PhD in Information Technology and Electrical Engineering

Università degli Studi di Napoli Federico II

PhD Student: Ugo Giordano

XXIX Cycle

Training and Research Activities Report – Third Year

Tutor: Prof. Stefano Russo

1. Information

PhD Student: Ugo Giordano
MS title: Computer Engineering – University of Naples Federico II
PhD cycle: XXIX – ITEE – University of Naples Federico II
Fellowship: Projects "SVEVIA" and "DISPLAY" of the "COSMIC" public-private laboratory (PON02_00669)
Tutor: Prof. Stefano Russo

2. Study and Training activities

a. Courses

Lecture/Activity	Type	Professor	Date	h	CFU
Project Management per la Ricerca	Ad hoc module	Guido Capaldo	20/03/15	15	**3**
Models, methods and software for Optimization	Ad hoc module	Antonio Sforza/Claudio Sterle	23-25/03-04/15	18	**4**
English Language Course	Ad hoc module	Geraint Thomas	-	-	**6**
Sistemi Real Time	Specialistic Module	Marcello Cinque	-	-	**6**
Dependability of Computer Systems	Specialistic Module	Cotroneo Domenico	-	-	**6**
Total					**25**

b. Seminars and other

Lecture/Activity	Type	Professor	Date	h	CFU
Reliability of electronic power devices And modules	Seminar	Alberto Castellazzi	24-26/03/15	6	**1.2**
The Memories of Tomorrow: Technology, Design, Test, and Dependability	Seminar	Elena Ioana Vatajelu	24/04/15	3	**0.6**
Design and writing scientific manuscripts for publication in English language scholarly journals, and related topics	Seminar	Barnet Parker	15-17/06/15	12	**3**
Beyond the data: how to achieve actionable insights with machine learning	Seminar	Matteo Santoro	10/11/15	2	**0.3**
Winter School: securing Critical Infrastructures	Doctoral School	-	17-24/01/16	36	**7.2**
Adversarial Testing of Protocol Implementations	Seminar	Cristina Nita Notaru	23/02/16	2	**0.4**
Programmable network conjunction	Seminar	Roberto Bifulco	26/02/16	2	**0.4**
Bell Labs seminar: the future of Network Technologies	Seminar	-	28-29/04/16	4	**0.8**
Total					**25**

3. Research Activities

Title: In-production continuous testing for future Telco Cloud

3.1 SDN: toward the softwarization of future networks

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 equipment 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's 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 (from low- to high-level 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 layer1, and atop the data plane, that it controls en- acting 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 ONOSTM [10], while probably the most widely known API is OpenFlow [11].

3.2 Motivations and contributions

The SDN controller is a logically centralized entity. Scalability, performance and reliability requirements demand however for its engineering in a distributed manner. This raises advanced challenges in achieving failure resilience, meant as the ability to maintain an acceptable level of service even in presence of failures. This in fact depends not only on fault tolerance in the data plane, as in traditional networks, but also on the high resilience of the control functions, logically centralized yet actually distributed.
As controllers' technology develops and they progressively become mature for the market, the need to engineer and to assess SDN solutions' compliance with non functional requirements - such as scalability, high availability, fault tolerance and high reliability - becomes more compelling. **The in-production assessment of the failure resilience of SDN controllers** is the focus of my research activity during the third year. The goal is to devise an approach to evaluate the extent to which a network controller's mechanisms are able to react to failure conditions, so as to ultimately satisfy non functional requirements, especially availability, reliability and fault tolerance.

The literature on SDN resilience assessment is still at the beginning. The research activity aims to contribute to the advancement in testing and evaluation of SDN resilience, trying to go beyond what can be achieved by means of "traditional" software analysis and testing techniques.
Indeed, in academic research it is a common practice to evaluate algorithmic design choices by modeling or simulation without actual implementations, while in the industry, testing is typically used to evaluate the design and implementation of a system. However, an algorithmic component could not guarantee the proper operation of the whole system when other components affect the overall behaviour. This is especially true with the increasing complexity of softwarized networks, due to which troubleshooting in operation is expected to become more complex. Modeling and simulation may be insufficient to unveil design defects, and implementation defects are sometimes not unveiled by network testing. Even when in principle defects can be made manifest through testing, it may be very hard and time consuming to identify test scenarios able to reproduce potential failures.
The vision we embrace is that it is an important goal in the engineering of SDNs to be able to perform testing and assessment not only in emulated conditions before release and deployment, but also in-production, when the system is under real operating conditions. This vision is somehow inspired by Netflix's Chaos Engineering approach [12]. Companies like Netflix providing services over the Internet "push new code into production and modify configurations hundreds of times a day" [12]. Netflix was probably the first advocating the need for "injecting failures into the production system to improve reliability".
Within this vision, the aim of this research activity is pursued through the use of software failure injection, which serves to evaluate a system behaviour under (possibly unforeseen) failure conditions. Differently from software fault injection [13] - 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.

During my third year I proposes a software failure injection framework for in-production assessment of the effectiveness of the failure detection and mitigation mechanisms provided by SDN controllers, by reproducing specific failure scenarios. The framework consists of a methodology and a configurable software infrastructure for failure injection in a SDN controller. The distributed infrastructure encompasses - as main components - a workload generator, a failure injector, and data collectors. The experimental evaluation is based on an open source industrial network operating system.

In accomplishing my goal, I took benefit from a concrete experience in a very advanced industrial research center in USA. Most of the work has actually been done during a period at the prestigious Murray Hill NOKIA Bell Labs in New Providence, New Jersey. The need for controller's resilience assessment and the key ideas of the failure injector have **been developed in this experience, in**

the framework of NOKIA's continuous and in-production testing strategy for the future generation of network solutions. The experiments are based on the open source distributed network operating system ONOSTM [10] [14], which is the very heart of the testbed. The ONOSTM initiative is supported by several major industrial partners, including AT&T, Cisco, Ericsson, Google, Huawei, NOKIA.

Finally, the main contributions of my third year research activity lie in:

- The use of failure injection to face the problem of evaluating the resilience of SDN controllers, based on a failure model envisaged specifically for distributed controllers, identifying representative failures to be injected;
- An injection methodology, conceived for both in-fabric and in-production assessment. The methodology envisages the steps of (i) definition of the workload (according to the Intent Based Networking model [15]) 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 design of a configurable distributed infrastructure to support the (optional) load generation, failure injection and data collection tasks;
- The experimental evaluation, on a distributed testbed based on ONOS™.

3.3 A Vision for Future Telco

With the rapid growth and spread of devices demanding for network services, the nowadays mobile broadband networks will need to be prepared to deliver much more data per user per day at lower cost, despite the added challenge of unpredictable traffic patterns. Furthermore, 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 to implement new services rapidly. To address these requirements, telecommunication (Telco, for brevity) companies, such as NOKIA, are palling to migrate towards the so called "**Telco Cloud**" (TC) [54,55,56], becoming a **Telco Cloud Service Provider** (TCSP). Telco Cloud is meant to provide a dedicated cloud computing solution for network operator, to shift network functions away from dedicated legacy hardware platforms into virtualized software components deployable on general-purpose hardware. Such an approach, will increase the flexibility and agility of the networks promoting innovative solutions for future network services, and will significantly lower the costs of hardware appliances and management activities. Therefore, the virtualization of network services plays a crucial rule in the Telco's vision of future networks, however a "virtualizing everything" approach has its drawback.

Software Defined Networking is a key area, along with Network Function Virtualization, to enable a full Telco cloud ecosystem. However, the introduction of SDNs technologies has raised new challenges in achieving network resiliency and fault tolerance due to the emergence of new failures scenarios. Indeed, by decoupling the control plane from the data plane the overall network resilience depends not only on the fault-tolerance in the data plane, as in the traditional networks, but also on the high availability of the (logically) centralized control functions.
Moreover, given the issues related to a centralized solution, i.e., a single point of failure, and the complexity of the control plane functionalities, the SDNs are by nature suitable to be implemented as distributed system. Such an approach can lead to further threats to the network resilience, such as inconsistent global network state shared between the SDN controllers as well as a network partitioning.

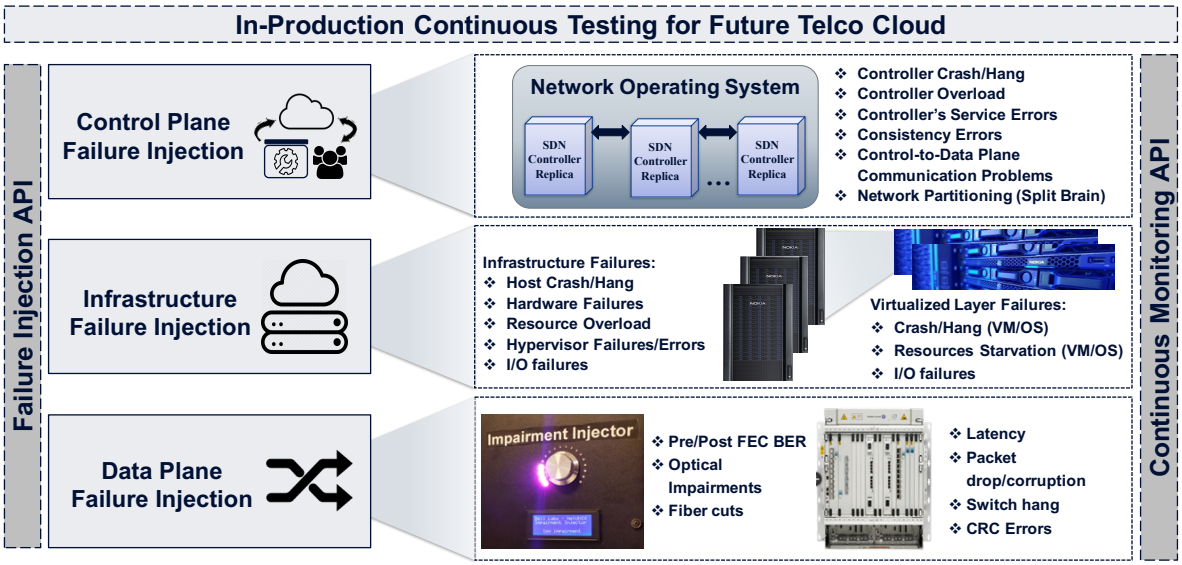

Figure 1 In-production continuous testing in Telco Cloud, a vision of future networks.

In addition, compared to the legacy network appliances, the adoption of technologies for virtualizing network services, introduce further threats to the networks reliability and resiliency, due to the known failures which periodically occur in data center [57,58].

Consequently, in such an operational 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), they are not suitable to identify failures that can affect complex distributed systems during production hours [35]. 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. Unlike the traditional testing techniques, failure injection allows to reproduce specific 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.

The vision we embrace is that 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 perform testing and assessment not only before release and deployment in emulated conditions, but also in-production, when the system is under real operating conditions. Figure 1 depicts an overview of such a vision, where failure injection techniques are exploited to continuously assess the reliability and resilience of the network services against widespread failure scenarios. This approach will provide continuous feedback on the capabilities of the softwarized network services to survive failures, which is of fundamental importance for improving the effectiveness of 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 need to be injected at different layers of the Telco infrastructure (see Figure 1), namely: (i) at data-plane layer, 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, e.g. by emulating hardware failures or causing resource starvation; and (iii) at control plane level, to emulate faulty network controllers, e.g. by corrupting the status of a controller or its communication with other replicas or the data plane layer. Within this vision, the research goal is to pursue 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 conditions, to evaluate the system behaviour under (possibly unforeseen) disruptive conditions. Specifically, the focus is on the resilience of control plane layer, and proposes a software failure injection framework for in-production

assessment of the effectiveness of the failure detection and mitigation mechanisms provided by SDN controllers. The framework consists of a method- ology and a configurable software infrastructure for failure injection in a SDN controller. The distributed infrastructure encompasses - as main components - a workload generator, a failure injector, and data collectors. The experimental evaluation is based on an open source industrial network operating system.
It has to be note that, although the failure model described in the next sections is meant to target specifically the control plane ecosystem, some of the proposed failure modes may overlap with those required to emulate faulty condition at infrastructure level (see Figure 1), e.g. a controller crash may correspond with the crash of the virtual machine hosting such controller.

3.4 Failure injection testing: The Netflix approach

In providing a methodology to assess the resilience and the failover mechanisms of the 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 op- erational 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 in a controlled experimental environment. Here, the key concept is represented by the execution of failure injection experiments within a live production environment, which has three main advantages:

- It allows a better assessment of the system by verifying the correctness of its behaviour under realistic production deployment and workload condition;
- It makes the system immune to possible failures;
- It helps preventing larger outages that can affect the overall system availability.

Such a methodology comes under the umbrella of the broader concept of "Chaos Engineering" [12], [51] 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 [52, 53] 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.
Although inspired by the Netflix's approach, the assessment methodology 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.

3.5 Assessment methodology

During the third year of the PhD I propose a methodology and a framework to validate the reliability and resilience of distributed SDN platforms. Specifically, **the proposed methodology aims to assess the effectiveness of the failure detection and mitigation mechanisms provided by the SDNs by reproducing representative failure scenarios**. To this end, I also designed a **Failure Injection infrastructure** to deliberately inject failures in the SDN ecosystem limiting intrusiveness, as much as possible.

The steps of the methodology are outlined in Fig. 2. There are three macro-steps:

- Definition of the workload, of the failure model, and of performance indicators;
- Experiments' execution;
- Computation of metrics and reporting.

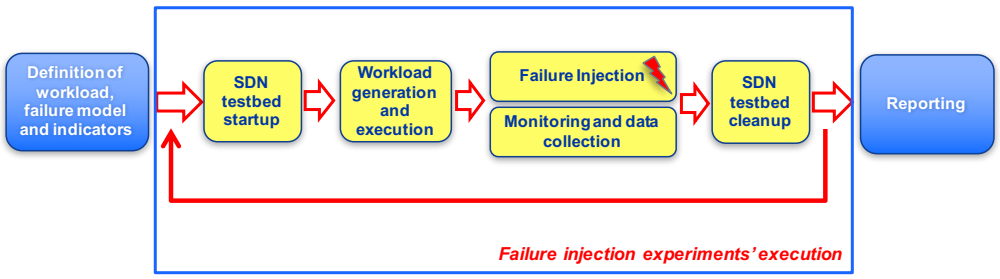

Figure 2 Failure injection based SDN resilience assessment methodology.

The execution of the failure injection experiments encompasses a number of tasks (Fig. 3). After the definition of the failure model and the workload parameters (step 1 in Fig. 3), the experiment is set up, instantiating controllers and data monitors on a distributed computing architecture (step 2). Then, the workload is generated (step 3), so as to stimulate the SDN to bring it in a state where to inject a failure selected from the failure model. During execution, a failure is injected (step 4), while the system is monitored and data are collected (step 5). After execution, the testbed is cleaned up by restoring the original status of the machines running the SDN controllers, and restoring the controller instances, before starting the next experiment (step 6).

It is worth emphasizing that the complexity of the environments where SDNs operate can lead to situations that are difficult to replicate with the traditional software testing approaches [35]. For this reason, the proposal has been conceived as a framework (method- ology and injection infrastructure) to assess the resilience of the SDNs in production. The support infrastructure (described in Section 4.3) has been designed as an extension of a generic SDN architecture, so as to support the injections of failures even in a production environment, with the aim of continuous testing.

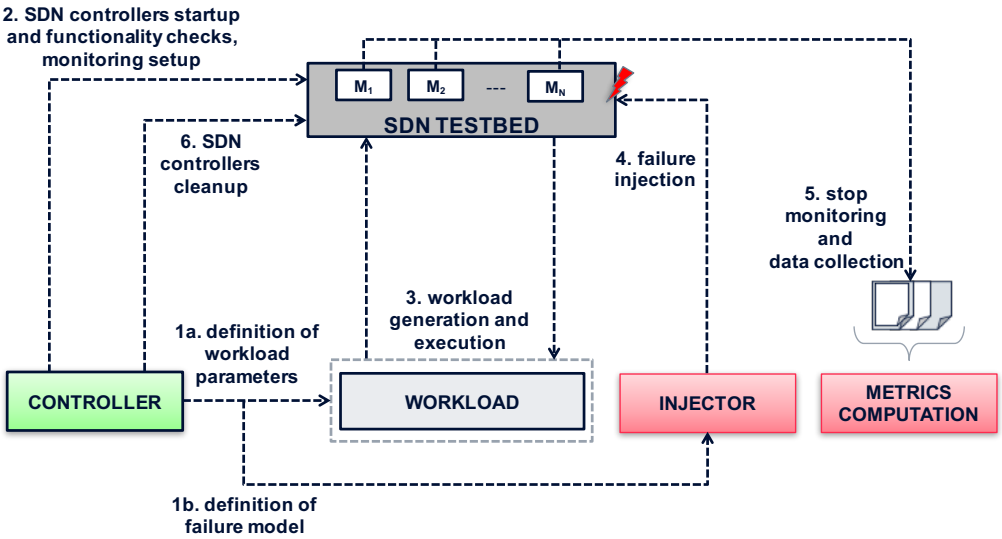

Figure 3 The steps of a failure injection experiment in the proposed methodology.

3.6 The failure injection framework

Software-Defined Networks are typically engineered as distributed systems given the draw- backs of centralized solutions in terms of scalability, performance, fault tolerance. This is true also for the newest SDN implementations, such as ONOS® (Open Network Operating System) and ODL® (OpenDaylight). For such a reason, we decided to adopt failures that belong to the most common failure classes observed in distributed systems [35,60,61,62]. These failures are injected by merely using API calls in a non-intrusive fashion.

Table 1 lists the failure classes considered in the proposed assessment methodology. Each class is intended to mimic different types of failure scenarios at different levels of the software stack. According to the level to which they apply, failures are classified in three main categories:

- **System Failures**: Relevant failure classes might affect the computational resources as well as I/O operations (e.g., physical/virtual CPU, memory and disk) of a target node. The failures that belong to this category are intended to evaluate the resilience of the SDNS to nodes crashes and resources depletion of the machines hosting the SDN controller instances. The failure types envisaged in this respect are system hang, starvation, outage and shutdown (at system and at single CPU level), as well as disk and memory saturation. Furthermore, a single controller instance might suffer from increased CPU utilization, for instance due to other compute-intensive jobs running on the same target machine. The corresponding failure types to mimic such a scenario are CPU or I/O burn. This class of failures are emulated by starting additional jobs that deliberately consume CPU cycles and allocate memory areas aiming to cause resource exhaustion, i.e. CPU and memory “hogs”.

- **Network Failures**: Network problems, such as link failures or latent communication, are among the ones that have always been faced by distributed applications [60,63, 64]. The most common consequence of these kind of failures is the partitioning of the network that split a system in multiple disjoint, disconnected partitions. As a result, even if a system is designed to be partition-tolerant, there are no guarantee that the modern distributed systems are able to cope with partitioned, unreliable networks [61]. To reproduce such network failures, message corruption or loss, partial or total network partitions as well as the permanent unavailability are introduced into the network interfaces. Furthermore, even latent communications and bandwidth limitation are emulated.

- **Service Failures**: This class of failures aims of mimicking the malfunctioning that may occur in the interaction between the SDN controller services. API calls may also be used to emulate a faulty controller instance by shut it down or to mimic an anomalous service behaviour by forcing the termination of specified system pro- cess. The corresponding failure types are emulated by process kill, and controller or dependency stop. Furthermore, the state of the controller is corrupted by a memory corruption, mimicking a hardware fault, or a programming error affecting the controller's memory.

Failure class	Failure type	Description
System Failures	System Hang	Stuck the system on an infinite loop without releasing the resources. The system's network interfaces are still responding.
	System Starvation	Cause the starvation of all the available system resources. The system first slow down, then crashes.
	System Outage	Cause a fatal error from which the system cannot safely recover (e.g. kernel panic).
	System Shutdown	Gracefully shut down the system.
	CPU Shutdown	Restrict the number of available CPUs.
	Disk Saturation	Cause the saturation of the disk, mimicking disk full errors.
	Memory Saturation	Cause the saturation of the memory, mimicking memory full errors.
	Burn CPU	Spawn CPU-bound processes, mimicking a faulty CPU and/or noisy process.
	Burn I/O	Spawn I/O-bound processes, mimicking a faulty disk and/or noisy process.
Network Failures	Black-hole	Abruptly drop the network communications towards a specific address or a subset of addresses.
	Packet Reject	Abruptly drop the inbound and/or outbound packets sent to specified address and/or port.
	Packet Drop	Quietly drop the inbound and/or outbound packets sent to specified address and/or port.
	Packet Latency	Induce artificial delays for packets sent to specified address and/or port.
	Packet Loss	Induce artificial losses for the packets sent to specified address and/or port.
	Packet Re-order	Induce a mis-ordering of certain packets sent to specified addresses and/or port.
	Packet Duplication	Induce duplication of certain packets sent to specified addresses and/or port.
	Packet Corruption	Induce a random noise by introducing an error in a random position of certain packets sent to specified addresses and/or port.
	Throttling	Induce a bandwidth limitation to the outgoing network traffic with specified addresses and/or port.
Service Failures	Kill Process	Quietly terminate the controller process (*SIGTERM* signal), mimicking a faulty service.
	Process Corruption	Randomly corrupt the state of the SDN controller process, mimicking a service misbehaviour.
	Controller Stop	Quietly stop of the SDN controller process .
	Controller Restart	Gracefully restart the SDN controller.
	Dependency Stop	Quietly stop one or more dependencies, i.e. modules, of the SDN controller.

Table 1 Failure Model.

It has to be pointed out that future SDN controllers are likely to be engineered to be deployable also in virtualized platforms, or in container technologies. This means that system failures should in principle to be injectable also at the virtual machine (VM) or container level. However, as this is true for any software system deployed in VM or containers (for instance, fault injection has been proposed for virtualized network functions [65]), we intentionally did not include in our failure model failures at the Hypervisor, host or container level, limiting it to failures whose injection allows to test controllers wherever they are in execution.

Each failure in the model is triggered according to a specified injection time, that is the exact time when the failure must be injected. In addition, in order to allow the design of more complex failure-injection scenarios, the failure can be injected in three different modes, namely:

(i) **transient mode**: the failures are injected only once and removed after a specified amount of time in order to emulate temporary failure;

(ii) **intermittent mode**: the failures are periodically injected and left in the system for a specified amount of time to emulate temporary, but recurrent failure conditions;

(iii) **permanent mode**: the failures are injected and never removed from the system to emulate persistent failure conditions.

3.7 The failure injection framework

The proposed assessment methodology has been complemented with a failure injection framework, which is composed of a set of of interoperating sub-components, namely:

- The **Workload Generator** is in charge of generating the workload for the SUT. By means of the Load Generator module, it equally distributes the requests towards the corresponding Load Generator API, which instruments the core northbound interfaces of the controller instances.

- The **Failure Injector** is a ready-to-use component which can be integrated seamlessly into the target system. It aims to inject different failure modes on the SUT. To this end, an instance of the FI Manager module is deployed on each hosts of the SUT to actually perform the injection, while the FI Controller module remotely coordinate the injection on all the target hosts.

- The **Data Collector** is in charge of collecting the data concerning to the SUT, namely the metrics concerning to the controller instances as well as the machines hosting such instances. The formers are collected by instrumenting the controller's source code with the Event Data Collector module, while the latter are collected by the Host Data Collector module. Then, these data are permanently stored through the Data Collector Server module.

Figure 4 shows how the main components of the proposed SDN assessment framework are integrated with the SUT and their interconnections. The next sub-sections describe in detail the outlined framework's components.

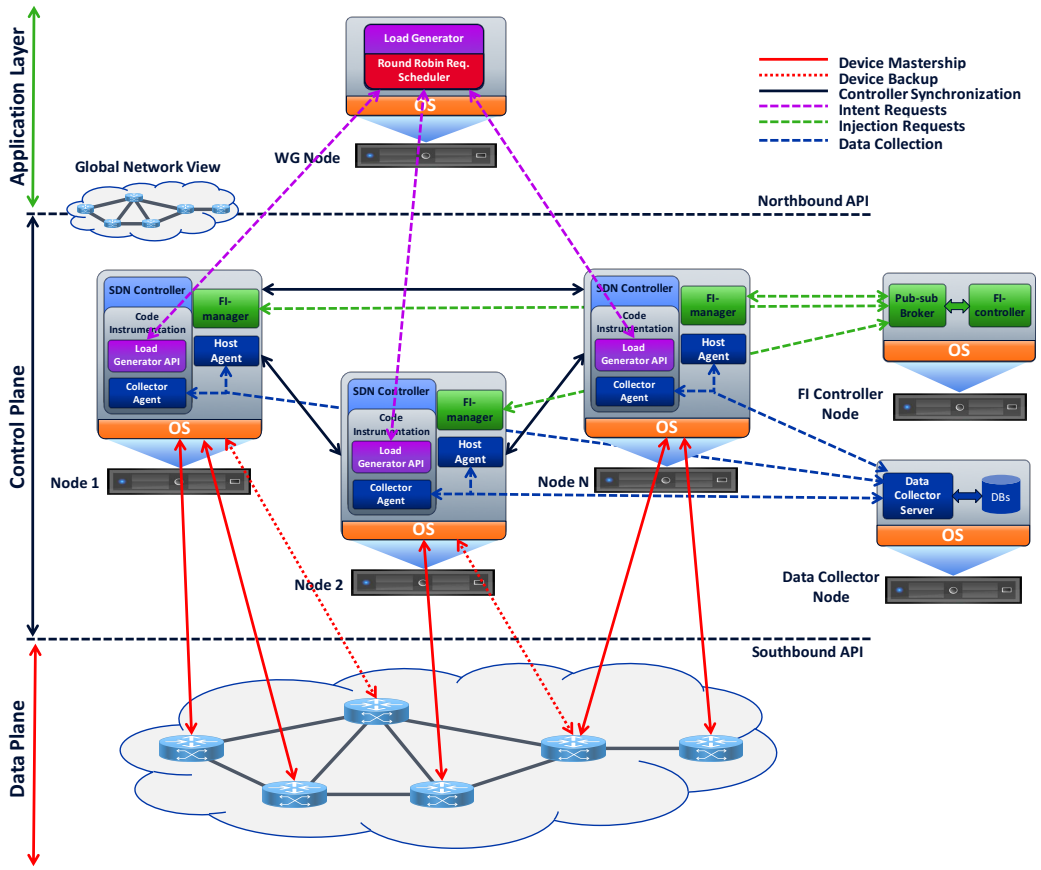

Figure 4 Architecture of the SDN failure injection infrastructure.

4. Products

a. Publications

Conference Paper

G.Carrozza, M.Cinque, U.Giordano, R.Pietrantuono, S.Russo,
Prioritizing Correction of Static Analysis Infringements for Cost-Effective Code Sanitization, Proc. IEEE/ACM 2nd International Workshop on *Software Engineering Research and Industrial Practice* (SER&IP), 37th "*IEEE International Conference on Software Engineering*",
Florence, Italy, May 17, 2015, pp. 25-31 DOI: 10.1109/SERIP.2015.13D, ISBN: 9781467370851, IEEE

b. Awards

Certificate of Appreciation from Nokia Bell Labs for the Summer '16 Internship for the contribution to the creation of the first version of "NetUnix", a Nokia Network Operating System.

5. Conference and Seminars

a. Conference

Winter School: "Securing *Critical Infrastructures*", Cortina d'Ampezzo, from January 17th to 24th, 2016

6. Activities abroad

I have spent the entire third year of the PhD course abroad, from April 2016 to March 2017 as PhD intern student at the prestigious Murray Hill NOKIA Bell Labs Bell Labs, a very advanced industrial research center in USA. The collaboration has been focused on the assessment of the resilience mechanisms provided by the nowadays SDN platform, such as ONOS™ and ODL™. The research activity has brought to the implementation of a methodology and framework for assessing the resilience of SDN platforms by means of failure injection in an in-production ecosystem. During this period, I acquired very well knowledge about the SDN technologies, having the pleasure to work with highly qualified people in the industry.

7. Tutorship

I have been teaching assistant for the course of Real-time Systems which is a course of the Master's degree in Computer Engineering, Department of Electrical Engineering and Information Technologies at University of Naples Federico II, academic year 2015/2016.

8. CS summary

	Credits year 1								Credits year 2								Credits year 3									
		1	2	3	4	5	6			1	2	3	4	5	6			1	2	3	4	5	6			
	Estimated	bimonth	bimonth	bimonth	bimonth	bimonth	bimonth	Summary	Estimated	bimonth	bimonth	bimonth	bimonth	bimonth	bimonth	Summary	Estimated	bimonth	bimonth	bimonth	bimonth	bimonth	bimonth	Summary	Total	Check
Modules	24	0	0	6	0	3	15	24	13	7	6	0	0	0	0	13	0	0	0	0	0	0	0	0	37	30-70
Seminars	6	0	0	3	0	2,7	2	7,7	8	1,8	3	0	0	0,3	8	13,1	0	0,8	0	0	0	0	0	0,8	22	10-30
Research	35	8	8	2	9	8	2	37	45	5	7	7	8	9	9	45	60	8	9	10	10	10	10	57	139	80-140
	65	8	8	11	9	14	19	69	66	13,8	16	7	8	9,3	17	71,1	60	8,8	9	9	10	10	10	58	198	120-240

9. Bibliography

[1] Diego Kreutz, Fernando M. V. Ramos, Paulo E. Verissimo, Christian E. Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-Defined Networking: A Comprehensive Survey. Pro- ceedings of the IEEE, 103(1):14–76, 2015.

[2] BrunoA.A.Nunes,MarcMendonca,Xuan-NamNguyen,KatiaObraczka,andThierryTurletti. A survey of Software-Defined Networking: Past, Present, and Future of Programmable Net- works. IEEE Communications Surveys Tutorials, 16(3):1617–1634, 2014.

[3] Enrique Hernandez-Valencia, Steven Izzo, and Beth Polonsky. How Will NFV/SDN Transform Service Provider OpEx? IEEE Network, 29(3):60–67, 2015.

[4] Hyojoon Kim and Nick Feamster. Improving network management with software defined net- working. IEEE Communications Magazine, 51(2):114–119, 2013.

[5] Allied Market Research. Global Software-Defined Networking Market: Opportunities and Fore- casts, 2015 - 2022. www.alliedmarketresearch.com/software-defined-networking-market (ac- cessed January 2017), 2016.

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

[7] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Mart ´ın Casado, Nick McKeown, and Scott Shenker. NOX: towards an operating system for networks. ACM SIGCOMM Computer Communication Review, 38(13):105–110, 2008.

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

[9] Jan Medved, Robert Varga, Anton Tkacik, and Ken Gray. OpenDaylight: Towards a Model- Driven SDN Controller architecture. In Proceeding of IEEE 15th International Symposium on a World of Wireless, Mobile and Multimedia Networks. IEEE, 2014.

[10] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Brian O'Connor Bob Lantz, Pavlin Radoslavov, William Snow, and Guru Parulkar. ONOS: towards an open, distributed SDN OS. In Proceedings of the 3rd Workshop on Hot Topics in Software Defined Networking (HotSDN), pages 1–6. ACM, 2014.

[11] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: Enabling innovation in campus Networks. SIGCOMM Computer Communication Review, 38(2):69–74, 2008.

[12] Ali Basiri, Niosha Behnam, Ruud de Rooij, Lorin Hochstein, Luke Kosewski, Justin Reynolds, and Casey Rosenthal. Chaos Engineering. IEEE Software, 33(3):35–41, 2016.

[13] Roberto Natella, Domenico Cotroneo, and Henrique S. Madeira. Assessing Dependability with Software Fault Injection: A Survey. ACM Computing Surveys, 48(3):44:1–44:55, 2016.

[14] ONOS Project. ONOS White Paper. http://onosproject.org/wp-content/uploads/2014/11/ Whitepaper-ONOS-final.pdf (accessed January 2017), 2014.

[15] ONOS Project. http://wiki.onosproject.org/display/onos/intent+framework. Website (ac- cessed January 2017).

[16] Hamid Farhady, HyunYong Lee, and Akihiro Nakao. Software-defined networking: A survey. Computer Networks, 81:79–95, 2015.

[17] Open Networking Foundation. www.opennetworking.org. Website (accessed January 2017).

[18] Dave 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 (Accessed January 2017), 2015.

[19] Open Networking Foundation. www.opennetworking.org/sdn-resources/openflow. Website (ac- cessed January 2017).

[20] ONOS Project. ONOS mission, http://onosproject.org/mission/. Webpage (accessed January 2017).

[21] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Transactions on Dependable and Secure Computing, 1(1):11–33, 2004.

[22] Catello Di Martino, Veena Mendiratta, and Marina Thottan. Resiliency Challenges in Ac- celerating Carrier-Grade Networks with SDN. In Proceedings of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), pages 242–245. IEEE, 2016.

[23] Soheil Hassas Yeganeh, Amin Tootoonchian, and Yashar Ganjali. On Scalability of Software- Defined Networking. IEEE Communications Magazine, 51(2):136–141, 2013.

[24] SethGilbertandNancyLynch.Brewer'sConjectureandtheFeasibilityofConsistent,Available, Partition-Tolerant Web Services. ACM SIGACT News, 33(2), 2002.

[25] Carolyn R. Johnson, Yakov Kogan, Yonatan Levy, Farhad Saheban, and Percy Tarapore. Voip Reliability: A Service Provider's Perspective. IEEE Communications Magazine, 42(7):48–54, 2004.

[26] Aditya Akella and Arvind Krishnamurthy. A Highly Available Software Defined Fabric. In Proceedings of the 13th ACM Workshop on Hot Topics in Networks (HotNets), pages 1–7. ACM, 2014.

[27] Jean-Francois Castet and Joseph 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.

[28] The ResiliNets Initiative. ResiliNets Wiki, https://wiki.ittc.ku.edu/resilinets wiki/index.php/ Definitions#Resilience. Website (accessed January 2017).

[29] Paul Smith, David Hutchison, James P.G. Sterbenz, Marcus Sch ¨oller, Ali Fessi, Merkouris Karaliopoulos, Chidung Lac, and Bernhard Plattner. Network resilience: a systematic approach. IEEE Communications Magazine, 49(7):88–97, 2011.

[30] Jean-Claude Laprie. Resilience for the scalability of dependability. In Proceedings of the 4th IEEE International Symposium on Network Computing and Applications (NCA), pages 5–6. IEEE, 2005.

[31] Sachin Sharma, Dimitri Staessens, Didier Colle, Mario Pickavet, and Piet Demeester. Open- Flow: Meeting carrier-grade recovery requirements. Computer Communications, 36(6):656–665, 2013.

[32] Leslie Lamport. Paxos made simple. ACM SIGACT News, 32(4):18–25, 2001.

[33] K. Mani Chandy and Leslie Lamport. Distributed snapshots: Determining global states of distributed systems. ACM Transactions on Computer Systems, 3(1):63–75, 1985.

[34] The Netflix Tech Blog. Netflix Chaos Monkey 2.0. http://techblog.netflix.com/2016/10/netflix- chaos-monkey-upgraded.html (accessed January 2017), Oct. 2016.

[35] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Sub- baiah Venkata, Jim Wanderer, Junlan Zhou, and Min Zhu. B4: Experience with a globally- deployed software defined WAN. ACM SIGCOMM Computer Communication Review, 43(4):3– 14, 2013.

[36] Song Huang, Zhiang Deng, and Song 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), pages 1–8. IEEE, 2016.

[37] Brandon Heller, Rob Sherwood, and Nick McKeown. The controller placement problem. In Proceedings of the 1st Workshop on Hot Topics in Software Defined Networking (HotSDN), pages 7–12. ACM, 2012.

[38] Paulo Fonseca, Ricardo Bennesby, and Edjard Mota. A Replication Component for Resilient OpenFlow-based Networking. In Proceedings of the IEEE/IFIP Network Operations and Man- agement Symposium (NOMS). IEEE, 2012.

[39] Navin Budhiraja, Keith Marzullo, Fred B. Schneider, and Sam Toueg. Distribyted systems, chapter The primary-backup approach, pages 199–216. ACM Press/Addison-Wesley Publishing Co., 2nd edition, 1993.

[40] Francisco J. Ros and Pedro 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), pages 31–36. ACM, 2014.

[41] Jie Hu, Chuang Lin, Xiangyang Li, and Jiwei 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), pages 147–152. IEEE, 2014.

[42] Murat Karakus and Arjan 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), pages 282–289. IEEE, 2016.

[43] Amin Tootoonchian, Sergey Gorbunov, Yashar Ganjali, Martin Casado, and Rob 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.

[44] Yimeng Zhao, Luigi Iannone, and Michel 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), pages 79–85. IEEE, 2015.

[45] Michael Jarschel, Frank Lehrieder, Zsolt Magyari, and Rastin Pries. A flexible OpenFlow- controller benchmark. In Proceedings of the European Workshop on Software Defined Network- ing (EWSDN), pages 48–53. IEEE, 2012.

[46] Amin Tootoonchian et al. Cbench: an Open-Flow Controller Benchmarker. https://github.com/mininet/oflops/tree/master/cbench.

[47] Keqiang He, Junaid Khalid, Sourav Das, Aaron Gember-Jacobson, Chaithan Prakash, Aditya Akella, Li Erran Li, and Marina Thottan. Latency in Software Defined Networks: Measurements and Mitigation Techniques. ACM SIGMETRICS Performance Evaluation Review, 43(1):435– 436, 2015.

[48] Jean Arlat, Martine Aguera, Louis Amat, Yves Crouzet, Jean-Charles Fabre, Jean-Claude Laprie, Eliane Martins, and David Powell. Fault Injection for Dependability Validation: A Methodology and Some Applications. IEEE Transactions on Software Engineering, 16(2):166– 182, 1990.

[49] Roberto Natella, Domenico Cotroneo, Joao Duraes, and Henrique Madeira. On Fault Represen- tativeness of Software Fault Injection. IEEE Transactions on Software Engineering, 39(1):80– 96, 2013.

[50] Domenico Cotroneo, Luigi De Simone, Antonio Ken Iannillo, Anna Lanzaro, and Roberto Natella. Dependability Evaluation and Benchmarking of Network Function Virtualization In- frastructures. In Proceedings of the 1st IEEE Conference on Network Softwarization (NetSoft), pages 1–9. IEEE, 2015.

[51] Principles of chaos engineering — The Netflix tech blog, 2015. [Online; accessed 31-January- 2017].

[52] Fit: Failure injection testing — The Netflix tech blog, 2015. [Online; accessed 31-January-2017].

[53] Peter Alvaro, Kolton Andrus, Chris Sanden, Casey Rosenthal, Ali Basiri, and Lorin Hochstein. Automating failure testing research at internet scale. In Proceedings of the Seventh ACM Symposium on Cloud Computing, pages 17–28. ACM, 2016.

[54] Jo ˜ao Soares, Carlos Gonc ¸alves, Bruno Parreira, Paulo Tavares, Jorge Carapinha, Jo ˜ao Paulo Barraca, Rui

L Aguiar, and Susana Sargento. Toward a telco cloud environment for service functions. IEEE Communications Magazine, 53(2):98–106, 2015.

[55] Peter Bosch, Alessandro Duminuco, Fabio Pianese, and Thomas L Wood. Telco clouds and virtual telco: Consolidation, convergence, and beyond. In Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on, pages 982–988. IEEE, 2011.

[56] Xu Zhiqun, Chen Duan, Hu Zhiyuan, and Sun Qunying. Emerging of telco cloud. China Communications, 10(6):79–85, 2013.

[57] Amazon S3 Team et al. Amazon s3 availability event: July 20, 2008. Retrieved November, 15:2008, 2008.

[58] Eric Bauer and Randee Adams. Reliability and availability of cloud computing. John Wiley & Sons, 2012.

[59] Erhan Cinlar. Introduction to stochastic processes. Courier Corporation, 2013.

[60] Ryan M Lefever, Michel Cukier, and William H Sanders. An experimental evaluation of cor- related network partitions in the Coda distributed file system. In Proceedings of the 22nd International Symposium on Reliable Distributed Systems (SRDS), pages 273–282. IEEE, 2003.

[61] Cbench. Network faults in distributed systems. https://github.com/mininet/oflops/tree/ master/cbench, 2014.

[62] Xiaoen Ju, Livio Soares, Kang G Shin, Kyung Dong Ryu, and Dilma Da Silva. On fault resilience of OpenStack. In Proceedings of the 4th annual Symposium on Cloud Computing (SOCC). ACM, 2013.

[63] Scott Dawson, Farnam Jahanian, Todd Mitton, and Teck-Lee Tung. Testing of fault-tolerant and real-time distributed systems via protocol fault injection. In Proceedings of Annual Sym- posium on Fault Tolerant Computing, pages 404–414. IEEE, 1996.

[64] Catello Di Martino, Zbigniew Kalbarczyk, Ravishankar K Iyer, Geetika Goel, Santonu Sarkar, and Rajeshwari Ganesan. Characterization of operational failures from a business data process- ing SaaS platform. In Companion Proceedings of the 36th International Conference on Software Engineering (ICSE), pages 195–204. ACM, 2014.

[65] Domenico Cotroneo, Luigi De Simone, Antonio Ken Iannillo, Anna Lanzaro, Roberto Natella, Jiang Fan, and Wang Ping. Network Function Virtualization: Challenges and Directions for Reliability Assurance. In Proceedings of the 25th IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pages 37–42. IEEE, 2014.

[66] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, Zhi-Wei Lin, and Varugis Kurien. Pingmesh: A Large- Scale System for Data Center Network Latency Measurement and Analysis. ACM SIGCOMM Computer Communication Review, 45(4):139–152, 2015.

[67] Devanandham Henry and Jose Emmanuel Ramirez-Marquez. Generic metrics and quantitative approaches for system resilience as a function of time. Reliability Engineering & System Safety, 99:114–122, 2012.

[68] Zwane Mwaikambo, Ashok Raj, Rusty Russell, Joel Schopp, and Srivatsa Vaddagiri. Linux kernel hotplug CPU support. In Proceedings of the Linux Symposium - Volume Two, pages 467–480, 2004.

[69] Jim Keniston, Ananth Mavinakayanahalli, Prasanna Panchamukhi, and Vara Prasad. Ptrace, Utrace, Uprobes: Lightweight, Dynamic Tracing of User Apps. In Proceedings of the 2007 Linux Symposium, volume one, pages 215–224, 2007.

[70] Volkmar Sieh. Fault-injector using UNIX ptrace interface. Internal Report 11/93, IMMD3, Universit ¨at Erlangen Nu ¨rnberg, 1993.