
**PhD in Information Technology and Electrical Engineering**

**Università degli Studi di Napoli Federico II**

# PhD Student: Luigi De Simone

**XXIX Cycle**

**Training and Research Activities Report – Second Year**

**Tutor: Prof. Domenico Cotroneo**

## 1. Information

**PhD candidate**: Luigi De Simone
**Date of birth**: 24/02/1986
**Master Science title**: Master's degree in Computer Engineering (cum laude), Universiy of Naples Federico II
**Doctoral Cycle**: XXIX
**Fellowship type**: PhD student grant
**Tutor**: Prof. Domenico Cotroneo
**Year**: Second

I received my MS degree (cum laude) in Computer Engineering from the Univesità degli Studi di Napoli Federico II in July 2013.

My master thesis focused on the dependability of the Linux OS, specifically the fault-tolerance of device drivers. Device drivers are software components with the most of the defects ("bugs") within an operating system, thus they are the main cause of operating system failures. I proposed a novel fault-tolerance approach based on run-time monitoring and fault-detection of a storage device driver, and I developed and tested the approach to a storage device driver of the Linux kernel.

I'm currently at second year of PhD program in Information Technology and Electrical Engineering (ITEE) at Federico II University of Naples, under the supervision of Prof. Domenico Cotroneo.

## 2. Study and Training activities

In this first year I attended the following courses and seminars.

| Title | Type | Hours | Credits | Dates | Organizer | Certificate |
|---|---|---|---|---|---|---|
| Project Management | Ad-hoc | | 3 | 30 Jan. 2015, 6,13,20 and 27 Feb. 2015 | Università degli Studi di Napoli Federico II | Yes |
| Designing and writing scientific manuscripts for publication in english language scholarly journals, and related topics | Ad-hoc | | 3 | 15, 16, 17 June 2015 | Università degli Studi di Napoli Federico II | Yes |
| Modelli matematici e calcolo scientifico nell'ingeneria e nell'innovazione tecnlogica, by Prof. Alfio Quarteroni | Seminary | 2 | 0,4 | 15/04/2015 | DIETI | Yes |
| Half Day Tutorial at DTIS2015 Conference. Title: The Memories of Tomorrow: Technology. Design, Test and Dependability, by PhD Elena Ioana Vatajelu | Seminary | 3 | 0,6 | 24/04/2015 | DIETI | Yes |
| Adversarial Testing of Protocol Implementations, by Prof. Cristina Nita Notaru | Seminary | 1,5 | 0,4 | 24/02/2016 | DIETI | Yes |
| Programmable network conjugations, by Dr. Roberto Bifulco | Seminary | 1,5 | 0,4 | 26/02/2016 | DIETI | Yes |

**Student: Name Surname**
luigi.desimone@unina.it

**Tutor: Prof. Domenico Cotroneo**
cotroneo@unina.it

**Cycle XXIX**

| | Credits year 1 | | Credits year 2 | | | | | | | | Credits year 3 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | 1 | 2 | 3 | 4 | 5 | 6 | | | | | |
| | Estimated | Summary | Estimated | bimonth | bimonth | bimonth | bimonth | bimonth | bimonth | Summary | Estimated | Summary | Total | Check |
| **Modules** | 20 | 23 | 10 | 3 | 3 | | | | | 6 | 9 | 9 | 38 | 30-70 |
| **Seminars** | 5 | 7,3 | 5 | 1 | | | | | 0,8 | 1,8 | 0,9 | 0,9 | 10 | 10-30 |
| **Research** | 35 | 37 | 45 | 6 | 7 | 10 | 10 | 10 | 9,2 | 52,2 | 50,1 | 50,1 | 139,3 | 80-140 |
| | 60 | 67,3 | 60 | 10 | 10 | 10 | 10 | 10 | 10 | 60 | 60 | 60 | 187,3 | 180 |

## 3. Research activity

**Title**: Dependability Evaluation of Cloud Computing Ecosystems

### Description and Study

### 3.1 Dependability evaluation of cloud computing infrastructures

In this first year of my PhD, the first goal of my research it has been to study and to understand which are the challenges and open problems behind the evaluation of a cloud computing ecosystems (CCE) **dependability**. As I have depicted in my study presented at *IEEE International Symposium on Software Reliability Engineering Workshops* (ISSREW, Napoli 2014) **[P1]**, it is necessary to conduct research and develop techniques and methodologies that allow us to build countermeasures against faults, with the purpose of preventing fault propagation within a CCE and, ultimately, of avoiding failures of the CCE as a whole.

Furthermore, it is important to provide techniques and methodologies for understanding how faulty components in the CCE can affect other components and the overall CCE services, and for predicting and quantifying the impact of fault propagation on the CCE as a whole.

Recent studies have been done in testing of cloud-based applications [1], using cloud platforms to perform testing of application [2], and studies related to verification of cloud services [3]. Furthermore, other studies [4] addressed the problem of reliability of cloud infrastructure.

Nevertheless, there is still a need for approaches specifically focused on the reliability of cloud services and infrastructures against faults.

In recent years, several studies and tools have proved that **Fault Injection Testing** is a valuable approach for assessing fault-tolerant systems [5]. Fault injection is an approach in which we deliberately introduce faults in a system. This approach can assess the robustness and performance of a system in the presence of faults, and to state if fault tolerance algorithms and mechanisms are effective.

In the CCE context, **Virtualization** is cornerstone technology to set up a CCE. Virtualization allows to abstract physical resources (e.g., CPUs, network devices, storage devices, etc.) in order to share and to provide resources, making a physical machine as a soft component to use and manage very easily.

Thus, to assure the dependability of cloud systems, it is necessary to assess the reliability of the virtualization environment as a whole, focusing both on VMs and on the Hypervisor, as well as on the Cloud Management Stack software that orchestrates them (such as the well-known OpenStack framework) to efficiently manage cloud infrastructures.

Recent studies have faced with testing of components that constitute CCE, adopting fault injection to assure a high-level of reliability of cloud systems. Unfortunately, these tools are not meant for the evaluation of CCE architecture as a whole.

In addition to hardware faults, a system can be affected also by software faults and configuration faults. A fundamental part of fault injection testing is the definition of the *Fault Model*, which is a description of the types of fault that the system is expected to experience during runtime. It drives fault injection tests specifying *what* to inject, *when* to inject and *where* to inject. It is very challenging to define realistic fault models that take into account all the specifics of each CCE elements, given the complexity of these systems. Furthermore, in the CCE context, software and operator faults have not yet been studied deeply, thus there is another big question to answer.

Failures in CCEs may involve fault propagation, and due to complex interactions between different layers, it is very challenging to predict and quantify which is the impact that such a propagation could have on the CCE as a whole.

The idea is to leverage fault injection techniques to conduct such a fault propagation analysis. We can inject faults (hardware, software and configuration faults) in each layer, to understand how these faults propagate through different components and layers within CCEs. This analysis can give useful information about if there is (or not) a fault propagation path from less critical components/layers to more critical components/layers. Furthermore, we can discover new failure modes, and localize failures to the greatest extent possible. This work aims to develop framework, tools, mechanisms, and algorithms in order to detect faults and prevent their propagation within CCEs.

## 3.2 Dependability evaluation of NFV infrastructures

During the first and second year I have collaborated with a global leader company of TLC solutions, within an industrial research project with the objective to propose methodology and tools to evaluate dependability of Network Function Virtualization systems.

*Network Function Virtualization* (NFV) [6], [7] is an emerging solution to supersede traditional network equipment to reduce costs, improve manageability, reduce time-to-market, and provide more advanced services [8]. NFV will exploit IT virtualization technologies to turn network equipment into *Virtualized Network Functions* (VNFs) that will be implemented in software, and will run on commodity hardware, virtualization and cloud computing technologies located in high-performance data centers, namely *Network Function Virtualization Infrastructures* (NFVIs). Thus, NFVI can be seen as a complex cloud computing infrastructures.

### 3.2.1 Challenges and Direction for Reliability Assurance

In particular, within my research group, I have studied the challenges to assess the risks introduced by virtualization technologies for NFVI reliability **[P3]**. Towards this goal, we conduct the following activities:

- **Failure Mode and Effects Analysis of virtualization technologies in NFVIs**: we need to analyze the architecture of NFVI and its potential threats in order to understand what can affect reliability. The FMEA should consider not only hardware failures, but also failures due to software and configuration faults that can impact on virtualized resources (*e.g.*, virtual CPU, memory, network and storage);
- **Definition of Key Performance Indicators and Methodologies for NFVI reliability**: we will define measures for fault tolerance and performance, and provide guidelines to allow reliability engineers to systematically assess reliability by means of fault injection testing;
- **Design of novel Fault Injection Techniques**: because of the challenges in NFVIs (e.g., black-box technologies), the most advantageous injection target seems to be represented by the interfaces of the Compute, Hypervisor and Network domains. The errors and corruptions to be injected should be defined on the basis of the FMEA;
- **Validation using NFV products and technologies**: we will conduct a proof-of-concept validation of the fault injection approach on commercial NFV products, based on virtualization technologies such as VMware and LXC.

### 3.2.2 Dependability benchmark of NFVI

It can be easily seen that the "softwarization" process of network functions raises performance and reliability concerns. NFVIs should be able to achieve resiliency in spite of *faults* occurring within them, such as hardware, software and configuration faults. The incidence of these faults is expected to be high, due to the large scale and complexity of data centers hosting the NFVI, and due to the massive adoption of several off-the-shelf hardware and software components: while these components are easily procured and replaceable, NFVIs will need to recover from faulty components in a timely way while preserving high network performance.

The prospective NFVI requirements and architecture, currently being defined by the ETSI [9], and presented in this section, includes fault tolerance mechanisms that will be adopted in the emerging NFVIs. According to these design principles, NFVI fault tolerance mechanisms will include *fault detection*, *fault localization*, and *fault recovery* (Fig. 1).

**Fault Detection** mechanisms of the NFVI are aimed at noticing the failure of a component (such as a VM or a node) as soon as the failure occurs, in order to timely start the fault treatment process. **Fault Localization** mechanisms identify which components, among all components in the NFVI, have failed. **Fault Recovery** mechanisms of the NFVI will perform a recovery action to remediate to the faulty component.

Given the complexity of this fault management process, it becomes important to get confidence that NFVIs can achieve its strict performance and reliability requirements, which is the goal of the experimental approach proposed in the research paper **[P4]**. In that study, we show a dependability evaluation and benchmarking methodology for NFVIs. Based on fault injection, the methodology analyzes how faults impact on VNFs in terms of performance degradation and service unavailability.

The methodology includes three parts, which are summarized in Fig. 1.

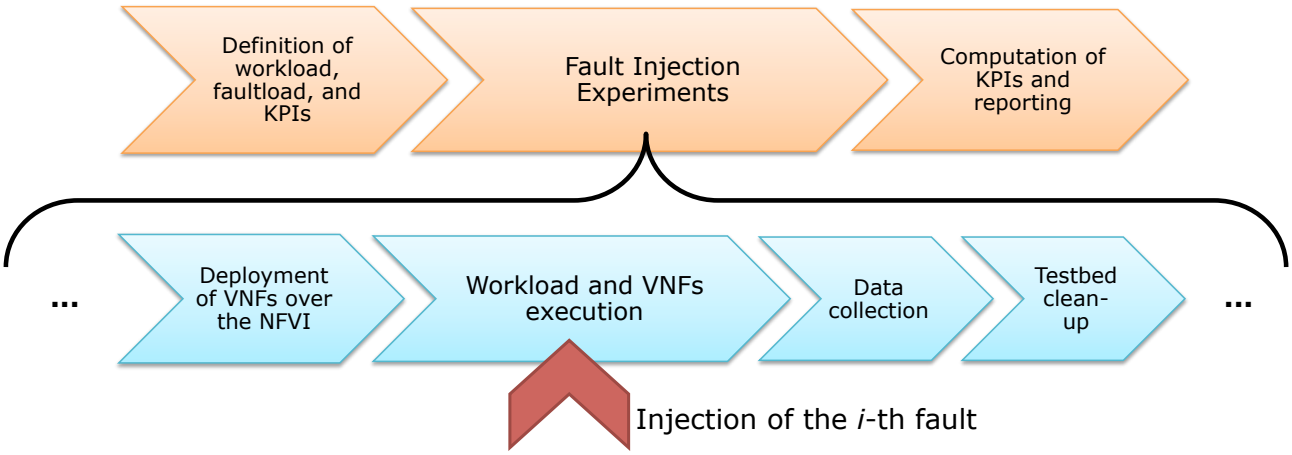


Figure 1. Overview of dependability evaluation methodology.

The first part consists in the definition of *key performance indicators* (KPIs), the *faultload* (i.e., a set of faults to inject in the NFVI) and the *workload* (i.e., inputs to submit to the NFVI) that will support the experimental evaluation of an NFVI. Based on these elements, the second part of the methodology consists in the execution of a sequence of fault injection experiments. In each fault injection experiment, the NFVI under evaluation is first configured, by deploying a set of VNFs to exercise the NFVI; then, the workload is submitted to the VNFs running on the NFVI and, during their execution, faults are injected; at the end of the execution, performance and failure data are collected from the target NFVI; then, the experimental testbed is cleaned-up (e.g., by un-deploying VNFs) before starting the next experiment. This process is repeated several times, by injecting a different fault at each fault injection experiment (while using the same workload and collecting the same performance and failure metrics).

#### 3.2.2.1 Key Performance Indicators

To evaluate performance and dependability of an NFVI, we consider the quality of service as perceived by its users. First, we define metrics for evaluating performance of an NFVI, which will be based on the responsiveness of VNFs running on the NFVI (referred to as *VNF latency* and *VNF throughput*). It is important to note that, while latency and throughput are widely adopted for characterizing performance of several types of systems, we specifically consider latency and throughput *in the presence of faults*, to quantify the *impact* of these faults, and evaluate whether the impact is too strong to be neglected.

1) ***VNF Latency and Throughput****:* The *VNF latency* is the time required by a network of VNFs to process incoming traffic, which can be evaluated by measuring the time between a unit of traffic (such as a packet or a service request) enters the network of VNFs, and the time at which the processing of that unit of traffic is completed (e.g., a packet is routed to a destination after inspection, and leaves the VNFs; or, a response is provided to the source of a request). The *VNF throughput* considers the rate at which traffic units are successfully processed, e.g., processed packets or requests per second, in the presence of faults.
2) ***Experimental Availability****:* Availability is a key aspect of quality of service. According to the *TL 9000* definition for telecommunication systems [10], [11], availability is “the ability of a unit to be in a state ready to perform a required function at a given instant in time”. The NFVI and its VNFs can become unavailable because of faulty components, causing service disruptions such as user-perceived outages, data losses and corruptions. It must be noted that, in general, availability cannot be predicted in probabilistic terms by the sole application of fault injection. Fault injection specifically

focuses on evaluating the reaction of a system *given that a fault already occurred*. The availability also depends on the probability of occurrence of faults, which relies on other factors beyond the possibilities of fault injection and of our evaluation methodology, such as the reliability of individual components.

3) ***Risk Score**:* The *Risk Score* (RS) provides a brief and concise measure of the impact of faults within the NFVI, such as the risk of experiencing service unavailability and performance failures. We take into account several factors in the evaluation of risk, including:
   a. the **type of service** and its **criticality** (in terms of number of users and importance of the service for the users);
   b. the **impact of faults** on the service as perceived by the end-users;
   c. the relative **frequency of occurrence of faults**.

   The Risk Score summarizes these factors, to provide an indication of risk for system designers, and to guide further analysis and improvements. In particular, the higher is the RS, the higher is the risk of service failures and, consequently, the worse is the capability of the underlying NFVI infrastructure to tolerate faults and assure service availability.
   The Risk Score is a weighted sum of the number of service failures in fault injection experiments. It is defined as:

$$RS = \sum_{\substack{\text{Weighted}\\ \text{average}\\ \text{over all}\\ \text{faults}}} \left( \%_{\substack{\text{Performance}\\ \text{failures}}} + \%_{\substack{\text{Availability}\\ \text{failures}}} \right)$$

#### 3.2.2.2 Fault Model

Fault injection in distributed systems encompasses two main fault categories: faults affecting I/O components (e.g., virtual network and storage), and faults affecting computational components (e.g., virtual CPUs and virtual memory). Faults in virtualized infrastructures (including hardware faults in OTS equipment, and software and configuration faults in the virtualization layer) mostly manifest as disruptions in ***I/O traffic*** (e.g., the transient loss or corruption of network packets, or the permanent unavailability of a network interface) and **erratic behavior** of the **CPU** and **memory** subsystems (in particular, corruption of instructions and data in memory and registers, crashes of VMs and physical nodes, and resource leaks).

These types of faults can be injected by emulating their effects on the virtualization layer. In particular, **I/O faults** and **Compute faults** (Fig. 2) can be emulated, respectively, by deliberately injecting I/O losses, corruptions and delays, and by injecting code and data corruptions, by forcing the termination of VMs and of their hosting nodes, and by introducing CPU and memory “hogs” (i.e., tasks that deliberately consume CPU cycles and allocate memory areas in order to cause resource exhaustion). Faults can be injected either in a specific VM (e.g., traffic from/to a VM), or in an NFVI node (affecting the hypervisor and all VMs deployed on the node).

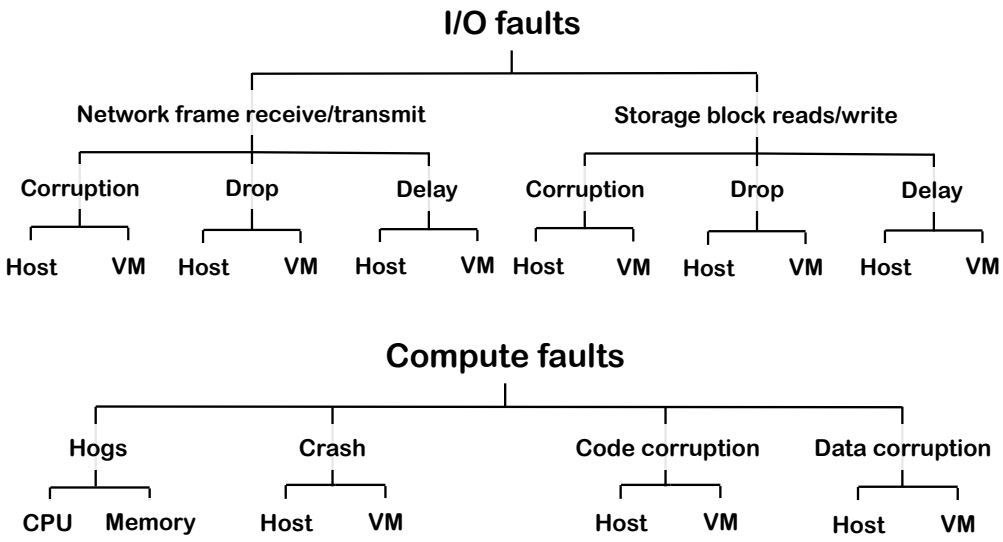


Figure 2. Faultload for the dependability evaluation of NFVIs.

#### 3.2.2.3 Workload

During fault injection tests, the NFVI has to be exercised using a workload. In order to obtain reasonable and realistic results from fault injection, these workloads should reflect the workloads that VNFs will face in production: in this way, the experiments will provide a realistic picture of performance and dependability of the NFVI. Realistic workloads are typically generated using load generators and performance benchmarking tools. Our dependability benchmarking methodology is not tied to a specific choice of workload.

#### 3.2.2.4 Case study and results

To show the application of the dependability evaluation methodology, we perform an experimental analysis of a virtualized IP Multimedia Subsystem (IMS) deployed over an NFVI.

The VNFs running on the NFVI under evaluation are from the *Clearwater* project [12], [13], which is an open-source implementation of an IMS for cloud computing platforms. Figure 8 shows the components of the Clearwater IMS that are deployed on the NFVI testbed. They are:

- ***Bono***: the SIP edge proxy, which provides both SIP IMS Gm and WebRTC interfaces to clients.
- ***Sprout***: the SIP registrar and authoritative routing proxy, and handles client authentication.
- ***Homestead***: component for retrieving authentication credentials and user profile information.
- ***Homer***: XML Document Management Server that stores MMTEL service settings for each user.
- ***Ralf***: component that provides billing services

The goal of this analysis is to provide examples of results that can be obtained from fault injection. We consider a commercial virtualization platform (the *VMware ESXi* hypervisor) running real-world, open-source NFV software. In these experiments, we adopt fault injection to analyze:

- Whether degradations/outages are more frequent or more severe than reasonable limits;
- The impact of different types of faults, to identify the faults to which the NFVI is most vulnerable;
- The impact of different faulty component, to find the components to which the NFVI is most sensitive.

The case study on the IMS showed how the methodology can point out dependability bottlenecks in the NFVI and guide design efforts. In particular, we have found these fundamental insights:

- Faults have a strong impact on availability! ***Compute faults*** and ***Sprout-VM faults*** have the strongest impact;
- Over than 10% of requests exhibit a latency **much higher than 250ms**;
- The **overall risk score (55%)** is quite high and reflects the strong impact of faults on the infrastructure;
- In our experiments, automated VM recovery was too **slow** and **availability** resulted **low**.

### 3.2.3 Quantitative Assessment of Isolation Properties in Container-based Virtualization

During my second year I have also started studies about application of new emerging virtualization technology known as container-based virtualization, within the DEEDS research group at TU Darmstadt, under the supervision of Prof. Neeraj Suri.

Virtualization technologies are the core enabler of cloud computing. Several techniques [14], [15] are currently available, and the most important ones are:

- Full-virtualization;
- Para-virtualization;
- Container-based virtualization.

Container-based virtualization, also called *Operating System-level* virtualization, allows running multiple appliances without hardware virtualization. Resource virtualization in OSs is an old idea, based on the concept of process. The idea behind container-based virtualization is to enhance the abstractions of OS processes (now called containers), by extending the (host) OS kernel. In container-based virtualization, a container has its own virtual CPU and virtual memory (like in traditional OS processes). In this kind of virtualization, it is also added abstractions for a virtual filesystem (i.e., the container perceives a filesystem structure that is different than the host's), virtual network (i.e., the container sees a different set of networking interfaces), IPC, PIDs, and users management. These virtual resources are distinct for each container in the system.

We are witnessing an increasing use of container-based virtualization in cloud infrastructures. Examples are Google [16], [17], Amazon [18], and Microsoft [19], which are already providing cloud services running on containers. Containers are expected to be fast, thus providing high performance, since there is no extra overhead due to emulation of devices. Moreover, containers make virtualization more manageable, since creating and moving containers is easier and faster. NFV infrastructures require extremely low packet processing overheads, controlled latency, automatic recovery from faults, and extremely high availability (99.99% or higher); container-based virtualization can be one of the more suitable solution for NFV.

In general, virtualization has to address the critical problem of guarantee the *isolation* among virtual instances [20]. In the more general sense, isolation means the fact that something is independent and disentangled to the behaviors of other things. Thus, the virtualization layer has to be in complete control of virtualized resources, and applications running on a virtual domain must have the illusion to be completely isolated from others. Unlike full-virtualization, in container-based virtualization, the "*virtualization surface*" (i.e., the isolation boundary) between virtual domains and the host is minimal, thus the likelihood of having interferences (e.g., failures in a container may affect the entire infrastructure) can be very high!

The isolation property has two main flavors:

- **Performance isolation**: the capability of isolating or limiting the impact of resource consumption (e.g., CPU, network, disk) of container on the performance degradation of the other containers, and the host kernel;

- **Memory isolation**: the capability of isolating code and data between containers, and between containers and the host kernel.

**Title**: Reliability evaluation in device drivers

### 3.2.4 Run-Time Monitoring for I/O Protocol Violations in Storage Device Drivers

Another important piece of my research is about reliability in device drivers. It is well known that device drivers are the most bug-prone part of the OS [21], [22], [23], and represent a critical component of every storage stack in IT systems. Bugs affecting storage device drivers include the so-called *protocol violation bugs*, which silently corrupt data and commands exchanged with I/O devices. Protocol violations are very difficult to prevent, since testing device driver is notoriously difficult. To address them, we present in **[P5]** a monitoring approach for device drivers (MoIO) to *detect I/O protocol violations* at run-time. The approach *infers* a model of the interactions between the storage device driver, the OS kernel, and the hardware (the *device driver protocol*) by analyzing execution traces. The model is then used as a reference for detecting violations in production. The approach has been designed to have a low overhead and to overcome the lack of source code and protocol documentation. We show that the approach is feasible and effective by applying it on the SATA/AHCI storage device driver of the Linux kernel, and by performing fault injection and long-running tests.

# 4. Products

In this second year, I have produced the following products.

## 4.1 Publications

### Conference Paper

**[P4]** Domenico Cotroneo, **Luigi De Simone**, Antonio Ken Iannillo, Anna Lanzaro, Roberto Natella, "*Dependability Evaluation and Benchmarking of Network Function Virtualization Infrastructures*", at 1st IEEE Conference on Network Softwarization (NetSoft), pp. 1 – 9, 13-17 April 2015 London, DOI: 10.1109/NETSOFT.2015.7116123
**BEST PAPER AWARD**

**[P5]** Domenico Cotroneo, **Luigi De Simone**, Francesco Fucci, Roberto Natella, "*MoIO: Run-time monitoring for I/O protocol violations in storage device drivers*", at Software Reliability Engineering (ISSRE), 2015 IEEE 26th International Symposium on, pp. 472 – 483, 2-5 Nov. 2015 Gaithersbury, MD, DOI: 10.1109/ISSRE.2015.7381840

# 5. Activity abroad

During my second year I spent 2 months (September 2015 – October 2015) at TU Darmstadt within the DEEDS group, under the supervision of Prof. Neeraj Suri. I have started studies about application of new emerging virtualization technology known as container-based virtualization, in order to quantitatively assess isolation properties from the perspective of dependability.

# 6. Tutorship

During the second year I have been teaching assistant for the course of Operating Systems, a.a. 2014/2015. Furthermore, I have been MSc thesis co-advisor on topic about verification of fault tolerant mechanisms in NFV.

## References

[**P1**] De Simone, L., "Towards Fault Propagation Analysis in Cloud Computing Ecosystems," Software Reliability Engineering Workshops (ISSREW), 2014 IEEE International Symposium on , pp.156,161, 3-6 Nov. 2014, DOI: 10.1109/ISSREW.2014.47

[**P2**] Cotroneo, D.; De Simone, L.; Iannillo, A.K.; Lanzaro, A.; Natella, R.; Jiang Fan; Wang Ping, "Network Function Virtualization: Challenges and Directions for Reliability Assurance," Software Reliability Engineering Workshops (ISSREW), 2014 IEEE International Symposium on , pp.37,42, 3-6 Nov. 2014, DOI: 10.1109/ISSREW.2014.48

[**P3**] Cotroneo, D.; De Simone, L.; Iannillo, A.K.; Lanzaro, A.; Natella, R., "Improving Usability of Fault Injection," Software Reliability Engineering Workshops (ISSREW), 2014 IEEE International Symposium on, pp.530,532, 3-6 Nov. 2014, DOI: 10.1109/ISSREW.2014.37

[1] X. Bai, M. Li, B. Chen, W.-T. Tsai, and J. Gao, "Cloud testing tools," in *Proc. Intl. Symp. SOSE*, 2011, pp. 1–12.

[2] L. Ciortea, C. Zamfir, S. Bucur, V. Chipounov, and G. Candea, "Cloud9: A software testing service," *SIGOPS Operating System Review*, vol. 43, no. 4, pp. 5–10, Jan. 2010.

[3] S. Bouchenak, G. Chockler, H. Chockler, G. Gheorghe, N. Santos, and A. Shraer, "Verifying cloud services: Present and future," *SIGOPS Operating System Review*, vol. 47, no. 2, pp. 6–19, Jul. 2013.

[4] A. Bessani, R. Kapitza, D. Petcu, P. Romano, S. V. Gogouvitis, D. Kyriazis, and R. G. Cascella, "A look to the old-world sky: EU- funded dependability cloud computing research," *SIGOPS Operating Systems Review*, vol. 46, no. 2, pp. 43–56, Jul. 2012.

[5] R. Natella, D. Cotroneo, J. Duraes, and H. Madeira, "On fault representativeness of software fault injection," *Software Engineering, IEEE Transactions on*, vol. 39, no. 1, pp. 80–96, Jan 2013.

[6] NFV ISG, "Network Functions Virtualisation - An Introduction, Bene- fits, Enablers, Challenges & Call for Action," ETSI, Tech. Rep., 2012.

[7] NFV ISG, "Network Functions Virtualisation (NFV) - Network Operator Perspectives on Industry Progress," ETSI, Tech. Rep., 2013.

[8] A. Manzalini, R. Minerva, E. Kaempfer, F. Callegari, A. Campi, W. Cerroni, N. Crespi, E. Dekel, Y. Tock, W. Tavernier *et al.*, "Manifesto of edge ICT fabric," in *Proc. ICIN*, 2013, pp. 9–15.

[9] NFV ISG, "Network Function Virtualisation (NFV) - Resiliency Requirements," ETSI, Tech. Rep., 2014.

[10] E. Bauer and R. Adams, *Reliability and Availability of Cloud Computing*, 1st ed. Wiley-IEEE Press, 2012.

[11] Quality Excellence for Suppliers of Telecommunications Forum (QuEST Forum), "TL 9000 Quality Management System Measurements Handbook 4.5," Tech. Rep., 2010.

[12] Clearwater, "Project Clearwater - IMS in the Cloud," 2014. [Online]. Available: http://www.projectclearwater.org/

[13] G. Carella, M. Corici, P. Crosta, P. Comi, T. M. Bohnert, A. A. Corici, D. Vingarzan, and T. Magedanz, "Cloudified IP Multimedia Subsystem (IMS) for Network Function Virtualization (NFV)-based architectures," in *Proc. ISCC*, 2014.

[14] A. S. Tanenbaum and H. Bos, *Modern Operating Systems*, 4th ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2014.

[15] D. C. van Moolenbroek, R. Appuswamy, and A. S. Tanenbaum, "Towards a flexible, lightweight virtualization alternative," in *Proceedings of International Conference on Systems and Storage*, 2014, pp. 8:1–8:7.

[16] Google Inc. Google Cloud Platform. [Online]. Available: https: //cloud.google.com/container- engine/

[17] ——. Kubernetes. [Online]. Available: http://kubernetes.io/

[18] Amazon. Amazon EC2 Container Service. [Online]. Available: http://aws.amazon.com/it/ecs/

[19] Microsoft Azure Team. New Windows Server containers and Azure support for Docker. [Online]. Available: http://azure.microsoft.com/blog/2014/10/15/ new- windows- server- containers- and- azure- support- for-docker/?WT. mc id=Blog ServerCloud Announce TTD

[20] E. Bugnion, S. Devine, M. Rosenblum, J. Sugerman, and E. Y. Wang, "Bringing Virtualization to the x86 Architecture with the Original VMware Workstation," *ACM Transactions on Computer Systems (TOCS)*, vol. 30, no. 4, 2012.

[21] A. Chou, J. Yang, B. Chelf, S. Hallem, and D. Engler, "An empirical study of operating systems errors," in *SOSP '01*. Average execution time [s]

[22] A. Ganapathi, V. Ganapathi, and D. A. Patterson, "Windows XP Kernel Crash Analysis," in *LISA'06*.

[23] N. Palix, G. Thomas, S. Saha, C. Calve`s, J. Lawall, and G. Muller, "Faults in Linux: Ten years later," in *ACM SIGARCH Computer Architecture News*, vol. 39, no. 1, 2011.