A Self-healing Platform for the Control and Management Planes Communication in Softwarized and Virtualized Networks

Natal Vieira de Souza Neto[a], Daniel Ricardo Cunha Oliveira[b], Maurício Amaral Gonçalves[c], Flávio de Oliveira Silva[d] and Pedro Frosi Rosa[e]
Faculdade de Computação, Universidade Federal de Uberlândia, Uberlândia, Brazil
{natalneto, drcoliveira, mauricioamaralg, flavio, pfrosi}@ufu.br

Keywords: Self-management, Self-healing, Fault Tolerance, SDN, NFV.

Abstract: Future computer networks will possibly use infrastructures with SDN, and NFV approaches to satisfy the requirements of future applications. In these approaches, components need to be monitored and controlled in a highly diverse environment, making it virtually impossible to solve complex management problems manually. Several solutions are found to deal with the data plane resilience, but the control and management planes still have fault tolerance issues. This work presents a new solution focused on maintaining the health of networks by considering the connectivity between control, management, and data planes. Self-healing concepts are used to build the model and the architecture of the solution. This position paper places the solution as a system occurring in the management plane and is focused on the maintenance of the control and management layers. The solution is designed using widely accepted technologies in the industry and aims to be deployed in companies that require carrier-grade capabilities in their production environments.

1 INTRODUCTION

Software-Defined Networking (SDN) and Network Functions Virtualization (NFV) are critical enablers for future computer networks, such as 5G (Yousaf et al., 2017), and therefore, unresolved problems in these approaches demand careful attention. Fundamental issues in SDN include network healthiness, controller channel maintenance, consistency, bugs and crashes in SDN applications, and so on (Abdelsalam, 2018) (Cox et al., 2017). NFV also has pending challenges, mainly associated with resources management, distributed domains, and the integration with SDN (Mijumbi et al., 2016).

In traditional architectures, the network health is managed by distributed protocols running on the network itself. In SDN, there are no specific protocols for detecting and healing failures (such as congestion and inconsistency), for network recovery, and link availability. The SDN controller should address all these issues, and the most adopted SDN controllers conveniently have procedures to deal with failures occurring in the data plane. An unresolved problem – addressed in this paper – is the communication maintenance between the data plane with control and management planes, primarily focused on in-band traffic.

The solution designed in this paper aims to deal with the control and management planes connectivity failures. Mainly, the network switches are managed by an SDN controller and require uninterrupted connectivity with this controller. Similarly, the NFV management components require constant connectivity with resources under management.

Our solution catalogs the control and management components (and the logical topology involving them), maintain backup paths in the logical topology, predicts failures, and applies the alternative routes to recover the system's health. The solution architecture adopts self-healing concepts and is focused on the control and management planes connectivity, which is not found in other works. Besides that, we propose a platform running entirely at the application level, which is not found in other projects either.

The remaining of the paper is structured as follows. Section 2 introduces the connectivity issues related to control and management planes. Section 3 presents the solution, Section 4 gives some related work, and finally, Section 5 presents the conclusion.

[a] https://orcid.org/0000-0001-5047-4106
[b] https://orcid.org/0000-0003-4767-5518
[c] https://orcid.org/0000-0002-6985-638X
[d] https://orcid.org/0000-0001-7051-7396
[e] https://orcid.org/0000-0001-8820-9113

Neto, N., Oliveira, D., Gonçalves, M., Silva, F. and Rosa, P.
A Self-healing Platform for the Control and Management Planes Communication in Softwarized and Virtualized Networks.
DOI: 10.5220/0009465204150422
In *Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSER 2020)*, pages 415-422
ISBN: 978-989-758-424-4

2 BACKGROUND

The main problem of applying self-healing in computer networks is the number of nodes running simultaneously in the environment (Thorat et al., 2015), and is related to the data plane. However, there is another crucial problem when using SDN/NFV: the controller and Operations, Administrations and Management[1] (OAM) connectivity will eventually fail. Therefore, the maintenance of the controller channel reliability is essential (Rehman et al., 2019).

A computer network is usually represented as a graph. A node represents a Network Element (NE) – such as switches, routers, gateways etc –, and edges represent links between NEs. Common problems are hardware or software failures, and broken or congested links (d. R. Fonseca and Mota, 2017). An SDN environment has a second graph representing connections between the control elements. Besides that, the infrastructure resources eventually provided by NFV platforms are controlled by systems running in a management plane.

In SDN/NFV, a failed component on the data plane can be isolated without significant problems. However, the isolation of control or management planes elements may eventually leave the network in a non-operational state. An example is the routing process. Typically, routes in an SDN architecture are defined by the SDN routing module running in the control plane. If this process is isolated, the routes will not be defined. In NFV, the orchestrators and managers need stable communication with the computer nodes constantly. This paper assumes that a self-healing service should react to failures in the control or management planes. Failures in the data plane are not addressed here, as the SDN controller should ensure the data plane health (Chandrasekaran et al., 2016).

Figure 1 shows a common SDN environment. The control plane is separated from the data plane. The data plane is composed by some NEs and their connections (links). The control plane includes the SDN controller which is centralized or distributed, and the OAM systems operating in the environment. It could be inferred that some SDN applications are running in the controller. Other elements also run in this kind of environment, but the figure is a basic representation.

It is important to note the representation of the connections between NEs and control plane elements shown in Figure 1. Continuous lines represent physical connections and dashed lines mean logical connections. The $SDNC_1$ has just one physical connection (with NE_2) but has logical connections with all NEs. It means that $SDNC_1$ controls NE_1, NE_2, ..., NE_8 and has complete information about the topology shown in Data plane. The OAM_1 also has logical connections with all NEs. This happens because the OAM could be monitoring and operating resources plugged by the NEs.

[1]In this paper, OAM means all elements used in control and management planes that are not placed inside the SDN controller (including NFV components).

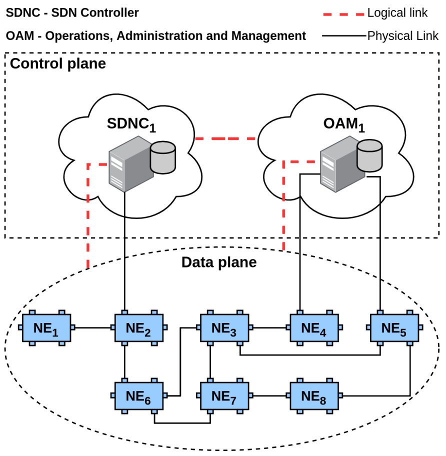

Figure 1: SDN common environment.

The control plane is logically, but not physically separated from the data plane, that is, the control primitives are separated from the data packages, but they also pass through the data plane switches. The first problem, in this case, is that a failure in a switch will affect both planes. For instance, if NE_2 has a problem, the entire control plane will be isolated. In this way, the self-healing system performs solutions to avoid overload of the NE_2. The basic idea is to predict failures before they become real. An example is the congestion of control links. The self-healing system may conclude that one or more links will be congested in the future. If the congested links cause damages in the communication between control and management elements with the NEs, the failures need recovery with high priority.

A feasible solution is to maintain alternative paths for the communication between NEs and controller, as well as for the communication between OAMs and their managed resources. Our solution aims to notice that a path will be congested or have failed, and apply an alternative path. The backup paths technique is actually essential to deal with fault tolerance in the control plane (Rehman et al., 2019). We designed an architecture, which takes advantage of autonomic computing fundamentals (such as self-healing), to build a system to deal with fault tolerance for the control and management planes connectivity.

3 SELF-HEALING PLATFORM FOR ADVANCED NETWORK LAYERS COMMUNICATION

The solution proposed in this paper is a novel platform that should monitor and maintain the health of the management and control planes connectivity. Figure 2 shows a new view of the key layered planes found in the standards specifications. The Client Layer is out of scope in this document, while the other planes compose the network layer. Our solution is placed at the Management Layer, and aims to monitor the connectivity between the other network layers. The delimited domain of our proposal is the Infrastructure Layer topology in addition with the components from Management, Application and Control Layers.

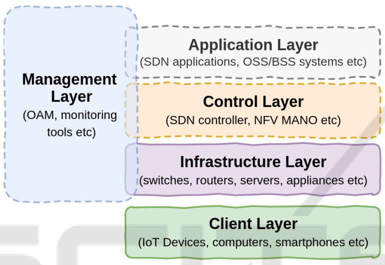
Management Layer
(OAM, monitoring tools etc)
Application Layer
(SDN applications, OSS/BSS systems etc)
Control Layer
(SDN controller, NFV MANO etc)
Infrastructure Layer
(switches, routers, servers, appliances etc)
Client Layer
(IoT Devices, computers, smartphones etc)

Figure 2: Network planes by layers.

The components from the Management, Application and Control Layers are physically (or virtually) placed on elements from the Infrastructure Layer. It means that the dashed layers in Figure 2 are just logical planes. Sometimes, the SDN controller is deployed in an out-of-band control network. In other approaches, SDN controller deployment is executed using in-band traffic, which means that the NEs are in the Infrastructure Layer (data plane) and the control and data primitives share these NEs.

3.1 Design and Specification

We propose a software architecture based on microservices and event-driven concepts. Figure 3 shows the architecture design. The Infrastructure Layer contains NEs and servers monitored by the platform. This section gives a brief description of the platform components.

The system components illustrated in Figure 3 are executed on containers. Our first implementation is deployed on a container manager built by ourselves, but it is not complex to run the solution on a commercial container orchestration solution. We assume that the network is already initialized, and then the monitoring is performed by the Collecting Entity (CoE) and the Control Primitives Interceptor (CPI).

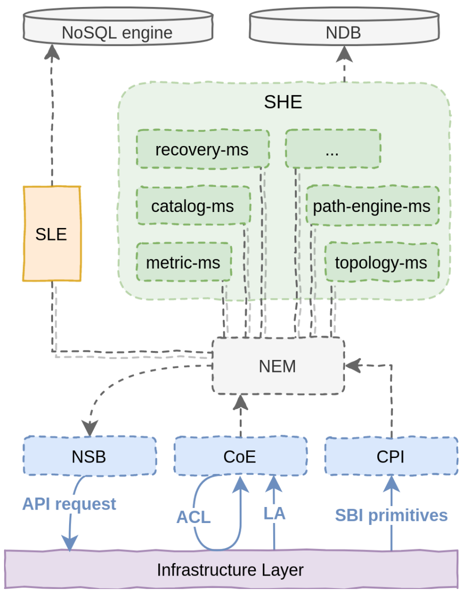

Figure 3: System architecture.

The CoE has two services to collect network information: topology-service and metric-service. The first one collects network topology information, such as added/removed links, new switches plugged, port up/down etc. The second one collects metrics in the NEs, such as port usage in bytes. There are two techniques used by CoE: the Autonomic Control Loop (ACL), and the Local Agent (LA). The ACL monitors the NEs periodically, while LAs are agents (inserted into NEs) which summarize information and send them to the CoE. The other collector component is the CPI, which is a proxy placed in the Southbound Interface (SBI). It copies and parses OpenFlow control primitives looking for metric primitives (such as ofpmp-port-stats).

According to our experience, the three strategies (ACL, LA and interception) are enough to monitor the entire network and cover all monitoring techniques found in the literature. We intend to perform an evaluation of the benefits and disadvantages of these techniques in the future.

All data collected by the CoE and CPI are converted into events and published in the Network Event Manager (NEM). The NEM is a publisher/subscriber message broker where the events are published in topics, in a way that more than one application can subscribe to one or more topics. The Self-Healing Entity (SHE) includes the microservices (ms) used to apply our self-healing algorithms. The current five ms specified in the SHE are summarized as follow:

- topology-ms: it subscribes to topology information topics and its function is to build the logical topology between control components (SDN controller) and their managed NEs, and the logical topology between NFV components and their servers (servers where the managed resources are deployed). Examples of NFV components include Virtual Infrastructure Manager (VIM), VNF Manager (VNFM) and so on;
- catalog-ms: it creates a catalog that contains information about the control and management components, their NEs, and also their management servers. Moreover, it contains information related to the necessary commands to integrate with control and management components;
- path-engine-ms: once the logical topologies have been built, this function calculates all possible paths between each node to its control component;
- metric-ms: the algorithms executed in this microservice analyze all metrics received. If a metric indicates a network control path failure, the metric-ms publishes an event in the NEM;
- recovery-ms: after a failure in a control path, the recover-ms is responsible to read the catalog, build the recovery commands, and publish the recovery events in the NEM.

The SHE microservices communicate with each other using events, through the NEM, just like the Self-Learning Entity (SLE). The SLE runs prediction algorithms, looking for a future failure in the control paths. If a failure is predicted, the SLE sends an event to the NEM. The recovery-ms receives the event and performs actions to avoid the failure. At this moment, our development is focused on basic math functions (such as linear growth of link usage) to find link congestion, but Machine Learning (ML) algorithms could be applied by the SLE in the future.

Prediction functions and ML require high storage volume. Because of this, all metrics collected are used at real-time by metric-ms, but also stored in a dedicated NoSQL database. All other information is stored in the Network Database (NDB). The NDB stores topology, catalogs, operational information and so on.

All decisions took by the recovery-ms are published as events in the NEM and received by the Network Service Broker (NSB). The NSB integrates with the SDN controllers and NEs. The SDN controllers usually provide open Application Programming Interface (API), and the NSB converts the event (received from the NEM) in the message format accepted by the SDN controller API.

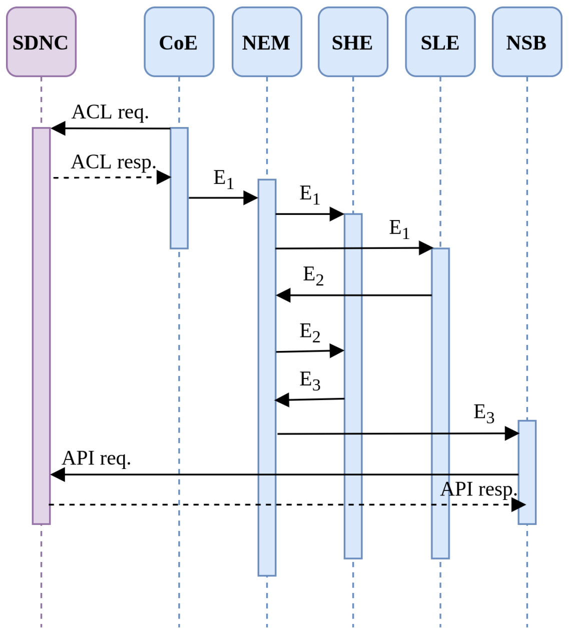

Figure 4: Recovery use case sequence diagram.

Figure 4 presents a sequence diagram showing how a metric (collected by the CoE) is converted to events and travels through the architecture. The CoE applies the ACL in an SDN controller (SDNC) requesting information about some devices (NEs). In this example, the SDNC could be the ONOS platform, which exposes metric information about all devices in a HTTP REST API. The CoE performs the request and converts the metric in an Event (E). The E_1 is published in a topic in the NEM, and all components subscribed to this topic receive E_1 (in this example, SHE and SLE).

In Figure 4, E_1 did not impact SHE. However, the SLE predicted a problem after the analysis of E_1. The SLE publishes a new event (E_2) in NEM, and this event is received by SHE, because the recovery-ms subscribed to the topic of this event. The SHE publishes an E_3 with information about the necessary modifications in NEs. The E_3 is received by the NSB which creates a message and sends it to the SDNC. The NSB also receives the response, with the information indicating whether the flow rule modifications were applied.

Figure 4 demonstrates that the integration between the system components is executed by events. The synchronous communications in Figure 4 are only the requests and responses directed to the SDNC. The figure gives just an example, but all other communications in the platform are analogous.

3.2 The Control Path Catalog

The integration of our solution with the control/management components intends to be autonomous. The CoE must discover management components, but this is not specified here due to lack of space. Therefore, we assume that the NDB has information about the SDN controllers in the topology, and about NFV management components. The CoE and NSB can integrate (through open APIs) with the components to read information about managed elements and to procedure recovery respectively.

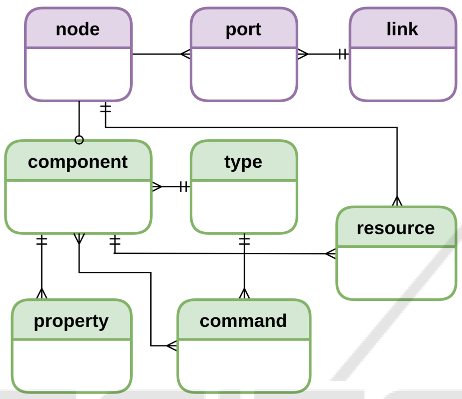

Figure 5: NDB entity-relationship model.

Figure 5 gives the NDB diagram. The attributes in each table are not displayed due to lack of space. The 'node', 'port' and 'link' tables store the network infrastructure topology, which is retrieved from SDN controllers running in the domain. The other tables represent the catalog proposed in this paper. The 'component' table stores information about SDN controllers and NFV management components. Each control/management element in the domain is recorded in the 'component' table, such as VNFMs, VIMs, SDN controllers and other OAMs. The 'type' table distinguishes the different technologies (virtual VNFM, physical VNFM, virtual SDN controller, containerized VIM etc).

The 'property' table stores the common properties that each component could have. As an example, the endpoint of an SDN controller REST API is stored in this table. The 'command' table stores the commands that may be applied in a component. An SDN controller can handle many commands (get devices, get metrics, modify flow rule etc), and the 'command' table stores the meta-data used by the NSB to build a proper command. The relation between 'type' and 'command' is required because each row in 'type' has different commands, even to a same component: a same SDN controller product can respond to different commands in two distinct versions, for example.

The key table in the model is the 'resource' table. It is a 2-tuple: (component, managed resource). Each component of control or management has *n* managed resources. As an example, an SDN controller has *n* controlled NEs. Another example is a VNFM, which has *n* managed Virtual Network Function (VNF). Figure 5 brings the main tables used by SHE to build the three logical topologies (data, control and management) and the commands catalog (used by the NSB). Other operational tables are not represented.

The entity-relationship model allows the explanation about the NSB module: if the NSB needs to apply a path modification to a control path, it can query the catalog to fetch the command. The logical topology stored in the NDB gives the SDN controllers in which the NSB will request path alterations. The NSB fetches the management components to find the endpoint of each component, and then fetches the commands in the catalog.

The operations performed by the CoE to fetch the NEs managed by a specific SDN controller, and the resources managed by some NFV components utilize the NDB catalog analogously. The logical topologies and the control paths described in this Section will be explained in more details in Subsection 3.3.

3.3 Management and Control Topology

The problem addressed by our solution can be separated in two distinct contexts: (i) the control plane communication; and (ii) the management plane communication. To apply our services in (i) means that the platform must ensure the communication paths between the NEs and the SDN controller. In (ii), the platform must ensure the communication paths between the OAMs and their managed resources. The (ii) is more complex than (i) because managed resources and OAMs run on conventional servers (containerized, virtual or physical), therefore the communication paths are usually defined by the routing application (at run time).

In the communication between the SDN controller and its managed NEs, the initial paths are defined in the network bootstrapping. In technologies like OpenFlow, the switch and SDN controller have protocol procedures to define the control path. In this case, our platform has information about the initial paths, and needs only to recover them from failures. This is achieved by calculating alternative paths (in the path-engine-ms), identifying (or predicting) the congested or broken links (in metric-ms and SLE), and applying an alternative path to the control traffic (in recovery-ms and NSB).

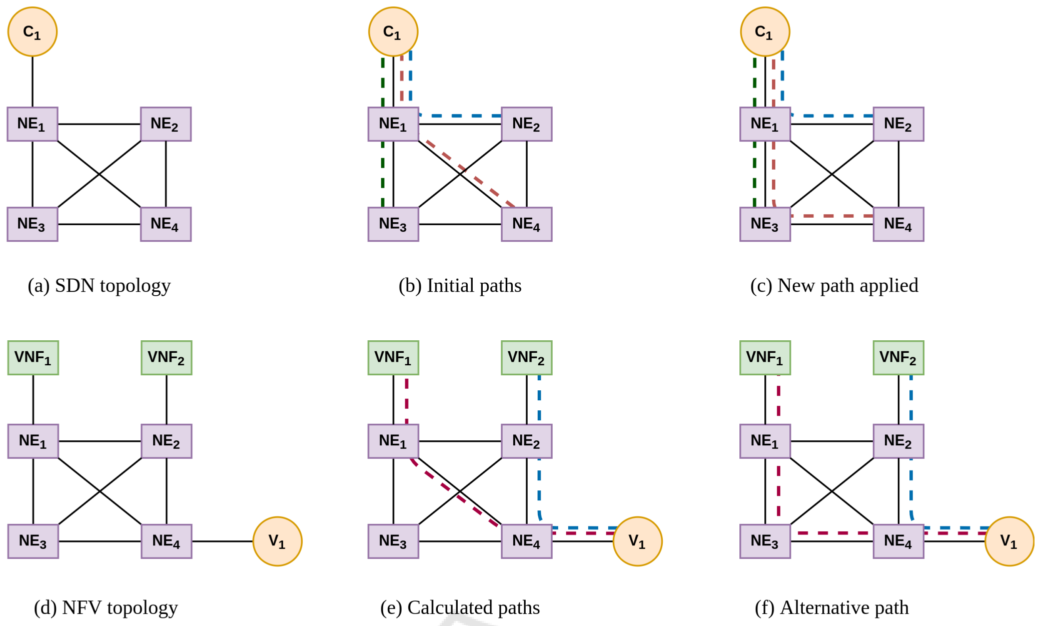

Figure 6: Management and control paths example.

The algorithms performed by the SHE are exemplified in Figure 6. A simple topology is presented, in which each NE is physically connected to all others. As shown in Figure 6(a), there is an SDN controller (C_1) in the domain. Figure 6(b) shows the communication paths (dashed lines) between each NE and C_1. These paths were previously defined by the OpenFlow procedures and are not part of our solution. When our platform starts, the CoE retrieves the network topology and paths shown in Figure 6(b). After this, the path-engine-ms calculates all alternative paths and stores them in the NDB. If the SLE concludes that the NE_1-NE_4 link will be congested soon, the recovery-ms defines an alternative path to the communication between NE_4 and C_1. Then, the NSB applies the new path, i.e. it requests a flow modification operation in C_1. If the communication between our system and C_1 is a problem, the NSB requests the operations to the NEs directly. Figure 6(c) shows the new path applied.

The topology between NFV management components and their managed resources has an additional issue: there are no initial paths. Figure 6(d) shows a topology with a VNFM (V_1) managing two deployed VNFs (VNF_1 and VNF_2). Since the catalog has information about V_1, VNF_1 and VNF_2, the first operation performed by path-engine-ms in to define priority paths between V_1-VNF_1 and V_1-VNF_2. The path-engine-ms calculates the best paths based on metrics already collected by the CoE (to avoid congested links) or applying the Dijkstra algorithm to decide the path with less hops. When the paths are defined, the NSB applies them to the network topology, i.e. it inserts flow rules in the network to create static and priority rules.

Figure 6(e) gives a computing example of the paths defined by the path-engine-ms. The defined management paths are V_1-NE_4-NE_1-VNF_1 and V_1-NE_4-NE_2-VNF_2. After this, the services used for the control plane communication are used as well: if a failure is detected in the NE_1-NE_4 link, the framework will apply alternative paths as presented in Figure 6(f).

The utilization of static and priority rules strategy overwrites the routes defined by the SDN routing application. This strategy requires several evaluation tests to validate the effectiveness in real production environments. We believe that this strategy can be better in some situations, because the routing algorithms could choose congested links or paths with considerable latency.

Figure 6 shows a control plane communication in (a), (b) and (c), and a management plane communication in (d), (e) and (f). It is noteworthy that the control plane topology could be different, and the management plane topology can have many OAMs and managed resources. Our architecture based on microservice and event-driven was designed to scale-out easily. In this way, the containers in our platform can manage the two advanced planes without any major. It is also important to mention that the platform itself is an OAM and is inserted in the catalog. It means that the platform components need priority paths with SDN controllers and other OAMs.

3.4 Status Quo and the Road Ahead

This is a work in progress paper and, at the current moment, we have developed the main components presented in Figure 3 by using widely accepted technologies in the industry. The components are applications running on Docker containers and managed by our own orchestrator. CoE, SHE, SLE, NSB and CPI were developed using Spring Boot framework (v2.1.2.RELEASE); the NEM is a RabbitMQ (v 3.8.0-rc.1) cluster; and the NDB and NoSQL engine are a cluster of Apache Cassandra (v3.11.4). The CoE services were developed by using SNMP, LLDP and ovsdb protocols. The CPI services are not finished, but a simple proxy between NEs and SDN controller is available.

We are now programming the microservices used by the SHE and SLE. Some initial experiments are necessary to decide which algorithms are better for each microservice. For example, the path-engine-ms may be implemented with different methods: to calculate the routes itself using Dijkstra or a similar algorithm; or to retrieve the paths already calculated by the SDNC. We will perform some experiments before the decision and implementation of these microservices.

After the implementation, the evaluation will be made. Firstly, we will evaluate the benefits of our platform, comparing them with an SDN and NFV environment without the platform. We are preparing a laboratory with the GNS3 emulator (for the network topology), ONOS platform (as SDN controller), Openstack (as VIM or OAM_1), and Open Source MANO (as VNFM or OAM_2). Secondly, we will prepare scenarios with different topologies to evaluate our platform performance.

4 RELATED WORK

As stated by (Rehman et al., 2019) and (d. R. Fonseca and Mota, 2017), the controller channel reliability is still a challenge in SDN. Reference (Rehman et al., 2019) still states that the path backup technique is essential for fault tolerance in the SDN scope, and (d. R. Fonseca and Mota, 2017) states that new fault tolerance platforms/tools are required. Our work addresses the controller channel reliability problem and applies the path backup approach as well (by building a new platform).

Our proposed solution considers an in-band communication even in the control plane. This means that the control decisions are made outside the data plane, but the traffic of control primitives is made using the same NEs used for the traffic of data primitives. Reference (Schiff et al., 2016) also believes in the in-band control, claiming that the deploy of out-of-band control in carrier-grade networks could be expensive, and the transition from legacy network is more complex. Our proposal is an architecture designed to monitor and recover control and management paths, while (Schiff et al., 2016) presents techniques for control plane. The techniques given by (Schiff et al., 2016) can even be used in the SHE.

(Basta et al., 2015) presents a control path migration protocol for virtual SDN environments. The protocol runs over OpenFlow and performs the migration involving the SDN controller, hypervisor and OpenFlow switch, by adding a hypervisor proxy. The migration phases described in the reference are insights to our work. However, we have worked in a platform running at the application level, placed on the Management Layer.

Reference (Canini et al., 2017) shows that self-organizing can be a valid idea for fault tolerance considering in-band SDN platforms. The predictive recovery was presented in (Padma and Yogesh, 2015), and (Neves et al., 2016) applies four self-* properties in 5G environments.

Our work considers an NFV environment deployed together with SDN technology because we believe that future networks will have these technologies in their domains, or at least in parts of the domain. In (Foukas et al., 2017) and (Mijumbi et al., 2016) some challenges in NFV are discussed and (Yousaf et al., 2017) shows how SDN and NFV are complementary technologies.

It can be concluded that many initiatives are using self-healing approaches to address fault tolerance and resilience questions in SDN, but the initiatives act on specific points. Our solution is indeed to be a complete platform in the future, prepared to work in the many different SDN and NFV environments. It means that the control and management plane connectivity is the first goal of our work, but the architecture designed can address other aspects in the future.

5 CONCLUDING REMARKS AND FUTURE WORK

This paper proposes a new platform to deal with fault tolerance aspects in the communication between components from the control and management planes. The platform is built on an architecture designed to scale-out when sizeable topologies are used. It is essential to mention that the platform is in practice, a system running at the application level. It means that no modifications are necessary for the NEs, SDN controller, NFV components, etc.

We have built a new system, despite other architectures and projects found in the literature, because we believe that future telecommunications and cloud computing networks will eventually use SDN and NFV platforms and will require management tools/platforms as well. Even if an SDN or NFV based network already exists, it is possible to deploy our solution.

This paper presented our platform and its main components. Additionally, the essential services for control/management plane communication were described. The reason to design a whole platform instead of an isolated service is that other self-healing functions may be implemented in the future using the platform. Examples of these functions include the healing of hardware resources (memory, CPU, disk, etc.) in a datacenter, instantiation of new NEs for high traffic, software reboot, reset of components, and so on.

Since this is a position paper, the evaluation of the solution is in progress. Our platform needs to be compared with other solutions. We intend to prepare an environment with virtual and physical NEs and diverse NFV components. To do this, we will use a laboratory environment inside our university connected to a facility in a local telecommunications company.

We believe our evaluation experiments can be initialized as soon as possible. The tests intend to prove the effectiveness of this Management Layer platform against solutions placed on Control Layer or inside SDN applications.

As future work, our research group long for finish the development, effectiveness, and performance tests; develop other self-healing functions inside the SHE and design other self-* capabilities in the architecture.

ACKNOWLEDGEMENTS

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (Capes) - Finance Code 001. It also received support from the Algar Telecom.

REFERENCES

Abdelsalam, M. A. (2018). *Network Application Design Challenges and Solutions in SDN*. PhD thesis, Carleton University.

Basta, A., Blenk, A., Belhaj Hassine, H., and Kellerer, W. (2015). Towards a dynamic sdn virtualization layer: Control path migration protocol. In *2015 11th International Conference on Network and Service Management (CNSM)*, pages 354–359.

Canini, M., Salem, I., Schiff, L., Schiller, E. M., and Schmid, S. (2017). A self-organizing distributed and in-band sdn control plane. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2656–2657.

Chandrasekaran, B., Tschaen, B., and Benson, T. (2016). Isolating and tolerating sdn application failures with legosdn. In *Proceedings of the Symposium on SDN Research*, SOSR '16, pages 7:1–7:12, New York, NY, USA. ACM.

Cox, J. H., Chung, J., Donovan, S., Ivey, J., Clark, R. J., Riley, G., and Owen, H. L. (2017). Advancing software-defined networks: A survey. *IEEE Access*, 5:25487–25526.

d. R. Fonseca, P. C. and Mota, E. S. (2017). A survey on fault management in software-defined networks. *IEEE Communications Surveys Tutorials*, 19(4):2284–2321.

Foukas, X., Patounas, G., Elmokashfi, A., and Marina, M. K. (2017). Network slicing in 5g: Survey and challenges. *IEEE Communications Magazine*, 55(5):94–100.

Mijumbi, R., Serrat, J., Gorricho, J.-L., Latré, S., Charalambides, M., and Lopez, D. (2016). Management and orchestration challenges in network functions virtualization. *IEEE Communications Magazine*, 54(1):98–105.

Neves, P., Calé, R., Costa, M. R., Parada, C., Parreira, B., Alcaraz-Calero, J., Wang, Q., Nightingale, J., Chirivella-Perez, E., Jiang, W., et al. (2016). The selfnet approach for autonomic management in an nfv/sdn networking paradigm. *International Journal of Distributed Sensor Networks*, 12(2):2897479.

Padma, V. and Yogesh, P. (2015). Proactive failure recovery in openflow based software defined networks. In *2015 3rd International Conference on Signal Processing, Communication and Networking (ICSCN)*, pages 1–6.

Rehman, A. U., Aguiar, R. L., and Barraca, J. P. (2019). Fault-tolerance in the scope of software-defined networking (sdn). *IEEE Access*, 7:124474–124490.

Schiff, L., Schmid, S., and Canini, M. (2016). Ground control to major faults: Towards a fault tolerant and adaptive sdn control network. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)*, pages 90–96.

Thorat, P., Raza, S. M., Nguyen, D. T., Im, G., Choo, H., and Kim, D. S. (2015). Optimized self-healing framework for software defined networks. In *Proceedings of the 9th International Conference on Ubiquitous Information Management and Communication*, pages 1–6.

Yousaf, F. Z., Bredel, M., Schaller, S., and Schneider, F. (2017). Nfv and sdn—key technology enablers for 5g networks. *IEEE Journal on Selected Areas in Communications*, 35(11):2468–2478.