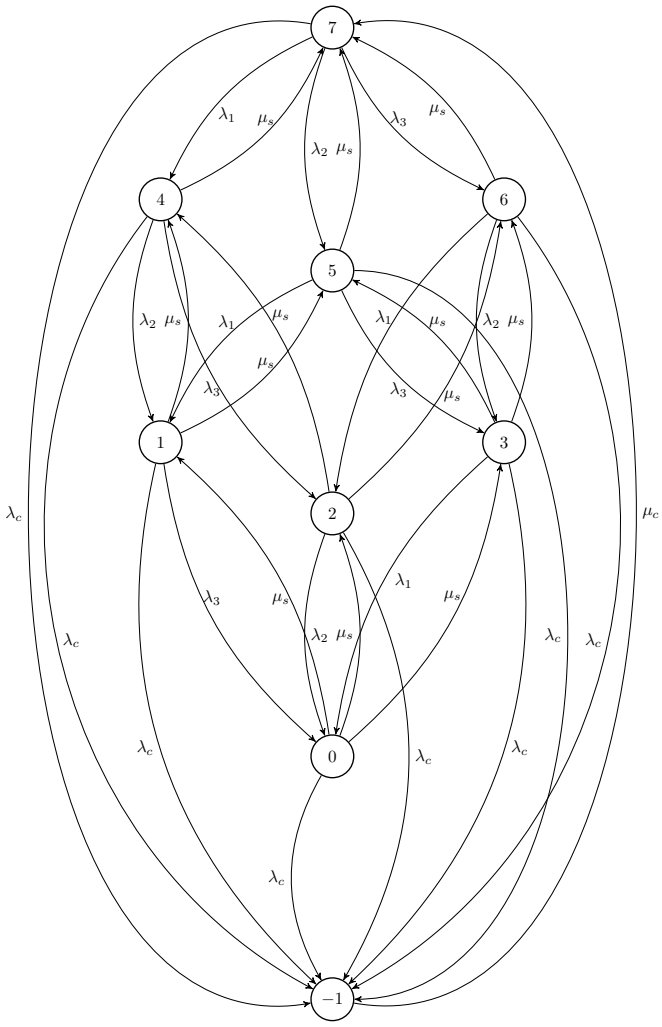

Figure 4: A multi-state model of the SDN controller supervising 3 Virtual Provider Instances.

The arising multi-state model of the SDN controller is a Continuous-Time Markov Chain (CTMC), with every state conveying the information on the working conditions (and thus the serving capacity) of the k VPIs. For the sake of ease, a CTMC model is depicted in Figure 4, where $k = 3$ VPIs are supposed to work on the same controller and where:

- $2^k + 1$ represents the total number of states, whereas the mapping between the state number and the triples with all the permutations of the up-down conditions of the VPIs is illustrated in Table 1. The up and down conditions of the VPI i are indicated by S_i and $\overline{S_i}$, respectively, and the serving capacity of the VPI is correspondingly n or 0;
- 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 triple $(\overline{S_1}, \overline{S_2}, \overline{S_3})$ for $k = 3$;
- from the state -1, only one transition towards a completely repaired controller is presumed.

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)	$(\overline{S_1}, \overline{S_2}, \overline{S_3})$	$(0, 0, 0)$

The state probabilities $p_j(t)$, $j = -1, 0, 1, .., 2^k - 1$ can be obtained by solving the following system of differential equations describing the CTMC in Figure 4:

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

with initial conditions $p_7 = 1$, $p_i = 0$ with $i = -1, 0, ..., 6$, representing an initial fully working system.

It is worth noting that the performance levels in every state of the CTMC are conveyed in a k-dimensional vector containing the serving capacity of the VPIs in that state, as reported in Table 1 for $k = 3$.

Since some redundancy is needed in real-case applications, the performance levels vectors of l-th node (in a controller redundancy configuration) are in the set

$$\mathbf{g}^{(l)} = \{\mathbf{g}_{-1}^{(l)}, \mathbf{g}_0^{(l)}, ..., \mathbf{g}_{2^k-1}^{(l)}\}, \quad (2)$$

where $\mathbf{g}_j^{(l)} = \left(g_{1,j}^{(l)}, ..., g_{k,j}^{(l)}\right)$ is the performance k-tuple, with $g_{i,j}^{(l)}$ the serving capacity of the i-th VPI (i.e., offered to the i-th network) provided by the l-th controller node in the state $j = -1, 0, ..., 2^k - 1$. Thus, the stochastic process $\mathbf{G}^{(l)}(t) \in \mathbf{g}^{(l)}$ represents the performance level of the l-th node at any instant $t \geq 0$, whose probability $p_j^{(l)}(t) = \Pr\{\mathbf{G}^{(l)}(t) = \mathbf{g}_j^{(l)}\}$ is computed by (1).

The steady-state probabilities of the CTMC, necessary for a performability evaluation in long runs, are:

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

that can be computed by solving the equations (1), where all the derivatives are posed equal to 0 and $p_j^{(l)}(t)$ are replaced by $p_j^{(l)}$, together with the normalizing condition

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

4. MULTIVARIATE UNIVERSAL GENERATING FUNCTION FOR AVAILABILITY EVALUATION

We start by considering the SDN controller as fully working when it is able to meet a required performance level (also referred to as *demand*) for each network, so a demand vector $\mathbf{W}(t) = (W_1(t), ..., W_k(t))$ is introduced.

In most practical cases, a certain level of redundancy may be necessary to meet demand. In this work, we consider the Master-Slave configuration for the controller architecture, where replicas of a single controller (core and software/VPIs parts) are admitted: in this configuration, all VPIs associated to a single domain are synchronized and share the same information on the network, so it becomes irrelevant which VPI replica has managed a specific flow entry.

Then, the SDN controller is modeled as a logical node with h parallel elements without *flow dispersion* [12], and the stochastic process describing the serving capacity offered to the i-th network is the maximum capacity offered by all the VPIs replicas responsible of managing the domain i, viz.

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

being $G_i^{(l)}(t)$ the i-th element of the k-dimensional random process $\mathbf{G}^{(l)}(t)$.

The steady-state values of the random processes $G_i(t)$, for $i = 1, ..., k$ and $t \longrightarrow \infty$, can be represented by a (discrete) random vector $\mathbf{G} = (G_1, ..., G_k)$ with a multivariate probability function $p_{\mathbf{G}}(\cdot)$, given by the steady-state distribution of the overall CTMC model provided by the parallel nodes composing the controller.

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

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

For large t, the controller initial state has no practical effect on its availability. Therefore, given one and the same constant demand level $W_i(t) = w, i = 1, ..., k$, the *steady-state availability* of the SDN controller $A_{SDN}(w)$ can be determined 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 (vectorial) state j of the SDN controller with parallel elements, whose overall model is a CTMC with m states, and $\mathbf{1}(True) = 1$ and $\mathbf{1}(False) = 0$.

A suitable procedure to evaluate the system availability is based on the Universal Generating Function (UGF) method, originally introduced by [18], and typically called u-transform.

The UGF method can be considered a hierarchical approach that avoids to address the solution of the overall state space model describing the performance of the whole system under analysis, which likely turns out 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 series and parallel connections. See [12] for further details.

The UGF of a discrete random variable X is a polynomial-shape function $u(z) = \sum_{i=1}^{I} q_i z^{x_i}$, where X has I values x_i and $q_i = \Pr\{X = x_i\}$. The UGF of the random variable representing the performance levels of a multi-state system allows to evaluate system availability. In addition, the UGF of a system can be efficiently computed by composing the UGFs of all subsystems by series and parallel operators. However, the UGF approach must be extended to the multivariate case in order to handle performance random vectors, such as $\mathbf{G}^{(l)}$ and $\mathbf{G}$.

Consequently, we define the *Multivariate* UGF (MUGF) $u(\mathbf{z})$ of the k-dimensional random vector $\mathbf{G}$, with values in the set $\{\mathbf{g}_1, ..., \mathbf{g}_m\}$ and multivariate probability function $p_{\mathbf{G}}(\cdot)$, as

$$u(\mathbf{z}) = \sum_{j=1}^{m} p_{\mathbf{G}}(\mathbf{g}_j) \prod_{i=1}^{k} z_i^{g_{i,j}}, \tag{8}$$

where $\mathbf{z} = (z_1, ..., z_k)$.

Let h be the number of parallel elements without flow dispersion composing the SDN controller, whose serving capacity is ruled by (5). The MUGF of the controller can be calculated by the following π operator:

$$\begin{aligned} u_{SDN}(\mathbf{z}) &= \sum_{r} p_r \prod_{i=1}^{k} z_i^{g_{i,r}^{SDN}} = \\ &= \pi\left(u_1(\mathbf{z}), u_2(\mathbf{z}), ..., u_h(\mathbf{z})\right) = \\ &= \sum_{j_1=-1}^{2^k-1} \sum_{j_2=-1}^{2^k-1} \cdots \sum_{j_h=-1}^{2^k-1} \prod_{l=1}^{h} p_{j_l}^{(l)} \prod_{i=1}^{k} z_i^{\max_{l=1,\ldots,h} g_{i,j_l}^{(l)}}, \end{aligned} \tag{9}$$

where $p_{j_l}^{(l)}$ are the steady-state probabilities in (3) corresponding to the performance k-tuple $\mathbf{g}_{j_l}^{(l)}$ in (2). From (9), we derive p_r and $g_{i,r}^{SDN}$ that are used to compute the steady-state availability of the SDN controller by (7).

In order to minimize the cost of the SDN controller, it is relevant to find the minimal number of parallel h^* that meets a condition on the steady-state availability A_0, viz.

$$h^* = \arg\min_{h \in \mathbb{N}} \left(A_{SDN}(w, h) \geq A_0\right). \tag{10}$$

The problem in (10) is a simple "redundancy optimization problem" [19].

5. A NUMERICAL EXPERIMENT

In this section, we provide a numerical example as an application of the methodology proposed in the previous paragraphs. We assume that every Service Provider has negotiated the same Service Level Agreements (SLAs), resulting in the same number of coexisting OpenFlow sessions managed by the controller. It corresponds to a service demand level equal to $w = 4800$ sessions per time unit, chosen to be close to the highest performance level of a single VPI (serving capacity). Each VPI has a serving capacity of $n = 5000$ sessions per time unit while failure and repair rates are: $\lambda_1 = 3.858 \times 10^{-7}$ s^{-1} (corresponding to 1 fault per month for S_1), $\lambda_2 = 7.716 \times 10^{-7}$ s^{-1} (corresponding to 2 fault per month for S_2), $\lambda_3 = 1.157 \times 10^{-6}$ s^{-1} (corresponding to 3 fault per month for S_3), and $\mu_s = 1.388 \times 10^{-4}$ s^{-1} (corresponding to a mean repair time of 2 hours for all VPIs). In our model, different failure rates for each VPI have been assumed in order to take into account the possible differentiation in terms of capabilities. On the contrary, one and the same value μ_s has been considered by presuming common repair operations for each VPI.

Table 2: Numerical data

Parameter	Value
n	5000 sessions/time unit
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 sessions/time unit
A_0	0.999999

Table 3: Steady-state probabilities and performance levels of the controller after redundancy optimization

Probability	Performance (sessions/time unit) Triples
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	$(5000, 5000, 5000)$

Node core failure and repair rates are $\lambda_c = 1.268 \times 10^{-7}$ s^{-1} (corresponding to 4 core faults per year) and $\mu_c = 3.472 \times 10^{-5}$ s^{-1} (corresponding to a mean repair time of 8 hours for a technical activity on site), respectively. Table 2 summarizes the illustrative example data. It is worth noting that all the reported values are chosen arbitrarily, but in keeping with system engineers experience.

With the considered values, the "six 9s" availability (increasingly desirable in telecommunication systems) of the SDN controller, is reached (and even exceeded) with 4 parallel elements; besides, the steady-state availability is equal to:

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

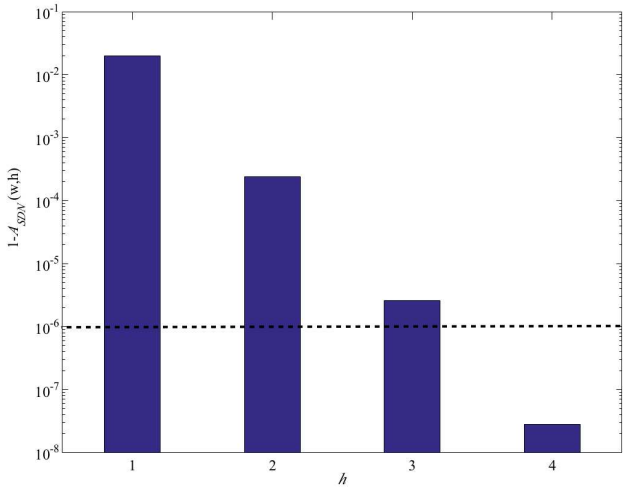

Figure 5: Unavailability $1 - A_{SDN}(w, h)$ of the SDN controller architecture for $h = 1, 2, 3, 4$ parallel elements. The horizontal dashed line represents the required steady-state unavailability $1 - A_0 = 10^{-6}$ of the SDN controller architecture.

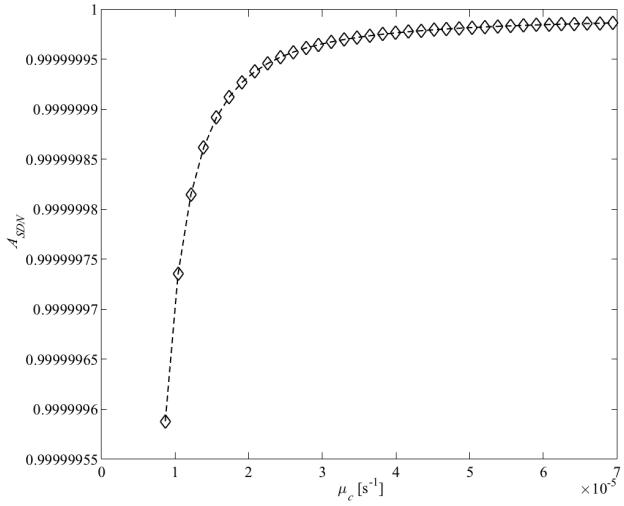

Figure 6: Influence of core repair rate for $N = 4$ redundancy levels.

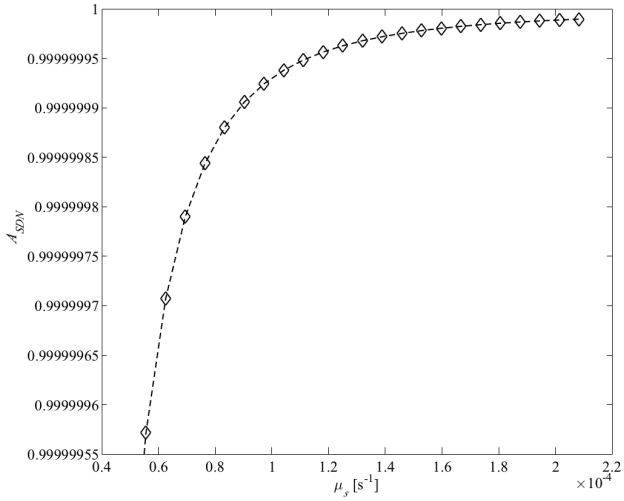

Figure 7: Influence of software instances (VPIs) repair rate for $N = 4$ redundancy levels.

The steady-state probabilities of a single node are derived by solving (1) and (4), while the output performance levels and steady-state probabilities of SDN controller after redundancy optimization are shown in Table 3. The experiments have 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 i7-4960HQ 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 with the modern standard requirements of very high availability. Besides, a sensitivity analysis reveals that the "six 9s" condition is reached also in the case of changing parameters μ_c and μ_s as shown in Figures 6 and 7, respectively. In particular, some reduction of the repair rates can be admitted, and thus a larger mean repair time for the technical activities can be allowed, while meeting the "six 9s" condition.

6. CONCLUSIONS

Software Defined Networking is a novel network concept based on the idea of decoupling control plane from data forwarding plane in order to improve network flexibility by introducing a protocol called OpenFlow. In our work, we developed a performance analysis of the most critical element of an SDN infrastructure, namely the controller, in the presence of random failures, aiming to obtain the so called "six 9s" of availability, increasingly required by the telecommunication world. Such an analysis has been carried out by considering the controller hosting and supervising some software instances, in a virtualized environment, able to manage a set of SDN capable devices. By choosing as a performance metric the number of coexistent OpenFlow sessions handled by each VPI, the performability analysis has been focused on a typical Master-Slave configuration of the SDN controller, where all parallelized software instances are synchronized thus sharing the same information on the network. The parallel redundancy optimization problem has been faced adopting the UGF approach extended to a multivariate case denoted as *Multivariate* UGF (MUGF), useful to deal with performance random vectors characterizing the steady-state conditions. Future works will be devoted to the investigation of other performance metrics, such as the response delay of the SDN controller to the requests of the switches, and on the extension of the proposed approach to a new emerging virtualization technology, strictly related to SDN, called Network Function Virtualization (NFV).

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] A. D. Malloy, U. Varshney, and A. P. Snow. Supporting mobile commerce applications using dependable wireless networks. *Mobile Networks and Applications*, 7(3):225–234, 2002.

[15] 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.

[16] 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.

[17] 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.

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

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

[20] 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.

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