Bootstrapping and Plug-and-Play Operations on Software Defined Networks: A Case Study on Self-configuration using the SONAr Architecture

Maurício Amaral Gonçalves[a], Natal Vieira de Souza Neto[b], Daniel Ricardo Cunha Oliveira[c], Flávio de Oliveira Silva[d] and Pedro Frosi Rosa[e]

Faculty of Computing, Federal University of Uberlândia, Uberlândia, Brazil

Keywords: Self-management, Self-driving, Intent-based, Autonomic Computing, SON, SDN, NFV, Future Internet, 5G.

Abstract: The autonomous network concept has gained strength with the growing complexity of the current networks, especially after the definition of 5G requirements and their key performance indicators. This concept is challenging to implement in legacy networks, and it has become feasible with the emergence of network softwarization, which enables the deployment of functionalities through a logically centralized control plane abstraction. The network softwarization simplifies the management process, reduces operational costs (OPEX), enhances protection against failures, and enables complex requirements such as high-performance indicators and IoT support. The Self-Organizing Networks Architecture (SONAr) project uses cutting-edge technologies and concepts such as SDN, NFV, and Machine Learning. It proposes a new architecture aimed at the design of self-management in computer networks, oriented by declarative intents. In this work, we introduce the SONAr project by describing its components and specifications. We also present a case study that shows the self-configuration property that includes bootstrapping and plug-and-play operations using SONAr components focusing on strategies applicable to OpenFlow based networks. We explain the decisions made in the implementation and present comparative results between them.

1 INTRODUCTION

Infrastructure for future computer networks will have great use of Software Defined Networking (SDN) and Network Function Virtualization (NFV) (Ramirez-Perez and Ramos, 2016), aiming to satisfy requirements from modern applications. The use cases which should take the benefits of these future networks are 5G, Internet of Things (IoT), health applications, low latency communications, massive video streaming, and so on (Barona López et al., 2017). SDN and NFV bring many advantages to computer and telecommunications networks, mainly related to scalability, fast deployment of new protocols, and programmable services (Cox et al., 2017). Unfortunately, these approaches also bring some new management issues to the network.

In traditional networks (before SDN and NFV), the configuration was generally applied to infrastructure elements manually with low-level abstraction. This strategy required highly complex planning to provide communication services with many configuration details per network element. At the same time, the SDN and NFV configuration bring new challenges related to bootstrapping and control plane availability.

In mobile networks, operated by telecommunications companies, the concept of Self-Organizing Networks (SON), specified by 3GPP standards (3GPP, 2018b)(3GPP, 2018a)(3GPP, 2018a)(3GPP, 2018b) defines the use of SON for network automation. The idea of autonomic computing was also used in software engineering and defined by IBM (Ganek and Corbi, 2003a).

However, SON functionalities used by mobile networks focuses only on the Radio Access Network (RAN). The core of the network has neither standards nor complete architectures to handle such self-* properties. The self-* refers to a set of properties such as self-healing, self-configuration, self-optimization, self-protection, and so on (Boutaba and Aib, 2007).

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

Gonçalves, M., Neto, N., Oliveira, D., Silva, F. and Rosa, P.
Bootstrapping and Plug-and-Play Operations on Software Defined Networks: A Case Study on Self-configuration using the SONAr Architecture.
DOI: 10.5220/0009406901030114
In *Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSER 2020)*, pages 103-114
ISBN: 978-989-758-424-4

The first step in developing an architecture for network self-management is to understand the requirements of administrators and operators. This understanding is useful both for the design of the solution and for gaining the trust of administrators, who fear the loss of control and dependence on tools in the administrative process (Duez et al., 2006). Inappropriate automation can increase the complexity of management, what is known as "irony of automation" (Bainbridge, 1983), or even propagate errors on a larger scale than with manual intervention. On the other hand, it will be impossible to manually manage the complexity of future networks. Because of this, autonomic computing should be explored to self-manage the environments (Sanchez et al., 2014).

In this work, we introduce the Self-Organizing Networks Architecture (SONAr), presenting its complete specification. SONAr's design principle is to have a vendor-independent solution, i.e., a self-management product that can be deployed in any environment. Data centers, home networks, telecommunications core networks, enterprise networks, and others can take advantage of using SONAr to apply self-management in their infrastructure. SONAr aims to use self-* properties to handle and manage infrastructure, hardware and software, and network communication in general.

We also present a case study that explains the self-configuration property that includes bootstrapping and plug-and-play operations. This experimental evaluation of the self-configuration SONAr components focuses on strategies applicable to OpenFlow based networks. We explain the decisions made in the implementation and present comparative results between them.

The paper is structured as follows. Section 2 presents the background and related work. Section 3 shows the requirements and challenges of management in future networks. Section 4 presents the SONAr solution with an overview of its main components. Section 5 shows the results from our experiments regarding self-configuration, and finally, Section 6 presents concluding remarks and future work.

2 BACKGROUND

The concept of Self-Organizing Networks(3GPP, 2018) has been explored by telecommunication networks standards (3GPP, 2018b)(3GPP, 2018a)(3GPP, 2018b)(3GPP, 2018a) as a way of dealing with the growing complexity of the infrastructure and the need for coexistence with legacy technologies (Feng and Seidel, 2008). In the same way, autonomous computing concept (Ganek and Corbi, 2003b) proposes the automation of administrative and operational processes as a way to reduce the complexity of computing systems, minimize operational costs (OPEX) and improve Quality of Service (QoS). The term is an analogy to the autonomous nervous system, which is the mechanism responsible for important bodily functions (e.g., breathing and blood supply) without conscious intervention. Similarly, we believe that networks must be able to self-manage with minimal human intervention.

In this section we show how SDN (McKeown et al., 2008) and NFV (Cui et al., 2012) could be used together to apply SON and autonomic computing in the core network of future networks. At the end, we present some related works, which have explored some points, and correlate them with SONAr.

2.1 SDN and NFV

Since SDN has programmable and centralized control, the configuration operations would be simplified. The SDN provides a high-level abstraction which allows the services deployment as a combination of flows and functions.

NFV is usually presented together with SDN as a way to make provisioning for network functions more flexible. By considering this approach, the SDN controller is usually deployed as a virtual function. Besides that, other Virtual Network Functions (VNF), such as routers, gateways, firewalls etc, need to work without interruptions. It means that the SDN and NFV need to guarantee that the VNFs are movable in the available infrastructure, keeping routes and other network configurations (Cox et al., 2017)(Barona López et al., 2017).

Some tools and frameworks for management are available in SDN. OF-Config is a protocol based on NETCONF, designed for SDN management. The protocol is restricted to OpenFlow. AdVisor is another example, which provides a management interface. These tools help the operation and implementation of management requirements, such as bootstrap, isolation, planning, monitoring etc (Wickboldt et al., 2015).

In future networks, the management keeps some challenges such as provide transparent mobility, low latency communication, minimal cost, and supporting millions of new devices.These problems need to be solved or mitigated using advanced techniques, like autonomic computing and self-management (Barona López et al., 2017).

2.2 Related Work

SELFNET project (Neves et al., 2016) presents an architecture for orchestration, administration and access control in SDN and NFV environments. It integrates SDN, SON, NFV, cloud computing and Artificial Intelligence (AI) to create an autonomic network management. Some sensors and actuators are deployed in the control plane and data plane, with the mission to execute the autonomic computing actions (monitoring, optimizing, recovery etc). SELFNET is a promising work in progress, designed for 5G networks. SONAr and SELFNET have some mutual functions and could work together in the future, since SONAr is designed to be independent of the network type (5G, core network, home networks etc can use the benefits of the solution).

In (d. R. Fonseca and Mota, 2017) the authors explore fault management in SDN. This paper shows the benefits provided by SDN, and describes the new failure points which were created with the SDN approach. It is important to know the different possible failures: in the data plane, in the controllers, or in the SDN applications.

In (Abdallah et al., 2018) the authors present a framework for FCAPS (Fault, Configuration, Accounting, Performance, Security) model. The paper shows the functions that can be assigned by the controller, and those that should be assigned by the Management Plane. It also shows a brief description of the use of OF-Config, instead of OpenFlow, for management needs.

As described in this section, many groups around the world have applied self-management through SDN and NFV. There is no project completed in this scenario (i.e. self-management in network core). Administrators and network operators have many requirements not met by the proposed frameworks.

3 SELF-MANAGEMENT REQUIREMENTS FOR FUTURE NETWORKS

This section introduces the main application requirements into future network environments, as well as the requirements of network operators, that is, requirements related to network management.

3.1 Future Internet Applications and QoS Requirements

The Future Internet is being developed as a response to the strict requirements and complexity of new applications – some of which are already being explored in a limited scope. Some good examples are VR/AR applications, drone control (together with 4K video transmission), tactile Internet, in-cloud gaming, autonomous driving, among others. All of these cases have a certain degree of high bandwidth and throughput as a common requirement, and low latency response is crucial to their successful implementation, especially when considering real-time interaction. In these particular cases, the senses of participating humans must be taken into account. For example, auditory interactions require a response time of 100 ms, while the visual reaction requires about 10 ms and tactile feedback, round trip delay time less than 1 ms (Simsek et al., 2016).

For a VR/AR total immersive experience indistinguishable from the reality, a network throughput of 5.2 Gb/s would be required for each user, albeit in a more realistic approach for practical applications, a throughput of 100-200 Mb/s with a latency of 13 ms give most people a good immersive experience without a feeling of motion sickness (Bastug et al., 2017).

Another important issue is the network reliability. Some 5G typical applications demands less than 10^{-7} failure rate, which corresponds to 3.16 seconds of unavailability per year (Simsek et al., 2016).

In general, the main requirements for a new generation network – e.g. high flexibility, plasticity, easiness of (re)configuration, implementation and control, reliability and cost-effectiveness – are not reachable using the legacy or current working network protocols.

3.2 Main Requirements of Real Operators and Telecommunications Industry

Implementing a self-managed SDN network is not an easy task. From the network operator's point of view, several challenges remain, if aspects such as backward compatibility, choice of deployment strategy, security issues, and especially business cases are closely observed (Zaidi et al., 2018). However, when considering the benefits of a Self-Organizing Network in the Future Internet scenario and the limitations already discussed, it is very clear that a SDN approach combined with SON architecture is a smart and viable solution. The main requirements presented by net-

work operators are: self-configuration, self-healing, self-optimization and self-protecting. Without these requisites, it will be difficult - if not impossible - to successfully deploy the Future Internet applications in a commercial network:

- Self-configuration: Setting up network elements usually means a lot of work, done by skilled engineers with a deep understanding of the operator's network, every time a new Network Element (NE) is put into service or moved from one place to another. It is not difficult to predict the amount of work that will be needed only to configure the radio access network – including front, mid and backhaul – of a complex and high density 5G macro/small/pico cells network of a city such as Beijing or São Paulo. The Future Internet will require plug-and-play solutions not only for the deployment of a complex system, but also to make its plasticity feasible;
- Self-healing: Ultra-high reliability will be a key issue in the Future Networks. Nowadays, the fault recovery management and recovery rely on the traditional distributed protocols – such as EIGRP or OSPF – to handle failures, which can not provide the required 99.999% reliability and may lead to some important issues (d. R. Fonseca and Mota, 2017). A quick response to faults is an indispensable aspect to be observed in a new network architecture, and self-healing is an important feature to be considered in any Self-Organizing Networks architecture proposal and an indisputable requirement for the telecommunications industry;
- Self-optimization: self-optimization process is a response to the increasing complexity of today's networks. As they become bigger, interconnected, multi-vendor and, thus, more and more complex, the work to achieve efficiency also becomes more difficult bordering on the impractical. Until a few years ago, self-optimization tools were restricted – and even full-blown – to mobile radio access networks (RAN). The concept was first introduced in 3GPP Rel. 8 (Moysen and Giupponi, 2018), as an answer to the huge amount of network parameters that had to be managed by the RAN optimization engineers. With SON being introduced in the access and transport networks, it is a natural path that self-optimization process should be also part of the evolution, as it will increase the system efficiency, thus reducing CAPEX/OPEX on nodes and links;
- Self-protection: Intrinsic self-protecting systems are also an important feature in any SON architecture. The main types of threats are: DDoS attacks, user to root (U2R) attacks, remote to local (R2L) attacks and probe attacks, and a number of router-based defense mechanisms have been proposed to detect and prevent them (Hariri et al., 2005). Needless to affirm that an inherent, fast-converging and intelligent self-protection system will be indispensable for future networks management and a great demand from the operators point-of-view.

The background and the requirements for future networks already presented give the basement for SONAr. Section 4 brings the new architecture design, the main components specification (and their integration), and the layer placement of the solution.

4 SONAr DESIGN AND SPECIFICATION

Figure 1 presents the SONAr design, its components and their distribution by network layers. The main modules are described in this section.

4.1 Self-Organizing Entities

The Self-Organizing Entities (SOEs) are responsible to perform the basic algorithms related with the autonomic computing fundamentals. The intelligence to heal, optimize, configure, protect, and plan is performed by these entities. Usually, the Network Event Manager (NEM) receives events from all components of the architecture. The Self-Organizing Entities subscribe to specific topics on NEM, and then receive the events related with self-* properties.

Self-Healing Entity (SHE) performs algorithms to predict or detect faults in all of the components depicted in Figure 1. In case of failures, it is necessary to recover from them. Note that faults in the Infrastructure Layer within SDN elements (such as OpenFlow switches) are usually handled by the SDN controller. Unfortunately, some legacy devices are not covered by controllers, and need special attention. In addition to the infrastructure, SHE needs to detect failures in the Control Layer, such as bugs or crashes in controllers, Virtual Infrastructure Manager (VIM), VNF managers etc. SONAr components may also fail, and SHE must have recovery functions. Furthermore, SHE itself may fail and need to recover itself as soon as possible.

The Self-Configuration Entity (SCE) works at the time of the network bootstrap and the NE plugging. In bootstrapping, the SCE configures all flows within the

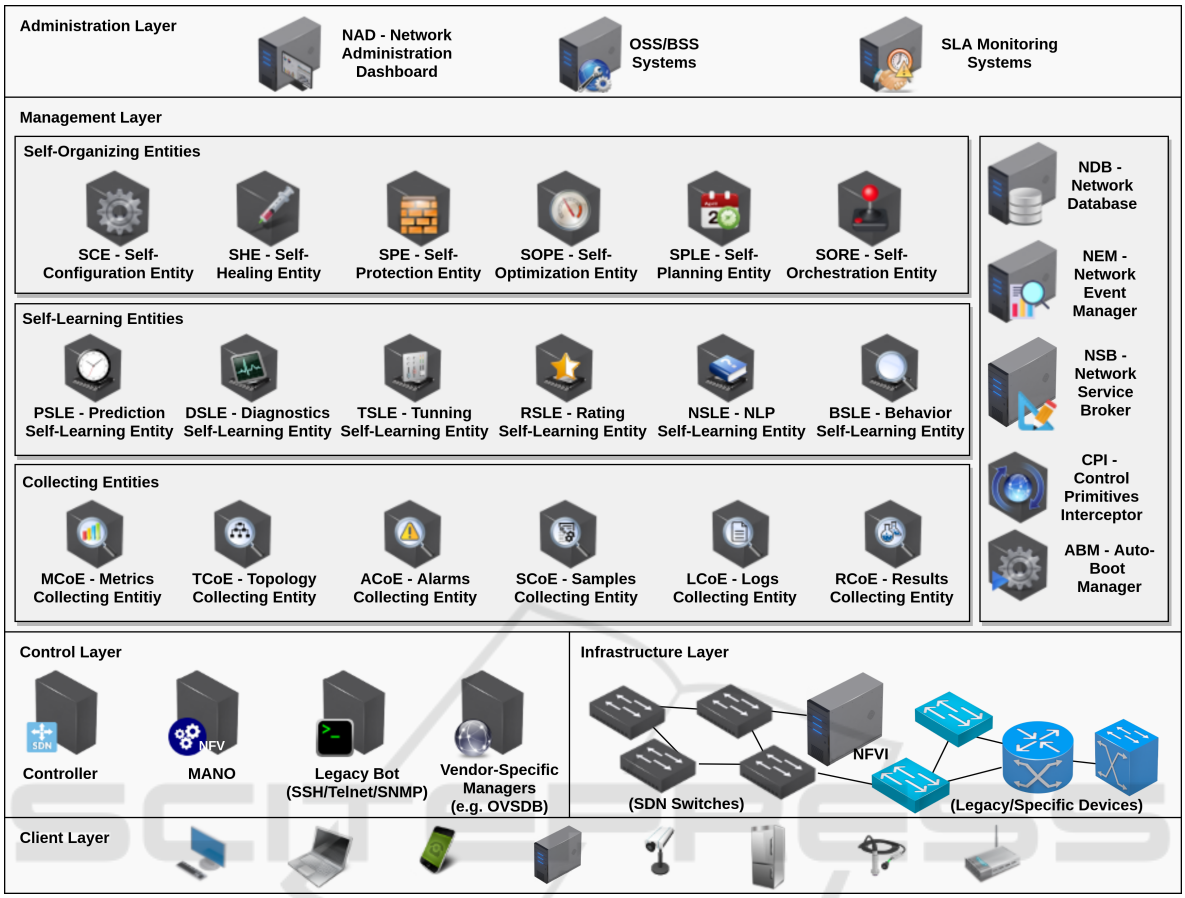

Figure 1: SONAr Design.

NE to move the network from an out-of-service state to a running state. When the network is running, the SCE works at the moment a new NE is plugged in the environment. In current infrastructures, an operator physically connects the new NE and configures the NE and its neighbors, e.g., the address (usually IP), host name, and routes are manually configured. The purpose of the SCE is to be responsible for discovering these settings and executing them automatically.

SCE is also responsible for the safety of new components. In traditional architectures, when a new equipment is connected to the network by an operator, it is discovered by the topology and begins to function. Routing protocols, such as OSPF, start creating routes for this new equipment. Security may become a problem in this scenario if this new equipment is not valid or authorized. With SONAr, the SCE needs to guarantee the identity of new components.

The Self-Optimization Entity (SOPE) subscribes statistical topics and analyzes possible network improvements. Considering a SDN and NFV environment, the improvements are usually related to reallocation of resources. In the Infrastructure Layer, if the SDN is an OpenFlow implementation, SOPE can rearrange the flow rules within the OpenFlow switch to use the resources intelligently. Network slice management is also covered by SOPE. Note that SOPE collaborates with the Diagnostics Self-Learning Entity (DSLE). Almost all improvements are made after the analysis of the history. Differentiation between DSLE and SOPE functions is important: DSLE performs AI algorithms while SOPE receives DSLE outcomes and performs algorithms to improve the use of network resources and infrastructure.

A common example of improvement is to avoid congestion: if the DSLE identifies that an NE or a port will be congested in the future, it will notify SOPE to perform an optimization procedure. SOPE will then analyze the event sent by DSLE and rebalance flows between routes.

The Self-Protection Entity (SPE) ensures the protection of all layers in Figure 1 against attacks. SPE has the services prepared for isolation and recovery. The first step is to isolate the compromised compo-

nents, preventing attacks from reaching other components. After isolation, the SPE will recover the attacked component, clearing the remains of the attack, and then reinserting the clean component into the infrastructure. Like SOPE, SPE is also based on diagnoses made by DSLE.

Architectures and frameworks in academy and industry have created mechanisms to protect the Infrastructure Layer. The current challenge is to ensure the protection of all layers shown in Figure 1. The Control Layer has components that can be attacked, such as the SDN controller: once it has control of any network topology, a successful attack on this component gives the attacker control of all infrastructure. In addition, the SPE needs to isolate attacks on the Infrastructure Layer, that is, attacks on an NE must be restricted to that NE and can not reach the Control Layer. Attacks on the Self-Management Layer and Administration Layer follow the same analogy.

The Self-Planning Entity (SPLE) works in conjunction with all Self-Organizing Entities and aims to schedule actions of other entities. The main idea is to realize changes that will be made in the future and to schedule them to be made in a timely manner. Another basic function is to ensure that repetitive actions of SHE, SCE, SPE and SOPE are applied.

SHE, SCE, SPE, SOPE and SPLE are separate entities that perform services that can interact. These interactions can eventually cause inconsistencies. For instance the optimization: SOPE could determine an improvement in the network and to make this improvement, a route would need to change from one port to another in a given NE. It may be that the SPE blocked this target port in advance to prevent an attack. SOPE's decision may eventually replace the previous decision of the SPE. The Self-Orchestration Entity (SORE) was designed to control the actions created by other Self-Organizing Entities.

The SORE will receive all the actions of other entities and will organize them in a queue, prioritizing the actions. The operator could previously determine the priorities in the SONAr dashboard. To avoid inconsistencies, SORE works with the concept of transaction: if a flow rule in an NE is initiated, other flow rules in the same register are queued. After the first flow rule is completed, the SORE will execute the others. Rollbacks are also performed by the SORE.

4.2 Self-Learning Entities

The Self-Learning Entities (SLEs) are responsible for providing AI based mechanisms to overcome the absence of human intervention. These entities use knowledge bases that can be built dynamically in an action/consequence analysis, manually in a guided learning strategy or imported from third parties similarly to antivirus bases (which allows the easy set-up for new networks). SLE implements machine learning algorithms to provide scenario prediction, parameter adjustment, problem/opportunity diagnosis, behavioral pattern detection, infrastructure classification and translation of intentions. These activities are usually complex and slow, and therefore need to be addressed by specific mechanisms separated from Self-Organizing Entities.

The Prediction Self-Learning Entity (PSLE) applies the K-Nearest Neighbors (KNN) algorithm to classify metrics, services, time and topology to provide a discrete representation of the scenario.

The Tuning Self-Learning Entity (TSLE) provides hyper-parameter optimization for all SONAr components, including those composing SLE. For this, it monitors the quality of the scenarios, evaluation metrics and service levels through a classification algorithm such as the KNN.

The Diagnostic Self-Learning Entity (DSLE) implements a correlation algorithm for root cause analysis, service level assurance, security issues detection and network degradation diagnostics. This is done by analyzing the metrics and thresholds, verifying the occurrence of alarms and monitoring log patterns and events. The knowledge base for DSLE can be constructed with administrator-driven learning or deep-learning based on data patterns of previous correct diagnostics. This entity creates events in the NEM with the specific diagnosis and all data available.

The Rating Self-Learning Entity (RSLE) evaluates devices, links, services, components and algorithms based on their performance, costs, usage statistics, current occupation, capabilities, and service level history for each of the supported policies. With a mature knowledge base, RSLE can be used to define the best infrastructure elements to be used at a specific time to configure a service according to its policies.

The NLP Self-Learning Entity (NSLE) implements the Intent Translator through Natural Language Processing (NLP) and Neural Networks. NSLE can infer policies, functions and filters from a service based on Declarative Intents expressed in natural language. The knowledge base for translating intentions can be constructed from guided learning or service description analysis (which is basically an intention defined in a formally-described service). NSLE also implements the feedback delivery aspect of the service validator, which is done by analyzing metrics such as RSLE and DSLE.

The Behavior Pattern Self-Learning Entity (BSLE) is responsible for detecting network usage

patterns and identifying the types of content being transmitted. This is an important feature for self-management since it makes the network sensitive to changes in context and allows activation of services under specific conditions.

4.3 Collecting Entities

The Collecting Entities (CoEs) are responsible for collecting data from the infrastructure and all other components of the architecture. This data is useful for monitoring the state of the network and providing the guarantee of the level of service.

These entities are also responsible for triggering events when: new data becomes available, data changes, limits are reached and reportable patterns are detected. Basic events can be used by SLEs and Self-Organizing Entities, which can use raw data as well as analyze and make decisions. The implementation of each CoE varies according to the specificity of the infrastructure, which can occur through proprietary protocols such as OVSDB and CDP, specific management systems, control components such as the MANO and SDN controller, CLI robots using ssh/telnet, or open protocols such as SNMP, LLDP, and syslog.

Each Collecting Entity implements a specific algorithm to collect and process the related data type. The Metrics Collecting Entity (MCoE) collects metrics from devices, links and servers through specific protocols, monitor thresholds and trigger events. The Topology Collecting Entity (TCoE) applies discovery algorithms, monitor changes on topology and trigger specific events when it occurs. The Alarms Collecting Entity (ACoE) collects events with an active or reactive strategy, for example by using SNMP traps or data polling, and triggers more specific events such as those defined by the DSLE. The Samples Collecting Entity (SCoE) uses strategies like Deep Packet Inspection (DPI), port mirroring and network sniffing to infer content types from a particular instance of communication, which can be used to enable specific services and to detect usage anomalies or security issues. The Results Collection Entity (RCoE) is responsible for conducting periodic examinations of resources and components in order to collect specific data that cannot be obtained otherwise, such as latency and availability. Finally, the Logs Collecting Entity (LCoE) collects logs and looks for predefined patterns (specified by administrators or imported from an existing knowledge base) and fires events when they match.

CoE plays a vital role for self-management and can be used in any solution. At the same time, it is possible to use current collector applications, such as those implemented in SDN controllers, with a polling and processing strategy to power the SONAr components.

4.4 Network Event Manager

The Network Event Manager (NEM) is a publisher/subscriber provider, able to handle events categorized into topics. Event processing can occur synchronously or asynchronously, the second being provided by a message queue solution. All SONAr components trigger events, even for notification when performing actions. Typically, SLEs and SORE are subscribed into topics, but SONAr defines a flexible architecture where any Self-Organizing Entities can enroll on a specific topic directly and take advantage of it. NEM also manages the event relationships generated by DSLE correlation to avoid unnecessary processing and optimize management.

SONAr is a completely event-oriented architecture, so NEM has an important role in connecting all the other components. The asynchronicity provided by NEM is essential for the self-management of the network, considering the Key Performance Indicators (KPIs) of future networks.

4.5 Network Database

The Network Database (NDB) stores all SONAr information: metrics, topology data, alarms, samples and logs collected by CoE; administrative data used by the dashboard; management primitives intercepted by CPI; events sent to the NEM; events created by any entity (SHE, SPE, SOPE, SCE etc); and prediction and analysis information (including input parameters and results).

Events sent to the NEM are first used by the Self-Organizing Entities, and stored in the NDB because they will be used by the SLE Machine Learning algorithms. From these events, the SLE can analyze information and adjust models to predict network behavior. The SLE analyzes data and creates new data (results), which are also stored in the NDB. The history of events received and created by the architecture entities are also maintained in the NDB. The SONAr dashboard can query this history and operators have graphical view of the behavior of the network over time.

The Self-Organizing Entities perform actions in the topology, and these actions are also stored in NDB. They can be used in the future to determine if a specific action solves a problem or not. SLE could also take advantage of this information. An example

is SHE: sometimes more than one action may be required to recover from a failure. The SHE performs the action (through the SORE) and, if the action does not solve the problem, the next action is chosen. The results of these actions are stored in the NDB, and in the future the SORE queries the NDB and decides which action, in this specific situation, has the highest success rate.

The NDB is designed to be distributed to ensure high availability and scalability. Implementation details are beyond the scope of this work, but it is recommended that NDB be implemented using well-known NoSQL databases such as Cassandra, MongoDB, ElasticSearch, etc. (for high availability purposes).

4.6 Controller Interceptor

The Control Primitives Interceptor (CPI) is a crucial component of SONAr. It is intended to be placed in the Southbound Interface (SBI), working as a proxy between the data plane and the SDN controller. The control primitives that cross this interface are captured and analyzed by the interceptor. If a primitive has management semantics, it is sent to the SONAr components and SDN controller. Otherwise, it is sent only to the SDN controller.

The control primitives that travel in SBI are usually related to applications and hosts connected in the topology, but some of them are related to primitives of metrics and notifications. First, the CPI receives a primitive and determines if this primitive has management semantics. After that, it parses the primitive by performing a shaping and discard analysis. The shaping will create a new primitive used by SONAr components. The discard will drop malicious primitives that were sent to the SDN controller by an attacker. The new shaped primitive will be sent to Network Event Manager (NEM), as an event. Then, other SONAr components will catch the event and treat it.

4.7 Other Elements

The Network Administration Dashboard (NAD) is an administrative tool for interacting with Self-Management Layer, allowing: to visualize topology, metrics, alarms, logs and events; to manage services; to enable guided learning and manual diagnostics; to provide service level and network status monitoring; and to allow the definition of parameters and goals.

Auto-Boot Manager is a solution for starting the network. It is responsible for: allocating resources; instantiating VNFs; deployment of applications; configuring devices and servers; and establishment of control and management flows. It follows an implementation plan that lists infrastructure details and a specification of required components and where they should be deployed, or can be fully automated by a strategy of self-discovery and self-configuration.

Even following a deployment plan, the Auto-Boot Manager constantly monitors the infrastructure to detect new devices and changes in general, and allows the Self-Organizing Entities to apply settings as needed. It combines a DHCP server, a FTP server containing configuration scripts, an Auto-Attach Manager as defined by the IETF for SPBM, and an OVSDB Manager. It controls the assignment of identification (usually IP) for devices; queries basic information (using the CoE code); applies the SPBM strategy to establish control and management flows; pushes configurations for devices and servers; and deploys and initiates component control and management. The Auto-Boot Manager is also directly responsible for auto-attaching of devices, servers, and links in a plug-and-play strategy. To do this, it constantly changes the basic configuration of flows and deployment (in active mode) or allows the reconfiguration called by the SCE (in reactive mode).

The Network Service Broker (NSB) is used by customers and administrators to configure communication services based on intentions inferred from the behavior/content or described in natural/formal language. These services are configured by SCE and guaranteed by other SOEs.

5 EXPERIMENTS

A prototype was developed to evaluate the SONAr architecture, implementing its main components to test its ability to initialize a software-defined network. This prototype involves: the Auto-Boot Manager, which is responsible for starting and configuring all infrastructure components; the Topology Collecting Entity, which is responsible for running the discovery process and for triggering events; the Self-Configuration Entity, responsible for calculating the routes, enabling OpenFlow and configuring the controller in the devices, and pushing the flow configuration through the controller; the Network Event Manager, which is responsible for managing the events exchanged by SCE and TCoE; the Network Database, responsible for storing the data, and the SDN controller used to control the devices.

To facilitate the deployment of the components, they were implemented as microservices running on Docker containers. Integration between components is done by event and database sharing. We also im-

plemented our own Container Infrastructure Manager (SONAR-CIM), which provides a RESTful API for managing containers with Docker support through the Spotify Docker Client API 8.16.1. In addition, we have implemented a custom DHCP server to provide dynamic IP address assignment . Our DHCP server supports the standard method (listening for broadcast UDP datagrams on port 67) and the OpenFlow method (processing PACKET-IN and returning PACKET-OUT).

All components were implemented using JAVA 8 and Spring Boot 2.1.2. We have also used RabbitMQ 3.8.0 for implementing the NEM, Apache Cassandra 3.11.4 for NDB, and ONOS 2.1.0 for SDN controller. We have built a Docker image of NE using Linux Alpine 3.9, net-snmp 5.7.3, lldpd 1.0.3 (with SNMP support) and Open vSwitch 2.10.1. The boot script of our NE starts the snmpd, lldpd and ovs-vswitchd; creates the OVSDB database and listens to remote and local connections; creates a bridge interface and runs the DHCP client. The SONAr server is a VirtualBox 6.0 VM with Ubuntu Server 16.04, 8194MB of RAM, 22GB of HD and a processor with 8 cores operating at 2.00GHz. In SONAr Server we installed JDK 1.8.0-151 and also net-snmp 5.7.3 and lldpd 1.0.3 (with SNMP support). The experiments were simulated using GNS3 2.1.16 (Neumann, 2014) in a machine with Ubuntu 18.10, 16 GB of RAM, 40 GB of HD and a processor with 8 cores operating at 2.00GHz.

The experiments were based on comparative tests with topologies of different proportions. To make this possible, we created topologies with 1, 2, 4, 8, 16, 32, 64, and 128 switches in GNS3 to simulate the scenarios described in the next sections.

5.1 Network Bootstrapping Scenario

When SONAr is started, it enters in the Discovery Stage (initiated by TCoE) in which it tries to discover all connected devices using BFS search and multi-threaded processing. In this process, NEs have already been started and operate as conventional switches (with learning switch mechanism). When all NEs are discovered with SNMP without failures or a timeout is reached, TCoE notifies the change of state through NEM.

Upon receiving the event that indicates the end of the Discovery Stage, the SCE starts the Routing Stage, when the configuration is calculated in a multi-objective analysis. In this experiment, we simplified this process using the Dijkstra algorithm to calculate the shortest paths between the server and the NEs. In this step, we also calculate a dependency tree, where the subsequent node depends on the configuration of the previous node. This is required to maintain connectivity to devices throughout the process. Configuration is also calculated at this stage, including flows to process ARP, DHCP, and routes between switches.

Finally, after the routing stage, the SCE starts the last stage, called the Configuration Stage. In this step, the SCE connects the grouped NEs according to the previously calculated dependencies using a multi-threading strategy, and configures the bridge, protocols, and controllers. The SCE waits for the effective connection of the network element before continuing. When the NE is connected to the Controller (optionally using a CPI), the SCE sends the flow configuration to the Controller as a configuration block. Upon all devices have been configured, this stage ends and SONAr changes the state of the network to “started”.

5.2 Device Plug-and-Play Scenario

When a new NE is connected to the network, such as an OpenFlow switch, it usually initiates a self-configuration process in which an address is requested through the DHCP protocol (Device Boot stage). SONAr implements a DHCP server capable of providing unique addresses and notifying management components such as TCoE, which starts the initial discovery process to find the attachment point of the new element (Discovery Stage - pt.1). After that, an access route can be calculated and deployed in order to provide connectivity between the SONAr components and the new device (Route Calculation and Deployment Stages). Once the access routes are deployed, the TCoE becomes able to perform the discovery routine with the plugged device (Discovery Stage - pt.2) and, in the end, notify the other components about the new available device.

Upon receiving the event that indicates a new available device, SCE starts the configuration process (Configuration Stage) in which the device is configured to use a specific controller (or CPI) and a default configuration is deployed according to the administrative goals defined as policies.

5.3 Results

Experiments were performed comparing the SONAr performance in topologies with different proportions. We tested the time spent by SONAr to initialize and extend networks with 1, 2, 4, 8, 16, 32, 64 and 128 devices. In each sub-experiment at least eight tests were performed (one for each network), and results with errors or abnormal delays were discarded.

In each test the configuration of the switches was reset and the components were removed from the

server in order to test boot on networks from scratch.

As can be seen in Figure 3, we tested three different network bootstrapping strategies: *boot 1* – connecting devices directly to controllers (without CPI) and configuring devices starting from the servers to the most distant nodes (from roots to leaves); *boot2* – connecting devices via CPI to controllers and configuring devices starting from servers to the most distant nodes (from roots to leaves); and, *boot 3* – connecting devices via CPI to controllers and configuring devices starting from the most distant nodes to the servers (from leaves to roots). The division of this experiment in three aims to identify the overhead resulting from the use of CPI and the differences between the configuration orders, which is due to the fact that when configuring a particular switch it is necessary to wait for the deployment of all flows before to proceed configuring the next accessible switches. This occurs because when configuring the OpenFlow protocol on the switch bridge the default learning-switch behavior is disabled preventing neighboring devices from being accessed without specific flows being created. By starting from the "leaves" all switches can be accessed by default learning-switch behavior.

Figure 2 demonstrates that the total bootstrapping time in all experiments increases linearly according to the number of devices. It also shows that the overhead introduced for using the CPI component is minimal and does not change the linear behavior of bootstrapping, which can be observed when comparing the times between *boot 2* and *boot 3* strategies. The results also show that when comparing *boot 1* and *boot 2* strategies in networks with more than 4 devices positioned at a maximum distance of 3 hops, the order of configuration from leaves to roots presents a reduction of between 20 and 40 % of total time.

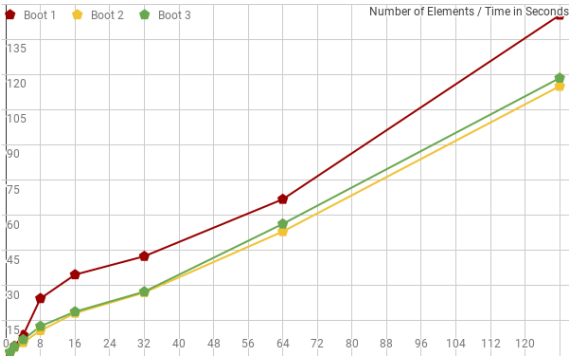

Figure 2: Time Comparison between the Three Bootstrapping Strategies Tested.

Figure 3 presents the time spent in each stage of bootstrapping using the strategy *boot 3*. As can be observed, the time of all stages increases linearly according to the number of devices. The routing time is almost insignificant compared to the time of other stages, and this is certainly related to the simplification of the multi-objective analysis as a shortest-path problem and by the absence of any integration with the external system. The Figure 3 also shows that the growth of the discovery time through the scenarios tends to be lighter, and this is explained due to the multi-threaded breadth-first search strategy and the topology design of the scenario. Every time a device is discovered, more devices become eligible for discovery, and this causes an acceleration of the process. The growth of configuration time is more uniform and tends to increase linearly at almost the same rate as the number of devices. The Overhead time is basically the time difference between process startup and termination excluding Discovery, Routing, and Configuration times.

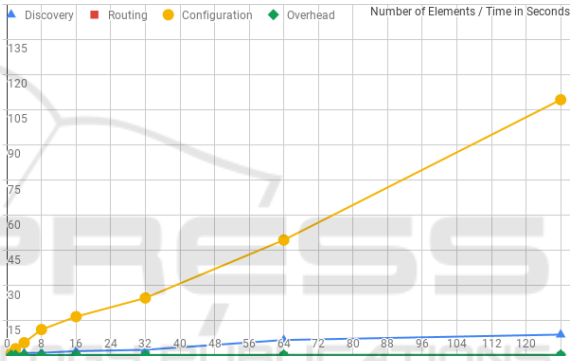

Figure 3: Time Spent at Each Stage of Bootstrapping Strategy 3: With Interceptor and Configuring Devices from Leaves to Root.

As can be seen in Figure 4, we tested three different network plug-and-play strategies: *pnp 1* – connecting devices directly to controllers (without CPI); *pnp 2* – connecting devices via CPI to controllers; and *pnp 3* - connecting devices via CPI to controllers and using the "delayed discovery" approach. The strategy *pnp 3* is faster because it avoids waiting for the device to became available to discovery. This is because the new switch needs to perform routines with the ARP and LLDP protocols to be able to communicate and discover neighboring switches. Performing the discovery stage right after configuring access flows can cause first attempts to fail until ARP/LLDP runs. In order to reduce this waiting time, a strategy called "delayed discovery " was proposed in which the new switch is only discovered after being configured. This strategy only works with CPI, because we need to know the interconnection point of the new switch without performing a discovery procedure. Figure 4 also shows that using a CPI can improve the plug-and-

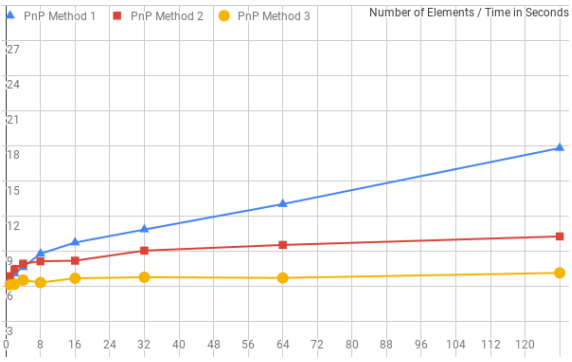

Figure 4: Time Comparison between the Three Plug-and-Play Strategies Tested.

play speed due to the ease of knowing the attachment point by simply checking the PACKET_IN primitive from OpenFlow protocol which is intercepted by CPI.

Figure 5 shows the time spent at each stage of the plug-and-play with the *pnp 3* strategy. Six measurements were made: Switch Boot (time spent executing the switch initialization script: starts with initialization of the container and ends with the assignment of an address via DHCP); Waiting (indicates time spent communicating between components, reaction time to events, and synchronization: comprises the difference between the total plug-and-play time and the sum of times of the other steps); Channel Routing (time spent calculating the best path to access the new switch); Channel Configuration (time taken to apply the new flow rules to the topology); Discovery (time taken to discover and validate the new switch); and, Configuration (time taken to configure the new switch via OVSDB and deploy streams via SDN Controller). As shown in the Figure 5, the time used with the *pnp 3* strategy in all topologies tested showed low variation, what indicates that the time for plug-and-play devices with SONAr using this strategy tends to be constant.

The experiments presented in this section show that SONAr provides a solution capable of automating

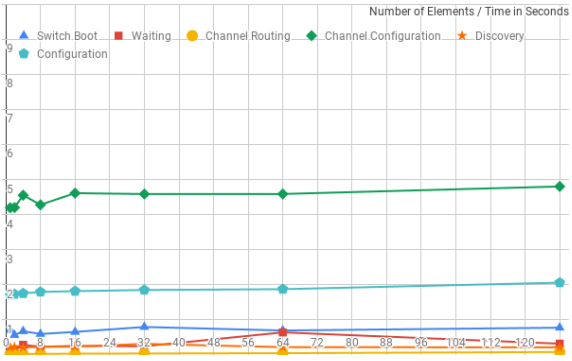

Figure 5: Time Spent at Each Stage of Plug-and-Play Strategy 3: Using CPI and "delayed Discovery".

the bootstrapping and plug-and-play of a self-defined network and demonstrated the viability and validity of the proposal of this work. The results also show the general behavior of the time spent at each stage, which is useful in network planning and also proves the polynomial complexity of the solution.

6 CONCLUSION

This paper presented a conceptual solution to enable self-organizing networks focusing on self-configuration property operations. Initially, SONAr was presented with details about its components, layers and design. Then, a case study with self-configuration was conducted to verify the viability and applicability of SONAr solution to address some fundamental aspects of self-management: bootstrapping and plug-and-play. The results were satisfactory and demonstrated that SONAr can be applied to reduce operating costs related to self-configuration. We also conducted a review of the requirements of telecommunications companies and their administrators/network operators. These requirements allowed the specification of SONAr presented in this paper.

Despite the gains in terms of OPEX, since SONAr will automatically perform the configurations of the network, the flexibility and the speed – with which the infrastructure will respond to incidents – is essential to highlight the improvement in network security, since the equipment no longer being accessed by humans for the purpose of (re)configurations.

SONAr solution is actually completely specified in our research group and, for sake of space, a summary was presented in this document. It is important to note that this project is being developed from experiences with a local telecommunication operator. Most components showed in Figure 1 are already finished and some components are being developed at the present moment. The experiments presented in this paper show the booting and configuration scenarios, which are the first steps to pass the computer network from the non-operational to the operational state. The experiments related to healing, optimizing, protection and others are not presented here due to space limitations.

The future works are to: (i) finalize the development of all SONAr components, using programming languages and frameworks widely used in industry; (ii) list many use cases to evaluate all aspects of a real telecommunications company; and (iii) compare SONAr with other projects, integrating it where possible.

ACKNOWLEDGEMENTS

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001. This research also received the support from PROPP/UFU.

REFERENCES

3GPP (2018a). Telecommunication management; Self-configuration of network elements; Concepts and requirements. Technical Specification (TS) 32.501, 3rd Generation Partnership Project (3GPP).

3GPP (2018b). Telecommunication management; Self-Organizing Networks (SON); Concepts and requirements. Technical Specification (TS) 32.500, 3rd Generation Partnership Project (3GPP).

3GPP (2018). Telecommunication management; Self-Organizing Networks (SON); Concepts and requirements. Technical Specification (TS) 32.500, 3rd Generation Partnership Project (3GPP).

3GPP (2018a). Telecommunication management; Self-Organizing Networks (SON) Policy Network Resource Model (NRM) Integration Reference Point (IRP); Requirements. Technical Specification (TS) 32.521, 3rd Generation Partnership Project (3GPP).

3GPP (2018b). Telecommunication management; Self-Organizing Networks (SON); Self-healing concepts and requirements. Technical Specification (TS) 32.541, 3rd Generation Partnership Project (3GPP).

Abdallah, S., Elhajj, I. H., Chehab, A., and Kayssi, A. (2018). A network management framework for sdn. In *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–4.

Bainbridge, L. (1983). Ironies of automation. *Automatica*, 19(6):775–779.

Barona López, L., Valdivieso Caraguay, Á., Sotelo Monge, M., and García Villalba, L. (2017). Key technologies in the context of future networks: operational and management requirements. *Future Internet*, 9(1):1.

Bastug, E., Bennis, M., Medard, M., and Debbah, M. (2017). Toward Interconnected Virtual Reality: Opportunities, Challenges, and Enablers. *IEEE Communications Magazine*, 55(6):110–117.

Boutaba, R. and Aib, I. (2007). Policy-based Management: A Historical Perspective. *Journal of Network and Systems Management*, 15(4):447–480.

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.

Cui, C., Deng, H., Telekom, D., Michel, U., Damker, H., Italia, T., Guardini, I., Demaria, E., Minerva, R., and Manzalini, A. (2012). Network Functions Virtualisation. *SDN and OpenFlow World Congress*.

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.

Duez, P. P., Zuliani, M. J., and Jamieson, G. A. (2006). Trust by Design: Information Requirements for Appropriate Trust in Automation. In *Proceedings of the 2006 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '06, Riverton, NJ, USA. IBM Corp.

Feng, S. and Seidel, E. (2008). Self-organizing networks (SON) in 3gpp long term evolution. *Nomor Research GmbH, Munich, Germany*, pages 1–15.

Ganek, A. G. and Corbi, T. A. (2003a). The dawning of the autonomic computing era. *IBM systems Journal*, 42(1):5–18.

Ganek, A. G. and Corbi, T. A. (2003b). The dawning of the autonomic computing era. *IBM systems Journal*, 42(1):5–18.

Hariri, S., Guangzhi Qu, Modukuri, R., Huoping Chen, and Yousif, M. (2005). Quality-of-protection (QoP)-an online monitoring and self-protection mechanism. *IEEE Journal on Selected Areas in Communications*, 23(10):1983–1993.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74.

Moysen, J. and Giupponi, L. (2018). From 4g to 5g: Self-organized network management meets machine learning. *Computer Communications*, 129:248–268.

Neumann, J. C. (2014). *The Book of GNS3*. No Starch Press, San Francisco, CA, USA, 1st edition.

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.

Ramirez-Perez, C. and Ramos, V. (2016). Sdn meets sdr in self-organizing networks: fitting the pieces of network management. *IEEE Communications Magazine*, 54(1):48–57.

Sanchez, J., Yahia, I. G. B., Crespi, N., Rasheed, T., and Siracusa, D. (2014). Softwarized 5g networks resiliency with self-healing. In *1st International Conference on 5G for Ubiquitous Connectivity*, pages 229–233.

Simsek, M., Aijaz, A., Dohler, M., Sachs, J., and Fettweis, G. (2016). 5g-Enabled Tactile Internet. *IEEE Journal on Selected Areas in Communications*, 34(3):460–473.

Wickboldt, J. A., Jesus, W. P. D., Isolani, P. H., Both, C. B., Rochol, J., and Granville, L. Z. (2015). Software-defined networking: management requirements and challenges. *IEEE Communications Magazine*, 53(1):278–285.

Zaidi, Z., Friderikos, V., Yousaf, Z., Fletcher, S., Dohler, M., and Aghvami, H. (2018). Will SDN Be Part of 5g? *IEEE Communications Surveys & Tutorials*, 20(4):3220–3258.