A Test-Based Self-Healing Mechanism for Service Function Chains

Cuong Le
Department of Information Communication Convergence
Soongsil University
Seoul, Korea Republic of.
cuonglv@soongsil.ac.kr

Myungsik Yoo
School of Electronic Engineering
Soongsil University
Seoul, Korea Republic of.
myoo@ssu.ac.kr

***Abstract*— Network functions virtualization (NFV) is a network architecture concept that uses the technologies of IT virtualization to virtualize entire classes of network node functions into building blocks. The network functions of an NFV network do not require any dedicated hardware. They only need a general server to be implemented and executed. Moreover, NFV has an important concept that is the service functions chain (SFC). It includes more than one network functions that work together to form a network service. NFV brings advantages to deploy network service. However, it also has challenges that need to be considered, especially fault recovery. Almost current solutions are passive self-healing. Although there are some efforts toward active self-healing, a comprehensive solution still has not been proposed. The proposed mechanism is based on testing to build rule-based solutions for self-healing in the deploying phase. Even though our approach has not yet achieved completely active self-healing, it is partly proactive than existing works.**

Keywords—NFV, SFC, Container, Self-healing, Multi-agent

I. INTRODUCTION

The Network Function Virtualization architectural framework was proposed by the European Telecommunications Standards Institute (ETSI) [2] to transform the way that network operator architects and manage networks by using virtualization technology. Virtualized network functions (VNFs) are software applications that deliver network functions such as file sharing, directory services, and IP configuration, etc. They are packaged as Virtual Machines (VMs) or containers on commodity servers to replace specialized network appliances. Depending on service requirements, an ordered set of VNFs and subsequent "steering" of traffic through them termed Service Function Chain (SFC) is created to meet. The NFV architecture framework proposed by ETSI is presented in Figure 1.

Besides those benefits, NFV creates many challenges that require more effort to research and collaboration to resolve. One of the most critical issues in NFV is fault recovery, which is how to return the system to its conventional operation as one or more failures appear. Due to the software nature of NFV, NFV developers usually design self-healing techniques, but the current implementation is quite simple, such as using ping or periodically healthy checking to acknowledge the degraded VNF and replace it with a new one. Nevertheless, it is expensive because VNF replacement often requires longer downtime and more resources. Additionally, the root cause of the failure can be fixed by just restarting the service inside the running VNF or allocating more resources.

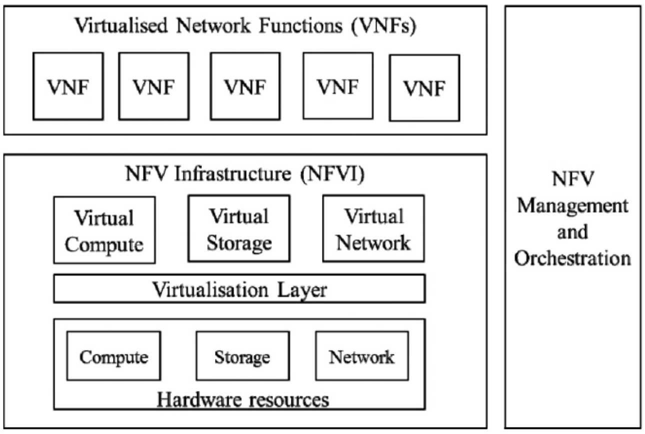

Figure 1: NFV Architecture

In [3], the authors introduced a mechanism that supports performance evaluation and functional correctness of SFCs before deploying. They aimed at providing a holistic solution for testing service function chains based on agents. However, they have not mentioned about solutions for failure occurred in deploying. In this paper, we focus on fault recovery service function chains in deploying. We use the proposed mechanism in [4] to make an architecture using a rule-based diagnosis algorithm in testing. Owing to that, we can apply fault recovery for service function chains in deploying. In [5], a mechanism distributed monitoring using Docker as the next-generation container virtualization technology is presented. The main idea presented herein is the use of containers to distribute software components.

The rest of the paper is structured as follows. Section II introduces the technologies and related concepts that will be used. Section III presents the architecture and the workflow of the proposed self-healing mechanism. Finally, section IV concludes this paper by summarizing the achievement of research and discussing possible future steps and enhancements of the proposed mechanism.

II. BACKGROUND

A. Self-healing

Self-healing represents the autonomic capability of a component or system to detect and recover from problems (manifestations of faults, errors, failures, and other forms of degradation), continue to function smoothly [1], and maintains it in a healthy state. More mechanisms have been developed and improved to achieve self-healing [2]. Self-healing characteristics are availability, fault-tolerance, maintainability, survivability, reliability, stabilization, and human-assistance if needed. The self-healing architecture takes the shape of a control-loop mechanism composed of three blocks, namely, diagnosis, impact analysis, and recovery, which receives information from the controlled components through detection failure agents and feedbacks action upon them through the mechanisms.

Two following data types are considered by a self-healing system: alarms that contain the state of the network elements to detect failures (wrong states) and performance metrics of resources to measure degradations (degraded states). These metrics are various: availability, performances or resiliency of services, network functions, and resources. Metrics can be common (delay, jitter, or throughput), specifications (e.g., the time of response of the controller to the switches' requests in Software-Defined Networking (SDN)). In case of failure or degradation, the diagnosis block looks up the root reason and trigger the proper recovery algorithm to resolve that specific problem. This execution is converted into reconfiguration or re-adjustment actions on the managed system to change its state.

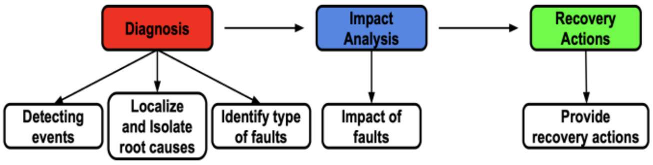

Figure 2 The Faults Management Process

B. OpenStack Cloud Platform

OpenStack is a set of software tools for building and managing cloud computing platform for public and private cloud. Mostly it is deployed as an Infrastructure-as-a-Service (IaaS). It is a major player in cloud computing. All of its resources and components are controlled and provisioned through REST APIs with centralized authentication methods. It has a user-interface dashboard giving cloud administrators the control while allowing users to provision resources and services through a modern web interface. OpenStack offers a couple of cloud-related services like networking, storage, image services, identity, etc. are offered by OpenStack [6]. They can be clubbed with a few components provided by the third-party to get a customized cloud optimization to support the cloud-native apps. Besides standard IaaS features, OpenStack has extra components that offer orchestration, failure management, and service management, backup and disaster recovery, etc.

C. Docker Platform

Docker is a containerization platform [7]. It extends existing container technology by providing a single lightweight [8], API for managing Docker images and execution of containers, and even a tool for defining and running multi-container applications. Docker can also be used to deploy an application regardless of environment settings from the environment to another one [9]. By using Docker to create any of our network functions in service function chains as a container, we not only save resources but also simple to deploy [10].

Docker container uses the same OS as the hosting node, i.e., no OS installation [11], and only consume resources when they run an application.

To overcome the drawbacks of existing works, a test-based self-healing mechanism for the SFC is proposed. Detail architecture of the proposed system is described in Section III.

III. The Test-Based Self-Healing Mechanism For Service Function Chains

Solving issues of the previous work, the proposed mechanism is trying to provide the ability to recover the service function chain in deploying rather than only testing. From there, this approach towards a solution more active. In this paper, our contribution is a proposed self-healing mechanism for SFC that not only helps SFC provide reliable on-demand network services but also makes sure it has the ability to self-recovery in deployment. The procedure includes the testing phase in which all of the problems are completely solved and saved as a rule-based self-healing solution for use in the deploying phase.

A. Detail architecture

The architecture of the proposed mechanism comprises four main components, the executor agent, the controller, the fault analyzing, and the executor, as shown in Figure 3. The executor agent built as a Docker container is installed in each VNF component. It will execute test commands inside the VNF following by the instruction of the controller. It also collects data sent to the controller from the VNF. Besides, the executor agent can execute the solution script inside the VNF following by the solutions of the controller faults when needed to deal with VNF faults.

Agents have to capability to communicate with each other to test, watch, and troubleshoot the VNFs within an SFC (throughput, anomaly, requests, etc.). Different kinds of agents can interact with the controller by an appropriate agent driver in the controller. The most important self-healing technique is to leverage the controller-agents connection.

The controller has an important role in both testing and deploying phases. It can get test cases from testing libraries and inject faults into VNFs through agents. Besides, it may receive fault logs by agents and transfer to the fault feature analysis. The controller also sends the rules to agents.

The fault feature analysis receives fault logs from the controller and extracts them. The result will be used to build the rules and store in the solution storage used when needed.

The checking will determine whether the fault extracted is known or unknown. The unknown fault storage will collect all unknown faults. The executor responsible to find and sent the appropriate rules for any known fault to the controller.

B. Workflow

The mechanism completely building and developing a service function chain is including 2 phases:

In the testing phase, the engineer will update the testing libraries using the unknown fault log from the deploying phase. The controller will get the test case from testing libraries. Then, it requires the agent to inject fault (e.g., loads, packages, etc.) into the VNF. The controller may receive the fault log sent back from the agent. After that, it will transfer the log to the fault feature analysis. The result will be used to create self-healing rules. The rules are created to solve various issues of the service function chain if it appears in the deploying stage.

In the deploying phase, the controller may receive the fault log sent from the agent. After that, it will transfer the log to the fault feature analysis. Herein, if the fault is known, the executor will find appropriate rules built from solution storage. It applies to the controller. And then, the rules will be executed by agents inside the VNF. In contrast, if the fault is unknown, they will be stored in the unknown faults storage to be built and improved testing libraries.

By this approach, we can actively solve the fault by testing them before they appear.

IV. Conclusions and Future Works

Even though self-healing in NFV is a mature topic, state-of-the-art works only focus on complete procedure creating test library and self-healing of service function chains. Nowadays, everything towards automation and continuous cost reductions using humans to manually trouble shooting and trigger corrective when needed to fulfill these recoveries are considered complexity and too expensive.

We aim to complete procedure from testing to deploying a service function chains. This paper introduces a mechanism for building a rule-based in the testing stage. Besides, in deploying, the mechanism saves information failures that still have not any solution in order to build the new test cases for testing.

Future works will have an implementation of this proposed mechanism to illustrate its feasibility and extensibility. Besides, machine learning approaches shall also be considered.

Acknowledgment

“This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2020-2017-0-01633) supervised by the IITP(Institute for Information & communications Technology Planning & Evaluation)”

References

[1] S. Cherrared, S. Imadali, E. Fabre, G. Gössler and I. G. B. Yahia, "A Survey of Fault Management in Network Virtualization Environments: Challenges and Solutions," in IEEE Transactions on Network and Service Management, vol. 16, no. 4, pp. 1537-1551, Dec. 2019, doi: 10.1109/TNSM.2019.2948420.

[2] ETSI, Network Function Virtualisation (NFV); Architectural Framework[Online].Available: https://www.etsi.org/deliver/etsi_tr/103400_103499/103495/01.01.01_60/tr_103495v010101p.pdf

[3] Trinh Nguyen, Myungsik Yoo (2019). An Agent-Based Mechanism for Testing SFC. 한국통신학회논문지, 44(10), 1949-1955

[4] H. Mfula and J. K. Nurminen, "Self-Healing Cloud Services in Private Multi-Clouds," *2018 International Conference on High Performance Computing & Simulation (HPCS)*, Orleans, 2018, pp. 165-170, doi: 10.1109/HPCS.2018.00041.

[5] S. Dhakate and A. Godbole, "Distributed cloud monitoring using Docker as next generation container virtualization technology," 2015 Annual IEEE India Conference (INDICON), New Delhi, 2015, pp. 1-5, doi: 10.1109/INDICON.2015.7443771.

[6] “OpenStack Trains Release Notes”, [online] Available: https://www.openstack.org/trains

[7] “Docker”, [online] Available: https://docker.com/

[8] “LXC”, [online] Available: https://linuxcontainers.org/

[9] J. Nickoloff, “Docker in Action”, Manning, 2016

[10] I. Miell, A. H. Sayers, “Docker in Practice”, manning, 2016

[11] J. Carnell, “Sping Micoservices in Action”, Manning 2017

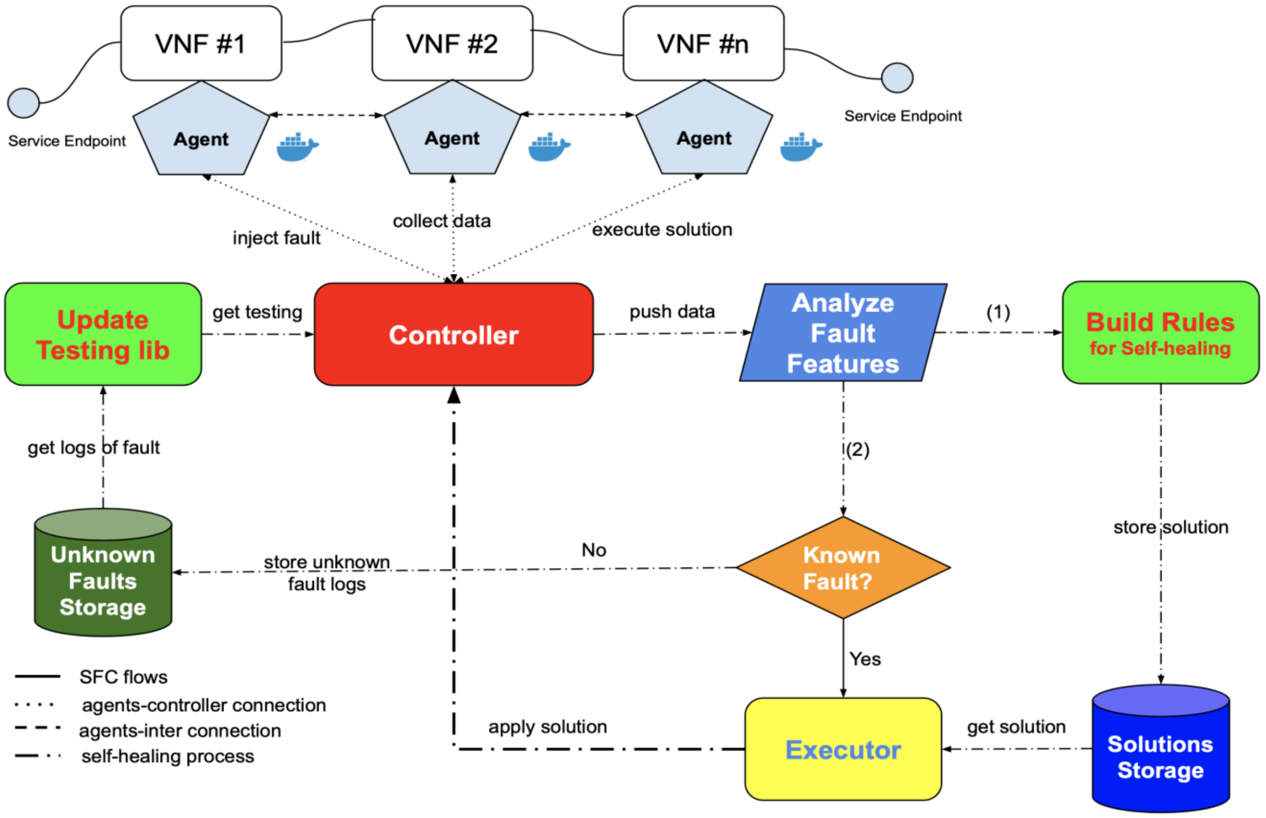

Figure 3: The architecture of the proposed mechanism