UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II


# UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

## PH.D. THESIS IN

### INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

# IN-PRODUCTION CONTINUOUS TESTING FOR FUTURE TELCO CLOUD


## UGO GIORDANO

**TUTOR: PROF. STEFANO RUSSO**

**CO-TUTOR:**
**DR. MARINA THOTTAN, NOKIA BELL LABS**
**DR. CATELLO DI MARTINO, NOKIA BELL LABS**


**XXIX CICLO**


SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE DELL'INFORMAZIONE

# IN-PRODUCTION CONTINUOUS TESTING FOR FUTURE TELCO CLOUD

By
Ugo Giordano

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
AT
"FEDERICO II" UNIVERSITY OF NAPLES
VIA CLAUDIO 21, 80125 – NAPOLI, ITALY
NOVEMBER 2017

*"To my grandfather Ugo, the best man I have ever meet in my life."*

# Table of Contents

# List of Tables

# List of Figures

# Acronyms

**API** Application Programming Interface.

**ASIC** Application-Specific Integrated Circuit.

**BER** Bit Error Rate.

**CAPEX** CAPital EXpenditure.

**CM** Campaign Manager.

**CPU** Central Processing Unit.

**CRC** Cyclic Redundancy Check.

**DC** Data Collector.

**DDoS** Distributed Denial of Service.

**DOE** Design Of Experiments.

**EJB** Enterprise Java Beans.

**EM** Experiment Manager.

**FEC** Forward Error Correction.

**FI** Failure Injection, Failure Injector.

**FIT** Failure Injection Testing.

**IBN** Intent-Based Networking.

**IPC** Inter-Process Communication.

**IPS** Intents operations per unit time.

**ISP** Internet Service Provider.

**JMS** Java Message Service.

**JMX** Java Management Extensions.

**JSE** Java Standard Edition.

**JVM** Java Virtual Machine.

**LG** Load Generator.

**MTBF** Mean Time Between Failures.

**MTTF** Mean Time To Failure.

**MTTR** Mean Time To Repair.

**NBI** Northbound Interface.

**NFV** Network Function Virtualization.

**NFVI** Network Function Virtualization Infrastructure.

**NOS** Network Operating System.

**ODL** OpenDayLight (registered trademark).

**ONF** Open Network Foundation.

**ONOS** Open Network Operating System (registered trademark).

**OPEX** OPerating EXpenditure.

**OS** Operating System.

**OSGi** Open Service Gateway Initiative.

**Pub/Sub** Publish/Subscribe.

**REST** Representational State Transfer.

**RMI** Remote Method Invocation.

**SBI** Southbound Interface.

**SCP-CLUB** SDN Control PlaneCLoUd-based Benchmarking.

**SDN** Software-Defined Networking, Software-Defined Networks.

**SFI** Software Fault Injection.

**SLA** Service Level Agreement.

**SUT** System Under Test.

**TC** Telco Cloud.

**TCSP** Telco Cloud Service Provider.

**TM** Topology Manager.

**VM** Virtual Machine.

**VNF** Virtualized Network Function.

**WG** Workload Generator.

# Acknowledgements

I would like to thank my advisor, Prof. Stefano Russo. I'm really thankful for all the tips and advice he gave me during these three years, and for having supported me in pursue the PhD.
Much of the work for this dissertation has been performed during a period at the prestigious NOKIA Bell Labs in Murray Hill, New Jersey, USA, under the supervision of Dr. Marina Thottan and Dr. Catello Di Martino. I am very grateful to NOKIA for the opportunity to work in a top-level research environment. I would like to thank specifically Dr. Thottan and Dr. Di Martino for the inception of the ideas underlying the SCP-CLUB benchmarking framework and the failure injector architecture, the useful discussions, their patient support and their compelling stimuli. Working with them at NOKIA Bell Labs has been as challenging as exciting.
I am also grateful to all friends (because we are friends, not just colleges) of the DEpendable Systems and Software Engineering Research Team (DESSERT) at DIETI, Federico II University, and at the CINI "Carlo Savy" laboratory in Napoli. Thanks to Stefano, Luigi, Flavio, Antonio, Roberto P., Roberto N, Mario, Anna and Alma. A special thank goes to Raffaele, Fabio and Salvatore (vaiiii vaiii!!!). They always supported me during the hard time of this PhD, and sharing a beer with them has always been a pleasure. Really thanks guys, hoping to have been a good friend and supporter for you too.

Thanks to my father, my mother, my bro, and my sisters, and my grandchildren Giulia and Michele (they always made you smile in all situations),

to my family; because without their support I would never be able to accomplish this last step. Thanks for all!
Thanks to my Bernardo and Giovanna, my second family. Thanks for letting me to be part of your family.
Thanks to YOU, Rosaria, my best friend, my greatest support, my biggest comfort, my strongest motivation, my truest smile, my deepest love, my favorite, my forever. Had it not been for her, I might still be in darkness, or I would have invented her. Sincerely, today, I don't know where I would be if she wasn't been on my side, where she has always been. There, close to me, for me. I will love you until death do us part!!! I'll always love you, as if it were the first day, as if there was not a tomorrow (I love you).

Naples, Italy — Ugo Giordano
October 4th, 2017

**Disclaimer**: The views and opinions expressed in this dissertation are those of the author and do not necessarily reflect the official policy or position of any person, organization or company other than the author.

Chapter 1

# Introduction

## 1.1 Context

Computer networks are nowadays at the basis of most critical infrastructures, and of many services we access in our daily activities - be they business, consumer, social or private. **Software Defined Networking** (SDN) has emerged in the very last few years - from the initial work done at University of California at Berkeley and Stanford University in 2008 - as a paradigm capable of providing new ways to design, build and operate networks. This is due to the key concept underlying it, namely the separation of the network control logic (the so-called **control plane**) from the underlying equipment (such as routers and switches) that forward and transport the traffic (the **data plane**) [1].

Thanks to the clear separation of the two *abstraction levels* - the logic level, corresponding to the control plane, and the physical one, i.e. the data plane - SDN is claimed, and by many experts strongly believed, to be about to introduce a big revolution in computer networking [2]. Along with Network Function Virtualization (NFV), SDN is expected to have a positive

impact on network management costs [3]. Indeed, the logical level may host the network control logic in a *programmable* and *highly flexible* way: advanced network services become software defined, supporting much easier enforcement of networking policies, security mechanisms, reconfigurability and evolution than in current computer networks.

The SDN flexibility is due to the *separation of concerns* between network configuration and policies definition and lower-level equipments for traffic switching and routing, a direct consequence of the separation of abstraction layers. The many advantages promised by SDN in engineering and managing computer networks and in operating their services are very attractive for network operators and Internet Services Providers (ISP). Network operation and management are challenging, and providers face big issues in configuring large networks, enforcing desired policies, and evolving to new technologies - all in a very dynamic environment [4]. This is easily comprehensible thinking, for instance, at the huge difficulties that major technological changes encounter to be applied in large networks. The transition from the Internet network protocols IPV4 to IPV6 is just an example: started about a decade ago, it is probably still far to be completed. And it has to be considered that protocols, which are at the heart of computer networks, are basic blocks from the point of view of the highly demanding modern and future fixed and mobile applications and services.

According to Allied Market Research, the SDN market is expected to reach $132 billion by 2022 [5]. Players in this market include telecommunication operators, ISPs, cloud and data center providers, and equipment manufacturers. Beside the decoupling of service, software and hardware

technology innovations in networking, there is probably a fundamental reason for such big expectation raised in the networking industry. The history of major advances in computer science and engineering is a history of raising the level of abstraction. This is true for instance for programming languages, for operating systems and middleware technologies, for software design (up to modern model-driven techniques) [6]. Abstraction and separation of concerns are fundamental engineering principles, which in the case of SDN may well support its wide spread.

The logical entity hosting software-defined core network services in the control plane (e.g. routing, authentication, discovery) is typically known in the literature as **SDN controller** (or simply controller). Very recently, the concept of controller has evolved to that of **network operating system** (NOS), an operating system - which can possibly run on commodity hardware - specifically providing an execution environment for network management applications, through programmable network functions. In the logical SDN architecture, the controller is below the application layer[1], and atop the data plane, that it controls enacting the policies and the services required by applications. The separation of the planes is realized by means of well-defined application programming interfaces (API) between them. Relevant examples of SDN controllers are NOX [7], Beacon [8], OpenDaylight [9] and ONOS® [10], while probably the most widely known API is OpenFlow [11].

[1]The applications atop the control plane are actually management programs accessing the controller programming interface to request network services or to enforce policies.

## 1.2 Motivations and contributions

Today's networks will need to adopt a new approach to support the predicted growth in scale, diversity and complexity of use cases [12]. With new services and applications emerging continuously, devices will be connected much more often, and consequently, a distinct competitive market advantage could be created by those network operators capable of implementing new services rapidly.

In order to meet evolving market demands, and increase the flexibility and agility of networks promoting innovative solutions for future network services, telco companies look with interest at migrating towards the emerging ***Telco Cloud*** (TC) paradigm [13] [12] [14], becoming so-called ***Telco Cloud Service Providers*** (TCSPs). Telco Cloud is meant to provide a dedicated cloud computing solution for a network operator, to shift network functions away from dedicated legacy hardware platforms into virtualized software components deployable on general-purpose hardware. This logically allocates cloud and networking capabilities into a multi-service programmable fabric built to precisely meet each of the different service requirements, changing network conditions, unpredictable traffic patterns, continuous streams of apps and services and short innovation cycles.

This ability to focus on what is needed is achieved through the combined use of **Software Defined Networking** (SDN) [1], **Network Function Virtualization** (NFV), and cloud technologies [15], [16] in telco cloud infrastructures.

While the *softwarization* and *cloudification* of the network provides unparalleled level of automation and flexibility, and a drastic reduction of the

network operating margins, it also presents significant challenges, mainly due to the variety of knobs (e.g., SDN-, cloud-, and software- related parameters) to properly fine tune in [16] order to obtain specific levels of service.

In the SDN world, performance it is not only related to the behaviour of the data plane. As the separation of control plane and data plane makes the latter significantly more agile, it lays off all the complex processing workload to the control plane. This is further exacerbated in distributed network controller (e.g., ODL [9], and ONOS [10]), where the control plane is additionally loaded with the state synchronization overhead. Misconfiguration of the control plane can negatively impact the overall network performance, cause customer insatisfaction and, in more extreme cases, network unavailability. Understanding the performance of the SDN control plane in a telco cloud and the factors that influence it is fundamental for planning, sizing and tuning SDN deployments.

Furthermore, the introduction of SDNs technologies has raised advanced challenges in achieving **failure resilience**, meant as *the persistence of service delivery that can justifiably be trusted, when facing changes* [17], and **fault tolerance**, meant as *the ability to avoid service failures in the presence of faults* [18] (these definitions will be used hereafter to refer to the resilience and fault-tolerance of SDN technologies). The decoupling of the control plane from the data plane leads the dependency of the overall network resilience on the fault-tolerance in the data plane, as in the traditional networks, but also on the capability of the (logically) centralized control

functions to be resilient to faults. Moreover, the SDNs are by nature suitable to be implemented as distributed system, introducing further threats to the network resilience, such as inconsistent global network state shared between the SDN controllers, as well as a network partitioning. In addition, compared to the legacy network appliances, which rely on dedicated high-performance hardware, the adoption of technologies for virtualizing network services, introduces performance and reliability concerns, e.g., high overhead/latency and failures, due to new failure scenarios which periodically occur in data center [19] [20].

Consequently, as the controllers technology develops and progressively becomes mature for the market, the need to engineer and to assess the compliance of SDN solutions with non functional requirements – such as scalability, high availability, fault tolerance and high resilience – becomes more compelling. In such a context, the **traditional software testing techniques appear insufficient to evaluate the resilience and availability of a distributed SDN ecosystems**. Indeed, although these techniques are useful to validate specific system behaviours (e.g. the functional testing), full operational testing may be possible only in production, due to the impossibility to reproduce the entire ecosystem in a testing environment. Ultimately, even if a system can be reproduced in a test context, it is impractical, or even impossible, to fully reproduce all aspects and failure modes that can characterize complex distributed systems during **production hours** [21]. On the other hand, a widely recognized effective way to assess fault-tolerance mechanisms as well as to quantify system availability and/or reliability is *failure injection*. Failure injection allows to assess fault

tolerance mechanisms by reproducing multiple failure scenarios, such as a latent communication, service failure, or hardware transient faults. Furthermore, if applied in a controlled environment while the **system is in production**, the *failure injection* can lead to discover problems in a timely manner, without affecting the customers, and providing helpful insights to build better detection, and mitigation mechanisms to recover the system when real issues arise.

Therefore, along with the "*softwarization*" of network services, it is an important goal in the engineering of such services, e.g. SDNs and NFVs, to be able to test and assess the proper functioning not only in emulated conditions before release and deployment, but also **in-production** [22], when the system is under real operating conditions.

The goal of this thesis is to devise an approach to evaluate not only the performance, but also the effectiveness of the failure detection, and mitigation mechanisms provided by SDN controllers, as well as the capability of the SDNs to ultimately satisfy non functional requirements, especially resiliency, availability, and reliability. The approach consists of exploiting *benchmarking* techniques, such as the *failure injection*, to get **continuously** feedback on the performance as well as capabilities of the SDN services to survive failures, which is of paramount importance to improve the effectiveness of the system internal mechanisms in reacting to anomalous situations potentially occurring in operation, while its services are regularly updated or improved.

To the best of our knowledge, there is no available approach or tool that can be used to provide automation in the analysis of the SDN control

plane. The literature on SDN performance and resilience assessment is still at the beginning. The thesis aims to **contribute to the advancement in testing and evaluation of SDNs**, trying to go beyond what can be achieved by means of "traditional" software analysis and testing techniques.

Within this vision, this dissertation first presents **SCP-CLUB** (**SDN Control Plane CLoUd-based Benchmarking**), a benchmarking framework designed to automate the characterization of SDN control plane performance, resilience and fault tolerance in telco cloud deployments. The idea is to provide the same level of automation available in deploying NFV function, for the testing of different configuration, using idle cycles of the telco cloud infrastructure. Then, the dissertation proposes an extension of the framework with mechanisms to evaluate the runtime behaviour of a Telco Cloud SDN under (possibly unforeseen) failure conditions, by exploiting the **software failure injection**.

Differently from software *fault* injection [23] - a nowadays consolidated form of testing - failure injection focuses on deliberately introducing *failures* in the components of the system under assessment, or in their execution environment, under real or emulated load conditions, to evaluate the ability of the system internal mechanisms to react to anomalous situations potentially occurring in operation.

Overall, the framework provides an approach to implement an automated methodology for characterizing the performance and resilience of the SDNs, and consists of a configurable software infrastructure. The distributed infrastructure encompasses - as main components - a set of *management* tools, a *workload generator*, a *failure injector*, and *data collectors*.

The experimental evaluation of the proposed framework is based on the open source distributed network operating system, ONOS® [10] [24], which is the very heart of the testbed. The ONOS® initiative is supported by several major industrial partners, including AT&T, Cisco, Ericsson, Google, Huawei, NOKIA.

In summary the **main contribution of this thesis** are:

- An automated and configurable distributed infrastructure (named SCP-CLUB framework) for deploying and testing SDN controllers under various configurations. The infrastructure encompasses tools to support the management of the experiments, the load generation, failure injection and data collection tasks;

- An injection methodology, conceived for both development and in-production stage assessment. The methodology envisages the steps of *(i)* definition of the workload (according to the Intent Based Networking model [25]) to emulate actual operating conditions of a controller; *(ii)* workload generation and actual injection of failures, selected from the failure model, in the emulated load conditions; *(iii)* data collection and assessment analysis. Clearly, workload emulation (definition and generation) is not necessary for in-production tests, yet it is important at the current state of the practice given the limited availability of SDN on-field deployments;

- The experimental evaluation, on a distributed testbed based on ONOS® over Nokia AirFrame telco cloud technologies.

## 1.3 Thesis organization

The dissertation is organized as follows.

**Chapter 2** introduces the main concepts of Software-Defined Networking, expected to become the paradigm underlying the next generation of computer networks. It then presents a discussion on the current state of art, and the identified open challenge, namely the resilience assessment of SDN controllers.

**Chapter 3** surveys the literature on SDN architectures and platforms, and in particular on their resilience mechanisms, and on failure injection testing techniques and tools for the assessment of software intensive systems. It then discusses the opportunities envisaged in the proposed application of failure injection methods to the problem of assessing the resilience of SDN controllers.

**Chapter 4** presents the SCP-CLUB (SDN Control Plane CLoUd-based Benchmarking), a cloud-based benchmarking framework designed for performance analysis of a telco cloud-based SDN control plane. SCP-CLUB provides the automated tools to deploy and test SDN infrastructures, allowing telco operator to perform the assessment of SDNs in controlled as well as in-production environments. The Chapter describes the implementation of the proposed framework based on the ONOS® distributed SDN controller. The results of extensive experiments in an industrial telco cloud infrastructure are presented, showing the effectiveness of SCP-CLUB in automating performance evaluation campaigns.

**Chapter 5** presents the vision of continuous and in-production testing, and describes the improvements made to the SCP-CLUB framework for the

resilience assessment of SDNs. The central idea is the use of failure injection to assess the failure detection and mitigation mechanisms of SDN controllers. The failure injection methodology and the support infrastructure are presented along with the underlying failure model, listing the variety of injectable failure types at system, network and service level. Then, the chapter presents the ONOS-based implementation of the proposed failure injection framework, and discusses the experiments by injecting failures into ONOS®.

**Chapter 6** summarizes the problem addressed in this thesis and its main contributions.

This page intentionally left blank.

The only joy in the world is to begin.

*Cesare Pavese*

Chapter 2

# Software-Defined Networks

*The Chapter introduces the concept of software-defined networking and its logic architecture. It then presents the major dependability requirements for SDNs, preceded by a short introduction of the basic concepts of dependability. Finally, the open challenges in SDN dependability - and in particular, SDN resilience - are identified, which drive the work of this dissertation.*

## 2.1 Abstracting the Network: SDN

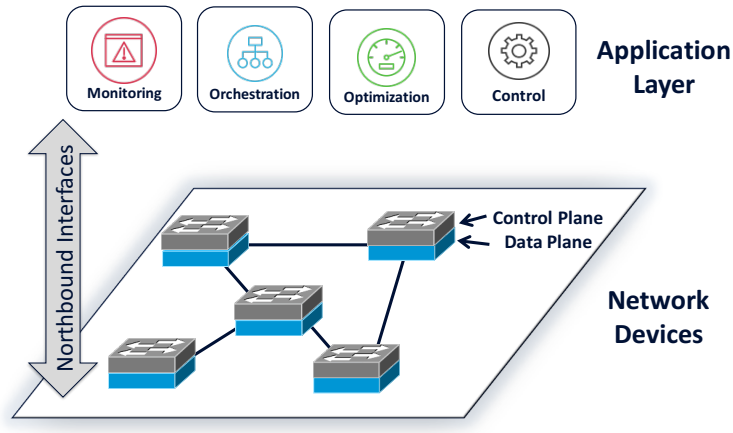
### 2.1.1 Key Concepts

**Software Defined Networking** (SDN) is an emerging paradigm to design, build and operate networks. It originated from work started at University of California at Berkeley and Stanford University in 2008, and it has increasingly gained momentum from both the research and industrial viewpoints in the computer networking sector.

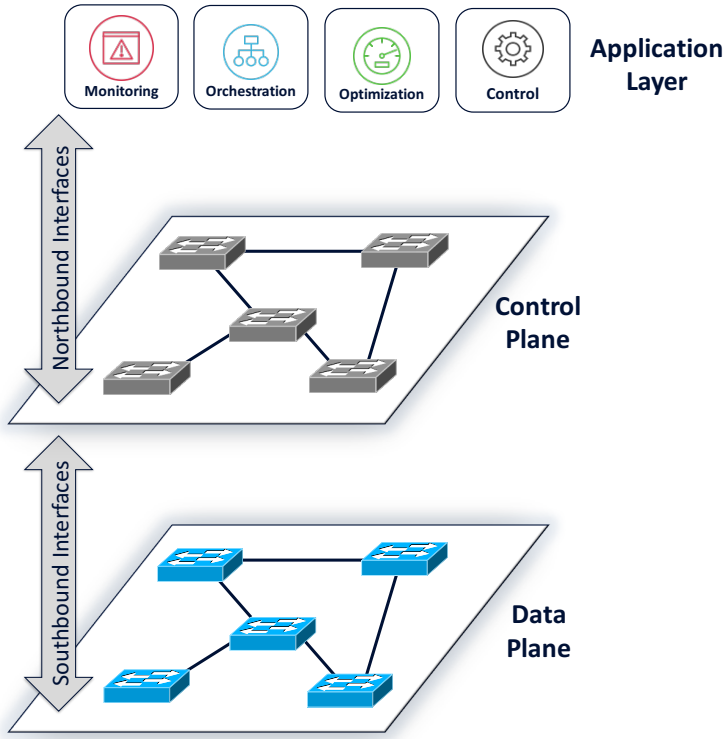
The driving motivation was the need for a major shift in networking technologies in order to support much easier configuration, management,

operation, reconfiguration and evolution than in current computer networks. Indeed, network operation and management are challenging tasks, and telecommunication operators and Internet and cloud service providers face big issues in configuring large networks, enforcing desired policies, and evolving to new technologies [4]. Computer networks are nowadays at the basis of most critical infrastructures, and of the many services we access in our daily activities - be they business, consumer, social or private. Large network's configuration and management is very difficult because enforcing high-level policies requires specifying them in terms of low-level commands of many proprietary, vertically integrated devices of different vendors [4]. These difficulties hamper the development and rapid provisioning of new advanced protocols and services to the highly demanding modern and future fixed and mobile applications. This motivation led to the definition of a new paradigm, envisaging a layered network architecture.

The key concept (Figure 2.1) is the *separation of the network control logic from the network equipments* that forward and transport the traffic [1]. Traditional networks are hardware-centric, and most network equipments (e.g., routers and switches) are *closed*, in the sense they incorporate both the control and data parts (Figure 2.1a), and have their own vendor-specific interfaces. Replacing or simply updating protocols and services is very complex because all equipments have to be replaced/updated [26]. In SDN




(a) Traditional networking




(b) Software-defined networking

Figure 2.1: Separation of data and control in SDNs.

(Figure 2.1b), network devices are simply packet forwarding devices residing in the so-called **data plane**, while the "brain" (the programmable control logic) resides in the **control plane**, distinct and above the data plane; network equipments are programmed via the control layer.

### 2.1.2 SDN Architecture

As shown in Fig. 2.2, the separation of the layers is realized by means of *open* application programming interfaces (API), called **Northbound Interface** (NBI, towards applications) and **Southbound Interface** (SBI, between the control and data planes). This concept allows co-existence of the new paradigm with the traditional one; indeed, several current commercial network equipments are hybrid, supporting both the new SBI and traditional protocols. This should ease the transition to SDN architectures.

The logical entity hosting software defined core network services in the control plane (e.g. routing, authentication, discovery) is known as **SDN controller** (or simply controller). It is responsible for enacting the policies and the services required by applications, by issuing commands are receiving events and status information from devices in the data plane (referred to as *SDN-enabled switches*) through the SBI.

Much effort in the scientific and technological SDN literature has been put on the Southbound Interface. The main southbound API is OpenFlow

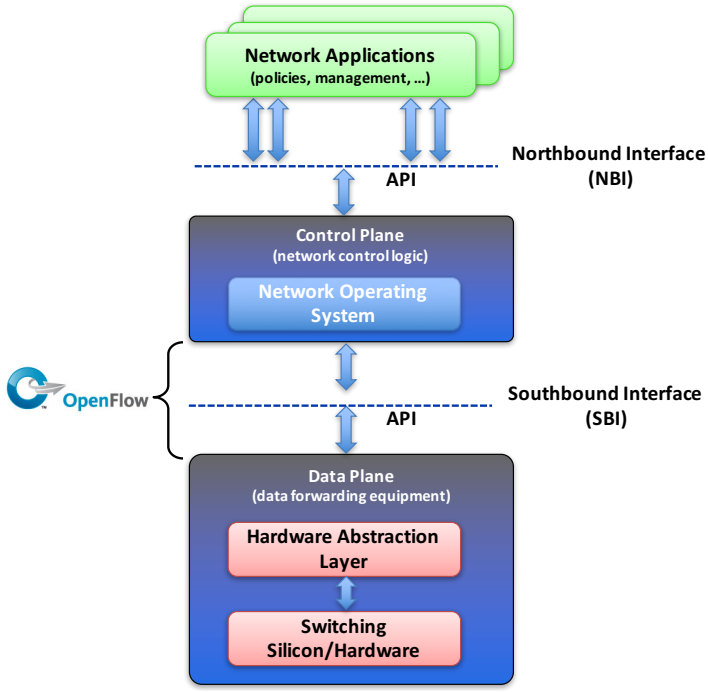



Figure 2.2: SDN architecture and interfaces.

[11], standardized by the Open Networking Foundation [27] (see Section 2.1.3), and supported by many leading network equipment vendors (such as IBM, NetGear, NEC, HP).

Less emphasis has been put so far on the northbound interface and protocols. The NBI is responsible to provide means to specify and request network policies and services in an abstract way, independent from way they are actuated by the controller. A promising proposal for the northbound interface is the **Intent-Based Networking** (IBN) model [28] [25], adopted in the ONOS project; it is described in Subsection 2.1.4.

### 2.1.3 Standards and technologies

From a technological and industrial viewpoint, several initiatives have started in the recent years to foster SDN development. A major initiative is the **Open Networking Foundation** (ONF), a non-profit organization launched in 2011 by Deutsche Telekom, Facebook, Google, Microsoft, Verizon, and Yahoo!. ONF currently counts tens of partner companies and it has as mission the "promotion and adoption of Software-Defined Networking through open standards development" [27]. Another relevant non-profit initiative is the **Open Networking Lab** (ON.Lab), established by service providers, network operators and network equipment vendors with the main goal of building open SDN tools and platforms. ONF and the ON.Lab announced in late 2016 they will join in 2017 under the ONF name to accelerate the adoption of SDN.

The first ONF achievement is the **OpenFlow™ Standard**, enabling remote programming of the forwarding plane [29]. OpenFlow provides the interface between the control and data planes, enabling a seamless communication between components in the two levels. OpenFlow was initially proposed for technology and application experimentation in a campus network [11]. It then gained momentum, up to be defined as an ONF standard for the southbound interface between the control and the data plane.

The **ONOS project** has been established to develop an open SDN-based vision of the next generation networks, to go beyond current networks, which are "closed, proprietary, complex, operationally expensive, inflexible" [30]. The explicit goal is "to produce the Open Source Network Operating System that will enable service providers to build real Software Defined Networks" [30]. This goal is pursued by a community of partners including many of the major industrial players in the field, be they network operators, Internet service providers, cloud and data center providers, and vendors, including AT&T, Cisco, Ericsson, Google, Huawei, NOKIA, NEC, NTT Communications, Samsung, Verizon.

The project has promoted the development of **ONOS™ (Open Network Operating System)**, claimed to be the first open source SDN network operating system. ONOS is not the first open source SDN controller, yet it is the first targeting scalability, high availability and high performance. It has been conceived to overcome the limitations of previous controllers such as NOX [7] and Beacon [8], which were closely tied to the OpenFlow API and provided applications with direct access to OpenFlow messages - in this sense, they did not provide the proper level of SDN abstraction to applications, hence the need for a real SDN *network operating system*[1].

[1]Note that the term network operating system (NOS) some decades ago referred to operating systems with networking features (such as the one by Novell); this obsolete usage has been changed in [7] "to denote systems that provide an execution environment for programmatic control of the network". This is what is currently still meant with the term in the context of SDN, which is probably why nowadays the terms *SDN controller* and *network operating system* are often used interchangeably.

### 2.1.4 Intent-Based Networking

In the ONOS view, the Intent-Based Networking model plays an important role in specifying the network needs through a policy-management service [24]. The idea of IBN is that applications should send requests or policies to the control plane in the form of **intents**, specified in terms of *what* and not in terms of actions to be taken in controlling the network, i.e. of *how* they should be actuated (the slogan is: *"tell me what you need not how to do it!"*). A simple example of an intent is the request to establish a point-to-point interconnection between two nodes, complemented by performance or service requirements, such as minimum bandwidth and duration.

An intent can be regarded as an object containing a request to the network operating system to alter the network behavior. It may consist of:

- Network Resources, the parts of the network affected by the intent;
- Constraints, such as bandwidth, optical frequency and link type requested;
- Criteria, describing the slice of traffic affected by the intent;
- Instructions, i.e. actions to apply to the slice of traffic of interest.

The IBN model abstracts the specification of the needs of workloads consuming network services (the "what"), from the way the network infrastructure satisfy those needs (the "how"). This is meant to be achieved

through a policy-management service at the northbound interface of the SDN controller; the latter is in charge of translating the network policies into corresponding control actions, e.g. flow rules installation.

The intents specify at a logical level the actions requested, then the SDN is in charge of satisfying them by directly interacting with the physical devices. By doing so, the IBN framework abstracts the network complexity allowing network operators and applications to describe policies rather than low level instructions for the devices.

## 2.2 SDN Dependability

### 2.2.1 Basic Dependability Concepts

**Dependability** of a system or a service is a concept encompassing several quality attributes (Fig. 2.3), namely *availability*, *reliability*, *safety*, *confidentiality*, *integrity* and *maintainability*, with confidentiality, integrity, and availability being part of the composite attribute *security* [18].

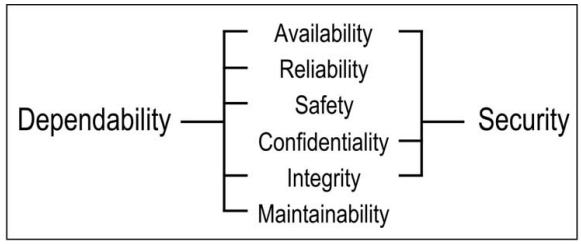



Figure 2.3: Dependability attributes (Avizienis et al., 2004).

**Reliability** expresses the continuity of correct service. It is the probability that the system functions properly in the time interval $(0, t)$:

$$R(t) = P(!failure\ in\ (0, t)). \tag{2.1}$$

Typically, reliability is evaluated through widely spread metrics such as Mean Time To Failure (MTTF), Mean Time Between Failures (MTBF) and Mean Time To Repair (MTTR).

**Availability** expresses the readiness for correct service. It is the probability that the system functions properly at time $t$:

$$A(t) = P(!failure\ at\ t)) \tag{2.2}$$

and it is often expressed as uptime divided by total time (uptime plus downtime); often, it is computed as the ratio:

$$A = \frac{MTTF}{MTTF + MTTR} = \frac{1}{1 + \frac{MTTR}{MTTF}} \tag{2.3}$$

which shows that for improving availability, it is important to reduce the ratio between MTTR and MTTF, by increasing the mean time to failure and/or by reducing the mean time to repair.

**Safety** is the absence of catastrophic consequences for the users and the environment. **Confidentiality** is an information security property; it is the property of a system to be able of not making available or disclosing or making understandable (protected) information to unauthorized individuals, entities or processes. **Integrity** is the property representing the

absence of improper alterations, may they concern - for instance - system, messages or data. **Maintainability** is the ability of a system to undergo modifications and repairs.

In this dissertation, we do not deal with safety, confidentiality, integrity and maintainability, and in the next Section we will focus on the dependability requirements placed on software-defined networks concerning availability, reliability and scalability.

### 2.2.2 SDN Dependability Requirements

In a software-defined network, although logically centralized, the controller is a physically distributed entity. This is because **dependability requirements** - mainly on scalability, availability and reliability - demand for its engineering in a distributed architecture. If, as claimed, SDN is going to become the technology of future networks, it has to fully address these requirements. Telecommunication operators, for instance, are unlikely to adopt SDN to replace existing carrier-grade networks unless SDN is proved to be able to provide at least the same quality-of-service, while providing greater flexibility and ease of management [16].

**Scalability** is not strictly a dependability attribute in the current classification, yet it is a major concern for SDNs. While certainly it is a fundamental design consideration for SDN controllers, this concern is often

overlooked. It is a common belief that - differently from current networks, where devices are often realized with specialized application-specific integrated circuits (ASICs) - a logically centralized but physically distributed software defined controller may not scale as the network grows. However, this appears to be a wrong belief. As argued by Yeganeh et al [31], "there is no inherent bottleneck to SDN scalability". The issues of scalability in SDN are similar as in any distributed system, and scalability is not inherently harder to achieve than in traditional networks. That is, if a distributed SDN is required to provide a unified network-wide view, solutions need to incorporate distributed consistency protocols. If there exist limits for distributed systems, they are probably those stated in the famous Brewer's Conjecture [32], claiming that it may not be always possible to achieve strong consistency, high availability and partition tolerance all together.

As for **availability**, the requirements on SDN are very stringent; these are not different from those of current networks. If SDN have to be used for basic yet critical services such as telephony, they are required to provide an end-to-end availability of five "nines" (i.e., 99,999%)[2] as in today carrier-grade networks [33]. In current networks, this is achieved at the cost of manual configuration and long deployment times [16]. The flexibility of SDN is very appealing from the point of view of telecommunication

[2]A five "nines" availability amounts to about five minutes of downtime per year.

networks operators, but they are not going to sacrifice availability for maintainability. Achieving such high levels of availability when the network is software defined may be hard and it requires careful design and accurate implementation, but *it is possible*. However, recent work by Akella and Krishnamurthy [34] has shown that availability issues for SDNs systems are more deeply rooted than those stemming from their complexity (see also Sections 2.4 and 3.1.2.2).

**Fault-tolerance** is (along with fault prevention, removal and forecasting) a means to increase the dependability of a system. It is clearly of paramount importance for SDN, since in the event of a controller failure the whole network can be compromised, because all applications and services depend on it. All SDN controllers are engineered with mechanisms to tolerate such events, and clearly fault-tolerance is one of the major techniques used to ensure a high level of resilience. This dissertation proposes failure injection as a testing technique to intentionally introduce failures, representative of failure events which can actually occur at various levels in SDN networks, so as to evaluate the controller's fault-tolerance mechanisms, and more in general to assess the extent to which a controller is able to provide the desired level of resilience.

## 2.3 SDN Resilience

The concept of **resilience** (or resiliency) has multiple definitions, as it has developed in different disciplines, including physics, psychology, ecology, engineering. Very likely, the term originally referred to a property of a physical material or of an entity, but the concept is now applied also to networked systems or organizations. Despite the different definitions, quoting from [35] there are "three elements present across most of them: the ability to change when a force is enacted, [to] perform adequately or minimally while the force is in effect, [to] return to a predefined expected normal state whenever the forces relents or is rendered ineffective".

In engineering, the term somehow intuitively conveys the notion of the ability to survive to unintentional or malicious threats (failures, attacks, etc.) and to resume normal operating conditions; we can say that the *resilience of a system* is often thought as its ability to provide and maintain an acceptable level of performance or service in presence of failures. It is worth to explicitly point out that resilience is a macroscopic-scale property of a system, i.e., a property of the system as a whole [35].

According to [36], computer **network resilience** *is the ability to provide and maintain an acceptable level of service in the face of faults and challenges to normal operation.* In [37] network resilience is defined as *the ability of a network to defend against and maintain an acceptable level of service in*

*the presence of challenges, such as malicious attacks, software and hardware faults, human mistakes (e.g., software and hardware misconfigurations), and large-scale natural disasters threatening its normal operation.*

In [38] the resilience of systems has been evaluated by means of a time-dependent indicator, known as figure-of-merit $F(\bullet)$, which is a quantifiable indicator of the performance of the system. As depicted in Figure 2.4, the state of a system is characterized by a value of $F(\bullet)$, directly affected by the two events (the disruptive event and the corresponding recovery action). Multiple indicators can be defined to provide a measurement of resilience, concerning reliability, network connectivity paths, flows, etc.

In Figure 2.4, the system is initially "functional" at time $t_0$, and this state remains constant until the occurrence of a disruptive event at time $t_e$, bringing the value of the delivery function of the system from its initial value $F(t_0)$, to the lower value $F(t_d)$. Thus, the system is assumed to function in a degraded mode from $t_e$ to $t_d$, when it reaches the point where

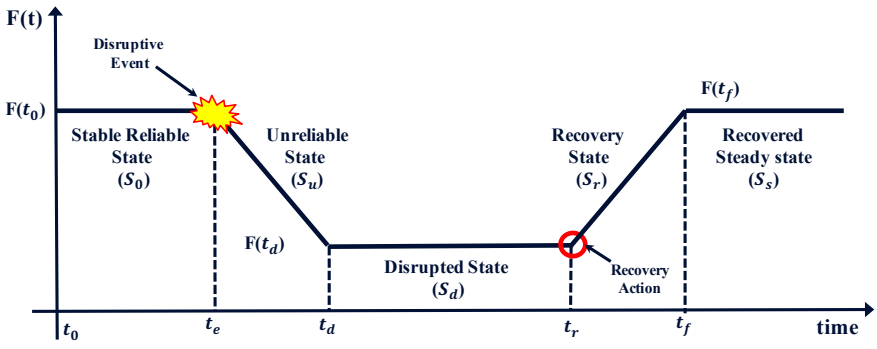



Figure 2.4: System behavior in the occurrence of a disruptive event.

the functionality is considered to be entirely lost. The system remains in such state until a recovery/repair action is initiated at time $t_r$ when the system regains functionality, although in a degraded mode. As a result of the resilience action, the system is considered recovered and fully-functional at time $t_f$, with delivery function value $F(t_f)$. However, the final state, reached by the system after the recovery action, does not necessarily have to coincide with the original state of the system, i.e., the figure-of-merit $F(t_f)$ can be equal, grather, or smaller than $F(t_0)$. Finally, the value of resilience corresponding to a specific figure-of-merit function can be computed as:

$$\Lambda(t) = F(t) - F(t_d)F(t_0) - F(t_d) \tag{2.4}$$

where $0 \leq \Lambda(t) \geq 1$ for $t \in (t_r, t_f)$ assuming that the recovery action succeeds in restoring the functionality.

In recent years, the concept of resilience has been broadened to incorporate a notion of dependability also with respect to changes; indeed, it is increasingly conceived as *the capability of a system to remain dependable in the present of changes*. This probably comes from the work by Laprie [39], who pointed out the need to address the growth of complexity of today so pervasive computing systems, a need deriving from changes which can be *functional*, *environmental* and *technological*. Laprie introduced *scalable resilience* as a concept of "survivability in direct support of the emerging pervasiveness of computing systems" [39].

For the purpose of this thesis, the following definition - essentially provided in [40] - will be used: **Network resilience is the ability to provide and maintain an acceptable level of service in the face of failures.**

Nowadays, resilience is a major requirement and design objective for computer networks. This is due to the fact that computer networks are at the basis of most critical infrastructures, subject to both unintentional faults/failures and to malicious (cyber-)attacks.

## 2.4 Open Challenges

Notwithstanding the large literature on SDNs, there are several still open problems related to the fulfillment of dependability and resilience requirements by SDNs. One of the reasons for this is that - quoting from [40] - "there is almost no practical way to experiment with new protocols in sufficiently realistic settings (e.g. at the scale carrying real traffic) to gain the confidence needed for their widespread deployment".

**Availability** may pose more threats to SDNs than those posed to generic distributed systems. Akella and Krishnamurthy [34] have shown that availability issues for SDNs systems are more deeply rooted than those stemming from the complex and critical inter-dependencies among the various network and distributed systems protocols they use. The authors provide a case for this, which is worth presenting here.

Let us consider the network in Fig. 2.5 (reproduced from [34]), where $C1$ to $C5$ are controller replicas in a distributed control plane, and $S1$ to $S9$ are switches in the data plane. Let us also point out that distributed controllers incorporate consensus protocols such as Paxos [41] to ensure they have a consistent view of network topology and data plane state even in the presence of failures or disconnections. When the links between $S4$ and $S6$ and $S4$ and $S8$ fail, a network partition occurs. In this case, switches in the data plane partition on the right cannot be updated since they can contact only a minority group ($C4$ and $C5$) of the replicated controllers. If the path between $S6$ and $S8$ managed by the control plane prior to the partitioning is not wholly contained in the right partition, $S6$ and $S8$ cannot communicate even if there is a path between them not affected by the failure. Such critical situations undermine high availability. Clearly, legacy network protocols do not suffer from this issue: when partitions happen, routers re-converge to new intra-partition routes. Quoting the authors, the case they provide shows that "current SDN designs fail to provide important fault-tolerance properties, which renders SDNs less available than traditional networks in some situations". While there exist solutions to the problem described (e.g. using partitioned consensus), they may address it only partially; specifically, there may be consistency pitfalls in case of concurrent events. The authors themselves proposed a solution - although they did not prove it - based on

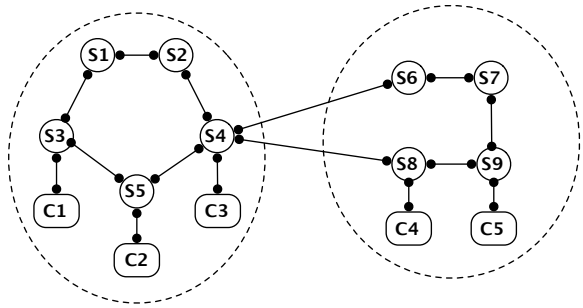



Figure 2.5: A case for availability threat in SDN (Akella, 2014).

a combination of the Chandy-Lamport snapshot algorithm [42], of reliable flooding and of whole quorum consensus. However such mechanisms are expensive, and while they appear to be sufficient, it is not clear if they are all necessary.

Sharma et al. focus on **fault tolerance** and **failure recover** in OpenFlow for deployment it in carrier-grade networks [40], as means to improve SDN resiliency. Carrier-grade networks pose a strict requirement that the network should recover from a failure within a 50 ms interval. They show that OpenFlow may not be able to satisfy this requirement, and they propose a recovery action in the switches without involving the controller. As for **reliability**, they briefly discuss as future work some possible approaches.

Di Martino et al. [16] present the **resiliency challenges** for future carrier-grade networks based on SDN. They build on an analysis of outages of current carrier-grade networks to identify three main factors impacting the effectiveness of their failover mechanisms, namely: *untested operational*

*context*, *multiple failures during the failover window*, and *human/procedural errors*. Based on this, the implications drawn include:

- The need for new resiliency techniques and for new in-production approaches to testing (the authors mention those used in the Chaos Monkey approach by Netflix [22] [43]). This is because SDNs are expected to reduce the service deployment time down to the order of seconds; this in turn may imply reduced testing and assessment.

- The need for new failover mechanisms. This is because many mechanisms are programmed in today's network devices, while in SDNs failover will demand for interaction between various layers and among different domains.

- Service business models driven by Service Level Agreements (SLA) will require richer resiliency specifications and validation. This is because SDN will allow high flexibility, service provisioning will be more dynamic (for instance, due to migration for resources' optimization), and service level will need to be validated dynamically.

Jain et al. [21] present *B4*, a practical example of a large scale implementation of an SDN-based WAN connecting the Google©'s data centers around the globe. The authors provide implementation details about one of the first and largest SDN deployments, which has shown to be efficient in

meeting both performance, and reliability requirements. However, despite the outstanding performance, *B4* experienced an outage due to a human error. Indeed, during a planned maintenance activity two physical switches were configure with the same ID causing substantial *link-flap* errors, i.e. their network interfaces continuously went up and down, and more protocol processing activities to discover the network topology. This has lead to subsequent failures of the Google's public network, affecting the network connectivity of their customers.

Unpredictable events underlines that, as long as a system has been shown to be robust as a result of testing activities, nevertheless it can fail in a **production environment**, where operating conditions change and evolve over time. Moreover, **traditional software-testing tools are inadequate** to verify the resilience of such complex distributed systems against all potential failure scenarios that can span across the whole system stack.

In summary, new approaches are needed for the **automated verification of SDN resilience** while exercised by real workload conditions, namely in production. Innovative approaches would allow to discover failure points otherwise difficult to detect by means merely of software testing, helping to design better detection techniques, and building the right mitigation means to recover the system when real issues arise.

This page intentionally left blank.

Every journey has a starting point.

-

Chapter 3

# Related work

*This Chapter surveys the literature on performance and resilience of software defined networks and on software fault injection (SFI). First, it analyzes existing work on the design of mechanisms for SDN controllers dependability, and on metrics and techniques for the evaluation of their performance, reliability, fault tolerance an resilience. It then introduces SFI, the technique proposed in this thesis for resilience assessment of SDN. Rather than providing a Systematic Literature Review, the goal is: i) to investigate the state of research on performance evaluation of SDNs as well as techniques to ensure controllers' resilience; ii) to provide a short background on SFI and on its application to dependability assessment.*

## 3.1 SDN Performance and Resilience

### 3.1.1 Sources

Despite the fact that Software-Defined Networking is still a relatively young field, the literature is rather large. This is due to the great appeal it has encountered in both the academic and industrial sectors.

From a scientific viewpoint, at its beginning the SDN literature spread

in publication venues traditional of the computer network and software dependability fields, such as the IEEE/IFIP Conference on Dependable Systems and Networks (DSN), the IEEE Symposium on Software Reliability Engineering (ISSRE), and all major journals and conferences on computer networks, and on systems/software dependability. Currently, specific conferences, conference series and journal special issues on SDN are really proliferating. The main ones include:

- The ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (**HotSDN**); the series started in 2012 in Helsinki (FN);
- The IEEE Conference on Network Softwarization (**NetSoft**), whose first edition was held in London (UK) in April 2015;
- The IEEE Conference on Network Function Virtualization and Software Defined Networking (**NFV-SDN**) (from 2015, and in 2017 the SDN-NFV Track in SAC Symposium at the IEEE International Conference on Communication (ICC 2017);
- The Research Track of the Open Networking Summit (**ONS**, since 2014 in conjunction with USENIX, the Advanced Computing Systems Association);
- The IEEE/IFIP Network Operations and Management Symposium (**NOMS**);
- The First International Workshop on Software Defined Networks and Network Function Virtualization (**SDN-NFV**), in conjunction with The Fourth International Conference on Software Defined Systems (SDS-2017);
- The European Workshop on Software Defined Networking (**EWSDN**), whose first edition was held in Darmstadt (D) in April 2012;
- The Special Issue on "Software Defined Networking", September 2015, and the upcoming Special issue on "SDN and NFV based 5G Heterogeneous Networks" of **IET Networks**;

- The Special Issue on "Future network: software-defined networking" of Frontiers of Information Technology & Electronic Engineering, July 2016;
- The Special Issue on "Management of Softwarized Networks", September 2016, and the (upcoming) Special Issue on "Advances in Management of Softwarized Networks" of IEEE Transactions on Network and Service Management (**TNSM**);
- The Special Issue on "Software-Defined Networking and Network Functions Virtualization for flexible network management" of the International Journal on Network Management (**IJNM**), November 2016;
- The Special Issue on "Management of SDN/NFV-based Systems" of the International Journal on Network Management (**IJNM**).
- The (upcoming) Special Section on "Network Virtualization, Network Softwarization, and Fusion Platform of Computing and Networking" of IEICE Transactions on Communications (**IEICE TC**);

The existing literature on the evaluation of performance, reliability and fault tolerance of SDN controllers has been searched in the above sources in the networking and dependability research fields, and more in general in all major scientific computer science and engineering databases, including IEEExplore, ACM Digital Library, Scopus, ISI Web of Science, Google Scholar. It is described in the following subsection.

### 3.1.2 Related Work

Despite the bulk of work in the scientific literature, the research on SDN performance and dependability can be considered still at the beginning.

Clearly, the decoupling of control plane from the network devices and the implementation of controllers as distributed systems inevitably make software-defined networks inherit the weaknesses associated with reliable distributed systems [34]. As recently stated in [16], "*the specific benefits and risks that SDN may bring to the resilience of carrier-grade networks remain largely unexplored*", and according to [44] "*the dependability of SDN itself is still an open issue*". The related work can be broadly categorized in *SDN benchmarking* (i.e., the evaluation of performance and scalability of SDN platforms), and *SDN dependability* (i.e., the mechanisms to ensure desired dependability features by proper design choices).

#### 3.1.2.1 SDN Benchmarking

**SDN performance benchmarking** is addressed in several studies [45] [46] [47] [48].

Cbench [45] is a benchmarking tool of this type based on simulating a configurable number of OpenFlow switches; it used in [46] to compute various controller performance metrics (response time, throughput, latency) of four controllers (NOX, NOX-MT, Beacon, and Maestro) based on OpenFlow, probably the most widely known southbound API [11].

Jarschel et al. [47] proposed a flexible OpenFlow controller benchmarking tool (OFCBench) to overcome various limitation of Cbench and previous tool (single threading, use of one controller connection for all emulated switches). They also proposed an analytic (queueing) model for predicting the performance of an OpenFlow architecture, in terms of packet sojourn time and probability of lost packets [49].

Cbench is also used by Zhao et al. [48], who presented an evaluation of five open-source controllers. However, it considers mainly centralized controllers; distributed controllers, which are gaining momentum also for addressing scalability, have more complex performance problems, ranging from the placement of the replicas to their synchronization.

Prior to [48], Tootoonchian et al. [46] presented a study of four publicly-available OpenFlow controllers, based on their Cbench tool [45]. This tool measures the number of flow setups per second that a controller can handle, and it supports two modes of operation: latency and throughput mode. A further experience including the evaluation of performance in a wide-area SDN is described in [21].

A crucial performance aspect for SDNs is **latency**. As pointed out in [50], critical SDN mechanisms such as *fast failover* and *traffic engineering* demand for the ability to program the data plane state at fine time-scales. In their study, He et al. state that "*timeliness is determined by: (1) the speed of*

*control programs, (2) the latency to/from the logically central controller, and (3) the responsiveness of network switches in interacting with the controller* [50]. While the first two factors are already being overcome by advances in distributed controllers, the authors' measurements show that the third one may be critical even with most modern switches. The *inbound* and *outbound* latencies (those concerning events generated by switches and those in the execution of rules provided by the controller, respectively) are high and variable. The study thus highlights the need for "*careful design of future switch silicon and software in order to fully utilize the power of SDN*".

**Scalability metrics** for the the SDN control plane are proposed in [51] and [52]. The first metric specific for SDNs appears to be the one by Hu et al. [51], who propose to compute scalability when the network scale varies from N2 to N1 as:

$$Scalability \quad \Psi(N1, N2) = \frac{\phi(N2)\frac{T(N2)}{C(N2)}}{\phi(N1)\frac{T(N1)}{C(N1)}}, \tag{3.1}$$

where $\phi(N)$ is the throughput of the control plane in processing network requests, $T(N)$ is the average response time per request, and $C(N)$ is the cost to deploy the control plane. The authors evaluate the metric with reference to three SDN control plane architectures – centralized, distributed and hierarchical - by building performance models for the response time, based on which they evaluate the scalability of the three structures.

While the previous metric considers throughput, average response time

and cost, the metric proposed in [52] does not consider the deployment cost, and assumes that the average flow processing time has to remain the same when scaling. The metric they propose is:

$$Scalability \quad f(W, O) = \frac{W}{O}, \tag{3.2}$$

where $W$ and $O$ are the workload and overhead, respectively; the former is the number of flows entering the data plane, the latter is the number of messages processed by the controller(s).

#### 3.1.2.2 SDN Resilience and Dependability

Several research groups have coped with the problem of defining proper mechanisms to ensure resilience and dependability for SDN controllers.

Heller et al. [53] have formulated the **Controller Placement Problem**, the one of deciding - given a network topology - how many controllers need to be used and where to place them to satisfy performance and fault tolerance requirements. They are concerned specifically with wide-area networks and with the minimization of propagation delays. Their study shows that the answers to the two questions depend on the topology and on the metric (a tradeoff has to be found between optimizing for worst-case or average-case latency), that for most topologies adding controllers provides almost proportional delays' reduction, but surprisingly, *in medium-size networks*

*one controller location can be sufficient to meet current typical real-time requirements*. Clearly, one is not enough for fault tolerance.

**SDN fault tolerance** is addressed by Fonseca et al. [54], who propose a primary-backup mechanism to provide resilience against several types of failures in a centralized OpenFlow-based controlled network. In *primary-backup replication* [55], one or more secondary (backup) replica servers are kept consistent with the state of the primary server, and as the primary server enters in a failure state, one (warm) backup replica is chosen to replace the primary server. Hence, this approach is well suited where there is a centralized control concentrating in one point of the network the information that need to be replicated. The approach has been implemented in the NOX controller, and has been shown to work in several failure scenarios, namely abrupt abort of the controller, failure of a management application (client running atop the controller), Distributed Denial-of-Service (DDoS) attack. The authors conclude that the OpenFlow protocol proved to be appropriate to support ease implementation of primary-backup replication.

Ross et al. [56] build on the previous work by Heller et al. formulating the **Fault Tolerant Controller Placement Problem**, as the problem of deciding how many controllers are needed, where they have to be deployed, and what network devices are under control of each of them, in order to

achieve at least five nines reliability at the southbound interface (the typical reliability level required by carrier-grade networks). They also propose a heuristic to compute placements with such reliability level. Again, the answers depend on the topology (rather than on the network size). However, *it is possible to achieve fault tolerance in SDN by careful selection of the placement of controllers.*

**SDN availability** design issues are addressed by Akella and Krishnamurthy [34]. They show that there may be situations where link failures can compromise the proper functioning of portions of a SDN. This is due to the fact that controller internal modules - specifically, distributed consensus protocols, mechanisms for switch-controller or controller-controller communication, and transport protocols for reliable message exchange - can have cyclical dependencies. This means that link failures can cause transient disconnections between switches and controllers or controller instances, which in turn undermine high availability. What appears to be particularly critical in SDN is the lack of robustness to failures which partition the topology of controllers. In fact, it has to be noted that - since in SDN the control has been taken out of switches and logically centralized in the control plane - it may happen for two switches to be unable to communicate even if a physical path between them does exist. The authors argue that current SDNs may be unable to offer high availability, and they should be re-architected by

including advanced mechanisms from the distributed systems theory, such as reliable flooding and global snapshots.

## 3.2 Fault Injection Testing

**Fault injection** [57] is the technique of introducing faults in a system to assess its behavior and to measure the effectiveness of fault tolerance or other resilience mechanisms. Although much more recent than hardware fault injection, **Software Fault Injection** (SFI) is today widely used too [23]. SFI developed as it became clear that software faults were becoming a major cause of systems' failures. It proved to be effective for fault-tolerance and reliability assessment for several classes of systems, such as distributed systems, operating systems, Data Base Management Systems, and it is nowadays recommended by several standards in critical systems domains [23].

SFI consists of applying small changes in a target program code, in a way similar to *mutation testing* (a well-known software testing technique), with the goal of assessing the system behavior in the presence of (injected) faults, which clearly have to be representative of potentially real faults - or, specifically, of *residual faults*, those which escape testing and debugging before software product release and may be activated in execution on-field [58]. In a SFI experiment, a fault is injected in the program, which is executed under a workload, in turn representative of real operating conditions.

Fault injection has been proposed by Cotroneo et al. [59] for dependability evaluation and benchmarking of Network Function Virtualization Infrastructures (NFVIs). NFV is a field closely related yet different from SDN. SDN and NFV share the goal of fostering innovation in networking by means of a shift to software-based platforms, that it to network programmability. NFV refers to the virtualization of specific in-network functions (e.g., firewalls and VPN gateways) in order to reduce the dependency on underlying hardware; this eases resource management, provides faster service enablement and lowers OPEX (Operating Expenditures) and CAPEX (Capital Expenditures). NFV solutions can operate in SDNs. While SDN is more concerned with control plane programmability, NFV mainly focuses on data plane programmability. As for SDNs, NFV inherits performance and reliability requirements from telecommunication systems. In their paper, the authors define some key performance indicators for Virtualized Network Functions (VNF), namely latency, throughput and *experimental availability*, and propose fault injection for evaluation and benchmarking of VNFs.

Differently from the *fault* injection approach pursued in [59] for NFV, we propose the use of *failure* injection for the assessment of the resilience mechanisms SDN distributed controllers. The aim it to devise a methodology suited for **in-production assessment**. In-production testing is gaining

importance in all those dynamic contexts where traditional software testing techniques *in fabric* (i.e. before deployment) are not deemed to be suited anymore: one famous such case is represented by Netflix [22]. In the framework of this dissertation, it means injecting failures at system, network or service level during executions under a workload (which is not emulated but real, in case of in-production testing), failures which have to be representative of events typically occurring in normal operation. The proposed failure injection methodology is described in the next Chapter.

## 3.3 Failure Injection Testing: the Netflix approach

In defining a methodology to assess the resilience and the failover mechanisms of SDN platforms we take inspiration from the failure injection approach proposed by *Netflix*®. It is a multinational company providing streaming and on demanded multimedia services to a wide range of users around the world. In doing so, they engineered a very complex ecosystem according to a "micro-services" architecture pattern, i.e. with multiple small and independent services working together to fulfill a specific goal. This leads to a dynamic operational context, where services are updated or added at runtime without ever interrupting the system, making it impractical, or even impossible, to perform testing activities aimed to assess possible system's deficiencies and identify potential failure modes.

Consequently, a methodology has been proposed to find possible weaknesses in a **production system** by observing its behaviour under the deliberate injection of failures. The execution of failure injection experiments within a live production environment has three main advantages:

- It allows a better assessment of the system by verifying its correct behaviour in realistic production deployment and load conditions;
- It helps making the the system immune to possible failures;
- It helps preventing outages that can affect system availability.

The methodology falls under the umbrella of the broader concept of "Chaos Engineering" [22], [60], which is defined as the "discipline of experimenting on a distributed system in order to build confidence in the system's capability to withstand faulty conditions in production". Specifically, this discipline provides few practical principles meant to facilitate the testing activities to uncover system weaknesses, namely:

- Definition of what is a normal system's behaviour, i.e. the system "*steady state*", considering some measurable output of the system;
- Build a control system and an experimental one, with the latter used for the failure injection experiment;
- Introduce disruptions on the experimental system to simulate real-world events, such as server crashes, network failures etc.;

- Compare the steady states of the experimental and control systems to find possible deviations from the normal behaviour and build confidence on system resilience.

Along with these chaos principles, Netflix proposes a *Failure Injection Testing* (FIT) platform to automate [61] [62] the injection and monitoring of arbitrary failures scenarios into specific targeted services or system subset, aiming to support the implementation of systems that are resilient to failure.

Though inspired by the Netflix's approach, the failure injection assessment methodology proposed in this dissertation differs in several aspects:

(i) It targets distributed SDN platform to perform a resilience assessment under failure scenarios, aiming to verify if such systems provide suitable failover mechanisms;

(ii) The framework reproduces failure scenarios which are representative for SDN ecosystems, e.g. faulty communications between SDN controllers, or a faulty controller's service;

(iii) It is meant to perform both *offline*, and *in-production* assessment, since the SDN technologies are still in very early stages to be deployed in a real production environment;

(iv) It provides measurements which give valuable insights into the performance and resilience of the SDNs.

What cannot be assessed
cannot be claimed.

-

Chapter 4

# SDN Control Plane CLoUd-based Benchmarking

*This chapter presents SCP-CLUB, a benchmarking framework for performance analysis of a telco cloud-based SDN control plane. First, an overview of the generic framework and its tool suite is given. Then, a detailed description is provided of the design and implementation of an ONOS-based instance of the framework. Afterwords, an experimental campaign is presented, showing the effectiveness of the proposed framework in automating very long sessions of experiments for benchmarking a telco cloud SDN. The last section presents and discusses the experimental results.*

## 4.1 Introduction

Telco Cloud is an emerging paradigm in the engineering of telecommunication infrastructures and services, which promises to make their management much more agile [12] [15]. Telecommunication (telco) operators are expected not only to be more and more software driven - with the adoption of softwarization technologies, such as SDN and NFV, in replacing the dedicated telco hardware boxes - but also to increasingly adopt a cloud computing approach to automate the management of their infrastructures and services.

The combined use of these softwarization technologies [1] [63], cloud and virtualization technologies has the potential to support in meeting the requirements of future networks in terms of elasticity, scalability, and resiliency to mutable network conditions, unpredictable traffic patterns, and continuous streams of services. However, it is still not clear for telco operators how these technologies can be best leveraged to meet such highly demanding requirements with carrier-grade service-level guarantees [16].

In a virtualized and cloud-based operational context for SDNs, there are several software layers and thousands of different setups, configurations, and parameters to be properly fine tune in order to obtain specific levels of service. For instance, Telco Cloud Orchestrators (such as XAAS, EC2 and VCLOUD) need automated tools and procedures to decide when to scale up or down the resources running the telco-cloud control (e.g., virtual machines and SDN controller instances). The ability of automating the performance analysis of these emerging technologies, combined to the cloud-based infrastructures is fundamental in the telco cloud scenario. No proper techniques and tools are available so far to address this task.

This chapter presents **SCP-CLUB**, an automated benchmarking framework for performance analysis of a telco cloud-based SDN control plane. SCP-CLUB provides the following features:

- a configurable load generator for SDNs benchmarking;

- cloud automation and orchestration tools to enable parallel benchmarking experiments for computing relevant performance assessment metrics using idle cycles of a telco cloud.

The load generator is based on the **Intent-Based Networking** (IBN) approach (see §2.1.4). Cloud automation supports performance evaluators in the many tasks of orchestrating and running benchmarking experiments and collecting data at various software levels for the analysis, with the ultimate aim of extracting actionable intelligence that can be used to manage (e.g., scale up or down) the cloud resources running the control network.

## 4.2 State-of-the-art Progress

As we have seen in Section 3.1.2.1, several authors have dealt with the problem of benchmarking SDN controllers. However, the literature focuses essentially on measuring the intrinsic performance of single controller instances with faked interactions with switches, or the performance at the southbound interface.

Probably the most widely used tool is Cbench [45], and the most comprehensive evaluation of the major open-source SDN controllers is carried out by Zhao et al. [48]. Their focus is on benchmarking one controller managing a variable number of switches. They consider Cbench suited to provide accurate results, in that it fakes switches just as "kind of traffic

generators able to send `packet_in` messages as fast as possible". Moreover, measurements are taken sending traffic through the local loopback interface, to eliminate also the link bottleneck.

Rather than focusing on the performance of single instances of controllers without significant requests from applications and with faked interactions with switches, SCP-CLUB takes a different perspective. It considers really distributed controllers in a cloud-based and virtualized deployment; by generating and submitting synthetic yet realistic work loads in terms of *intents* to be actually processed, it allows to investigate the performance of the control plane related to the whole workflow of installing and translating application requests, up to sending commands with flow rules to the data plane and checking their successful processing by the controlled switches. To this aim, both intent installation and withdraw requests are generated and submitted to the control plane at the northbound interface. SCP-CLUB allows also to evaluate the impact on performance of the main configuration parameters, so as to derive hints for their proper configurations.

SCP-CLUB has some similarities with the approach used in the Performance and Scale-out Test Plan [64] designed to characterize ONOS latencies, throughput and capacities (again with Cbench): both use a load generator producing self-adjusting intent install and withdraw requests, and both make measurements as the number of controllers instances in a cluster

scales up. There are however substantial differences.

First, the ONOS performance test plan and results described in [64] are based on "a set of Null Device Providers at the adapter level to interact with the ONOS core. The Null Providers act as device, link, host producers as well as a sink of flow rules". Using such dummy stubs, the ONOS performance figures typically reported in the literature intentionally disregard Openflow adapters and interactions with real or emulated switches. From the point of view of telco operators, this means that such figures can not be trusted to assess the suitability of a controller for use in a carrier-grade SDN. Moreover, the ONOS performance metrics available in the literature are made generating a load to the highest rate ONOS can sustain with synchronous requests. SCP-CLUB is designed to assess actual controller performance with concrete intent requests, complete processing of requests by the controller (intent compilation, installation and removal), and real or emulated topologies considering failures in the data plane (which indeed occur in reality). Finally, SCP-CLUB is based on cloud and virtualization technologies, and it is designed to analyze performance figures such as throughput and latency in real industrial SDN cloud testing datacenters.

The next section introduces the generic SCP-CLUB framework, while Section 4.4 presents an implementation based on the ONOS open source SDN controller [10]. Section 4.5 shows the results of experiments with a

This page intentionally left blank.

No amount of experiments can prove that I am right; a single experiment can prove me wrong.

*Albert Einstein*

Chapter 5

# SDN Resilience Assessment: a Failure Injection Tool Suite

*The Chapter describes the failure injection methodology and a framework for continuous assessment of the reliability and resilience of SDN technologies. To this aim, the already presented SCP-CLUB framework is extended with a configurable and distributed software infrastructure for failure injection. The Chapter then describes the steps of the methodology, which encompasses the definition of a workload to bring the SDN platform under assessment in a state where to inject failures according to the failure model; the workload is based on the Intent-Based Networking model. A failure model is presented, describing the variety of injectable failure types at system, network and controller level. Then the Chapter describes the logical architecture and components of the distributed software, along with its implementation details. Finally, the Chapter terminates with the experimental campaign, and results, aimed to evaluate the resiliency of the Open Network Operating System (ONOS).*

## 5.1 Assessment Methodology

### 5.1.1 Overview

The knowledge on how failures may affect software systems is of paramount importance to improve their resilience and reliability. With the complexity of modern distributed systems, the design of effective detection and

mitigation mechanisms can no longer rely exclusively on software testing techniques in controlled environments. Indeed, it is impractical to fully reproduce a complex operational context. In SDNs, for instance, it has been shown that a faulty SDN application can compromise or crash the whole SDN network [86]: while SDN controllers software is likely to stabilize, even the application plane may be a vehicle for dependability threats. Therefore, in the engineering of software network services, it is a key goal to be able to test the proper functioning not only in controlled environments, but also in-production, under real operating conditions.

Figure 5.1 depicts a high-level view of the approach, where failure injection is exploited to **continuously** assess the reliability and resilience of the network services against a wide range of failure scenarios. This will provide continuous feedback on the capabilities of the softwarized network services to survive failures, which is of fundamental importance for improving the system internal mechanisms to react to anomalous situations potentially occurring in operation, while its services are regularly updated or improved. To this end, failures are injected in different layers of the telco cloud infrastructure (Figure 5.1), namely: *(i)* at **data-plane** level, to emulate faulty network appliances, e.g. by injecting Bit Error Rate (BER) or packet latency and corruption at switchs' port level; *(ii)* at **infrastructure** level, to emulate faulty physical nodes or virtualized hosts; and *(iii)* at **control**

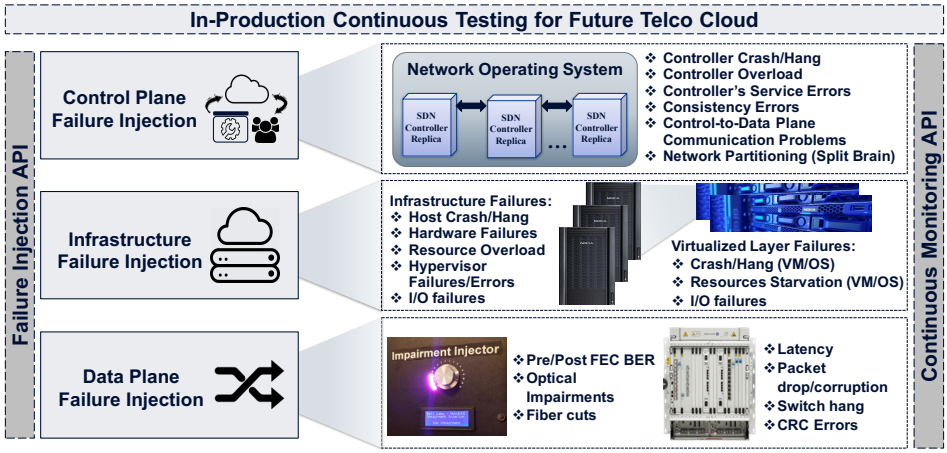



Figure 5.1: In-production continuous testing in Telco Cloud.

**plane** level, to emulate faulty network controllers.

With this approach in mind, the chapter's aim is pursued through the use of software failure injection to deliberately introduce failures in the components of the system under assessment, or in their execution environment, under real or emulated load scenarios, to evaluate the system behaviour under (possibly unforeseen) disruptive conditions. Specifically, it focuses on the resilience of the control plane layer, and proposes a methodology and a tool suite to validate the reliability and resilience of distributed SDN platforms.

### 5.1.2 Failure Injection Methodology

The proposed methodology aims to assess the effectiveness of the failure detection and mitigation mechanisms provided by the SDN technology by

That's all folks.

*Popular TV cartoons closing*

Chapter 6

# Conclusions

*We summarize the research problem addressed and the main contributions provided by this dissertation.*

In the era of the highly and always connected information society, Software-Defined Networking is a very hot research area, and SDN are foreseen to have a huge potential market in the forthcoming years. Among the several open issues, this dissertation has identified two still open research problems: *i)* assessment of SDN performance and *ii)* in production assessment of SDN failure resilience, with reference to real industrial telco cloud data centers. Their relevance is due to the fact that network operating systems (SDN controllers) are very complex distributed systems - subject to performance and dependability requirements as severe as those of current carrier-grade networks - for which current experimental evaluations cannot be trusted by telco operators, and traditional software testing techniques appear insufficient for dependability assessment, given the difficulty to reproduce the

many potential failures which can actually occur in operation and may affect at all levels their many software components.

The thesis has contributed to these research issues with a twofold proposal. The presented SCP-CLUB framework automates experimental campaigns for SDN performance evaluation in real telco cloud data centers. As for the assessment of SDN failure resilience, the proposal extends SCP-CLUB with the use of software failure injection. A methodology has been devised for *in-fabric* test, as well as for *in-production* assessment, with the aim of continuous testing. The methodology in complemented by an infrastructure specifically designed to be integrated with limited intrusiveness into distributed SDN controllers, in order to support the execution of failure-injection experiments.

The resilience assessment methodology and the infrastructure are conceived to be usable in a controlled test environment, as well as in a normal operational environment for future SDN platforms (including virtualized and container-based SDN controllers). For in-fabric test, a workload consisting of *intents* may be generated and then submitted to the platform. Failures are injected during the execution, which is monitored so as to gather data of interest for analysis. Failures belong to a failure model representative of typical classes of events occurring in operation at network, system and service level. For in-production assessment, failures are injected when

the controller under assessment is subject to normal operating conditions (under the real load). A set of metrics to evaluate controllers' resilience to failures have been proposed, too.

The proposed SCP-CLUB framework and the failure injection infrastructure have been implemented and experimented with reference to the open-source ONOS™ network operating system. The workload is automatically generated based on the Intent-Based Networking (IBN) model. The implementation is based on the Linux, Java, Apache ActiveMQ and Apache Karaf technologies.

The experimental evaluation has shown that the framework can be effectively applied to assess the performance of SDN deployments in telco clouds, and the controllers' resilience mechanisms (for failure detection and mitigation), as well as to quantify system availability and reliability. The experiments have been performed at the prestigious Murray Hill NOKIA Bell Labs in New Providence, New Jersey, USA, in the framework of a continuous and in-production testing strategy for the future generation of network solutions.

This page intentionally left blank.

# Bibliography


[1] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[2] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *IEEE Communications Surveys Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[3] E. Hernandez-Valencia, S. Izzo, and B. Polonsky, "How Will NFV/SDN Transform Service Provider OpEx?," *IEEE Network*, vol. 29, no. 3, pp. 60–67, 2015.

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

[5] Allied Market Research, "Global Software-Defined Networking Market: Opportunities and Forecasts, 2015 - 2022." `www.alliedmarketresearch.com/software-defined-networking-market`, 2016. [Online; accessed 21-September-2017].

[6] S. Russo, "Finding a way in the Model Driven jungle," in *Proceedings of the 9th India Software Engineering Conference (ISEC)*, pp. 13–15, ACM, 2016.

[7] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 13, pp. 105–110, 2008.

[8] D. Erickson, "The Beacon OpenFlow Controller," in *Proceeding of the 2nd Workshop on Hot Topics in Software Defined Networking (HotSDN)*, pp. 13–18, ACM, 2013.

[9] J. Medved, R. Varga, A. Tkacik, and K. Gray, "OpenDaylight: Towards a Model-Driven SDN Controller architecture," in *Proceedings of IEEE 15th International Symposium on a World of Wireless, Mobile and Multimedia Networks*, IEEE, 2014.

[10] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. O. Bob Lantz, P. Radoslavov, W. Snow, and G. Parulkar, "ONOS: towards an open, distributed SDN OS," in *Proceedings of the 3rd Workshop on Hot Topics in Software Defined Networking (HotSDN)*, pp. 1–6, ACM, 2014.

[11] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus Networks," *SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[12] P. Bosch, A. Duminuco, F. Pianese, and T. L. Wood, "Telco Clouds and Virtual Telco: Consolidation, Convergence, and Beyond," in *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 982–988, IEEE, 2011.

[13] J. Soares, C. Gonçalves, B. Parreira, P. Tavares, J. Carapinha, J. P. Barraca, R. L. Aguiar, and S. Sargento, "Toward a telco cloud environment for service functions," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 98–106, 2015.

[14] X. Zhiqun, C. Duan, H. Zhiyuan, and S. Qunying, "Emerging of telco cloud," *China Communications*, vol. 10, no. 6, pp. 79–85, 2013.

[15] J. Soares, C. Gonçalves, B. Parreira, P. Tavares, J. Carapinha, J. P. Barraca, R. L. Aguiar, and S. Sargento, "Toward a telco cloud environment for service functions," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 98–106, 2015.

[16] C. Di Martino, V. Mendiratta, and M. Thottan, "Resiliency Challenges in Accelerating Carrier-Grade Networks with SDN," in *Proceedings of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pp. 242–245, IEEE, 2016.

[17] J.-C. Laprie, "From dependability to resilience," in *38th IEEE/IFIP Int. Conf. On Dependable Systems and Networks*, pp. G8–G9, Citeseer, 2008.

[18] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[19] A. S. Team *et al.*, "Amazon s3 availability event: July 20, 2008," *Retrieved November*, vol. 15, p. 2008, 2008.

[20] E. Bauer and R. Adams, *Reliability and availability of cloud computing*. John Wiley & Sons, 2012.

[21] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, and M. Zhu, "B4: Experience with a globally-deployed software defined WAN," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 3–14, 2013.

[22] A. Basiri, N. Behnam, R. de Rooij, L. Hochstein, L. Kosewski, J. Reynolds, and C. Rosenthal, "Chaos Engineering," *IEEE Software*, vol. 33, no. 3, pp. 35–41, 2016.

[23] R. Natella, D. Cotroneo, and H. S. Madeira, "Assessing Dependability with Software Fault Injection: A Survey," *ACM Computing Surveys*, vol. 48, no. 3, pp. 44:1–44:55, 2016.

[24] ONOS Project, "ONOS White Paper." `http://onosproject.org/wp-content/uploads/2014/11/Whitepaper-ONOS-final.pdf`, 2014. [Online; accessed 20-September-2017].

[25] ONOS Project, "Intent-based framework." `http://wiki.onosproject.org/display/ONOS/Intent+Framework`. [Online; accessed 20-September-2017].

[26] H. Farhady, H. Lee, and A. Nakao, "Software-defined networking: A survey," *Computer Networks*, vol. 81, pp. 79–95, 2015.

[27] The Open Networking Foundation (ONF). `http://onlab.us`. [Online; accessed 20-September-2017].

[28] D. Lenrow, "Intent As The Common Interface to Network Resources." Presentation at the Intent Based Network Summit 2015, Palo Alto, CA, USA. Available at `www.ietf.org/mail-archive/web/i2nsf/current/pdfEhAfL7kT9F.pdf(AccessedJanuary2017)`, 2015.

[29] Open Networking Foundation, OpenFlow. `www.opennetworking.org/sdn-resources/openflow`. [Online; accessed January 2017].

[30] ONOS Project, "Onos mission." `http://onosproject.org/mission/`. [Online; accessed 20-September-2017].

[31] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On Scalability of Software-Defined Networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 136–141, 2013.

[32] S. Gilbert and N. Lynch, "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services," *ACM SIGACT News*, vol. 33, no. 2, 2002.

[33] C. R. Johnson, Y. Kogan, Y. Levy, F. Saheban, and P. Tarapore, "Voip Reliability: A Service Provider's Perspective," *IEEE Communications Magazine*, vol. 42, no. 7, pp. 48–54, 2004.

[34] A. Akella and A. Krishnamurthy, "A Highly Available Software Defined Fabric," in *Proceedings of the 13th ACM Workshop on Hot Topics in Networks (HotNets)*, pp. 1–7, ACM, 2014.

[35] J.-F. Castet and J. H. Saleh, "Survivability and Resiliency of Spacecraft and Space-Based Networks: a Framework for Characterization and Analysis," in *AIAA SPACE Conference*, American Institute of Aeronautics and Astronautics, 2008.

[36] The ResiliNets Initiative, "ResiliNets Wiki." `https://wiki.ittc.ku.edu/resilinets_wiki/index.php/Definitions#Resilience`. [Online; accessed 20-September-2017].

[37] P. Smith, D. Hutchison, J. P. Sterbenz, M. Schöller, A. Fessi, M. Karaliopoulos, C. Lac, and B. Plattner, "Network resilience: a systematic approach," *IEEE Communications Magazine*, vol. 49, no. 7, pp. 88–97, 2011.

[38] D. Henry and J. E. Ramirez-Marquez, "Generic metrics and quantitative approaches for system resilience as a function of time," *Reliability Engineering & System Safety*, vol. 99, pp. 114–122, 2012.

[39] J.-C. Laprie, "Resilience for the scalability of dependability," in *Proceedings of the 4th IEEE International Symposium on Network Computing and Applications (NCA)*, pp. 5–6, IEEE, 2005.

[40] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "OpenFlow: Meeting carrier-grade recovery requirements," *Computer Communications*, vol. 36, no. 6, pp. 656–665, 2013.

[41] L. Lamport, "Paxos made simple," *ACM SIGACT News*, vol. 32, no. 4, pp. 18–25, 2001.

[42] K. M. Chandy and L. Lamport, "Distributed snapshots: Determining global states of distributed systems," *ACM Transactions on Computer Systems*, vol. 3, no. 1, pp. 63–75, 1985.

[43] The Netflix Tech Blog, "Netflix Chaos Monkey 2.0." `http://techblog.netflix.com/2016/10/netflix-chaos-monkey-upgraded.html`, 2017. [Online; accessed January 2017].

[44] S. Huang, Z. Deng, and S. Fu, "Quantifying entity criticality for fault impact analysis and dependability enhancement in software-defined networks," in *Proceedings of the IEEE 35th International Performance Computing and Communications Conference (IPCCC)*, pp. 1–8, IEEE, 2016.

[45] A. Tootoonchian et al., "Cbench: an Open-Flow Controller Benchmarker." `https://github.com/mininet/oflops/tree/master/cbench`.

[46] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *Proceedings of the 2nd USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*, USENIX Association, 2012.

[47] M. Jarschel, F. Lehrieder, Z. Magyari, and R. Pries, "A flexible OpenFlow-controller benchmark," in *Proceedings of the European Workshop on Software Defined Networking (EWSDN)*, pp. 48–53, IEEE, 2012.

[48] Y. Zhao, L. Iannone, and M. Riguidel, "On the performance of SDN controllers: A Reality Check," in *Proceedings of the IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, pp. 79–85, IEEE, 2015.

[49] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, "Modeling and Performance Evaluation of an OpenFlow Architecture," in *Proc. of the 23rd Int. Teletraffic Congress (ITC)*, IEEE, 2011.

[50] K. He, J. Khalid, S. Das, A. Gember-Jacobson, C. Prakash, A. Akella, L. E. Li, and M. Thottan, "Latency in Software Defined Networks: Measurements and Mitigation Techniques," *ACM SIGMETRICS Performance Evaluation Review*, vol. 43, no. 1, pp. 435–436, 2015.

[51] J. Hu, C. Lin, X. Li, and J. Huang, "Scalability of control planes for software defined networks: Modeling and evaluation," in *Proceedings of the IEEE 22nd International Symposium of Quality of Service (IWQoS)*, pp. 147–152, IEEE, 2014.

[52] M. Karakus and A. Durresi, "A Scalability Metric for Control Planes in Software Defined Networks (SDNs)," in *Proceedings of the IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*, pp. 282–289, IEEE, 2016.

[53] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proceedings of the 1st Workshop on Hot Topics in Software Defined Networking (HotSDN)*, pp. 7–12, ACM, 2012.

[54] P. Fonseca, R. Bennesby, and E. Mota, "A Replication Component for Resilient OpenFlow-based Networking," in *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS)*, IEEE, 2012.

[55] N. Budhiraja, K. Marzullo, F. B. Schneider, and S. Toueg, *Distribyted systems*, ch. The primary-backup approach, pp. 199–216. ACM Press/Addison-Wesley Publishing Co., 2nd ed., 1993.

[56] F. J. Ros and P. M. Ruiz, "Five Nines of Southbound Reliability in Software-Defined Networks," in *Proceedings of the 3rd Workshop on Hot Topics in Software Defined Networking (HotSDN)*, pp. 31–36, ACM, 2014.

[57] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.-C. Fabre, J.-C. Laprie, E. Martins, and D. Powell, "Fault Injection for Dependability Validation: A Methodology and Some Applications," *IEEE Transactions on Software Engineering*, vol. 16, no. 2, pp. 166–182, 1990.

[58] R. Natella, D. Cotroneo, J. Duraes, and H. Madeira, "On Fault Representativeness of Software Fault Injection," *IEEE Transactions on Software Engineering*, vol. 39, no. 1, pp. 80–96, 2013.

[59] D. Cotroneo, L. D. Simone, A. K. Iannillo, A. Lanzaro, and R. Natella, "Dependability Evaluation and Benchmarking of Network Function Virtualization Infrastructures," in *Proceedings of the 1st IEEE Conference on Network Softwarization (NetSoft)*, pp. 1–9, IEEE, 2015.

[60] The Netflix Tech Blog, "Principles of chaos engineering." `http://principlesofchaos.org/`, 2015. [Online; accessed 31-January-2017].

[61] The Netflix Tech Blog, "Fit: Failure injection testing." `http://techblog.netflix.com/2014/10/fit-failure-injection-testing.html`, 2015. [Online; accessed 31-January-2017].

[62] P. Alvaro, K. Andrus, C. Sanden, C. Rosenthal, A. Basiri, and L. Hochstein, "Automating failure testing research at internet scale," in *Proceedings of the Seventh ACM Symposium on Cloud Computing*, pp. 17–28, ACM, 2016.

[63] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, 2015.

[64] ONOS Project, "ONOS Test Plan - Performance and Scale-out." `https://wiki.onosproject.org/pages/viewpage.action?pageId=3441823(accessedJuly2017)`, February 2017.

[65] Apache Software Foundation, Apache Karaf. `http://karaf.apache.org`. [Online; accessed 21-September-2017].

[66] W. Vogels, "Eventually consistent," *Communications of the ACM*, vol. 52, no. 1, pp. 40–44, 2009.

[67] A.-M. Kermarrec and M. Van Steen, "Gossiping in distributed systems," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 5, pp. 2–7, 2007.

[68] E. Brewer, "CAP twelve years later: How the "rules" have changed," *Computer*, vol. 45, no. 2, pp. 23–29, 2012.

[69] The Copycat framework. `http://atomix.io/copycat/`. [Online; accessed 20-September-2017].

[70] D. Ongaro and J. K. Ousterhout, "In search of an understandable consensus algorithm," in *USENIX Annual Technical Conference*, pp. 305–319, USENIX Association, 2014.

[71] The Raft Consensus Algorithm. `https://raft.github.io/`, 2014. [Online; accessed 10-September-2017].

[72] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks," in *Proc. of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (Hotnets-IX)*, ACM, 2010.

[73] Centec Networks. `http://www.centecnetworks.com`. [Online; accessed 19-September-2017].

[74] InfluxData, InfluxDB Time-Series Data Storage. `http://www.influxdata.com`. [Online; accessed 20-September-2017].

[75] Grafana Labs, Grafana. `http://grafana.com`. [Online; accessed 20-September-2017].

[76] Vmware Inc., Vmware PowerCLI. `https://www.vmware.com/support/developer/PowerCLI/`. [Online; accessed 21-September-2017].

[77] Pivotal Software Inc., Spring Boot. `https://projects.spring.io/spring-boot/`. [Online; accessed 21-September-2017].

[78] Iperf3. `https://iperf.fr/iperf-download.php`. [Online; accessed 21-September-2017].

[79] P. Zhang, H. Li, C. Hu, L. Hu, L. Xiong, R. Wang, and Y. Zhang, "Mind the Gap: Monitoring the Control-Data Plane Consistency in Software Defined Networks," in *Proc. of the 12th Int. Conference on emerging Networking EXperiments and Technologies*, pp. 19–33, ACM, 2016.

[80] R. Jain, *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling.* John Wiley & Sons, 1990.

[81] Vmware Inc., VMware ESXi. `https://my.vmware.com/en/web/vmware/evalcenter?p=free-esxi6`. [Online; accessed 21-September-2017].

[82] Centec Networks, V350 Series OpenFlow/SDN Switch. `http://www.centecnetworks.com/en/SolutionList.asp?ID=43`. [Online; accessed 19-September-2017].

[83] ONOS bug reports. `https://jira.onosproject.org/browse/ONOS-4978`. [Online; accessed 21-September-2017].

[84] ONOS bug reports. `https://jira.onosproject.org/browse/ONOS-6780`. [Online; accessed 21-September-2017].

[85] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker, "Applying NOX to the datacenter," in *Proceedings of the 8th Workshop on Hot Topics in Networks (HotNets-VIII)*, ACM, 2009.

[86] L. J. Jagadeesan and V. Mendiratta, "Programming the Network: Application Software Faults in Software-Defined Networks," in *Proceedings of the IEEE 27th International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp. 125–131, IEEE, 2016.

[87] R. M. Lefever, M. Cukier, and W. H. Sanders, "An experimental evaluation of correlated network partitions in the Coda distributed file system," in *Proceedings of the 22nd International Symposium on Reliable Distributed Systems (SRDS)*, pp. 273–282, IEEE, 2003.

[88] "Network faults in distributed systems." `https://github.com/aphyr/partitions-post`, 2014.

[89] X. Ju, L. Soares, K. G. Shin, K. D. Ryu, and D. Da Silva, "On fault resilience of OpenStack," in *Proceedings of the 4th annual Symposium on Cloud Computing (SOCC)*, ACM, 2013.

[90] S. Dawson, F. Jahanian, T. Mitton, and T.-L. Tung, "Testing of fault-tolerant and real-time distributed systems via protocol fault injection," in *Proceedings of Annual Symposium on Fault Tolerant Computing*, pp. 404–414, IEEE, 1996.

[91] C. Di Martino, Z. Kalbarczyk, R. K. Iyer, G. Goel, S. Sarkar, and R. Ganesan, "Characterization of operational failures from a business data processing SaaS platform," in *Companion Proceedings of the 36th International Conference on Software Engineering (ICSE)*, pp. 195–204, ACM, 2014.

[92] D. Cotroneo, L. D. Simone, A. K. Iannillo, A. Lanzaro, R. Natella, J. Fan, and W. Ping, "Network Function Virtualization: Challenges and Directions for Reliability Assurance," in *Proceedings of the 25th IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp. 37–42, IEEE, 2014.

[93] Apache Software Foundation, Apache ActiveMQ. `http://activemq.apache.org/`. [Online; accessed 21-September-2017].

[94] Apache Software Foundation, Apache Karaf Monitoring and Management using JMX. `https://karaf.apache.org/manual/latest-3.0.x/monitoring`. [Online; accessed 21-September-2017].

[95] Z. Mwaikambo, A. Raj, R. Russell, J. Schopp, and S. Vaddagiri, "Linux kernel hotplug CPU support," in *Proceedings of the Linux Symposium - Volume Two*, pp. 467–480, 2004.

[96] Linux Foundation, tbf – Token Bucket Filter. `http://linux.die.net/man/8/tc-tbf`. [Online; accessed 21-September-2017].

[97] Linux Foundation, NetEm - Network Emulator. `http://man7.org/linux/man-pages/man8/tc-netem.8.html`. [Online; accessed 21-September-2017].

[98] Linux Foundation, iptables. `https://linux.die.net/man/8/iptables`. [Online; accessed 21-September-2017].

[99] Linux Foundation, ip. `https://linux.die.net/man/8/ip`. [Online; accessed 21-September-2017].

[100] Linux Foundation, ptrace – process trace. `http://man7.org/linux/man-pages/man2/ptrace.2.html`. [Online; accessed 21-September-2017].

[101] J. Keniston, A. Mavinakayanahalli, P. Panchamukhi, and V. Prasad, "Ptrace, Utrace, Uprobes: Lightweight, Dynamic Tracing of User Apps," in *Proceedings of the 2007 Linux Symposium*, vol. one, pp. 215–224, 2007.

[102] V. Sieh, "Fault-injector using UNIX ptrace interface," Internal Report 11/93, IMMD3, Universität Erlangen Nürnberg, 1993.