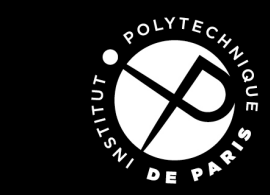
Toward auto-configuration in software networks

Wassim Sellil Atoui



INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2020IPPAS015

Thèse de doctorat

Toward Auto-configuration in Software networks

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à TELECOM SudParis

École doctorale n°ED 626 Ecole Doctorale de l'Institut Polytechnique de Paris (ED IP Paris)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Chatillon, le 21 septembre 2020, par

WASSIM SELLIL ATOUI

Composition du Jury :

Walid Gaaloul Professeur, TELECOM SudParis, France	Directeur de thèse
Imen Grida Ben Yahia Ph.D., Orange Labs, Chatillon, France	Encadrante de thèse
Filip De Turck Professeur,Université de Ghent, Ghent, Belgique	Rapporteur
Panagiotis Demesticha Professeur, Université du Pirée, Pirée, Grèce	Rapporteur
Laura Galluccio Professeur, Université de Catane, Catane, Italie	Examinatrice
Noel Crespi Professeur, TELECOM SudParis, France	Président du jury
Nour Assy Ph.D., Université internationale libanaise, Beirut,Liban	Examinatrice

To my mother,
to my father,
to my loving family
for their unconditional
love and support.
To Nyzark...

Acknowledgment

Foremost, I would like to express my sincere gratitude to both of my thesis supervisors, Dr. Imen Grida BenYahia and Pr. Walid Gaaloul for giving me this opportunity to pursue a Ph.D. at both Orange labs and Institut Polytechnique de Paris. Their guidance and devotion helped me achieve this work. It was a great honor and a pleasure to have them both as supervisors.

I would like to thank Dr. Nour Assy for collaborating with me on various aspects of this work.

A special thanks to Pr. Panagiotis Demestichas and Pr. Philip De Turck who evaluated and reviewed my manuscript.

I thank also Dr. Bertrand Decocq, the manager of team BRAINS at Orange Labs, for his guidance, kindness, and time. He was always available at times of need. It was an honor to work with him.

I am grateful to all my co-workers at Orange Labs and Telecom SudParis who helped me to integrate and made this experience memorable. Thanks also to all of my friends in France and Algeria for their support.

Last but not least, I am extremely grateful to my father, L'hadi, my mother, Moufida, my sister, Randa, and my brother, Amine, for their unconditional love and support throughout the last 29 years.

Résumé

Les réseaux logiciels ont le potentiel de porter l'infrastructure réseau à un niveau plus avancé, un niveau qui peut rendre la configuration autonome. Cette capacité peut surmonter la complexité croissante des réseaux actuels et permettre aux entités de gestion d'activer un comportement efficace dans le réseau pour une amélioration globale des performances sans aucune intervention humaine. Les paramètres de configuration peuvent être sélectionnés automatiquement pour les ressources réseau afin de faire face à diverses situations que les réseaux rencontrent, comme les erreurs et la dégradation des performances. Malheureusement, certains défis doivent être relevés pour atteindre ce niveau avancé de réseaux. Actuellement, la configuration est encore souvent générée manuellement par des experts du domaine dans d'énormes fichiers semi-structurés écrits en XML, JSON et YAML. C'est une tâche complexe, sujette aux erreurs et fastidieuse à accomplir par les humains. De plus, il n'y a pas de stratégie formelle, à part l'expérience et les meilleures pratiques des experts du domaine pour concevoir les fichiers de configuration. Différents experts peuvent choisir une configuration différente pour le même objectif de performances. Cette situation rend plus difficile l'extraction des fonctionnalités des fichiers de configuration et l'apprentissage des modèles susceptibles de générer ou de recommander automatiquement la configuration. De plus, il n'y a toujours pas de consensus sur un modèle de données de configuration commun dans les réseaux logiciels, qui a abouti à des solutions hétérogènes, telles que: TOSCA, YANG, Hot, etc. qui rendent la gestion de réseau de bout en bout difficile. Dans cette thèse, nous présentons nos contributions qui abordent les défis susmentionnés liés à l'automatisation de la configuration dans les réseaux logiciels. Pour aborder le problème de l'hétérogénéité entre les fichiers de configuration, nous proposons un cadre sémantique basé sur des ontologies qui peuvent fédérer des éléments communs à partir de différents fichiers de configuration. Pour le défi de génération automatique de la configuration, nous proposons deux contributions, une contribution qui considère les réseaux de neurones profonds pour apprendre des modèles de fichiers de configuration pour recommander la configuration et une autre contribution basée sur une approche basée sur un modèle configurable pour aider automatiquement la conception de la configuration des fichiers de description.

Abstract

Software networks have the potential to move the network management to a more advanced level, a level where the network can automatically configure itself. This ability can overcome the rapidly growing complexity of current networks, and allow management entities to enable an effective behavior in the network for overall performance improvement, with less human intervention Configuration parameters can be automatically selected for network resources to cope with various situations that networks encounter like errors and performance degradation. Unfortunately, some challenges need to be tackled to reach that advanced level of networks. Currently, the configuration is still often generated manually by domain experts in huge semi-structured files written in XML, JSON, and YAML. This is certainly a complex, error-prone, and tedious task to do by humans. Also, there is no formal strategy except experience and best practices of domain experts to design the configuration files. Different experts may choose the different configurations for the same performance goal. This situation makes it harder to extract features from the configuration files and learn models that could automatically generate and recommend configurations. Moreover, there is still no consensus on a common configuration data model in software networks, which resulted in heterogeneous solutions, such as TOSCA, YANG, Hot, etc. that making the end-to-end network management difficult. In this thesis, we address the problem of automating the configuration in software networks. We propose two main contributions for tackling the aforementioned challenges. To address the problem of heterogeneity between the configuration files we propose a semantic-based framework that automatically maps configuration elements from heterogeneous data models. The framework extracts ontologies from the configuration files and builds a common configuration view by mapping similar elements from the ontologies using our proposed algorithms. Regarding the problem of configuration generation, we propose two contributions to assist the service providers to design the configuration files and automate their generation. We propose a contribution that is based on deep neural networks and another one that is model-driven. The deep neural network contribution is an approach that learns from previously made configuration file models that recommend and complete the files with configurations. The model-driven approach assists service providers to design and generate configuration files. The approach is a configurable model that merges similar component elements from different configuration files into a single model. This model captures and learns configuration variabilities from the files. The model is a tree-structured graph that represents the component elements that appear in the descriptors along with configurable connectors that capture variable structures of configurations

Table of contents

List of Tables

List of Figures

Chapter 1

General Introduction

1.1 Research context: Software Networks

Software networks (or Software-driven networks) emerged as the next level in the evolution of computer networks. The evolution was necessary to cope with the challenges related to the new technology ecosystem, e.g. cloud computing and internet of things, in which traditional networks have struggled to keep up with the transition.

Traditional networks (legacy networks) are an impediment to today's technology era for many reasons. One of the main reasons is that the infrastructure of traditional networks is rigid. Any new changes added to the network would take time to be applied, requires human interventions, and in a lot of cases needs a physical hardware installation. Another reason is that traditional networks are dependent on closed and proprietary hardware that inhibits network users from exploiting alternative best-of-breed technologies. Moreover, the hardware in traditional networks is designed with a combined integration of the control plane and the data plane. The configuration of the hardware in this case is usually done manually in a box-by-box fashion.

Software networks provide answers to the traditional network limitations [58]. The network design is redefined by new paradigms that build the networks at a lower cost and with a greater scope for innovation in network services. Software networks offer new opportunities to transform the economics of business network while at the same time accelerating the ability to design and deploy new service capabilities independently of the operated network (fixed, mobile, long-distance, etc.) and the offered service (data, voice, content delivery, etc.) [66].

The main paradigms in software networks are Software-defined network (SDN) and [63] and Network function virtualization (NFV) [77]. On one hand, SDN enables network programmability by offering the capacity to initialize, control, change, and manage network behavior dynamically via open interfaces. On the other hand, NFV decouples network functions from proprietary hardware appliances and run them as software in virtual machines (VMs). The emergence these new paradigms in has created new hopes for the progress of network management. With software networks, networks will likely be upgraded with self-management capabilities that allow them to be autonomic. Autonomic networks could overcome the rapidly growing complexity of current networks, and adopt an optimal management behavior for overall performance

improvement in a fully autonomous way, without any human intervention [118]. Other benefits of autonomic networks are the following:

- Operation and maintenance costs will be reduced significantly, and human resources could be deployed to handle higher-value activities.
- Quality and efficiency will be improved given that human errors are reduced. Configuration parameters will be automatically selected for network resources that need to be configured.
- The parameters are adapted to new or unexpected situations like errors, failures, security threats, performance degradation, etc. [79].

Autonomic networks need to ensure self-configuration. A capability allows the network to adapt with less human interventions to new changes in the network by configuring new components seamlessly or modifying the parameters in the network according to the overall global state. In this thesis, we aim to investigate new approaches that contribute toward enabling auto-configuration in software networks.

1.2 Research problem: automatic configuration

In this section, we present the main challenges related to automatic configuration in software networks. Before that, we explain briefly how the configuration is currently made to motivate the direction of our contributions toward enabling automatic configuratio (auto-configuration) in software networks.

The key feature of software networks is their ability to be completely programmable. This means that theoretically, it is possible to write a piece of software that could manage automatically the entire network. The main enablers for programmability are application programming interfaces (APIs). They allow network components to expose their functionalities and be managed by other entities in the network. The APIs simplify the configuration of network components and enhance their agility. A set of complicated configuration operations, for example enforcing policies, could be specified in a single file and transmitted to the network device. The APIs in the SDN paradigm is situated between the abstraction planes (described in chapter 2). While in NFV, the programming interfaces are located between the management components (described in chapter 2).

The configuration in software networks is model-driven. It is specified in semi-structured files like JSON and YAML, following a predefined data model such as YANG, TOSCA, etc. The data model that is used has to be also the same used by the network components that are configured. Otherwise, the APIs would not be able to execute the operations that are expected.

automatic configuration in software networks will take advantage of the programmability of the network to automate the configuration. New network components could be configured using their APIs seamlessly without human intervention,

in a plug and play fashion. Likewise, the behavior of the network could be adapted to unexpected situations while maintaining the required performance by doing adequate configuration in the network.

To ensure automatic configuration in software networks, it is necessary to account for the APIs used by network components. Any generated configuration should be interpretable by the APIs of network components. We have identified three challenges that are important to be tackled to move toward automatic configuration. The challenges are namely, (i) the heterogeneity between configuration data models, (ii) the configuration generation, and (iii) the configuration propagation. We describe next each of these three challenges.

1.2.1 Heterogeneity between configuration data models

There are constantly new proposals and solutions that are emerging in software networks from different vendors, providers, open-source organizations, etc.. The network is a heterogeneous environment where components from different vendors cooperate and share information. It is important therefore to account for the data model that is used by the components when making the configuration, otherwise, it cannot be interpreted and processed correctly.

Unfortunately, there is still no consensus on a common data model used to make the configuration by network components from all the competition. There is currently a plethora of data models based on languages like TOSCA (Topology and Orchestration Specification for Cloud Application) [2] or YANG (Yet Another Next Generation).

To enable automatic configuration, it is important to automate the interpretation of configuration that is issued from heterogeneous data models and also to generate configurations with adequate data models. The reason is that, in the absence of a common data model, it is not possible to construct a global network view, to which the automatic configuration module can reason on or analyze to trigger configuration actions. It is also necessary that the generated configurations follow the same data model as the configured network components.

This situation raises two research questions:

- RQ1: How to automatically interpret configurations from heterogeneous sources?
- RQ2: How to automatically generate configurations to a targeted data model?

Note that a solution that is based on a static translation between heterogeneous data models is not viable in an environment that is constantly evolving. Keeping the translation updated to all the data models is a tedious task. We investigate in this thesis approaches that provide some insights on penitential solutions to overcome this challenge.

1.2.2 Configuration generation

To automatically configure network components without human intervention, auto-configuration should help generate and recommend configurations that are adequate with the expectation of network users. The problem is that there is no formal strategy on how to choose the best configuration for each given situation. The configuration files in software networks are often designed and created manually. Which is a highly complex, time-consuming, and tedious task.

This challenge raises the following research question:

- RQ3: How to automatically generate coherent configurations in software networks?

We investigate in thesis model-driven approaches that ease the generation of configurations by assisting service providers in designing configuration files. Also, we investigate approaches based on deep learning that automate the generation of configuration in an unsupervised manner.

1.2.3 Configuration propagation

automatic configuration could generate a configuration that triggers a chain reaction of other changes in network components so that the system remains in a coherent state. For example, changing the network interfaces of virtual functions in NFV will impact the network service in which this virtual function is implemented, and also the virtual functions that are linked to it. It is therefore important to identify the configuration relations between network components and also the order of changes. Formalizing the dependencies between configurations helps to create constraints that ensure that the system remains coherent. The relations could be used for root cause analysis to ensure the recoverability of the system by undoing previous changes, in case of failures.

The problem is that software networks are an environment where multiple network components cooperate. It is a complex task for a human to identify the dependencies between the configuration. This situation raises the following research question:

- RQ4 : How to identify automatically the dependencies between configurations in software networks?

1.3 Research contributions

In this thesis, we investigate solutions that are in the scope of answering the questions made in the previous section. We propose approaches that contribute toward enabling automatic configuration in software networks. To study new solutions to the aforementioned challenges, we have to focus our attention to specific uses cases. For

that reason, we choose two use cases, one in SDN and another one in NFV, as they are the main paradigms in software networks Our contributions could be generalized and adapted to software networks.

Our first contribution is related to the problem of heterogeneity between configuration data models. To investigate a solution, we focus our attention on an SDN architecture where multiple controllers are deployed. SDN controllers have to cooperate to share a global network view that allows them to localize network devices. The problem is that the SDN controllers may expose their functionalities (via APIs) using different data models. The difference could be in the structure of the data, the syntactical representation of the data, or the semantics of the data. The cooperation, in this case, becomes not possible as the SDN controllers could not process each other's data.

To address this problem, we propose a semantic-based framework that automatically maps configuration elements from heterogeneous data models. This contribution is in the scope of answering the RQ1 and RQ2. The framework builds an ontology from configurations of the SDN controller, referred to as local ontology, to represent the SDN controller concepts and the relation between them. The local ontologies are automatically extracted from the configuration files using our method. The global network view is constructed by mapping similar elements from the ontologies using our proposed algorithms. More concretely, our contributions can be summarized as follows:

- We propose a semantic-based framework that encompasses an approach that extracts a local ontology from each controller and incorporates algorithms that map the ontologies together to form the overall network view.

- We define a central global ontology model that represents the domain knowledge of SDN with OWL. The global ontology describes the concepts that constitute an SDN architecture.

- We adapt our framework to two scenarios:

  - A centralized scenario where the SDN controllers expose their network views to a centralized entity that builds the network view
  - A distributed scenario, where SDN controllers exchange in peer to peer their local network views and build locally the global view.

- We establish two ontology mapping algorithms. One for the centralized scenario, to map the controller's local ontologies to the global ontology and one for the distributed scenario to map the local ontologies of the SDN controllers.

- We propose an ontology extraction method from the SDN controller configuration files that are described in JSON format.

- We evaluate the performance of our framework, in both distributed and centralized scenarios, in terms of the matching accuracy of our mapping algorithms and the execution time. We test the performance over different network topologies in a heterogenous multi-controller SDN architecture composed by controllers like ODL and ONOS.

Regarding the problem of configuration generation, we propose two contributions to assist the service providers to design the configuration files and automate their generation. The first contribution is based on deep neural networks and the second one is a model-driven approach. These contributions are in the scope of anwsering the RQ3.

We consider an NFV architecture to implement and test our proposed solutions. In NFV, service providers have to associate with their virtual network functionalities description files before the onboarding of the VNFs. The description files indicate the deployment and operational behavior of the functionalities in terms of connectivity and resource requirements. The descriptor files are large files that are designed manually by the service providers, which is complex, tedious, and error-prone. Moreover, the descriptor files contain the configuration of numerous components that are dependent on each other, and there is no formal strategy to select the best configuration. This scenario is therefore adequate for our investigating solution to assist and automate the generation of configurations.

We propose at one hand approaches based on deep neural networks that learn from previously made descriptor file models that recommend and complete configurations. These contributions are briefly summarized as follows:

- We propose a learning framework based on neural network architectures that is divided into three phases: the preparation phase, learning phase, and model tuning phase.

- In the preparation phase, we process the descriptors files into a format that is suited for the neural network architectures. We use a word embedding approach to representing the data that is extracted from the descriptor files based on its semantics.

- In the learning phase: we propose two neural network architectures:

  - Convolutional neural network architecture to learn a recommendation model for the descriptors
  - Long short term memory architecture to learn a completion model for the descriptors

- We evaluate afterward the generated models (recommendation and completion) in terms of their accuracy of prediction.

On the other hand, we propose a model-driven approach that assists service providers to design and generate descriptor files. We propose a configurable model that merges similar component elements from different descriptor files into a single model. This model captures and learns all the configuration variabilities from the descriptors. The model is a tree-structured graph that represents the component elements that appear in the descriptors along with configurable connectors that capture variable structures of configurations. This approach is also in the scope of answering the RQ3. The model-driven contributions can be briefly summarized as follows:

- We formalize a configurable deployment descriptor model that capitalizes on a catalog of descriptors
- We propose an algorithm based on machine learning (K-medoids) to search and cluster similar elements from different descriptors and construct the configurable model.
- We propose an approach that extracts useful and implicit knowledge from the configurable model. The approach derives configuration guidelines by mining the configurable deployment descriptor model and a repository of VNFD models. The guidelines capture the dependencies and the relations between the configuration of VNFD elements.

1.4 Research publications

We have published our contributions in several scientific venues:

- **J2**: Wassim Sellil Atoui, Nour Assy, Walid Gaaloul, Imen Grida Ben Yahia: Assisting deployment descriptor design in NFV. International Journal of Network Management, 2020 (invited to submit an extended version from a conference paper)
- **J1**: Wassim Sellil Atoui, Nour Assy, Walid Gaaloul, Imen Grida Ben Yahia: Configurable Deployment Descriptor Model in NFV. Journal of Network and Systems Management 28, 693–718, 2020
- **C5**: Wassim Sellil Atoui, Imen Grida Ben Yahia, Walid Gaaloul: Token embedding for deployment descriptors in NFV. IFIP Networking, 2020
- **C4**: Wassim Sellil Atoui, Nour Assy, Walid Gaaloul, Imen Grida Ben Yahia: Learning a Configurable Deployment Descriptors Model in NFV. IEEE/IFIP Network Operations and Management Symposium, 2020
- **C3**: Wassim Sellil Atoui, Imen Grida Ben Yahia, Walid Gaaloul: Virtual Network Function Descriptors Mining Embeddings and Deep Neural Networks. IFIP/IEEE International Symposium on Integrated Network Management, 2019: 515-520

- **C2**: Wassim Sellil Atoui, Imen Grida Ben Yahia, Walid Gaaloul: Using Deep Learning for Recommending and Completing Deployment Descriptors in NFV. IEEE Conference on Network Softwarization, 2019: 233-2352
- **C1**: Wassim Sellil Atoui, Imen Grida Ben Yahia, Walid Gaaloul: Semantic-Based Global Network View Construction in Software-Defined Networks with Multiple Controllers. IEEE Conference on Network Softwarization, 2018: 252-256

1.5 Thesis outline

This doctoral thesis is organized in seven chapters:

- **Chapter 2: Background Information on Software Networks** introduces the basic concepts related to our research and needed to understand the rest of the work. In this chapter, we present two of the main paradigms in software networks. The first paradigm is Software-defined networks and the second one in network function virtualization. We present their architecture, their benefits, and the challenges that they need to overcome.
- **Chapter 3: State of The Art** provides an exploration and a thorough analysis of the state of the art around the two challenges that we are tackling in this thesis. Mainly we discuss the contributions related to : (i) handling automatically the heterogeneity between configurations and (ii) Generating automatically configurations.
- Chapter 4: **Handling the heterogeneity between configuration data models** presents our approach to handle the heterogeneity in software networks. More precisely, we focus on a use case in SDN and propose a framework that enables heterogeneous multi-controllers SDN platforms to construct a global network view.
- **Chapter 5: Deep Learning for automatic configuration generation** introduces our approach based on deep neural networks that learns from a set of configurations a model that could recommend or complete the additional configuration.
- **Chapter 6: Model-driven automatic configuration generation** introduces our model-driven approach that assists service providers in designing and generating deployment descriptors in NFV.
- **Chapter 7: Conclusion and Future Work** summarizes the proposed contributions and presents an outlook on the potential perspectives that we intend to tackle in the short-medium term.

Chapter 2

Background on Software Networks

Contents

2.1 Introduction

Software networks are new paradigms that enable the network programmability through a set of tools used to deploy, manage, and troubleshoot network devices. Compared to traditional networks, software networks accelerate service deployment and delivery, reduce the IT costs, increase resource flexibility, and provide greater cloud integration.

The term software in software networks refers to the possibility that intelligent software can be designed to manage and deal with a single node in the network or the

entire network as a unified single element. This is possible with the help of APIs that are used by the software to gather data or to run configurations and management operations. In this chapter, we present an overview of the two main paradigms of software networks, SDN, and NFV.

2.2 Software Defined Networks (SDN)

SDN is a paradigm inspired by cloud computing that overcomes the shortcomings of traditional network management and improves its performance [63]. In traditional networks, switches are manually set up by domain experts following a specific vendor CLI language. This becomes a very complicated and error-prone task for companies running strongly virtualized environments along with large networks. SDN decouples the vertical integration by separating the control plane from the data plane. The switches in the infrastructure are just forwarding devices freed from any vendor lock-down and the control logic of the devices is centralized in a single entity in the network called the controller. The controller is responsible for policy enforcement, network configuration, topology management, link discovery, flow table, etc..

2.2.1 SDN Architecure

The architecture of this paradigm is typically divided into three layers: infrastructure layer, control layer and application layer, and two interfaces: southbound interface and northbound interface. Figure 2.1 shows a simplified SDN architecture.

2.2.1.1 Infrastructure layer

This layer contains physical and virtual forwarding devices. The infrastructure layer support basic operations that are exposed to the SDN controller like packet forwarding, caching, transcoding, and monitoring [63]. The SDN controller dynamically installs packet processing rules onto the devices.

2.2.1.2 Control layer

The control layer is responsible for managing the infrastructure devices and making decisions on how packets should be forwarded by the devices. This layer could be composed of one or multiple controllers. Each controller is responsible for a set of network devices in the network infrastructure and communicates with these devices using the southbound APIs to get the state of the traffic or to push decisions about the traffic flow. Essentially it is a centralized set of software-based SDN controllers that combined together have an abstract view of the whole network infrastructure, enabling the management entities to customize policies across the infrastructure devices.

The control layer provides a set of functionalities to the application layer using the northbound APIs. These functionalities could be of the following [63]:

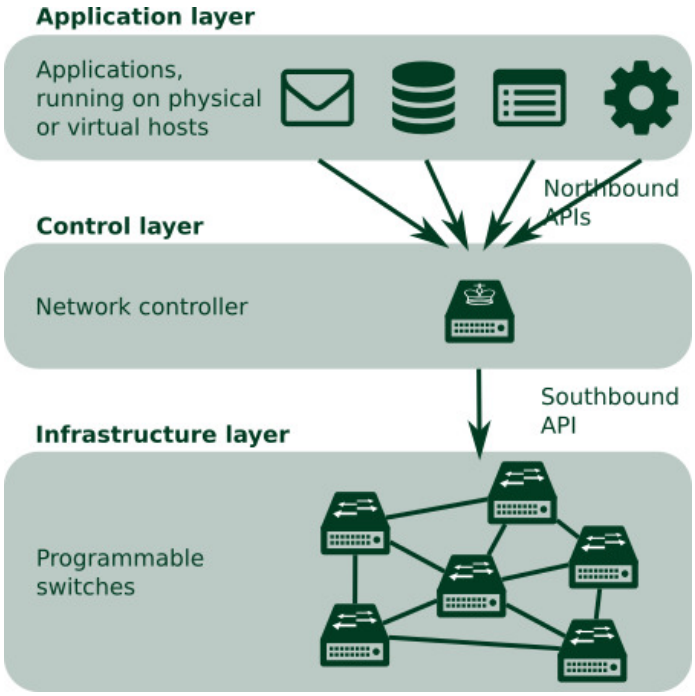


Figure 2.1: Overview of a typical SDN architecture [4]

- Topology discovery and maintenance
- Packet route selection and instantiation
- Path failover mechanisms
- Install of the forwarding rules on the forwarding tables based on the requested performance from the applications and the network security policy
- collect status information about the infrastructure layer.

2.2.1.3 Application layer

The application layer contains application and services that dictates the network behavior. The applications interact with the controller to achieve a specific network performance and fulfill the requirements of service providers. For this purpose, the SDN controller provides an abstracted view of the network and a set of functionalities via the northbound APIs that applications use in order to specify the desired network behavior.

2.2.2 Communication between SDN layers

The communication between the SDN layers is achieved through a set of APIs. There are typically two types: the northbound APIs and the southbound APIs.

Southbound APIs are communication interfaces used to connect the infrastructure layer to the control layer. Their main goal is to provide functionalities that enable communication between the SDN controllers and the infrastructure devices, in order to discover the network topology, push the flow table, and implement the control policies. The OpenFlow protocol [40] is one of the popular southbound API for SDN.

Northbound APIs are communication interfaces that connect the control layer to the application layer. Their goal is to enable applications to easily manage network resources and capabilities. Compared to the southbound APIs, northbound APIs do not have a standard specification. This situation created a plethora of solutions, each with diverse features that make it difficult to federate multiple SDN controllers together in the network.

Additionally to these two interfaces **East-West interface** could exist when multiple SDN controllers are employed in the network [119]. They ensure communication between the controller in a horizontal manner. The cooperation, in this case, is direct between the controllers, as opposed to the cooperation via a mediation entity in the application layer.

OpenFlow: communication protocol between the controller and network devices

OpenFlow [40]is a communication protocol that is used to connect the SDN controller to the infrastructure devices via the southbound interface. This protocol exposes a set of basic functionalities used for handling the forwarding flow between the infrastructure devices and managing them. The controllers send the forwarding plane in a flow table to the devices and the devices keep the controller updated about the network state, i.e. if the links are down or when receiving a packet for which there is no forwarding instruction. The OpenFlow communication can be either on a secure connection like the Transport Layer Security (TLS) or on an unprotected TCP connection.

The Forwarding tables in OpenFlow are defined by the controller to instruct infrastructure devices on how packets are forwarded. The devices, called OpenFlow switches could have one or more flow tables and a group table. The devices check the flow tables to make a decision on how to forward the packets. These Flow tables contain a list of flow entries, each of which determines how packets belonging to a flow will be forwarded. Flow entries consist of: a rule, an action, and a statistic, as shown in Figure 2.2:

Rule: is a header used to match incoming packets. There are several supported Ethernet headers that are specified in OpenFlow specification, custom headers can

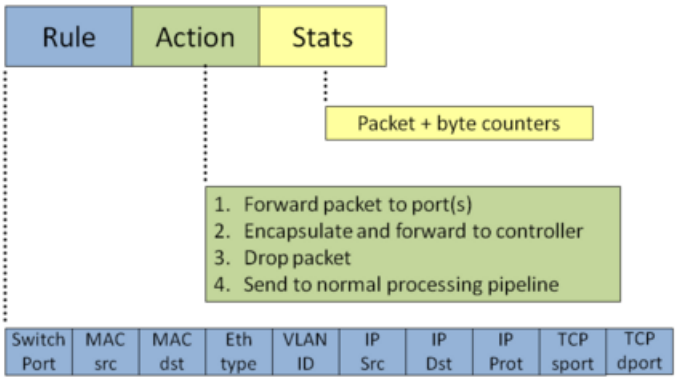


Figure 2.2: OpenFlow Flow table fields [5]

be additionally defined. The header contains either a specific value against which the corresponding parameter is compared or a value indicating that the entry that is not included.

Action: Define an action to take when a rule is matched with the packet traffic. There are several actions already defined in OpenFlow specifications. Examples of actions could be: forward to one or more ports, forward to the controller, drop the frame, and modify frame fields.

Statistics: This is a counter that is updated each time when a flow rule is matched with the packet traffic. It is to indicate the popularity of a specific flow. There are counters for every table, each flow, all the ports, and every queue. Also, a timer of the last activity and an initial set of the flow is maintained.

In OpenFlow , an OpenFlow Controller is a software server that interacts with the OpenFlow switch using the OpenFlow protocol. The protocol connects controller software to network devices so that controllers can configure network devices and inform where to forward packets. The interface to which the OpenFlow controller and the infrastructure switches communicate is called the OpenFlow channel. Through this interface, the controllers configure and manage the switches, receive events from the switches, and send packets out the switches (to add, update, and delete flow entries in flow tables). The OpenFlow channel is usually instantiated as a single network connection (using TCP or Encrypted connection).

2.2.3 SDN controller abstractions

The SDN controller enables three types of abstractions in the network, which constitute its major feature.

- Network state abstraction: SDN controllers provide a global network view to applications and services.
- Forwarding abstraction: The forwarding of packets in the network is handled indirectly by the controller. The network devices are executing instructions from the controller on how to forward the packets. This way, the infrastructure hardware is abstracted by the controller.
- Network operation abstraction: The controller allows a network application to express the desired behavior of the network without been responsible for implementing it.

2.2.4 Benefits of the SDN paradigm

SDN has improved a lot of aspects from traditional networks. For example:

- Centralized network management: The centralized view of the entire network eases the management and provisioning of new service in large networks.
- Control and data planes abstraction: This abstraction accelerates service delivery and provides more agility in provisioning both virtual and physical network devices from a central location.
- Holistic management: The management for physical and virtual devices can be elaborated in SDN through a single set of APIs, which is more accommodable for domain experts.
- Lower operating costs: SDN brings more efficiency in administrating the network. Network routines can be centralized and automated, which solves many administration issues and saves a lot of time.
- Reduced capital expenditures: in SDN, the infrastructure devices could easily be optimized and commoditized. They are repurposed easily by the controller. Therefore less expensive devices can be deployed for broader utilization since all the intelligence is centered at the controller.
- Easier content delivery: SDN provides the ability to shape and control data traffic is and makes it easier to implement quality of services (QoS) and ensure flawless user experience.

2.2.5 SDN issues and challenges

There are still open challenges regarding SDN that are actively tackled by the scientific community. Here are some of the considered topics:

- **Reliability and fault tolerance:** It is important that SDN stays available even in case failures in the controller [94]. An automatic solution should be triggered in that case to ensure the reliability of the system and the users should be informed about the coherent network state at any moment.
- **Scalability:** SDN have to handle the growing amount of traffic load and have to support new applications without impacting the level of service delivered to the users [82]. This capability needs to account for performance degradation caused by resource constraints, processing demands, and communication overhead.
- **Flexibility:** SDN should be designed in such a way that it could adapt its behavior to new changes and dynamic variations occurring in the network [52]. The network must be flexible enough to adapt and accommodate future developments and the needs of the network designers.
- **Elasticity:** SDN needs to be capable of ensuring the availability of its resources at every moment independently of the network load [52]. It needs to dynamically adapt to the workload changes inside the network. During higher demands, the maximum resources could be allocated to ensure a quality of service, while during lower demand, SDN should be able to reduce the resources to preserve the energy and lower the maintenance cost
- **Data model heterogeneity:** There are a plethora of data models used to define the northbound APIs like TOSCA, YANG, etc That makes it difficult to operate and federate different SDN controllers in the same architecture. This is could be the case for example when different controllers are dedicated to different specific domains. The controller has to cooperate to provide a coherent global view of the users and maintain a centralized abstraction of the network. Our first contribution is in the scope of this challenge. We study approaches that handle the heterogeneity between data models of SDN controllers.
- **Control abstraction:** SDN should be able to translate effectively the intention of the user or application to program the network. The programming interface of the framework of the SDN must coordinate the multiple asynchronous events at the switches to perform even simple tasks.
- **Performance and Security:** Security treats exists also in SDN. Network attacks could reduce the performance of the SDN. For example, a simple D-DOS attack may down the working of networks [29]. SDN should ensure the

integrity and the coherency of the managed resources and prevent the attacker from taking control in the network.

We listed here all the important challenges but we are addressing in this thesis scope only the heterogeneity.

2.3 Network Function Virtualization (NFV)

NFV is a software network paradigm that aims to transform the way that network operators manage networks. It is based on virtualization technologies that virtualize network functionalities and consolidate different network equipment into industry-standard high-volume servers and data center or at end-customer premises. NFV replaces custom-designed network equipment (vendor locked in) that dominated traditional network to off the shelf hardware that is used to deliver virtualized network functionalities. For example, what was in traditional networks router equipment can be in NFV virtual routing functionality that could be deployed on any hardware. The network functionalities in NFV can be chained together to form a network service, like an Intrusion detection system.

2.3.1 NFV Architecture

The architecture of NFV aims to add more agility and automation methods to deploy and manage widely distributed network infrastructure and resources. It includes three major components: the virtualized network functions (VNFs), NFV Infrastructure (NFVI) and NFV management and orchestration (MANO), as depicted in Figure 2.3.

Network Functions Virtualization Infrastructure

The NVFI includes at the bottom a physical layer that contains the computing hardware. This layer delivers the physical resources like the compute, storage, and network and software on which the VNFs are deployed and managed. Just above the physical layer sits a virtual layer that abstracts the hardware resources. It contains hypervisors and virtual machines and enables them to logically partition and provision the hardware resources in order to support VNFs. The virtualization of the resources has also the advantage of abstracting the type and localization of the hardware. The network resources could be therefore distributed across different locations in a transparent way to the user.

Virtual Network Functions

VNFs are virtual software that could run in one or multiple virtual machines on top of VNFI component. VNFs could be vRouters, vSwitches, vLoad Balancers, vFirewalls,

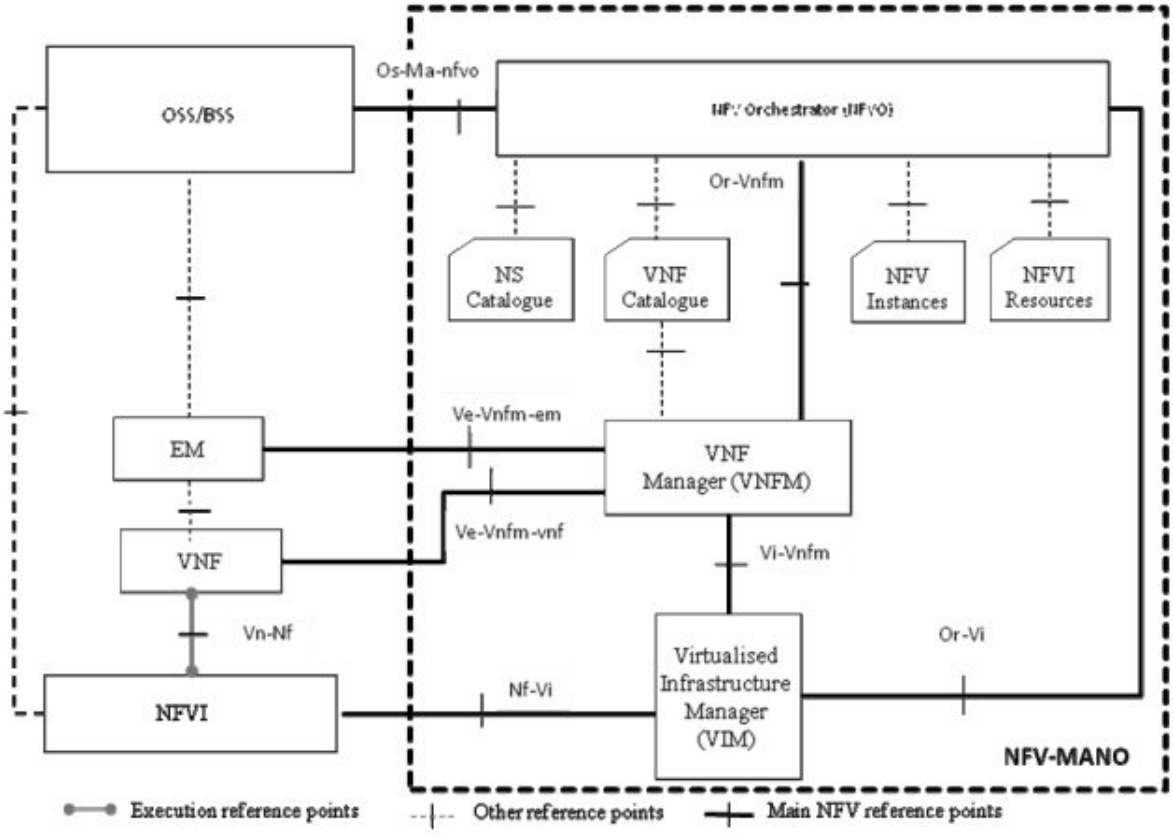


Figure 2.3: Overview of a typical NFV architecture [6]

etc. That could be chained together to form NSs like for example a virtual intrusion detection system or a virtual voice over IP. NFV enables the VNFs to be deployed on-demand without any delays. Preventing the need or on-site technical skills like in traditional networks. This component provides therefore the agility needed to respond and anticipate dynamic network performance or growing demands.

NFV Management and Network Orchestration

MANO is a framework developed and proposed by the ETSI working group [39]. The NFV MANO coordinates all resources of NFV (i.e the NFVI and the VNFs) running in a virtualized platform like a data center. It also defines the template to which service providers are using to describe the appropriate NFVI resources need for deploying the VNFs. The NFV MANO framework is mainly comprised of three functional areas:

- NFV Orchestrator handles VNF onboarding, lifecycle management, global resource management, and validation and authorization of NFVI resource requests.
- VNF Manager controls the VNF lifecycle management of instances, providing a coordination and adaptation role for NFVI and Element/Network Management Systems configuration and event reporting.
- Virtual Infrastructure Manager controls and manages the NFVI compute, storage, and network resources.

2.3.2 Communication between the NFV components

In NFV, the programming interfaces are located between the components of the NFV architecture, as illustrated in Figure 2.3. They are called reference points in the ETSI terminology. The reference points expose different functionalities that allow the NFV components to collaborate with each other. We summarize briefly in following the functionalities of the main reference points [39].

- Or-Vnfm: Defines the communication between the orchestrator and the VNFM. It specifies functionalities to configure VNFs and pass specific service data.
- Or-Vi : Defines the communication between the orchestrator and the VIM. It specifies functionalities to configure the NFVI.
- Vi-Vnfm: Defines the communication between the VNFM and the VIM. It also specifies functionalities that allow the VNFM to configure the NFVI.
- Ve-Vnfm: Defines the communication between the VNFM and the NFV. Specifies functionalities to onboard, deploy, and configure the VNFs, and also to manage their operational life cycle.

- Nf-Vi: Defines the communication between the VIM and the NFVI. Specifies functionalities to configure the physical hardware.

2.3.3 Benefits of the NFV paradigm

Like SDN, NFV also improves many aspects of traditional networks [58]. We describe in the following the main benefits of this paradigm.

- **Cost effective:** In NFV, lower-cost off the shelf hardware can be used to implement network functionalities. This one of the most important advantages in NFV.
- **Flexibility:** In NFV, the network operator can utilize off the shelf hardware instead of vendor-centric hardware. This adds flexibility in choosing the most efficient solution that suits the needs of the operator and its requirement. Also, the virtualization technology in NFV allows to allocate and remove dynamically resources on demand in the network. This improves dramatically the network management form traditional networks, where any improvement in the resources leads to the implementation of dedicated vendor locked-in hardware. This is very costly, time-consuming, and needs special skills from network administrators.
- **Rapid Network service deployment:** NFV allows the rapid deployment of network function and services compared to traditional networks. In traditional networks, the network functionalities are dependent on the hardware on which they are installed. Adding or deleting the functionalities will require a physical on-site intervention from network administrators, which is costly and time-consuming. NFV overcome this problem by virtualizing the functionalities. VNFs and NSs could be created and removed from the network on the fly and on-demand. Dedicated software can automate their deployment in the network and remove them when they are not utilized to free up resources. The deployment of VNFs and NSs can, therefore, be elaborated as fast as pushing a button and in a transparent way to the users.
- **Scalability and Elasticity:** NFV allow the dynamic change of resources capabilities. It can automatically augment and reduce the resources used by the VNFs in the network. For instance, if VNFs require additional CPU or storage resources, it can be requested from the VIM and allocated to the VNF. This was very costly in traditional networks where any changes in the resources require an upgrade of the physical hardware.
- **Rich Eco System:** NFV can integrate a wide variety of eco-systems and encouraging openness. It opens the virtual appliance to small players, nonprofit organizations, and academia. This encourages more innovation to bring new services and new revenue streams quickly at a much lower risk.

- **Operational Efficiency and Agility:** In NFV, the physical infrastructure could be shared by various VNFs. This centralized the tasks and eases their management. It centralizes the tasks associated with running the same management domain like for example the inventory and core business activity. This reduces operational overhead and maintenance costs.

2.3.4 NFV challenges

The challenges that the NFV paradigm faces is mainly related to the requirements that NFV is aiming to satisfy. We describe in the following some of the main requirements [77]:

- Deployment descriptor design and automation: deployment descriptors have to be defined for the network functionalities before onboarding them in the NFV platforms. They are designed by service providers manually and without any formal strategy, which is cumbersome and time consuming. Moreover, these descriptors have to be adapted for each NFV platform. In this thesis, we focus on this challenge and propose approaches that automate the design and generation of the deployment descriptors.

- Interoperability and Compatibility: One of the main goals of NFV is to provide an environment in which various solutions from different vendors could co-exist. It is therefore important to ensure that these solutions could communicate with each other. The design of standard interfaces is important to only ensure the communication between virtual appliances but also between the virtualized implementations and the legacy equipment.

- Reliability and Stability: Similar to SDN, reliability is also an important requirement for NFV. Network operators should guaranty the services that they are offering services and maintain the performance that they are promoting (e.g., voice call and video on demand).

- Security Resilience: NFV should ensure that the network functionalities are not affected by treats and architectural vulnerabilities. Also, NFV is expected to improve network resilience and availability by allowing NVFs to be re-instantiated to the coherent state after a failure.

- Manageability: NFV needs to dynamically instantiate VNFs in the right locations at the right time, allocate and scale hardware resources for them and interconnect them to achieve service chaining.

2.4 Conclusion

We presented in this chapter two of the main paradigms in software networks. The first paradigm is SDN, a paradigm inspired by cloud computing that decouples the control plane of infrastructure devices from the data plane. The devices in SDN are just forwarding devices freed from any vendor lock-down and the control logic of the devices is centralized in a single entity in the network called the controller. We showed that SDN has improved the performance form traditional networks and can lower operating costs and reduce capital expenditures.

The second paradigm in software networks showed in this chapter is network function virtualization. It is a paradigm that aims to transform the way that network operators manage networks. It is based on virtualization technologies that virtualize network functionalities and consolidates different network equipment into industry-standard high-volume servers and data center or at end-customer premises. We described the architecture of this paradigm and showed that it has many benefits like the rapid deployment of network service, the cost-effectiveness of operation, flexibility, the scalability, and the elasticity of management.

In the next chapter, we will present the related work made in the literature to position our contributions and shed light on what has already been done.

Chapter 3

State of The Art

Contents

3.1 Introduction

In this chapter, we review the works related to the context of our study. We highlight both the broader picture of the studied problem as well as a more focused picture on the use cases that are considered in this thesis. We start by examining the works related to the problem of heterogeneity between the configuration data models and

focus our attention on the use case of the cooperation between SDN controllers in a multi-controller architecture. Afterward, we analyze the works related to the problem of automating the configuration generation with a special focus on the use case of automating the design and generation of deployment descriptors in NFV.

3.2 Heterogeneity between the configuration data models

There are a plethora of data models in software networks that are used to define the configuration structure, syntax, and semantics. The diversity of data models creates a heterogeneity problem that prevents the end-to-end management of network components. We focus in this thesis on an SDN use case of a multi-controller architecture to study solutions for the heterogeneity problem. SDN controllers have to communicate to share information. It is therefore important to handle the heterogeneity between the controller's data models to enable them to interpret each other messages. We start in section 3.2.1 by examining the efforts made in SDN to enable multiple SDN controllers architecture. We aim at understanding the current situation in SDN on how the controllers cooperate and if heterogeneous controllers are considered in this architecture. Afterward, we investigate in section 3.2.3 the contributions in SDN that considered the use of semantic-based approaches. Finally, in section 3.2.3, we discuss the category of works related to handling the heterogeneity between data structures crosses multiple research areas. This gives us a better understanding of how this problem is treated in the literature and allows us to understand this problem and to position our contribution to the literature.

3.2.1 Multiple SDN controllers Architecture

SDN controllers must cooperate and share their local network views to construct a global network view. The global network view abstracts the underneath network infrastructure and offers a coherent view to the users. When the controllers in the network have heterogeneous data models, the cooperation becomes difficult in that case as the controllers could not interpret each other data. Despite some efforts to standardize the communication between the SDN controllers, to the best of our knowledge, there is no accepted standard to this date for the communication between the SDN controllers. The Internet engineering task force (IETF) proposed a draft on an SDN protocol, called SDNi, used for exchanging information between SDN controllers [114]. The protocol is proposed for the east/west interfaces of the controllers. It is responsible for coordinating behaviors between SDN controllers and exchange control and application information across multiple SDN domains. This work has not yet been finalized and implemented. It is therefore difficult to assess its benefits. The authors in [67], proposed an east/west interface, called EWBridge that allows heterogeneous SDN controllers to exchange their views in the network. EWBridge

abstracts the views of the controller in a "key-value" database. The controllers can access the global network view in a centralized fashion using a publish/subscribe mechanism. For this approach to work, all the controllers have to implement the EWBridge interface, which is not obvious.

The authors in [119] proposed Zebra, a framework that enables the communication between heterogeneous SDN domains. The framework has two modules: Heterogeneous Controller Management (HCM) module and Domain Relationships Management (DRM) module. HCM collects network information from a group of controllers with no interconnection and generates a domain-wide network view, while DRM collects network information from other domains to generate a global-wide network view. The framework adds a layer between the application and SDN controller layer to make the heterogeneous controller cooperate. Multiple heterogeneous controllers could be supported by the framework using a server-end component that collects information from each controller. It is unclear however in this work how the information is extracted from the controllers and integrated into a global view. The framework was extended in [92] and a system is built on top of the control plane, which can encapsulate the whole plane and provide a uniform interface for users to control it independently of the controller's type.

The authors in [49] proposed an east/west interface, called INT manager, to ensure the communication between SDN controllers. The INT manager interprets the network topology - data plane routers and their links managed of the SDN controllers into a virtual router. This virtual router presents all networks in the controller domain and networks reachable by it, with various path metrics for all of them. It is expected in this study that the controllers use the same data model of the interface INT manager.

The authors in [15] also proposed an East-West interface for multi-controller architecture in SDN, called Communication Interface for Distributed Control plane (CIDC). It is used for exchanging messages between controllers and for customizing the behavior of each controller in the network. The CIDC interface is composed of four modules, Consumer, Producer, Data Updater, and Data Collector that are used to communicate with the core elements of the controller. In this design, each controller plays the role of a Consumer for external events and a Producer for local events. Similarly to EWBridge, for this interface to work, the controllers have to be aware of the interface logic and adapt their communication to it.

The authors in [71] proposed an application interface for multi-controller architecture in SDN, called Data Distribution Services (DDS). The interface provides a global data store in which the SDN controllers publishes and subscribes to respectively write and read data from the datastore. It is unclear however how the datastore is modeled and if heterogeneous controllers can use it for communication.

Other contributors proposed SDN platforms to allow multiple SDN controllers to be deployed in the network. ONIX [62] and HyperFlow [112] are among the first physically distributed platforms of SDN controllers. ONIX uses a central network

Table 3.1: Comparison between multiple SDN controllers solutions

Multi-Controller SDN architectures	Approach	Heterogenous controllers	Automatic integration
SDNi [114]	East/West Interface	Yes	No
EWBridge [67]	East/West Interface	Yes	No
Zebra [119]	East/West Interface	Yes	No
INT manager [49]	East/West Interface	Yes	No
CIDC [15]	East/West Interface	Yes	No
DDS [71]	Application Interface	Yes	No
ONIX [62]	SDN Platform	No	No
HyperFlow [112]	SDN Platform	No	No
DISCO [90]	SDN Platform	No	No
Kandoo [47]	SDN Platform	No	No
Elasticon [31]	SDN Platform	No	No
ODL [111]	SDN Platform	No	No
ONOS [16]	SDN Platform	No	No

information base (NIB) that maintains a global view of the network, network applications can read and write into this base to manage the network elements. Hyperflow is an event-based architecture that supports OpenFlow, it implements NOX [43] controllers. DISCO [90], is a platform that can cope with wide area networks and overlay networks, it can be divided into two parts, an intra-domain part, that is responsible for monitoring the network and dynamically managing the network to maintain a satisfiable performance, it uses a central database for storing the controller's information, and an inter-domain part, that provides communication between domains. Kandoo [47] is a hierarchical platform for distributed controllers where root controllers take the responsibility of managing the network. Elasticon [31], is an elastic platform architecture, where a controller is added or removed dynamically depending on the network traffic load. OpenDayLight (ODL) [111] is an open-source platform, where a set of controllers collaborate as a cluster to achieve some performance criteria such as availability and scalability. ONOS [16], is also an open-source platform suited for high availability and scalability applications. It uses a central graph database to store network information.

We summarize in Table 3.1 the efforts made to enable multiple SDN controllers architecture. Unfortunately, none of the works proposed an approach to handle automatically the integration of heterogenous SDN controllers. The contributions that proposed interfaces to enable the communication between SDN controllers could allow different SDN controllers from different vendors to cooperate. The only matter with these kinds of approaches is that the controllers have to employ the predefined data model of the interface in order to use its functionalities and send/receive information from/to other controllers. The other contributions consist of SDN platforms that enable homogeneous SDN controllers to cooperate with each other. To the best of our knowledge, enabling the integration of heterogenous SDN controllers has not yet been considered. Given the importance of that matter for auto-configuration, we address this challenge in this thesis.

There are only a few contributions that considered semantic approaches in SDN.

Most of these contributions aim at providing solutions for data integration in SDN when the data is issued from diverse sources. Nevertheless, these works have shown the efficiency of semantic approaches, which is a good motivator for our choice to follow their lead into considering semantic approaches as a potential solution for handling the problem of heterogeneity in SDN.

3.2.2 Semantic approaches in SDN

In our work, we investigate a solution inspired by semantic data integration. Semantic data integration is an active research field that aims to harmonize heterogeneous data [24]. There are several contributions based on semantic data integration that have been investigated in SDN. The authors in [75] proposed a semantic-based approach for complementing the capabilities of OF-CONFIG and NETCONF for the remote configuration of network devices in SDN. They defined an Ontology-Based Information Extraction (OBIE) System from the CLI of network devices and proposed a learning algorithm that enables automated and highly precise interpretation of CLI configuration capabilities in heterogeneous (multi-vendor) network scenarios. They aim to exploit the knowledge already available in CLIs, to automatically reconcile the semantic and syntactic differences between heterogeneous configuration environments.

The authors in [27] studied wireless SDN, they proposed a cognitive radio ontology that abstracts the description of the communication scenario, RF devices, and policies. The proposed ontology provides awareness and supports reasoning by the controller and applies to any RF device. The authors [96] proposed a novel architecture of software-defined semantic routing networks that can provide semantic routing networks with high efficiency, fine-grained control, as well as the flexibility to support future advanced network technologies. The proposed solution normalizes various data resources into a high-dimensional semantic space and correlates the data semantics with their distance information, which can realize the intelligent discovery of internet data.

There are only a few contributions that considered semantic approaches in SDN. Most of these contributions aim at providing solutions for data integration in SDN when the data is issued from diverse sources. Nevertheless, these works have shown the efficiency of semantic approaches, which constitutes a good reason for our choice to follow their lead into considering semantic approaches as a potential solution for handling the problem of heterogeneity in SDN.

3.2.3 Handling the heterogeneity between information structures in the literature

We analyze in this section some contributions related to analyzing the heterogeneity between information structures. We group these contributions into six categories based on [103]: Similarity measures, Information fusion, information clustering, information classification, link prediction, and information ranking. Table 3.2, summarizes

Table 3.2: Categories of approaches that handled the heterogeneity between information structures

Category	Description
Similarity measures	Approaches that measure the similarity between the elements of heterogenous information structures. [103], [101], [110], [48], [50]
Information fusion	Approaches that merge and fuse heterogeneous information structures. [32], [88], [87], [36], [84], [81], [35], [37]
Information clustering	Approaches that group similar elements from heterogeneous information structures. [117], [108], [18], [116], [73]
Information classification	Approaches aim at learning a models using machine learning and other techniques that predict the label of the information elements. [56], [72], [53]
Link prediction	Approaches that analyze heterogeneous information structures to estimate the likelihood of the existence of a connection between two information elements. [91], [20], [107], [120], [25], [109]
Information ranking	Approaches that help to identify the relevant and important elements in heterogeneous information structures. [109], [86], [60], [113]

these categories.

3.2.3.1 Similarity measures

Measuring the similarity between the elements of different information structures is the basic operation in analyzing the heterogeneity. There are two types of similarity approaches to analyze an information structure. Feature-based approaches and link-based approaches [103].

Feature-based similarity approaches measure the similarity between elements of information structures relative to their characteristics. The characteristics are in most cases the attribute values of the elements. For example, a compute element in a configuration file could be characterized by the number of CPUs, the type of CPUs, the frequency of the CPUs, etc. The famous feature-based similarity measures are the cosine similarity, the Jaccard coefficient, and the Euclidean distance.

The link-based similarity approaches measure the similarity by taking into consideration the relationship between the elements in the information structure and also the meta path connecting these elements. A meta path contains different connec-

tions with different semantics. This situation leads to different similarities. In that regard, [101] introduced a framework that derives various similarity semantics based on the relation between different elements. The proposed framework includes a similarity measure called PathSim [110] that can be used to find similar peer elements. This measure can be useful in many scenarios compared with random-walk based similarity measures. This work was later extended in [48] and [50], where the authors augmented the framework with additional information like transitive similarity, temporal dynamics, and supportive attributes.

Another meta path-based similarity, called HeteSim, was proposed in [102]. This measure can be used to evaluate the relevance between heterogeneous elements in an information structure under an arbitrary path (connection path between two elements through following a sequence of element types). The measure was used in [65] for the identification of interactions between drugs and targets in drug research. They used this measure in an optimization algorithm called LSH-HeteSim. The algorithm is employed to mine the drug-target interaction in heterogeneous biological networks, where the relationship between drugs and targets is various. [76] proposed a similarity measure, called AvgSim, as an effort to reduce the high computation and memory demand in which other similarity suffered from. AvgSim can measure the similarity of the same or different-typed element pairs in a uniform framework.

3.2.3.2 Information fusion

Merging and fusing information from heterogeneous sources is useful for understanding scattered data and obtaining common knowledge about a domain. Fusion is the process of aligning and matching common elements from different information structures. This process plays a key role in many disciplines, such as data warehousing, e-business, or even biochemical applications. Our contribution to handling the heterogeneity between configuration data models matches this mining task.

In our work, we use ontologies as an information structure to represent the knowledge in the SDN domain. Ontologies are considered as an essential component for sharing knowledge and defining the concepts associated with a domain, in our case it is SDN. In our work, we aim to establish semantic correspondences between the concerned heterogeneous ontologies to construct a coherent federated view. The literature counts works in the same range from various disciplines, such as information retrieval, databases, learning, knowledge engineering, and natural language processing.

The authors in [32] proposed GLUE, a system that employs machine learning techniques to find mappings between two ontologies. For each concept in one ontology, GLUE finds the most similar concept in the other ontology. It uses multiple learning strategies, each of which exploits a different type of information, either in the data instances or in the taxonomic structure of the ontologies. In [88], the authors tackled the problem of the absence of links at the concept level and proposed an approach that

searches for alignments between concepts from ontologies. In their work. they hypothesized new composite concepts defined as disjunctions of conjunctions of (RDF) types and value restrictions that are used to generate alignments between these composite concepts. In [87], the authors proposed an extensional approach to generate alignments between ontologies. It is an algorithm that produces equivalence and subsumption relationships between classes from heterogeneous ontologies by exploring the space of hypotheses supported by the existing equivalence statements.

In [36] the authors proposed a matching system that aims to discover automatically the correspondence links between two intrinsically heterogeneous ontologies, through different calculations of similarity between their concepts. In [84], the authors adopted two alignment methods based on the ontology structure. The first, called Method of Similarity of Inheritance (MSI), applies the initial similarity method based on the concepts and integrates the inheritance relation (father/son) into the calculation of similarity. The second, called Method of Sibling Similarity (MSS), involves sibling relationships to enrich the similarity score between two concepts. In another work [81], the authors defined a method for the combination of three different types of similarity measures: lexical techniques, structure-based techniques, and semantic-based techniques. Also, authors in [35] defined a measure that focuses on matching ontologies based on the structure and aggregation of similarities calculated by different matches. In [37], the authors presented a measure of similarity based on the structure for ontology matching by the terminological, structural, and semantic levels.

LikeLike illustrated in this section, there are plenty of proposed approaches for merging different information structures and aligning their elements. Though, these approaches differ from each other mainly on their performance. The authors in [64] described multiple performance factors that merging approaches face in the context of sensor networks. We describe some of the performance factors in Table 3.3 and assign the works that measured those factors in their evaluation. Each approach is suited therefore for a particular task where a corresponding performance factors is important to be satisfied. In our case, we want at this stage to construct a global network view in SDN by merging different configuration elements. Our main focus is to enhance the matching accuracy. We take inspiration therefore from the approaches that also stratified this performance factor.

3.2.3.3 information clustering

Clustering approaches aim at grouping similar elements from different information structures together based on diverse parameters like their type, their connection, or their syntactic representation. The aim of clustering elements together can provide useful insights into many applications. For example, in topic modeling, a field that aims to automatically discover the main themes that pervade a large and unstructured collection of documents, the authors in [117] proposed a unified Topic Model cluTM

Table 3.3: Performance factors for information fusion approaches

Performance factor	Description	Contributions
Alignment accuracy	The correctness of alignment between elements.	[32], [36], [35], [37]
Confidence	Effective integration of elments from the same context (same information) increases improves the performance.	[88], [84]
Reliability	It quality of the results. When data is redundant it improves the reliability of the system and enhance the understanding of the data.	[88] [81]
Complexity	When there is too much data to be alligned, the complexity of fusion increases	[87]
Uncertainty	The uncertainty increases when bad data is fusing with good data based on estimation.	[81]
Unbelief	When the data is resulted form the merging of multi-modal data, there is a risque that results losses the semantic of the orginal data.	

by incorporating both the document content and various links in the text related to heterogeneous structures. cluTM combines the textual documents and the link structures by the proposed joint matrix factorization on both the text matrix and link matrices. Their approach derives a common latent semantic space shared by multi-typed objects. They proved that this approach enhanced semantic information.

Another application could be to identify the label or the type of elements from different information structures when the value of the attributes is incomplete or when the elements carry only partial attributes or even no attributes. The clustering in this case is challenging as the connection of different types may carry different kinds of semantic meanings, and it is a difficult task to determine their nature. The authors in [108] proposed a model-based clustering algorithm. It is a probabilistic model that clusters the objects of different types into a commonly hidden space, by using a user-specified set of attributes, as well as the connection from different relations. In [18], the authors proposed a density-based clustering model TCSC for the detection of clusters in heterogeneous information structures in which the elements are are densely connected as well as in the attribute space. TCSC enables the detection of clusters that show similarity only in a subset of the attributes. This could be more effective in the presence of a large number of attributes.

In [116], the authors provide an example of using world knowledge for domain-dependent document clustering. They indicate ways to specify the knowledge to domains by resolving the ambiguity of the entities and their types and represent the data with knowledge as a heterogeneous information structure. They proposed a clustering algorithm that can cluster multiple types and incorporate the sub-type information as constraints. Also in [73], the authors proposed a semi-supervised learning approach, called SemiRPClus, that clusters entities based their relation-path.

3.2.3.4 Information classification

Classification tasks aim at learning models using machine learning and other techniques to predict the label the information elements. As the information infrastructure contains elements with different types, the classification has to be of multiple types of elements simultaneously. The authors in [56], proposed GetMine, an algorithm that solves a transductive classification problem on heterogeneous information infrastructures that share a common topic and have only some elements in the structure that are labeled. The algorithm's goal is to predict labels for all types of the remaining elements. The authors in [72] proposed PathMine, an algorithm that clusters small labeled data on heterogeneous information infrastructures through a novel meta path selection model, and the authors in [53] proposed a method to label elements of different types by computing a latent representation of nodes in a space where two connected elements tend to have close latent representations.

3.2.3.5 Link prediction

Link prediction is an analysis made on heterogeneous information structures to estimate the likelihood of the existence of a connection between two elements. This task also received a lot of attention in the literature. The authors in [91] introduced a structured logistic regression model that can make use of relational features to predict the existence of connections. In [20], the authors considered the problem of collective prediction of multiple types of links in heterogeneous information structures. They introduced a relatedness measure, called RM, between different types of objects that they used to compute the existence probability of a connection between the elements. Other works in [107], [120], [25], [20] and [109], have considered to solve the link prediction problem in two steps. The first step consists of extracting meta path-based feature vectors and the second step is to train a regression or classification model to compute the existence probability of connections between elements.

3.2.3.6 Ranking

Ranking helps to identify the relevant and important elements in heterogeneous information structures. These approaches use raking functions to evaluate the elements and determine their importance. Many contributions were conducted for this task like PageRank [109] that evaluates the importance of elements through a random walk process, and HITS [86] that ranks elements using the authority and hub scores.

Ranking heterogeneous elements were heavily investigated for social networks, like the work in [60] and [113]. The authors in [60] propose SocialRank which uses social hints for image search and ranking in social networks. In [113], the authors proposed an approach that ranks the tweets. Their approach integrates both formal genres and inferred social networks with tweet networks to rank tweets. Ranking approaches are also investigated in QA systems to identify high-quality elements such as questions, answers, and users. The author in [51] devised an unsupervised heterogeneous information structure to co-rank multiple elements in QA sites. In [121], the authors proposed a framework for the heterogeneous cross-domain ranking problem. The framework discovers a latent space for two domains and minimizes two weighted ranking functions simultaneously in the latent space.

It is clear from the review of the state of the art regarding the categories of approaches that handled the heterogeneity between information structures that our task of handling the heterogeneity between SDN controllers to construct a global network view is under the category of information fusion. We, therefore, focus our attention on the related approaches and take inspiration from these works.

3.3 Automating the configuration generation

We consider the problem of automating the generation of deployment descriptors in NFV as a step toward automating the configuration in software networks. We review

first the existing works in NFV that considered also the problem of automating the generation of descriptors in section3.3.1. Afterward, in section3.3.2 we briefly discuss the efforts made in software engineering to automate the configuration generation.

3.3.1 Deployment descriptors generation in NFV

In NFV, service providers have to define manually the deployment descriptors in order to onboard the VNFs and NSs. This is certainly a cumbersome and error-prone task. Recent surveys on the current challenges in NFV orchestration indicated that the generation of deployment descriptors is a problem that needs to be tackled in order to enable full automation of network configuration and allow more agility in the orchestration [58] [104].

Unfortunately, the problem of automating the generation and design of deployment descriptors in NFV has received only a few attention in the literature. The authors in [74] proposed a tool, called VNF Onboarding Automation Tool (VOAT), and a methodology to aid the VNF designer to rapidly design and onboard new services and applications. VOAT takes as input the deployment descriptors files and translate them to metadata format and generates an appropriate service catalog for future deployment of the VNF packages in the production or test environments. The metadata inputs from the VNF vendor are provided in JSON files. These inputs are then translated to a data model based on the ETSI VNFD. A VNFD Parser then parses the VNFD and generates a HOT template associated with the VNF or network service package. The VOAT backend engine performs the onboarding procedure and places the artifacts in the appropriate deployment environment and catalog. This approach assists the creation of an appropriate deployment descriptor that follows the required data model, hot templates in their case study. However, this approach relies heavily on human intervention for the definition of the required artifacts needed for the deployment and does not automate the generation or recommendation of information related to the deployment.

The authors in [97] proposed a framework based on machine learning to identify the appropriate types of resources to be allocated to a VNF at deployment time. Their work helps identify the performance characteristics and affinity between physical resources in order to maximize performance and to optimize the number of allocated resources. The framework automatically generates different deployment configurations for given VNF characteristics in the form of heat templates. It has two distinct functions: the first one is the characterization of VNF workload performance using different resource allocations in a quantitative manner; the second one is modeling the relationships between VNF performance and resource allocations. The templates represent possible configurations for selected parameters (e.g. vNIC configuration, number of virtual CPUs, amount of RAM, etc.) within a given range of values. The framework interacts with an OpenStack cloud environment and generates different configurations in the form of Heat templates, which are deployment scripts based

Table 3.4: Illustration of the contributions made for automating the generation of deployment descriptors

Description automation approach	Proposed solution
VOAT [74]	Generation and design of deployment descriptors
Ressource allocation Framwork [97]	Automatic generation of deployment resources configuration
CPU consumption model [57]	Prediction of the CPU perofrmance
VNFD generator web application [80]	Generation and design of deployment descriptors

on the Heat Orchestration Template format. This work focus only on automating the configuration of the physical resources, which is an important step toward auto-configuration.

The authors in [57] investigated an approach based on machine learning to estimate the VNFs need in terms of resource requirements from descriptors. The aim of this work is to model the VNF requirement. That is because the behavior of VNFs is dynamic, complex, and depends on different factors, which makes developing accurate models a challenging task. They proposed a model that takes as input incoming traffic entering the VNF and output the amount of CPU required by the VNF to process that traffic. Once the prediction of the model is performed, the CPU's estimated value can be used by resource allocation algorithms to automatically adapt the amount of provisioned resources.

The authors in [80] proposed a VNFD generator web application to automatically create a VNFD template that the operators and VNF vendors can quickly validate and deploy. The VNFD generation process takes as input the VNFD properties like the name of the VNFD; the number of deployment units (VDUs). The attributes of the VNFD are then selected and their value determined. Overall, this approach helps to automate the creation of the file and need human intervention to determine all the component expected to be founded in the VNFD.

Table3.4 summarizes the efforts made for automating the generation and design of deployment descriptors in NFV.

Following the ETSI NFV recommendations [38], the VNFD descriptors define the VNF properties, such as resources needed (amount and type of Virtual Compute, Storage, Networking), software metadata (External and internal Connection Points, Virtual Links,etc.), lifecycle management behavior (scaling, instantiation,etc.), lifecycle management operations, and their configuration. The absence of a common data model to describe the deployment descriptors have led different template models to emerge. The most used modeling languages for that matter are YANG and TOSCA. In TOSCA, there are two profiles that are standardized by OASIS: the simple profile in YAML v1.0 for managing the life cycle of cloud applications and services, and the Simple Profile for NFV [2] that is based on ETSI NFV.

From the analysis of the existing works related to automating the deployment descriptors generation, we can notice that the proposed solutions offer only static automation, meaning that service provider has to define the template of the descriptors,

the structure of the files, the input of the model, the data set, etc., for the solution to predict configuration values or an estimation. In our work, we aim at minimizing service providers intervention by learning directly from deployment descriptors. We investigate in this thesis two approaches. The first one is based on deep neural networks that learn in an unsupervised manner from deployment descriptors models that could recommend and complete automatically configurations without human intervention. The second approach is model-driven. It aims at capturing the configuration variabilities into a single configurable model that could assist the automatic generation of deployment description.

The deep neural network techniques are powerful to generalize the prediction from a set of limited examples and to eliminate the errors caused by data noises. To the best of our knowledge, our contribution to learning from deployment descriptors using deep neural networks has not been yet investigated in the context of automating the deployment descriptor generation. Deep neural have been already showed to be successful in many tasks such as processing natural language and computer vision [59], [42]. In the next section, we show examples of works that considered deep neural networks in automating the configuration generation in software engineering.

The configurable model of our second contribution is inspired by the configurable process models in Business Process Management (BPM) [21]. Indeed, designing high-quality process models, similarly to deployment descriptors, is time-consuming, error-prone, and costly. The tasks in BPM need to be ordered as well as their execution plan, i.e. whether the tasks are executed in parallel or are alternatives to each. Configurable process models significantly reduce process modeling costs by allowing the reuse of existing process models and enabling their adaptation for different needs. [105]. To the best of our knowledge, our approach to learning from deployment descriptors using a configurable deployment descriptor model has not been yet investigated in the context of NFV.

Our proposed configurable model is closely related to the work introduced by the work in [93, 99]. The authors proposed a Configurable Event-driven Process Chain (C-EPC) to improve the configurability of Enterprise systems and reference models such as SAP R/3 reference model. Basically, the EPC notation consists of three main control-flow elements: *event*, *function* and *gateway*. An event can be seen as a pre- and/or post-condition that triggers a function. A function describes the kind of work which must be done. Three types of connectors, OR, exclusive OR (XOR), and AND are used to model the splits and joins. C-EPC adds two constructs to the EPC language: *configurable functions* and *configurable gateways*. A configurable function has two configuration alternatives: *ON* and *OFF*. A configuration *ON* means that the function is included in the process variant while a configuration *OFF* means that it is excluded. A configurable connector has configuration alternatives that are of equal or less restrictive behavior. For example, a configurable OR has the configuration alternatives: OR (no restriction is applied), XOR, and AND with possibly restricted outgoing/incoming branches. We propose a graphical representation of a

Table 3.5: Categories of contributions related to the configuration automation in software engineering

Categories of auotmating the configuraiton in software engineering	Description
Configuration prediction	Approaches used to identify and predict measured and observed configuration attributes. [26], [44], [3], [55]
Interpretability of configurable systems	Approaches used to model the impact of the configuration choices on the system quality. [33], [54], [61]
Configuration optimization	Approaches used to find configuration features that suit the expected performance requirements. [10], [14], [30], [7]
Dynamic Configuration	Approaches that adapt the configuration of the software with unexpected environmental and contextual changes. [12], [13], [98]
Configuration constraints mining	Approaches that identify the configuration constraints that ensure the coherency of the system. [13], [22], [69], [95]

VNFD model that consists of component instances, relations, and gateways. Our proposed composition, allocation, and connection gateways are different than those in the BPM domain (i.e. OR, XOR and AND gateways). The behavior of these operators is similar to that of BPM gateways. However, their configuration is different.

3.3.2 Automatic configuration in software engineering

In this section, we discuss a broader picture of the contributions related to automating the configuration of systems. More particularity, we emphasis on software engineering as software configuration automation is a very active field of study and also because NFV can be considered as a special software dedicated to managing networks. As illustrated in table 3.5, auto-configuration contributions can be grouped into five categories [89]: configuration prediction, interpretability of configurable systems, configuration optimization, dynamic Configuration, and configuration constraints mining.

3.3.2.1 Configuration prediction

Configuration prediction aims to identify and predict measured and observed configuration attributes. The measured attributes are quantitative values like the execution time, the latency, delay, etc., while the observed attributes are qualitative like the er-

rors, the quality of service, the type of resources, etc. Authors in [26] investigated an approach that anticipates the performance of an eventual application solution before been built. They proposed an empirical approach that determines the performance characteristics of component-based applications by benchmarking and profiling. The authors in [44] studied the correlation between feature selections of configuration and performance in configurable software systems. They proposed a variability-aware approach to performance prediction via statistical learning. The authors in [3] also investigated the same problem and proposed a data-efficient learning approach, called DECART, that combines several techniques of machine learning and statistics for performance prediction.

The authors [55] considered a strategy that transfers knowledge across environments to predict the performance using transfer-learning strategies. Their approach, called Learning to Sample, takes into consideration empirical insights about common relationships regarding influential options, their interactions, and their performance distributions. The approach selects the best samples in the target environment based on information from the source environment.

3.3.2.2 Interpretability of configurable systems

The interpretability of configurable systems aims to provide insights on how the configuration choices could impact the system quality. They focus on learning models that could accurately explain the performance behavior of a configurable system as a whole. In [34] the authors proposed a method that learns human-readable models that capture non-deterministic impacts explicitly of actions on the system. This model could be to revise the system model in offline. The authors in [33] presented an approach to analyzing the correlation between the selection of features embedding uncertain parameters and system performance. In [54], the authors investigated the benefits of using transfer learning into constructing performance models. They conducted an empirical study on four software systems, varying software configurations and environmental conditions, such as hardware, workload, and software versions, to identify the key knowledge pieces that can be exploited for transfer learning. In [61], the authors studied the tradeoffs between prediction error and model size and between prediction error and computation time. They identified several patterns across subject systems, such as dominant configuration options and data pipelines, that explain the influences of highly influential configuration choices and interactions.

3.3.2.3 Configuration optimization

Configuration optimization contains approaches that aim to find configuration features that best suit the expected requirements. In [10], the authors proposed, CherryPick, a system that leverages Bayesian Optimization to build performance models for distinguishing the best or close-to-the-best configuration from the set of configuration. In [14], the authors proposed an automatic configuration system that can

optimize producer-side throughput on Distributed message system. In [30], the authors proposed a learning algorithm to tackle the input sensitivity problem in program performance autotuning. The algorithm is a clustering method that automatically refines input grouping, feature selection, and classifier construction. In [11], the authors, the authors proposed a machine learning approach for predicting the performance of each configuration of optimization algorithms. Their contribution finds the most suitable configuration on a per-instance analysis based on a supervised machine learning model.

3.3.2.4 Dynamic Configuration

Dynamic configuration is related to systems that need run-time adaptations of software in order to react with unexpected environmental and contextual changes. In [12], the authors proposed to utilized the Markov decision process theory to present a reinforcement learning strategy that discovers the complex relationship between the system workload and the corresponding optimal configuration. In [13], the authors investigated an approach based on machine learning that finds Pareto-optimal configurations without needing to explore every configuration. This approach restricts the search space to such configurations to make the planning tractable. In [98], the authors proposed a transfer learning approach for Improving model predictions in highly configurable software.

3.3.2.5 Configuration constraints mining

Mining constraints are approaches that identify the configuration constraints that ensure the coherency of the system. For example, there are some configuration choices that are mutually exclusive. In this category, the contributions analyze and mine the constraints related to the configuration choices. Variability models are proposed to precisely define the space of valid configurations. Our second contribution to auto-configuration could be considered in this category. This is because our contribution can be used to mine the configuration constraints and help service providers to choose the best options. In [13], the authors described an approach that automatically mines constraints for runtime monitoring from event logs recorded in software-intensive systems of systems. In [22], the authors proposed a search-based technique that is able to repair a model composed of a set of constraints among the various software system's parameters. In [69], the authors proposed an approach that combines multi-objective search with machine learning to mine rules to automate the configuration in Product Line Engineering. In [95], the authors proposed an approach based on the combination of a kernel density estimation and a genetic algorithm to rescale a given configuration attribute-value profile to a given variability model.

After the review of the categories of existing works related to the configuration automation in software engineering, we can position our contributions vis-a-vis to the literature. Our deep neural network approaches can fit the categories of configuration

prediction and dynamic configuration. That is because our approach can not only predict configuration values in the deployment descriptors but also can be used to generate new configurations to adapt to changing network situations. For example, an NFV orchestrator can use our approach to recommend an existing deployment descriptor in its catalog in cases when an onboarded deployment descriptor is damaged, or when the resources described in the deployment descriptor cannot be satisfied by the orchestrator.

Our model-driven approach can be categorized in configuration prediction, configuration optimization, and configuration constraints mining. Indeed, the configurable model can be used to design the deployment descriptors, to select the best configuration values based on the catalog of deployment descriptors, and also can be used to mine the dependencies between the configuration choices to derive constraints that should be respected by service providers.

3.4 Conclusion

In this chapter, we explored relevant existing works to our objectives. We considered two challenges related to the auto-configuration: (i) handling automatically the heterogeneity between configurations and (ii) Generating automatically configurations.

Regarding the first challenge, we discussed the efforts made in SDN to overcome the heterogeneity problem in multi-controller architecture. We showed that there is a lack of approaches for the automatic integration of controllers. The efforts already made are either including only homogenous controllers in the architecture or using predefined APIs between controllers. Afterward, we described the contributions related to analyzing the heterogeneity between information structures. We grouped these contributions into six categories based on [103]: Similarity Measure, Information fusion, Clustering, Ranking, Link prediction, and Classification. Each category focuses on a particular mining task. We indicated that the challenge that we are tackling in this thesis is part of the information fusion categories as it aims at grouping multiple SDN controllers' views into a single coherent one. Regarding the second challenge, we focused on the efforts made in the context NFV to automate the generation of deployment descriptors. We showed that are only a few works that considered automating the generation of deployment descriptors and that these works only static automation, meaning that the service provider has to define the template of the descriptors, the structure of the files, the input of the model, the data set, etc., for the solution to predict configuration values or an estimation. Additionally, we reviewed the existing works that were made toward auto configuration by grouping them into five categories: Prediction, Interpretability, Optimization, Mining Constraints, and Dynamic Configuration.

We start presenting in detail our approaches in the next chapters. In chapter 4, we elaborate on our approach to handling the heterogeneity in SDN. In chapter 5, we introduce our deep learning techniques to recommend and complete deployment

descriptors in NFV, and in chapter 6, we present our model-driven approach to assist the design of deployment descriptors in NFV.

Chapter 4

Handling the heterogeneity between configuration data models

Contents

4.1 Introduction

Software networks are built and operated on various solutions from different sources. These solutions could offer different functionalities that satisfy different performance goals or compete with each other on the market on supplementary aspects like the service price or the usefulness of the service. Federating these heterogeneous solutions together is one of the main challenges for end-to-end network management and consequently for also an auto-configuration solution in the network.

The SDN paradigm addresses part of such challenges by separating the data plane (hardware) from the control plane (network functionalities). The control plane manages different network devices independently from any vendor. This way the network is not locked-in to a specific vendor and can hide the heterogeneity from network users. SDN uniforms and unifies network management. The underlying devices can be easily changed without any exceeding time or integration effort.

The problem of heterogeneity can prevent the management entity from collaborating with network components if the latter are not sharing a common understanding of how the information is defined. The information exchanged and the configuration requested could not be understood by both parties, they are not sharing the same syntactic and semantics of the data.

Currently, there are a plethora of data model languages used in software networks. Certainly, the common ones are TOSCA (Topology and Orchestration Specification for Cloud Application) and YANG (Yet Another Next Generation). TOSCA is a standard proposed by the Organization for the Advancement of Structured Information Standards (OASIS). it is used for defining portable deployment and automated management of services on a wide variety of infrastructure platforms. It describes services, platforms, infrastructure, and data components, along with their relationships, requirements, capabilities, and operational policies. YANG on the other hand is defined by the IETF. It is used to model the operations and content layers of NETCONF and also in the configuration of devices and services in NFV and SDN.

A potential solution for this heterogeneity problem is to find a consensus between network vendors and organizations on a common representation of network domain knowledge. This solution may be improbable in the short term. Network vendors diverge on their vision of software networks. This resulted in inventing different architectures, platforms, and software with different characteristics that make it very difficult for them to converge on the same understanding. Moreover, also a solution that manually does the translation between heterogeneous network components is not feasible in software networks. It is hard to keep up the translation updated for each heterogeneous solution and with the constant evolution and upgrade of this type of network.

To overcome the heterogeneity problem in software networks, we investigate in this chapter a semantic-based approach that searches for elements with similar meaning or semantics between heterogenous configuration files and maps them together. The

outcome of this approach is a common representation of the configuration files.

We focus our attention more particularly on SDN. Operating heterogeneous SDN controllers is a challenging task. The problem is the integration of the controller's local views into a global coherent view that encompasses the entire network. Each SDN controller may expose different functionalities using different models to describe its data. The difference could be in the structure of the data, the syntactical specifications of the data, or the semantics of the data. This makes a controller's local view uninterpretable by the others. To address this problem, we propose a semantic-based approach that handles the incompatibility of the information that is exchanged between controllers. The approach models the controller's network views as ontologies. The common elements of controllers ontologies are then matched together via semantic mapping techniques, which federates the controller's knowledge and exposes a global view of the network. The approach is a framework that could be implemented as a module in the SDN controller. The contribution of this chapter could be summarized as follows:

- We consider two scenarios: 1) the controllers share their local views to a central entity that collect them and integrates them into a single view. 2) the controllers share their local views, each controller construct locally the global view.
- We use a global ontology in the centralized global view that defines the SDN elements and their relations.
- We propose an extraction method from network data that are described in JSON format to an ontology.
- We suggest two mapping techniques. One for the centralized global view, to map the controller's local ontologies to the global ontology and one for the distributed global view that map controller's ontologies together.

This chapter is organized as follows: we start by discussing the use case of heterogeneous multiple SDN controllers in section 4.2. We then present our semantic approach for constructing the global network view in this use case in section 4.3. we evaluate this approach and analyze the results in section 4.4, and we conclude the chapter in section4.5.

4.2 Use case: Multi-controllers SDN architecture

The control plane of SDN is centralized when a single controller is operated in the network. Although this architecture is sufficient for many medium-sized networks, the performance requirements could not be met in large networks. Single controller architecture may not be adequate for scenarios where multiple network devices communicate in large distances. It is important in that case to ensure the availability of the network and decrease the latency of communication.

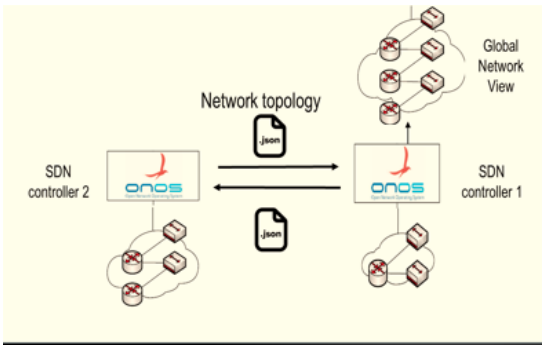


Figure 4.1: Multi-Controllers SDN architecture

Operating multi-SDN controllers in the networks can enhance the performance in terms of efficiency, scalability, and availability of the networks. The communication latency can be reduced when multiple controllers are distributed along with the network. Load balancers are used to make sure that the controllers are not overloaded and that the load is dispatched efficiently. Controllers can be provisioned dynamically to achieve higher performance and meet the needs of the networks. Moreover, multiple controllers can also be used as a backup to ensures the availability of the network. The redundancy of controllers prevents a single point of failure and improve the security of the control plane.

Concretely, an SDN multi-controller architecture is a set of controllers operating together to achieve some level of performance. Figure 4.1 shows an example of this architecture. There are different ways of designing the multi SDN controller architecture depending on various aspects that we are going to describe in the following.

4.2.0.1 Distributed vs Centralized control plane

A physically distributed SDN controllers could be used either as a single entity that controls the network or as the cooperation of multiple entities (controllers).

The management and control of multi-controllers could be logically centralized to preserve the original idea of the concept of SDN and also take advantage of multi controllers. In this architecture, there is a single layer above the controller that is responsible for distributing the load among the controllers. The layer gives the impression that there only a controller in the network. It abstracts the multi controller's architecture to the users.

On the other hand, in a distributed plane of multi-controller architectures, all the controllers have the same responsibilities and may divide the load among them. The controllers in that case need to be always aware of every change in the network and need to constantly cooperate to share the same information. instantly, thanks to network synchronization. In a logically distributed architecture, the controllers are physically and logically distributed. Additionally, every controller has just a view

of the domain it is responsible for, and it can take decisions for it, unlike a logically centralized design, where each controller makes a decision based on the global network view.

4.2.0.2 Flat Architecture vs Hierarchical Architecture

In a flat multi-controller architecture, controllers are positioned horizontally on one single level. There is only a single layer in the controller plane and each controller has the same responsibilities and has only a partial view of the network.

In other words, the entire network is divided into several domains, each domain is managed by a controller. The cooperation between the controllers is possible by west-east interfaces. They could communicate relevant information to contract the global network view.

In a hierarchical multi-controller architecture, SDN controllers are positioned vertically on multiple levels. The control plane in this case is divided into several layers, where each layer has a specific role. The controllers have in each layer have different responsibilities, and they can take decisions based on a partial view of the network. The cooperation between the controllers could be only with the controller from different layers. For example, a root controller could be used to controller and manage controllers from a different domain and maintain a global network view. The root controller can communicate with the domain controllers, while the domain controllers do not contact each other.

4.2.0.3 Dynamic Architecture versus Static Architecture

In a dynamic multi-controller architecture, the links and the positions between the controllers, as well as the switches, are changeable, which makes the network flexible.

In a static architecture, the links and the positions between the controllers and also the switches are unchangeable, which gives more stability and less overhead to the network in comparison to a dynamic architecture. In the other hand, when the design of multi-controllers is determined, the connections between switches and controllers also fixed and the static connection mapping cannot adapt the dynamic change of network traffic, and the presence of overloaded controllers or underloaded controllers will severely degrade the overall performance of the control plane.

The cooperation between controllers enables them to share information about their network state and construct a global view of the network. This view is used by different controllers for decision making and exposed to network management applications. Unfortunately, the communication between SDN controllers is not yet normalized and most of the current solutions rely on self-developed interfaces that handle the message exchange. Moreover, the diversity of SDN controllers adds more challenges to federate heterogeneous controllers for integrated end-to-end control and management.

The main challenge that arises when operating heterogeneous SDN controllers is the integration of the controller's local views into a global coherent view that encompasses the overall network. The problem is that each controller may expose different information using different conceptual models. The difference could be in the structure of the data, the syntactical specifications of the data, or the semantics of the data. This makes a controller network view uninterpretable by the others.

4.2.0.4 Existing SDN platforms

Several SDN platforms have already been deployed to manage a set of controllers. The platformers may differ in their design; they could be suited for a certain type of applications such as large scale networks, fault tolerance, end-to-end flow management, etc. They could also be implemented following different architectures and different data models. The authors in [17] and [123] have surveyed the use of multi-controller SDN in the literature, they confirmed that there is a lack of standardization for the communication between the SDN controllers and that, till to this date, there is no SDN platform that takes into account the heterogeneity of the controllers. In the following, we present some of the proposed platforms, their architectures, and the communication between their controllers.

ONIX [62] and HyperFlow [112] are among the first physically distributed platforms of SDN controllers. ONIX uses a central network information base (NIB) that maintains a global view of the network, network applications can read and write into this base to manage the network elements. Hyperflow is an event-based architecture that supports OpenFlow, it implements NOX [43] controllers. DISCO [90], is a platform that can cope with wide area networks and overlay networks, it can be divided into two parts, an intra-domain part, that is responsible for monitoring the network and dynamically managing the network to maintain a satisfiable performance, it uses a central database for storing the controller's information, and an inter-domain part, that provides communication between domains. Kandoo [47] is a hierarchical platform for distributed controllers where root controllers take the responsibility of managing the network. Elasticon [31], is an elastic platform architecture, where a controller is added or removed dynamically depending on the network traffic load. OpenDayLight (ODL) [111] is an open-source platform, where a set of controllers collaborate as a cluster to achieve some performance criteria such as availability and scalability. ONOS [16], is also an open-source platform suited for high availability and scalability applications. It uses a central graph database to store network information.

4.2.0.5 Platform architectures

The architecture in terms of co-operations between SDN controllers can be either be logically distributed or centralized. In the distributed architecture, the controllers have just a local view of their control domain and base their decisions on it without sharing any state of the network with the other controllers. Whereas in the logically

centralized architecture, the controllers cooperate to share a global view, this cooperation can be either established directly between the controllers or via a centralized entity.

In a direct SDN controllers communication, each controller exchanges its local view to its neighbors to synchronize the view in the network. This architecture can be cumbersome in traffic load because of the number of exchanged synchronization messages and may present a major challenge in keeping the controller's network views consistent. In indirect cooperation, each controller exposes its local view about the network in a central entity. A global view of the network is constructed by consolidating all of the controller's information. This allows the controllers to make decisions based on a global network view that exposes a coherent current state of the network. The global view is periodically updated by the central entity.

The architecture of the controllers can also be viewed in terms of the interconnection between them. There is two types of interconnection, a horizontal (flat) interconnection, in which the controllers are positioned at the same level in the network and have the same level of responsibility, and a vertical (hierarchical) interconnection in which the controller is set into several layers, each controller has the responsibility of its local domain and the domain that is underlying its layers.

The controllers have to cooperate to build a global view of the network that contains an up-to-date state of the data plane e.g. topology, traffic load, etc. This view appears to applications and policy engines as a single, logical network domain. Each multi-controller platform implements differently the global view construction. For example, Elasiticon, uses a distributed data store to gather the information about the network, Hyperflow propagate the network state among the controller using a publish/subscribe system, ONOS maintains the network state at each controller using a graph database and propagate it by a publish/subscribe system and ODL elect a head controller to collect the network states from the controllers in its cluster and update the global view.

4.3 Semantic-based framework for global network view construction

We describe in this section a semantic-based framework that enables a multi-controller SDN platform to construct a global network view from heterogeneous controllers. The framework enables the platform to interpret controllers of local network views when they are represented with different data structures and models. To do that, the framework extracts an ontology from each controller network view to represent its data and the relation between them. Then, it constructs a global view by matching similar ontologies elements together. We propose this framework for two scenarios of constructing the global network view.

Scenario 1: Centralized global network view construction. the construction of

the view is centralized (Figure 5.12 a). Each controller exposes its local network view to a central entity that contains the semantic-based mediation layer. The framework integrates all the local views into a coherent global one.

Scenario 2: Distributed global network view construction. (Figure 5.12 b), the construction of the global view is distributed. Each controller disseminates its local view to the others and implements the semantic-based mediation layer locally to construct the global view, by merging and integrating the other collected local views.

4.3.1 Exemplified Problem Statement

Building a global network view of heterogeneous local views of SDN controllers require that the exchanged information have to be understood across the different controllers. The problems are the heterogeneity in the structure and the semantic of the data. The structure heterogeneity is the difference in the way that the data is defined. A given SDN controller has its local understanding of the SDN domain and therefore structure the data accordingly. For example, the current two leading open source SDN controller platforms, Open Network Operating System (ONOS) [16] and Open Day Light (ODL) [111], expose differently the network state of their management domain. ONOS internally represents the infrastructure (topology, installed flows, etc.) in a protocol-agnostic model called object model, while ODL employs a model-driven approach to describe the network, the functions to be performed on it, and the resulting state. The ODL model-driven infrastructure uses YANG as a modeling language for interface and data definition that allows applications and plugins to be developed from a single model. Figure 4.2 shows an example of a network topology that is exposed by the ONOS controller in (a) and ODL controller in (b), using JSON format. This topology consists of two switches and a link that interconnects them. We notice from Figure 4.2 that the network data is expressed using the different structures and different syntax. The semantic heterogeneity on the other hand relates to the intending meaning behind the syntactical expression. The exchanged information could be interpreted differently across SDN platforms depending on the way that the data is defined. It is important therefore that the heterogeneous controllers use a common understanding of the data of the domain to establish cooperation or to build a consistent global network view.

4.3.2 Centralized global network view construction

In the centralized scenario, the global network view is constructed by consolidating the controller's local network views into a single representation. The framework of building this global view could be endorsed by an independent entity or by one of the controllers. The central entity requests periodically the local views from all the controllers in the network and constructs consequently a global view by harmonizing the local views into a single, formal, and coherent representation. This representation

(a- ONOS)


```
[{"id":"of:0000000000000001",
 "type":"SWITCH",
 "available":true,
  "mfr":"Nicira, Inc.",
 "hw":"Open vSwitch",
 "sw":"2.3.1",},
{"id":"of:0000000000000002",
 "type":"SWITCH",
 "available":true,
  "mfr":"Nicira, Inc.",
 "hw":"Open vSwitch",
 "sw":"2.3.1",}]
[{"src":
{"port":"1",
"device":"of:0000000000000001"},
"dst":
{"port":"1",
"device":"of:0000000000000002"},
"type":"DIRECT",
"state":"ACTIVE"}]
```


(b-ODL)


```
{"network-topology":
 "topology":
[ { "topology-id": "flow:1",
  "node":
[{"node-id": "openflow:11",,
"termination-point": [
 {"tp-id": "openflow:1:1"},
 {"tp-id": "openflow:1:2"}],
{"node-id": "openflow:2",
  "termination-point": [
 {"tp-id": "openflow:2:1"},
 {"tp-id": "openflow:2:2"}],],
   "link":
 [{"link-id": "openflow:1:1",
 "destination":
{"dest-tp": "openflow:2:1",
"dest-node": "openflow:2"},
"source":
{"source-node": "openflow:1",
 "source-tp": "openflow:1:1" }}]]}
```


Figure 4.2: The heterogeneity of network data representation between ODL controller and ONOS controller

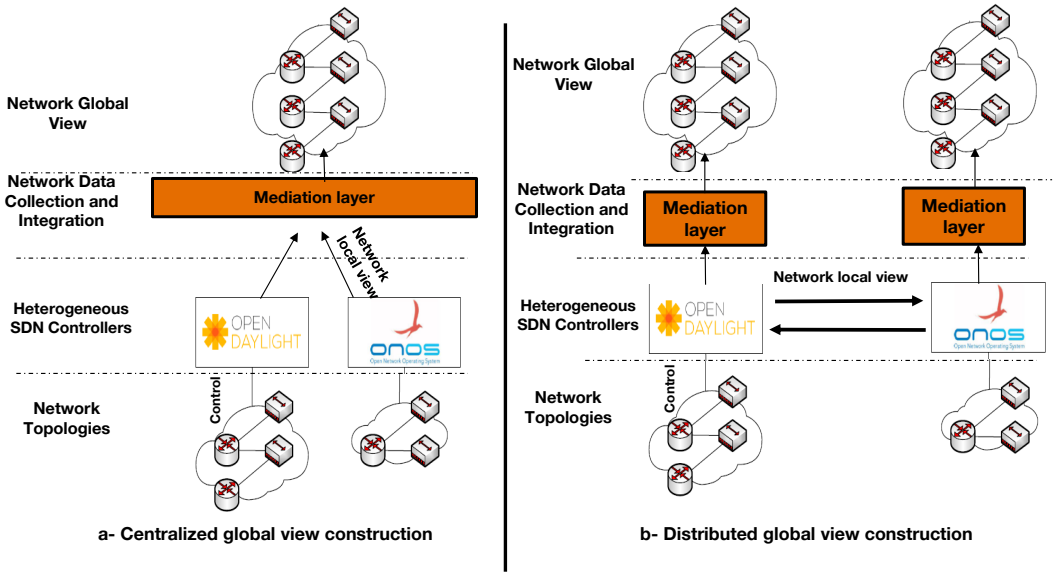


Figure 4.3: Different contexts for constructing the global network view

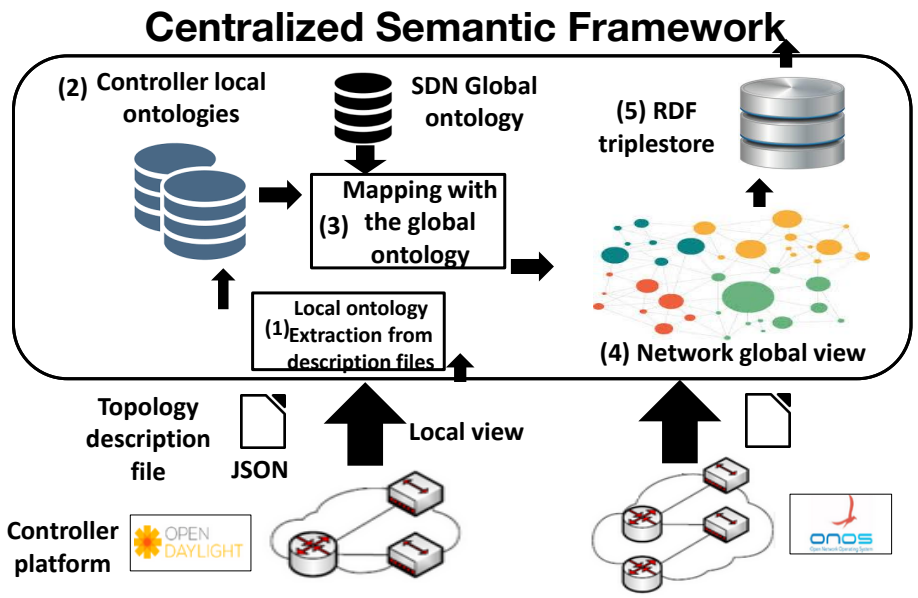


Figure 4.4: Mediation for the central global view scenario

could be afterward exposed to network applications for end-to-end network management or queried by the SDN controller for decision making. An advantage of this approach is that it provides a coherent and consistent single global view, shared by all the controllers. Constructing the global view require that the central entity extract information from the controller's local views and integrate them into a unified formal representation. Information extraction from the controller's local views could be a difficult task given the heterogeneity of their representation. We propose to use an ontology to describe a common understanding of the SDN domain. The ontology formally describes the different entities and the relation between them. We use this ontology with its global representation of the network to identify entities from the controller's local views and match them to their semantic meaning, irrespective of their syntactic representation. To do so, we build for each heterogeneous controller a local ontology from its description file of the exposed local view. This local ontology is used to describe the controller conceptual data model and the different entities that are represented in it. We map afterward this local ontology to the global one using semantic techniques. Mapping the local ontologies together with the global one clarifies the meaning of the controller's entities with the commonly defined ones. The semantic approach for the centralized global view construction is summarized in Figure 4.4, we discuss in the following the process of building the global ontology, extracting the local ontology from the controller's local view and mapping them with the global ontology.

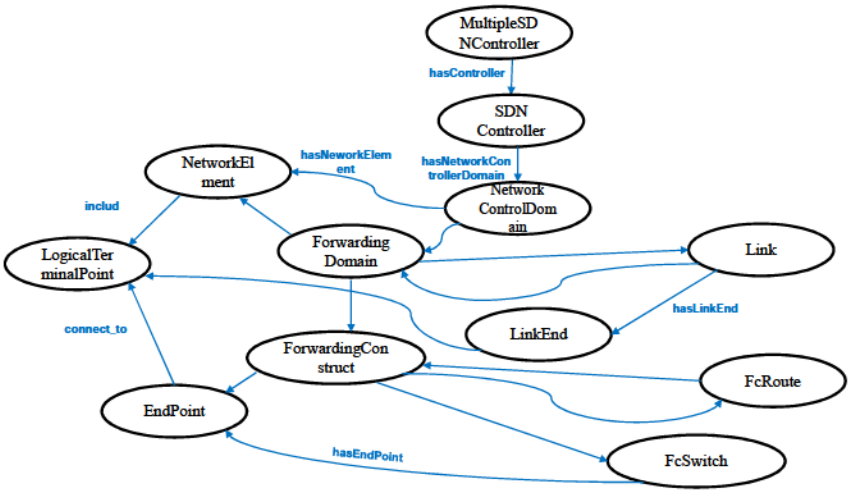


Figure 4.5: Brief representation of the global ontology

4.3.2.1 Global ontology model

We build a global ontology model that represents the domain knowledge of SDN with OWL. The ontology describes the entities that constitute SDN and the relations between them. This ontology is used as a mediator for integrating the data and hiding the heterogeneity between controllers network views. It has to be sufficiently comprehensive to be effective and that relevant information could be extracted from the controller's local view files.

To build our ontology, we relied on the common information model (IM) that was proposed by the Open Networking Foundation (ONF) [1]. This IM describes the objects of the SDN domain, their properties, and the relation between them. The IM objects are resources. A network view could only be expressed in terms of these objects. Figure4.5 shows a brief view of the global ontology for the multi-SDN topology. The global ontology takes into account the elements that are defined in the ONF IM, the relations, and the associations between them as well as the integrity constraints in the form of SWRL (Semantic Web Rule Language).

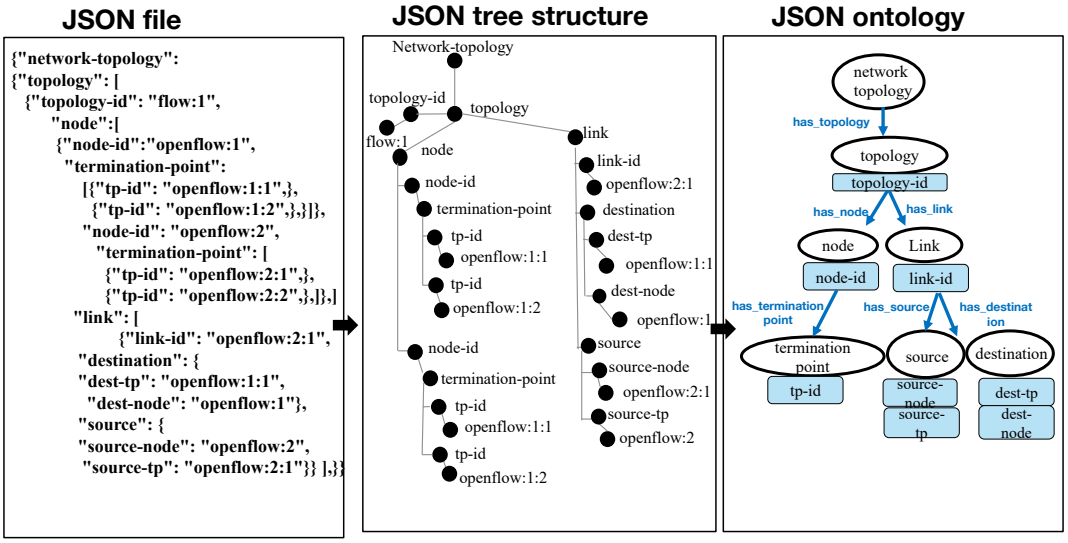


Figure 4.6: Extracting the local ontology from a JSON file

4.3.2.2 Extracting local ontologies form the controllers

We construct for each heterogeneous controller a local ontology that describes the conceptual model of its data. The ontology is generated from the description files of the controller's network view. We assume that the SDN controllers expose their service functionalities via an API that allows network hosts and applications to request the network topology. The topology can be subsequently returned in semi-structured data files e.g. JSON, XML, etc. These files are afterward parsed into a local ontology that identifies the elements which are described into the controller's network view and the relation between them.

The generation of the local ontology is performed by transforming the structure of the description file into a set of entities along with their properties and the interrelationships between them. We focus our attention on JSON format as it is commonly used by the controllers to expose their services and also because there is a lack of attention in the literature to ontologies extraction from JSON formatted data. Conversion methods from XML to OWL could be found in [19] and [45].

The data in JSON is stored in the form of key-value pairs and array data types that are grouped into objects. Typically, the description file consists of nested object structures, starting from a root object. Each object may contain other objects and/or attributes. The transformation to a local ontology is enabled by translating, starting from the root, the object name into an ontology entity (owl:Class), and the names of the attributes of the object into ontology relations between this entity and the type of the attribute. If the type of the corresponding attribute is a literal (e.g. integer,

string, boolean, etc.), the entity is matched with its corresponding ontology literal (rdf:datatype). Otherwise, if the value of the attribute is another object, the name of this object is translated to an ontology entity and linked to the first type with a property type (rdf:property). We apply this process recursively when the object attributes values are other JSON objects. The attribute value may also be in some cases an array (list) of elements (key/value pairs or only values). In this case, if the array is a set of key/value pairs, we consider that each element is an instance of the object. Therefore, the structure of one of the instances can define the object. Otherwise, if the array contains a set of values, then we consider the elements as a complex type and we associate this type to an RDF sequence in the ontology. We show an example in Figure4.6 of translating the structure of JSON into an ontology representation.

4.3.2.3 Mapping the local ontologies with the global ontology

The SDN global network view is constructed by mapping the controller's local views of the network with the global ontology. We use the global ontology to describe entities of the SDN domain along with their properties and their instances. Also, each entity in the ontology is augmented with a set of synonyms terms that helps to correspond it with other entities that have similar semantic but syntactical modeling. After extracting local ontologies from the heterogeneous controllers, we semantically map them to the global ontology.

The mapping phase aims to find similar entities between the controller's local ontologies and the SDN global ontology to correspond them together. We propose for that matter an approach inspired from [46] that maps entities based on their semantic similarity. The approach compares each entity in each controller's local ontology with all the global ontology entities and their synonyms terms. It takes into consideration the neighbors of the compared entities to enhance the matching accuracy. The most similar entity in the SDN global ontology is afterward mapped with the controller entity. The similarity between ontologies entities is calculated using the well-known Jaro–Winkler similarity metric. We note the similarity function $sim(e, e') \in [0, 1]$, where e is an entity of a controller local ontology O_l and e' is an entity in the SDN global ontology a global ontology O_g. The mapping approach can be summarized in algorithm 1.

For each entity in the controller local ontology (line 1), the mapping approach searches for a corresponding entity in the SDN global ontology that maximizes the semantic similarity (line 2-29). The process is executed in 4 steps:

- Step 1: (line 2-6) The algorithm searches for an entity in the SDN global ontology that has a similarity metric above a certain threshold. To do that, we compare all the entity's name and their synonyms with the controller entity name. We construct afterward a similarity table that contains the matched pairs, with a similarity above the threshold. The result is a set of candidate

Algorithm 1 Mapping between local and global ontology

Input: Local ontology O_l, threshold, scope

Output: Mapping between similar entities of local and global ontology

```
1  for each e ∈ O_l do
2      for each e' ∈ O_g do
3          for each (e'(name) and e'(synonyms)) do
4              if sim(e(name),e'(name_x)) ≥ threshold then
5                  add (e,e') to similarity_table
6      for each (e,e') ∈ thesimilaritytable do
7          weight(e,e')= sim_parents(e,e',scope)
8      for each (e(attribute) and e'(attribute)) do
9          if sim(e(attribute),e'(attribute_x)) ≥ threshold then
10             weight(e,e')++
11     for each e ∈ similarity_table do
12         Select e'_max  s.t.  weight(e,e'_max)= max(weight(e,e'))  ∀ e'
           Mapping(e)= e'_max
```

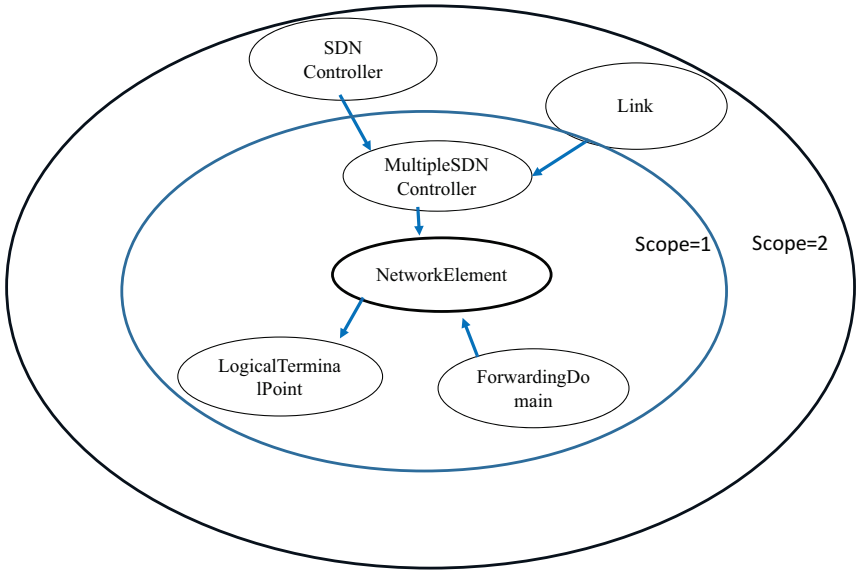


Figure 4.7: Exmaple of a scope of two areas around the entity "networkElment"

entities that could be mapped with the controller entity.

- Step2: (line 7-9) The algorithm compares the parents of the entities in the similarity table with the controller entity to further enhance the accuracy of the matching. For that, we define the scope of the search as a round area that encompasses the entities that are either directly connected or indirectly connected via other entities. Figure 4.7 shows an example of ontology, for which the entities around the entity switch with a scope = 2 are within a red circle. The similarity is calculated for each pair of parents of the two entities within the same area (same distance to the entity), the average value of the aggregated similarities is assigned as a weight of the entries in the similarity table.

- Step3: (lines 10-17) After comparing the parents of the entities in the similarity table, we compare in this phase their attributes. For each global entity in the similarity, we compare the similarity of its attributes and relations with the local entity.

- Step 4: (lines 18-21) We map the entity of the local ontology with the most similar entity in the similarity table. For that, the entity with the highest weight in the similarity table is selected and corresponded with the local entity.

- Step5: (line22-28) this step is considered when the similarity table in step 2 is empty (there is no similarity match with the global entities). A description model mismatches could be a reason for this case. Hence, We search for similar attributes and relations in the global ontology that could be corresponded to the local entity.

4.3.2.4 Interacting with the global network view

After mapping local ontologies with the global ontology, the data instances are afterward stored into an RDF triplestore as a triplet (subject, object, predicate). This allows network applications or controllers to interact with the global network view via a semantic query language like SPARQL. The query could be formulated in high level by using the terms defined in the global ontology. But they could also be reformulated into a controller specific terminology using a dedicated rewriting algorithm.

4.3.3 Distributed global network view construction

In this scenario, we build the global view of the network by matching local views of controllers. This is done in a distributed manner, where each controller gathers the other local views and build the global view locally. The distributed approach is more fault tolerant, the global network view is still available to the other. Also, it alleviates the load of accessing a single entity. Every node is responsible of the construction of the global view. This enhances the performance of the network in

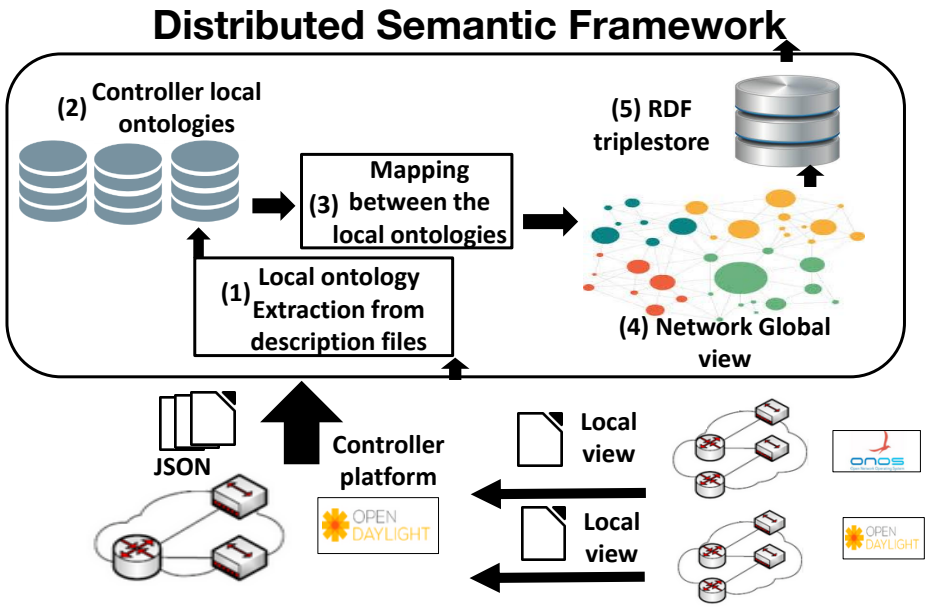


Figure 4.8: Mediation for the distributed global view scenario

terms of request latency. In the other hand, this approach adds more complexity of handling synchronization between controllers in order to keep a consistent global view. The consistency has a major impact on the SDN application performance.

Similar to the centralized global view construction, we use a framework that maps controllers local views together to form a global view. However, we do not rely on a shared common ontology to describe the SDN domain and link the local network views to it. It is improbable that SDN vendors agree on a common understanding of the SDN domain. Instead, we extract at the controller a local ontology from its topology description file that conceptualize the data model, and we do the same with the gathered local network views of the other controllers. We then map the local ontologies together to constitute a global knowledge of the network. Figure 4.8 depicts the framework for the distributed scenario. The extraction of the ontologies from the description file and the interaction with the global view are comparable to the centralized global view construction. We discuss next the mapping of local ontologies together at the controller.

4.3.3.1 Mapping local ontologies

After extracting a local ontology from controllers topology description files. A controller matches between these ontologies by searching for semantic mappings between their After extracting local ontologies from controllers topology description files. The controller matches between them by searching for semantic mapping between their

entities. It is a matching problem, where for each local entity in the controller ontology, we search for a corresponding entity in the other controller's local ontologies. The matching process is effectuated between two ontologies at a time. We propose an approach based on a lexicon that searchers for semantic similarities between entities. The lexicon is used to disambiguate the situation in which homonyms may occur.

The idea is to compare ontologies elements based on their semantics taking into account their local context. For that, we use the lexicon to identify the synonym sets of the compared entity. Then for each synset, we determine its accuracy by computing its similarity with the neighboring entities. We use the Wu-Palmer similarity metric that calculates the similarity between two entities based on their distance to a common closest hyperonym (ancestor). After determining the sense of an entity, it is compared with other ontology elements using the same similarity metric. Let O_c and O_d, the controller's local ontology, and a local ontology extracted from the collected topology description file. The approach could be summarized in algorithm 2.

Algorithm 2 Mapping between local ontologies

Input: O_c, O_l, scope

Output: Mapping of similar entities of local ontologies

```
13 for each entity e ∈ O_c do
14     entity_similarity = 0
       matched_entity = ∅ for each entity e' ∈ O_d do
15         synset_similarity = 0
           accurate_synset = ∅
           S=Get synsets of e' from the Lexicon for each synset s ∈ S do
16             TotalSim = 0 for each parent p of e' in its scope r do
17                 TotalSim = TotalSim+ sim(s,p)
18             TotalSim= TotalSim / ||p|| if TotalSim > synset_similarity then
19                 synset_similarity = TotalSim
                   accurate_synset = s
20         if sim(e,accurate_synset) > entity_similarity then
21             entity_similarity = sim(e,accurate_synset)
               matched_entity = e'
22 map(e,matched_entity)
```

The controller launches the mapping algorithm to possibly match the entities in its local ontology (O_c) with one of the other controllers ontologies (O_d). For each entity in O_c, the algorithm searches for a most similar entity in O_d. Let e be an element in O_c that we want to map to O_d. To that purpose, for each $e' \in O_d$, the algorithm determines first the best semantic of the term that is used in O_d to describe e' (line 7-15). It uses the lexicon to show the different synset that could be corresponded to this term (line 7). Then the algorithm determines the best synset by computing it

similarity with the neighbors of the e' using the Wu-Palmer similarity metric. The algorithm considers only the neighbors that are within the scope r (see 4.3.2.3 for further clarification about the scope r). The best synset is then compared with the element e using the Wu-Palmer similarity metric (13-16). After comparing all the entities in O_d, the algorithm chooses the most similar entity to e and correspond them together (line 16-18).

4.4 Evaluation

4.4.1 Evaluation environment

To evaluate the performance of our framework, we have carried out some tests over different network topologies in a multi-controller SDN architecture. We considered mainly two controller platforms, ODL and ONOS. They are currently the most deployed open controller platforms [17]. The topologies of network elements are generated for each controller using a Mininet emulator.

The proposed framework is developed using protégé for defining the global ontology (centralized scenario) based on the ONF-CIM TR-512 [1]. Note that we restrict our attention in this evaluation only on an ontology that describes the topology elements of the network and it is out of the scope of this work to propose a generic ontology that incorporates all of the SDN elements. The local ontology extraction method form JSON files and the mapping algorithms between ontologies are developed using a self-defined program in JAVA.

To test the framework in the distributed and centralized scenarios, we request the local network view of the controllers in a JSON format using their REST APIs and add them afterward as input in the two frameworks. The output is a global network view stored in an RDF base.

We evaluate the proposed frameworks in terms of execution time as it has a big impact on the performance of the SDN. We measure for that the execution time of the local ontology extraction, the mapping algorithms, and the interaction with the global view. We also evaluate the accuracy of the global view by measuring the success rate of the mapping algorithms between the ontologies.

4.4.2 ontology extraction from JSON files

We test the local ontology extraction process using local network views files. Each file describes different network topology instance from a different controller. Table 4.1 highlights the settings of the evaluation and the process execution time to create the local ontology. We notice that the extraction process has a low execution time and can escalate relatively good to the topology files.

Table 4.1: Execution time of ontology extraction

Controller	Switches/Links	File-size	lines	Execution time
ONOS	15/14	35Ko	421	0,013s
ODL	15/14	47Ko	614	0,018s
ONOS	99/98	290Ko	2921	0,192s
ODL	99/98	324Ko	4109	0,2s
ONOS	399/398	490Ko	11230	0,312s
ODL	399/398	530Ko	19230	0,401s

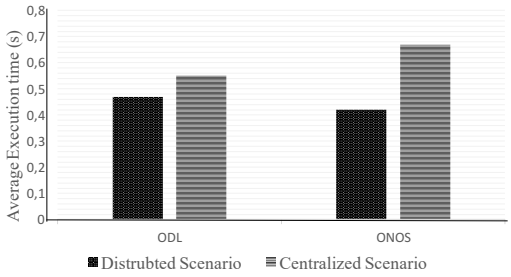


Figure 4.9: Average execution time of the mapping algorithms in distributed and centralized scenarios

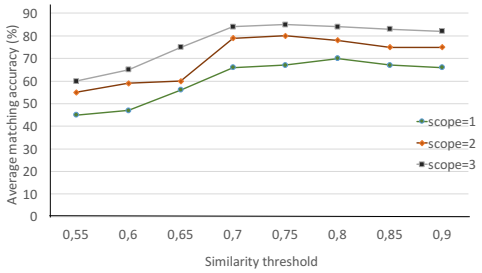


Figure 4.10: Average matching accuracy of the mapping algorithm in the centralized scenario

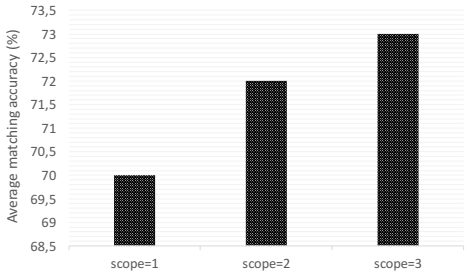


Figure 4.11: Average matching accuracy of the mapping algorithm in the distributed scenario

4.4.3 Ontology mapping

We run the ontology mapping algorithms in two scenarios (centralized and distributed global view construction) using different controller's topologies. We have developed for that purpose a basic lexicon that incorporates terms that are potentially used in the topology description files, along with their synonyms. The results are shown in Figure 4.9,4.10 and 4.11.

The algorithms have a relatively low computational time (as shown in Figure4.9) that makes them a potential solution for the SDN global network view construction. This time can be considerably reduced if the mapping results are stored for future use. That avoids the recomputation of the mapping for similar controllers. We can notice also that the mapping algorithms in the distributed scenarios are faster than the centralized ones. That makes sense, given that the algorithm in the centralized scenarios searches through all the concepts in the global ontology iteratively to find a good similarity match with the controller ontology.

Figure4.10 and 4.11 show the average accuracy matching of the mapping algorithms when are operating over different topologies. The centralized scenario is shown to be more accurate than the distributed scenario. This is because the global ontology offers a more generalized structure that can help to ease the matching process. Also, we can notice that the matching accuracy can be enhanced when the algorithms consider a high scope in the mapping process. Moreover, Figure4.10 show that the similarity threshold in the centralized is a linear function with accuracy. When the threshold increase, the accuracy also increases until it reaches a certain point where the results stabilize or even diminish. This could be explained by the fact that a high threshold makes the matching process more laborious which may result in no matching at the end.

Table 4.2: Interaction with the RDF triplestore

E2E topology (Switches/Links)	Average processing time
30/28	0,127s
198/196	0,18s
439/437	0,214s

4.4.4 Interaction with the global network view

We evaluate here the average time for an RDF triplestore to return an end-to-end topology of the network. The time is the duration that it is needed to send a request to the data store and receive an answer from it. We also use in this test several topologies, as shown in table 4.2, to test scalability in terms of processing time. The results in table 4.2 show that the interaction does not incur high processing time despite the high data to be returned and can, therefore, be suited for dynamic network operations.

4.5 Conclusion

We considered in this chapter the problem of heterogeneity between network components. This problem can prevent the management entity from collaborating with network components if the latter are not sharing a common understanding of how the information is defined. This is the first research question that we have specified in chapter 1: How to automatically interpret configurations from heterogeneous sources?. To address this research question, we focused our attention on a use case in SDN, heterogenous multi-controller SDN architecture.

The Operating heterogeneous controllers in software-defined networks (SDN) is a challenging task. Controllers have to cooperate to build a global view of the network. This view can be shared with network applications for end-to-end management or to other controllers for decision making. We proposed a framework that enables heterogeneous multi-controllers SDN platforms to construct a global network view. The framework can be used in two scenarios. In a centralized manner, where a central entity collects local network views from controllers and build afterward centrally a global view, or in a distributed manner, where controllers share with each their network views and construct locally the global view. The idea behind the framework is to use an ontology to represent the conceptual data each heterogeneous controller and afterward map the similar semantic data together to form a coherent and global knowledge of the network. The evaluation of the framework shows that it can be considered accurate and can have a low computational execution time. However, the framework is dependent on a lexicon (and on a global ontology in the centralized scenario) that needs to be defined manually and adapted constantly with the evolution of the SDN. This limitation is left for future works.

The work in this chapter is published in C1 (see chapter 1, section 1.4).

Chapter 5

Deep Learning for automatic configuration generation

Contents

5.1 Introduction

Auto-configuration in software networks are expected to enable management entities and network components to automatically adapt their configuration to unexpected situations to preserve the required network performance. Network elements are therefore required to generate automatically adequate configuration parameters with the current network state.

Automatic configuration generation could be done via predefined policies made by domain experts. The policies will indicate the behavior that should be considered in terms of configuration, given every possible situation that could occur in the network. This is a cumbersome task that possibly could not be efficient as covering all situations occurring in the network is very improbable.

Another alternative solution, which proven itself efficient in different scientific domains, is pattern recognition. It is the process of recognizing patterns from a given data set by using Machine Learning or deep learning algorithms. Pattern recognition is unsupervised and does not require human assistance, it can be defined as the classification of data based on knowledge already gained or on statistical information extracted from patterns and/or their representation. A Pattern is a special arrangement of data that can characterize a system. Learning patterns can afterward be used by predictive analytics methods to envisage what probable income these patterns present.

In our case, pattern recognition can be used to learn from the past made configuration to help network components predict new configurations and adapt to the current network state, without human intervention. Learning automatically patterns from configuration could also be beneficial for other applications. For example, the configuration patterns can be helpful to categorize the configurations based on their utilization and to assist the design of configuration files. Domain experts are still defining the configuration files for software networks manually and without any formal strategy expect their experience. Categorizing the configuration can help the extraction of insights regarding the impact of configuration on different contexts. It can also assist the configuration file design by recommending configuration parameters for the domain experts.

In this chapter, we investigate deep learning approaches to learn from configuration file models that can recommend and complete configurations. The recommendation model in one hand aims to learn patterns from a set of configuration files that helps the selection of appropriate configuration files described configuration as input. The completion model, on the other hand, learns from the sequence of terms in the configuration files a pattern that helps to predict what is needed to be added in a particular configuration file.

Both of these approaches could be used by management entities like an orchestrator to augment their abilities with the capability to select, recommend, and complete configuration files. In the case of an NFV orchestrator, we argue that it is important

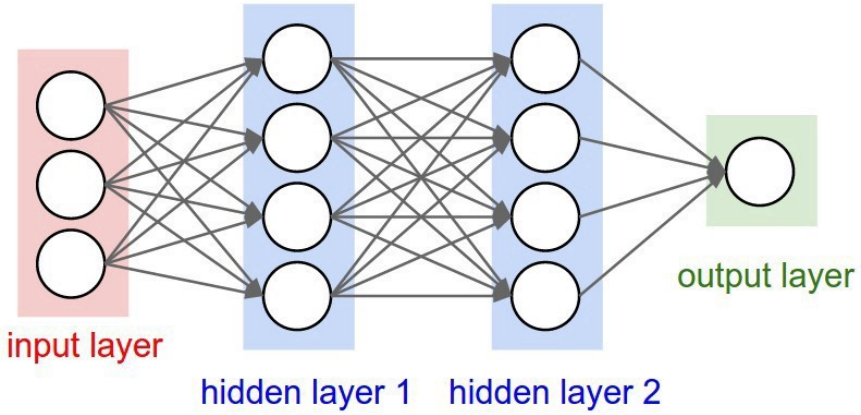


Figure 5.1: A basic multi-layer artificial neural network

to adapt the requirement related to deploying VNFs based on the network condition. This is because the ultimate goal of service providers is to satisfy a certain performance. It is therefore up to the orchestrator to readjust the resources and align them with what is available in the network, while satisfying the performance required, instead of relying solely on what was predefined by the service provider.

This chapter is organized as follows: we start by discussing the use case of onboarding the VNFs in NFV 5.2. We then present background information on deep neural networks in 5.3 and afterward present our deep neural network framework for recommending and completing configuration files 5.4. we evaluate this approach and analyze the results in section 5.5, and we conclude the chapter in section 5.6.

5.2 Background information on Deep learning approaches

Deep learning techniques have gained recently a lot of attention in the scientific community, they proved their efficiency in multiple fields such as computer vision [100], natural language processing [83], audio recognition [23], machine translation [28], etc. Deep learning is a subfield of machine learning methods that are based on artificial neural networks. The artificial neural networks are inspired by our understanding of the biology of our brains and the interconnections between the neurons.

Basically, deep learning architectures use multiple layers of artificial neural networks to progressively extract higher-level features from the raw data. Given a large data set, deep learning algorithms will learn a model that can generalize the prediction of the output by learning the weights of the connection between the neurons. Figure 5.1, shows a basic multi-layer neural network structure, composed of input, hidden, and output layers. Each layer is composed of neural cells that are connected to other cells in different layers. Each connection i has a weight (w_i) associated with it, it indicates the intensity of the link between the cells.

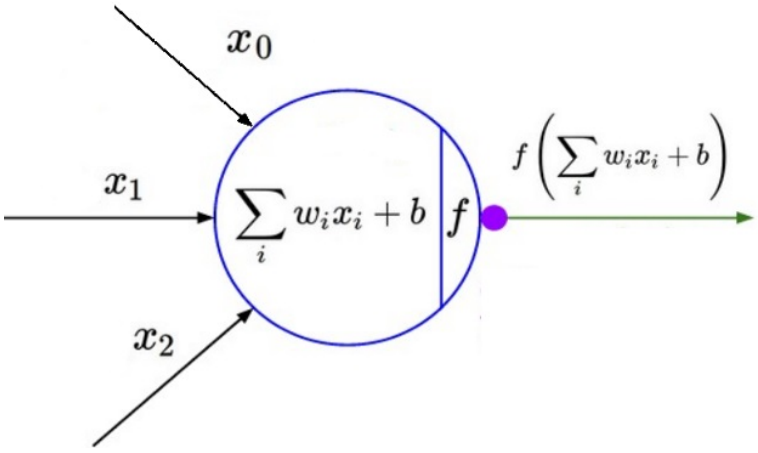


Figure 5.2: Activation function of neural cell

The neural networks are trained using the dataset to minimize the difference between its prediction and the expected output. In the learning phase, all the weights of the neural interconnections are firstly initialized. The initialization could be random or following a predefined strategy. Then, the input layer takes in a numerical representation of the data for the dataset and passes it through the hidden layers to the output layer, which outputs later the prediction. The neuron cells in each layer determine when to pass the information to the next layer. The cell output is a function of the predecessor neurons and their connections as a weighted sum. A bias term b can be added to the result. This function is called an activation function. Figure 5.2 depicts the activation function, x_i is the information of the predecessor neuron cells.

The neural networks adjust all the weights of the network to improve the accuracy of its result. This is done by taking pairs from the dataset of the input data and the expected output to minimize the difference between the prediction and expected output. A loss function like the mean squared error is used to calculate the prediction error. The weights are afterward readjusted to compensate effectively the error among the connections. This can be done by calculating the gradient (the derivative) of the cost function associated with each cell with respect to the weights. This approach is called backpropagation. By doing this, the neural network learns the association/pattern between given inputs and outputs and allows the deep learning model to generalize to other data that are not included in the data set.

There are various architectures of deep learning each is dedicated to a special kind of task. In our chapter, we are considering Convolutional Neural Networks (CNN) that was originally used for computer vision given its ability to extract automatically features from the data and Long Short Term Memory (LSTM) that were used in Natural language processing and time series data to learn patterns in data sequences.

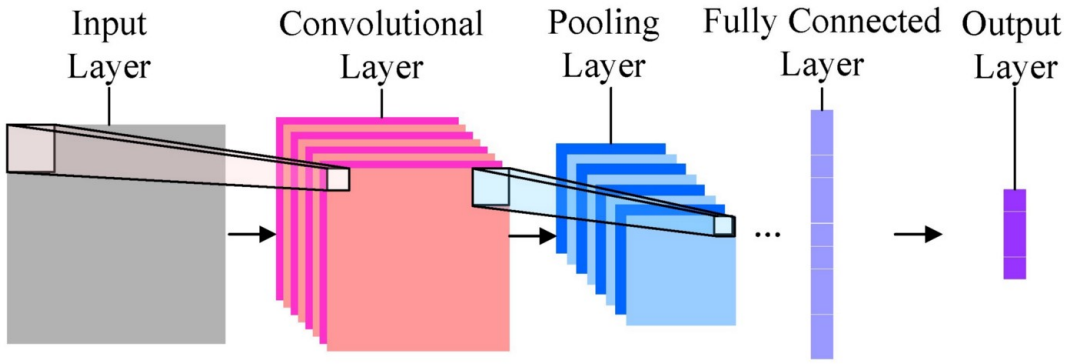


Figure 5.3: General overview of a CNN architecture

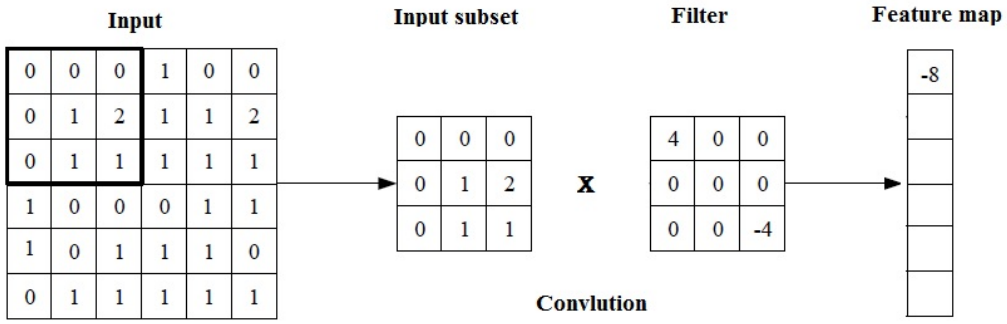


Figure 5.4: Convolution operation between the input and the filter

In the following, we describe briefly these two architectures.

5.2.1 Convolutional Neural Networks (CNN)

The CNN architecture is mainly characterized by three types of layers: convolution layer, pooling layer, and fully-connected layer. These layers could be stacked together multiple times to achieve the feature learning process. Figure 5.3 illustrates a simplified architecture. We briefly describe next the function of each layer.

The Input layer: like the other neural network architectures, holds a numerical representation of the data. This could be for example pixels of an image or words of a text.

The Convolutional layer is the most important and vital part of CNN. It is responsible for the feature extraction from the dataset. It applies a series of filters, also known as convolutional kernels to each data in the dataset. The filters are numerical matrices that are used on a subset (which is also a matrix) of the input values. The size of the subset is similar to the size of the filters. These filters are in most cases small in spatial dimensionality and spread along with the entire depth of the input

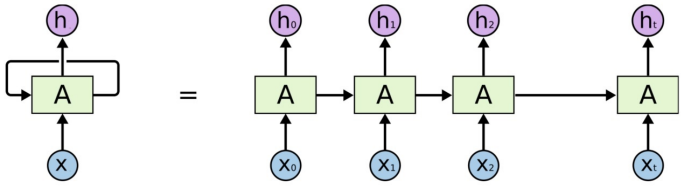


Figure 5.5: Architecture of recurrent networks

data. Each filter is multiplied to the subset by a dot product called convolution. It is that operation that can extract high-level features such as edges, or word contexts from the data. The convolutions are all summed up to have a single number, for a filter at a given time.

Each filter constitutes afterward a feature map, a vector containing the convolution results of the filter. The feature map is filled by moving the filter position in the input data with a certain stride value until it parses the complete length of the input data. Moving on, the vector is added with the result of the convolution operation at each position.

Every filter will have a corresponding feature map, which will be stacked along the depth dimension to form the full output volume from the convolutional layer. Figure 5.4, illustrates a single convolution operation between the input and the filter to fill the feature map.

The pooling layer aims to gradually reduce the dimensionality of the input representation and further reduce the computational complexity of the model. It operates over each feature map in the input, generated by each filter. It scales the dimensionality of the feature map by using a function that selects the appropriate value in the feature map. In most cases, this comes up to max-pooling, the maximum value in each feature map is selected.

The fully connected layer is a layer that is fully connected to all possible outputs. This is analogous to the way that neurons are arranged in traditional neural networks. It is the last block of the CNN architecture and it is used to make the classification.

CNN has been used extensively for many applications like facial recognition [100], natural language processing [83], anomaly Detection [106], recommender engines [122], etc. In this chapter, we are using CNN to learn from a model that can recommend configuration files given an input specification.

5.2.2 Long short-term memory (LSTM)

LSTMs are a special type of recurrent neural networks. Recurrent neural networks are a traditional neural cell in which the neural connection is a loop. The network is

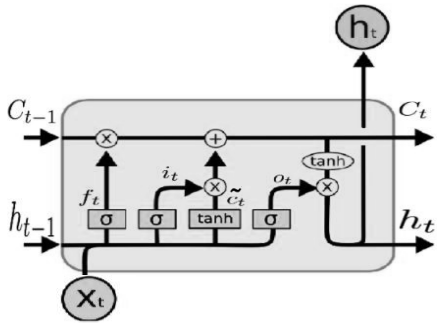


Figure 5.6: Long short-term memory cell architecture

therefore formed by cycles and it takes as input the current information and also the previous time step to influence predictions at the current time step. By doing this, recurrent networks have the ability to store information for an arbitrary duration and resist noise. They are trained to predict what could be the next data to output given the current context. Figure 5.5 illustrates the basic architecture of recurrent networks. There are three main components, the input data, the hidden layer, and your output layer, denoted as x, A and h above. At each transition, the network takes in the new data and the hidden layer from the previous time step, thus continuing to learn from past data points as it moves through the entire dataset.

LSTMs have basically emerged primarily to overcome the main problem of recurrent networks, the vanishing gradient problem. It is a problem that occurs when learning from large datasets. The readjusting of the weights of the neural network layers in this becomes not efficient. Each of the neural network's weights receives an update proportional to the partial derivative of the error function with respect to the current weight. The problem is that this gradient could become extremely small, and thus not stopping the weights from readjustment.

LSTMs have the ability to memorize long-term dependencies and overcome the vanishing/exploding gradient problem. It contains an internal state variable that is passed from one cell to the other and modified by operation gates. A simple LSTM cell contain generally three gates, as illustrated in Fig 5.12:

• Forget gate (f_t): this operation helps to remove irrelevant information from the input and thus keeping only what is important to remember. It takes the output of the previous state, h(t-1), and performs on it a sigmoid function (σ). W and b are respectively the weight matrix and bias term of f_t

$$f_t = \sigma(W_f.[h_{t-1,x_t} + b_f]) \tag{5.1}$$

• Input gate(i_t): this operation decides which element from the present input to be updated and creates a vector for new candidates to add to the cell state ($\hat{C}_t$).

$$i_t = \sigma(W_i.[h_{t-1,x_t} + b_i]) \tag{5.2}$$

$$\hat{C}_t = tanh(W_C.[h_{t-1,x_t} + b_C]) \tag{5.3}$$

• Output gate(o_t): this operation decides what to output from the current state.

$$o_t = \sigma(W_o.[h_{t-1,x_t} + b_o]) \tag{5.4}$$

The current state of the cell C_t is updated based on the previous gate operations and a filtered version of this state h_t is passed out as output to the next cell:

$$C_t = f_t * C_{t-1} + i_t * \hat{C}_t \tag{5.5}$$

$$h_t = o_t * tanh(C_t) \tag{5.6}$$

LSTM has been used successfully for various application like language modeling [83], machine translation [23], Image captioning [100] Hand writing generation [83] Image generation [85] Question answering [70], Video to text [115], etc.

We use LSTM in our work to learn from the configuration files a model that can be used to complete the files with additional information.

5.3 use case: Learning from deployment descriptor in NFV

We consider in this chapter a use case of VNF onboarding in NFV frameworks to test our proposed deep learning approach for recommending and completing configuration files.

NFV enables service providers to have an ability to bring quickly new services to the market to grow and to reduce the cost of network services from both CAPEX and OPEX perspective. It also allows them to deploy their services on industry-standard high volume servers, switches, and storage located in data centers, network nodes, and end-user premises.

Service providers have to associate with VNFs and NSs the descriptions related to their deployment. The descriptions are written in semi-structured files like XML, JSON, and YAML. They are called deployment descriptors. They describe for a network service [1], the virtual network function(VNFs) that compose it, the topology of interconnecting the VNFs, and the data flow direction between them. For VNFs, the deployments descriptors indicate the deployment and operational behavior in terms of connectivity, interface, and virtualized resource requirements [2]. Figure 5.7 shows an example of a deployment descriptor that is onboarded with a VNF by a service provider.

The ETSI NFV has released a specification that defines the requirements for the structure and format of a VNF deployment descriptor (VNFD) and network services(NSD). Figure 5.8 illustrates the high-level structure of a VNFD. The VNFD

is composed of one or many virtual deployment units (VDUs) that describe the deployment resources and operation behavior of a VNF component (VNFC). VDUs are virtual machines that host the VNF or parts of it. Each part of the VNF is a VNFC and can be deployed on one or more VDUs. Each VDU is characterized by, among others, the software image loaded on it and the resources needed to deploy it. Figure 5.9 illustrates a VDU deployment view. A VDU describes mainly the virtual compute (VC), virtual storage (VS) and virtual memory (VM) resources that are necessary for deploying a VNFC and it could be linked via connection points (CPD) to other VDUs or to external VDUs that belong to other VNFs via external CPD. Virtual links in the VNFD indicate how the VDUs are connected and via which CPD.

The NSD in the other hand contains multiple VNFDs for each VNF that compose the service and multiple Physical Network Function Descriptors(PNFD) for each legacy network function. The VNFs and PNFs are interconnected via Virtual links that are described in a virtual link descriptor (VLD). A VNF Forwarding Graph Descriptor (VNFFGD) can be defined to describe the topology of interconnection between VNFs and PNFs and how they can exchange information with each other. Figure 5.10 illustrated an overview of an NSD.

The onboarding of new functionalities in the NFV frameworks is a very challenging task. According to the SDxCentral report, it is one of the top two major problems for VNFs, along with monitoring. The reason is that the deployment descriptors are complex and tedious to design manually or with static automation, they contain numerous components that are dependent on each other. Also, there is no formal strategy to assist the creation of the deployment descriptors except the experience and best practices of the domain experts. This leads to a plethora of strategies for deployments. Different domain experts may choose different configurations for deploying the same network service/function, which makes it even harder to learn from past deployment strategies and identify deployment characteristics related to a network service/function. Moreover, despite some efforts to find a consensus on a common information model to use for modeling the deployment descriptors like the ETSI NFV initiative [1], there is still an absence of a common data model for the deployment descriptors, which resulted on a diversity of solutions, such as: TOSCA, YANG, Hot, etc. This diversity obliges service designers to constantly adapt the descriptors to the heterogeneous NFV platforms in order to enable the integration of the network services/functions. Therefore, designing the deployment descriptors for the network services/functions in such a highly dynamic environment becomes a highly complex, time-consuming, and tedious task.

It is therefore beneficial to capitalize on past VNFDs to help design or generate new VNFDs. To illustrate this observation, we give an example of two VNFDs.

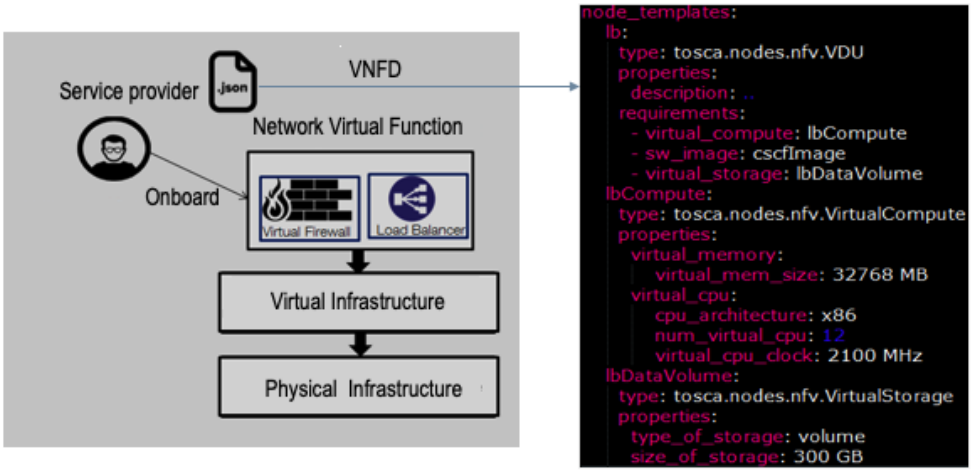


Figure 5.7: Example of a deployment descriptor onboarded by a service provider with a VNF

5.4 Deep learning framework for configuration recommendation and completion

5.4.1 Overview

In this section, we propose an approach based on deep neural networks to learn from deployment descriptors a model that could complete and recommend a description for deploying network functionalities in NFV platforms. We focus particularly our attention on learning from VNFDs as they are fundamental for VNF and NS onboarding and deployment. Nevertheless, this approach is agnostic to a data model or a given type of file. It could be applied therefore for also the NSDs or other configuration files.

Our approach could be used as a solution that could enable network orchestrators to complement a given network deployment descriptor with appropriate data, select and recommend descriptors to ease the design and automate the onboarding and deployment phase of VNFs.

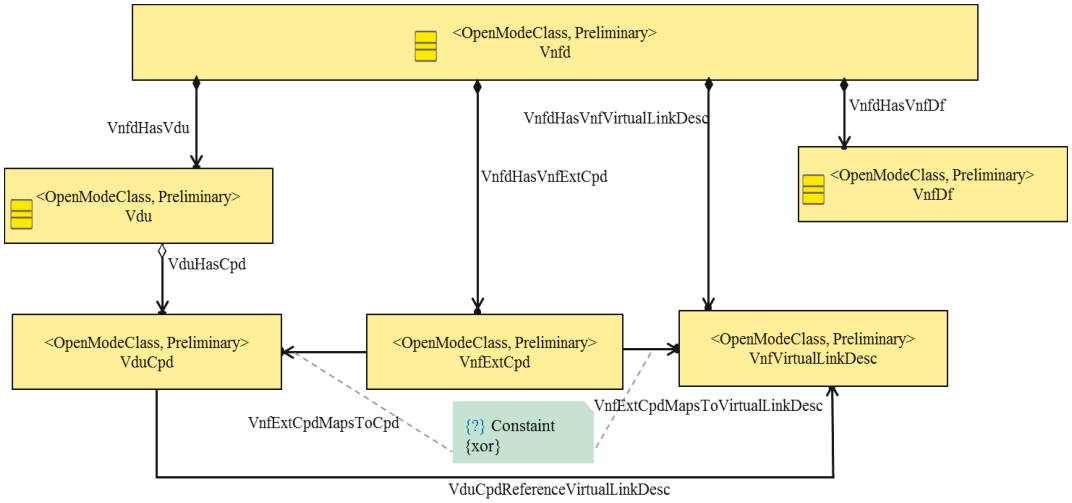


Figure 5.8: A high-level representation of a VNFD structure [38]

We assume that there is a catalog of multiple VNFDs following the same data model and the same format on which we are learning a model to recommend and complete the descriptors. Our approach is a framework that encompasses three different phases. Figure 5.11 shows an overview of the proposed framework.

The first phase of the framework is the data preparation phase, the goal of this phase is to pre-process the VNFDs into a format that can accurately be analyzed by deep neural network techniques. The output of this phase is to set a numerical representation of the fundamental data units that compose the set of VNFDs. To do that, we use a word embedding approach called word2Vec [78] to learn a representation that reflects the semantic of each data unit in the set of VNFDs. Doing so, this phase helps to learn patterns that can characterize not only the VNFDs but also the language on which they are defined.

The second phase of the framework is the learning phase. It takes as input the pre-processed data. This phase is based on two deep neural networks architectures, that is CNN and LSTM (see section 5.2 for their description).

CNN is employed for learning a model that could recommend an existing VNFD from the catalog. CNN is best suited for this task as it can learn features, without supervision or a liberalization step, that could characterize the deployment descriptors. Thus helps the selection of the appropriate VNFD given a partial description as input.

There are a lot of application scenarios for recommending deployment descriptors. For example, a service provider could use this framework to learn from a database

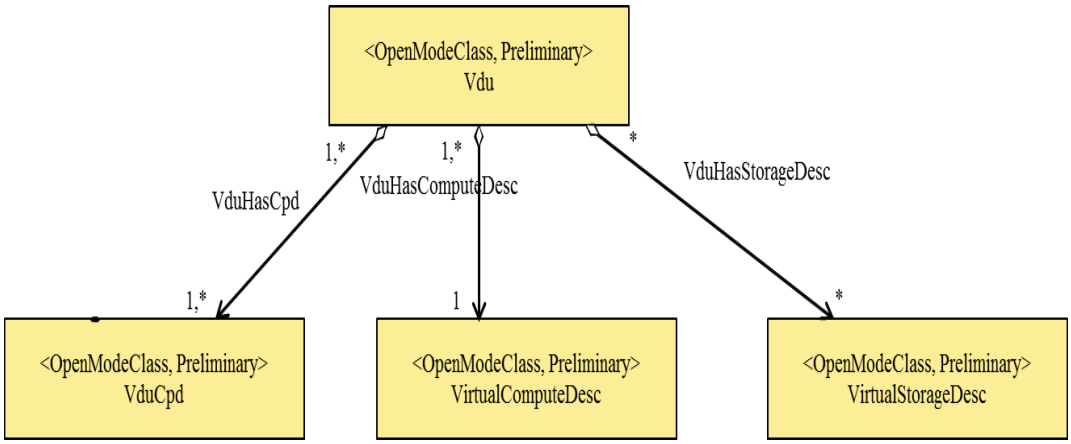


Figure 5.9: Composition of Virtual deployment unit in a VNFD [38]

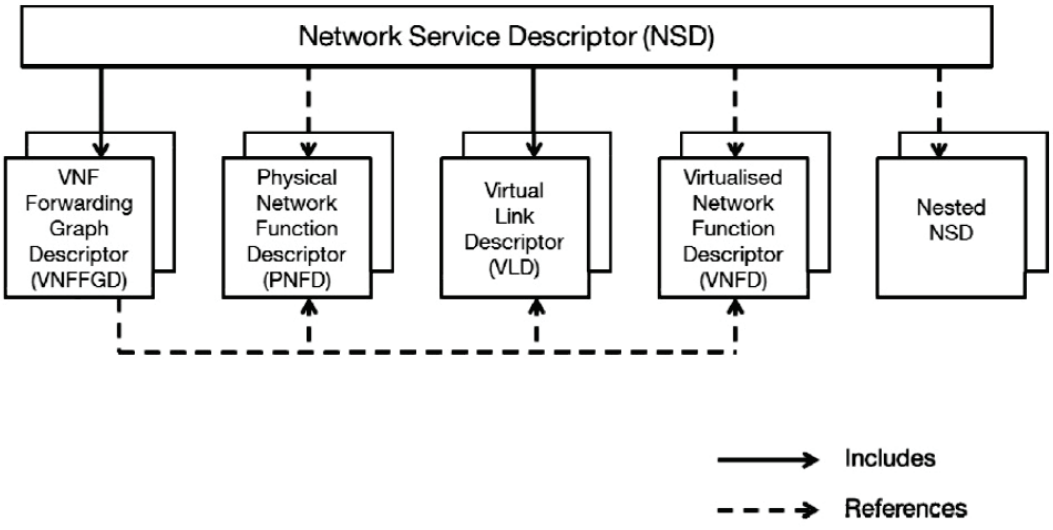


Figure 5.10: Overview of a NSD architecture

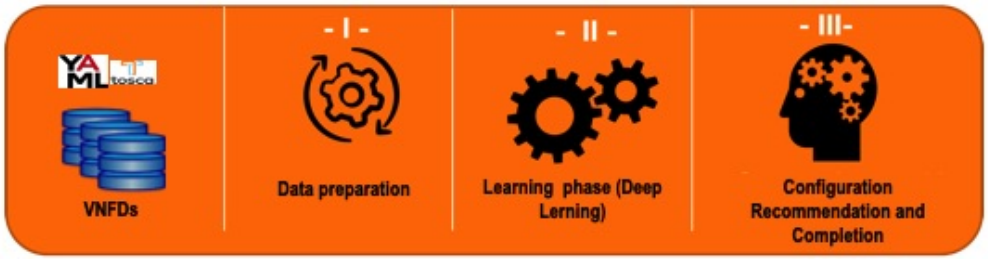


Figure 5.11: Overview of the deep neural network framework

of previously made VNFDs and use the generated model to help him design and adapt new VNFDs. The framework can help him select fragments of VNFDs that are adequate to a given description and assemble them at the end to form a new VNFD. Another scenario could be to help an orchestrator adapt to unexpected situations. For example, it could be a situation where the network requirement of an already onboarded VNFD could not be satisfied due to: overloading of resources utilization in the network, oversizing of requirements by the domain experts, or to the technical problem in the VNFD. The orchestrator could, therefore, overcome these situations by selecting another existing VNFD in its repository that is closely similar to the initial VNFD in terms of performance expectation and deploy the VNF with the new descriptor. Thus the performance can still be satisfied with different acceptable configurations.

LSTM is on the other hand utilized to learn a model that completes the VNFDs with additional data. The choice of using LSTM for this task is more than appropriate, given that LSTM proved its efficiency over different similar tasks in Natural language processing. Its ability to learn arbitrary long-term dependencies in an entire sequence of data allows it to be a powerful approach to be used to learn from a sequence of descriptions in the VNFD files a model that could generate entirely new plausible sequences of the description given an initial input.

The LSTM technique could be used by an orchestrator to adapt the VNFDs to new situations by adding additional descriptions, or changing unfeasible description like for example a policy that satisfies the expected performance and allow the VNF to be deployed in the current network state. Another important use case for applying the completion model is to ease the VNFD generation by service providers. The model can help to complete the description with a coherent description that can be tailored afterward by the service provider.

During the third phase of the framework, the generated LSTM and CNN models are executed and tested. Their hyperparameters are tuned over multiple training phases to enhance the accuracy and performance of the models. The final result is afterward used in the design time of the VNFD or in the deployment time.

In the following, we describe each of the aforementioned phase of the deep neural network framework.

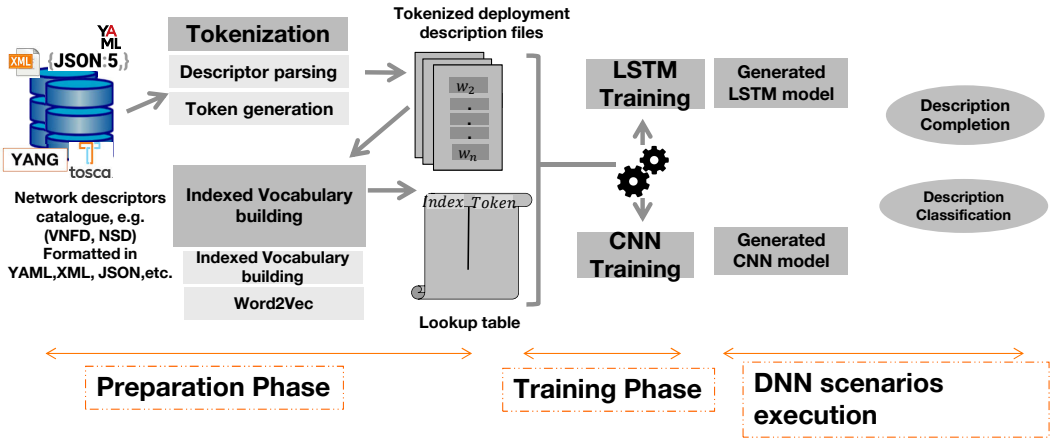


Figure 5.12: DNN-based framework for VNF deployment descriptors mining

5.4.2 Preparation phase

The objective of this phase is to transform the VNFD files in the catalog into an understandable and processable format by the deep neural network techniques to enable them to learn and capture structural patterns from the files.

Like we mentioned previously, the deployment descriptors including the VNFDs could be stored with different formats such as YAML, XML, JSON, and modeled with different data models such as TOSCA, YANG. It this important in the preparation phase to process the data to not only capture the language used to describe the VNFDs but also the format in which those VNFDs are written. This helps the neural networks to learn a model that could generate an output with a coherent syntax and format. To simplify the learning at this stage, we assume that the catalog of VNFDs includes only files with a similar format and similar data model.

In the preparation phase, all the elements that constitute the VNFD files are converted to vectors and projected to a vector space, such that the vector space could include all the possible vectors extracted from the VNFDs. An element of the VNFD is a vector with a numerical value. The deep learning techniques takes as input the set of vectors that constitute the VNFDs to extract relevant patterns from the VNFDs. In this phase there two steps. In the first step, called tokenization, we show how we extract the elements from the VNFDs, and in the second step, called vectorization, we propose two approaches to convert the elements into vectors and project them into a vector space. The first approach is based on the appearance of the VNFD elements and the second one is based on word embeddings. We describe next the two steps of the data preparation phase.

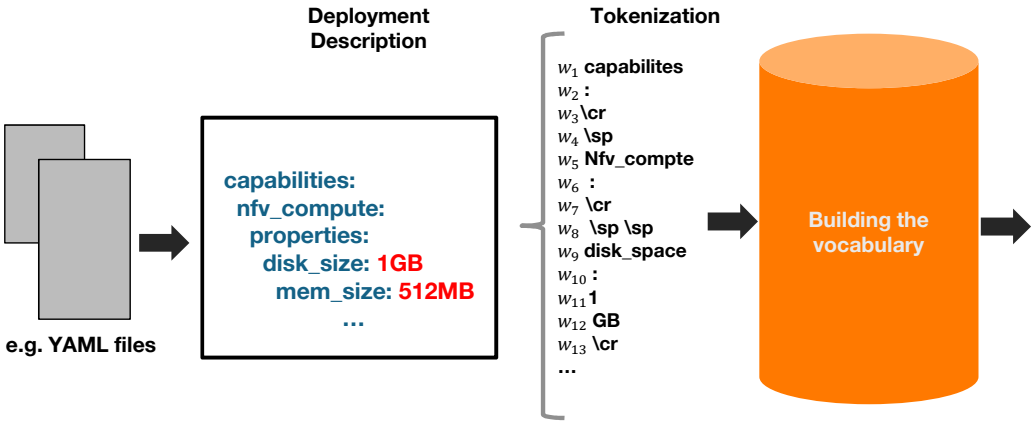


Figure 5.13: Example of tokenization using a deployment descriptor file

5.4.3 Tokenization step

The tokenization step aims at extracting the relevant elements from the VNFDs. This task breaks down each VNFD into a set of independent elements, called tokens. The goal is to do the decomposition in a way that captures characteristics of the VNFDs in terms of lexical characteristics and also the file format characteristics. To do so, we consider as a token each term in the VNFDs that have a strong significance in the corpus, to define the files format or the data model of the VNFD. All special characters that define the structure of the files are also considered as tokens.

For example in the case of YAML files, the whitespace indentation is considered as a token, given that it has a logical role in the descriptor file, it denotes the data structure adopted within the descriptor. For JSON files, the brackets are also considered token as they define the structure of the file. The same thing is with tag characters in XML files. In Fig.5.13, as an example, we show a partial VNFD YAML file that is tokenized.

5.4.4 Vectorization step

The vectorization step aims to project to a vector space the tokens that were extracted in the previous step. The output of this step is a look-up table that contains all the tokens and their corresponding vector representation. We propose two methods, the first one called appearance-based vectorization and the second one is a method of word embedding adapted to deployment descriptors, we called it token embedding.

5.4.4.1 Appearance-based vectorization

Appearance-based vectorization is a basic method that consists of indexing the elements/tokens based on their appearance in the code corpus. Each index is a vector

that is encoded in binary format with a size that could fit the indexation of all the elements of the corpus. We assume that similar elements across different VNFDs have the same index. The output of this method could be seen as a list of indexed vocabulary, where vectors are ordered based on the appearance of tokens in the VNFDs.

5.4.4.2 Token embedding

Word embeddings are methods used for language modeling and feature learning. They are essentially used to vectorize words or phrases from the vocabulary with real numbers. Conceptually it involves a mathematical embedding from a space with many dimensions per word to a continuous vector space with a much lower dimension [78].

The token embedding approach is a word embedding approach that learns a representation for each unique token extracted from the set of VNFDs. The goal is to learn a representation in a vector space such that the tokens that share common context in the corpus are located near one another in the space. Figure 5.14 illustrates a projection of the token embeddings extracted from the VNFD after reducing their dimensions to a 3D space. The black condensed area in the figure are tokens that have close embeddings, meaning that they appear frequently together inside the VNFDs.

The learning approach of the token embedding is a generalization of the SkipGram approach that is implemented in the word2vec software [78]. SkipGram performs a proxy task to learn the word embeddings, which is finding the probability distribution of tokens in the vocabulary to their context (other tokens within a given distance). It uses a simple neural network model with one hidden layer. The actual goal is to learn the weights of the hidden layer. These weights represent the embedding of the token.

The neural network is trained on a large set of VNFDs. It takes as input a target token in the middle of a window of tokens and predicts the probability for every token in the vocabulary of being the context token. The context token is a word in the proximity of the target token. The context size is parametrized to indicate to which distance from the target token (in both directions) is considered as a context token. The output probabilities relate to how likely each vocabulary token nearby the target token.

In the training phase, the neural network is fed with pairs of tokens (target token, context token). For example as an analogy to natural language, given the sentence "Paris is the capital of France", for a target word "capital" and a context size 3, the training pairs of words are : (capital, of), (capital, France), (capital, the), (capital, is), (capital, Paris). After training the neural model. For example, it will likely output "France" when Paris" is given as input than the word "Sahara".

More formally, given a repository of VNFDs with a token vocabulary size L. The input of the neural network is a one-hot encoded vector w_i of size L. The weights between the input layer and the hidden layer are represented by $L \times E$ matrix, denoted U, where E is the vector embedding size and $w_i * u_i$ is the embedding of the token

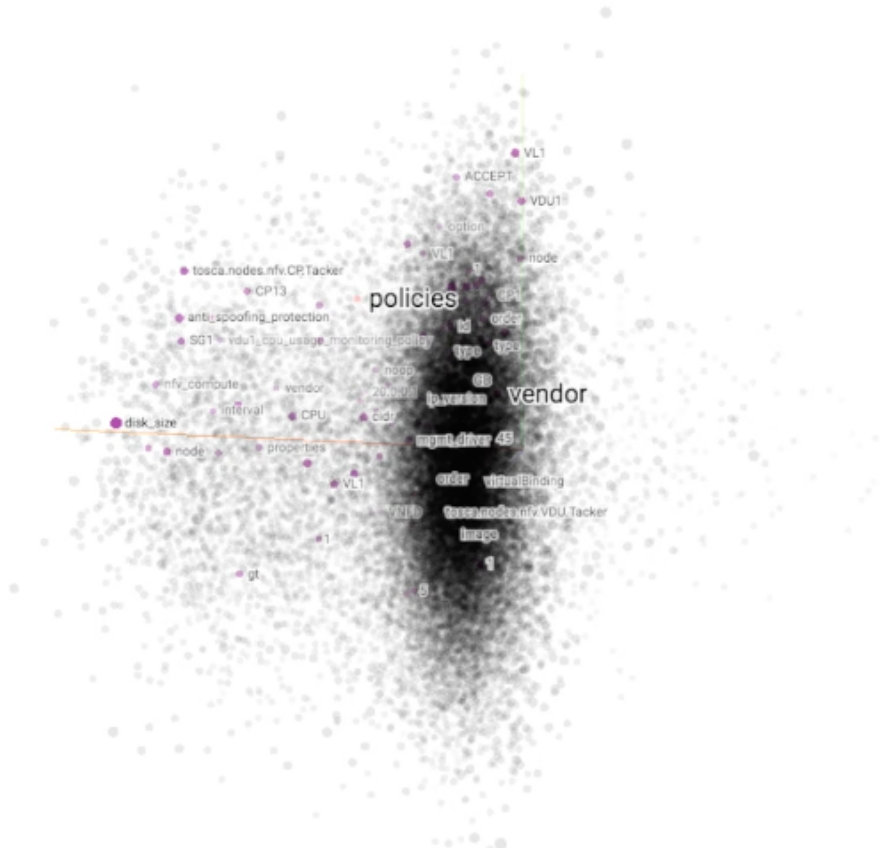


Figure 5.14: Projection of VNFD token embedings in a 3D vector space

w_i. From the hidden layer to the output layer there is a different weight matrix, that represents the context, denoted, V, which is also a $L \times E$ matrix.

The conditional probability of observing the target token w_i given a context token w_j is

$$p(w_j|w_i; U, V) = \frac{1}{1 + e^{-u_i{}^\top v_j}}$$

The sigmoid function is strictly increasing, the larger the scalar product of the representations of two tokens (which means that the two vectors are similar), the higher the conditional probability.

where w_i and w_j are the model parameters to be learned. The goal is to maximize the log-probability of the observed pairs belonging to the data, the objective is therefore:

The maximization of the likelihood of the data set D (set of pairs target and context tokens) is:

$$argmax_{(U,V)} \prod_{(w_i,w_j)\in D} \sigma(u_i^\top v_j)$$

where $\sigma(u_i^\top v_j) = p(w_j|w_i; U, V)$

Since logarithm is a strictly monotonic increasing function, maximizing the likelihood function is equivalent to maximizing the log of this function

$$\text{argmax}_{(U,V)} log(\textstyle\prod_{(w_i,w_j)\in D} \sigma(u_i^\top v_j)) = argmax_{(U,V)} \textstyle\sum_{(w_i,w_j)\in D} log(\sigma(u_i^\top v_j))$$

Negative sampling is proposed to avoid the situation when the problem admits a trial solution like choosing U and V as two matrices whose coefficients are all fixed to a constant to obtain for every pair of tokens a high scalar product and therefore a conditional probability close to 1 The negative sampling term is:

$$\text{argmax}_{(U,V)} \textstyle\sum_{(w_i,w_j)\in D} \left(log(\sigma(u_i^\top v_j)) + \sum_k E_{w_{j'}\sim q(w_{j'})} \left[log(\sigma(-u_i^\top v_{j'})) \right] \right)$$

It is thus a question of maximizing $p(w_j|w_i; U, V)$ for the pairs of tokens taken from the corpus (the "positive" pairs), and to maximize $1 - p(w_j|w_i; U, V)$ for "negative" pairs of tokens, where the context token is drawn randomly according to $q(w_{j'})\alpha f(w_{j'})^{\frac{3}{4}}$ (that is, the frequency of occurrence in C, smoothed). The value of k, a hyper-parameter, controls the ratio between the number of positive pairs and the number of negative pairs. In practice, we choose k between 1 and 15.

The problem can be formulated as a problem of binary supervised classification, where it is a question of distinguishing pairs of positive tokens from pairs of negative tokens. Let the data set be labeled D', composed of triplets (w_i, w_j, γ_{ij}), where $\gamma_{ij} \in \{-1, +1\}$ indicates whether the token pair is positive or negative. We build D' according to the following procedure. For each pair $(w_i, w_j) \in D$, we add to D' the

triplet $(w_i, w_j, 1)$ and we add k triples $(w_i, w_{j'}, -1)$, the $w_{j'}$ being drawn randomly according to q. Finally, the log-likelihood of this dataset is expressed as :

$$argmax_{(U,V)} \sum_{(w_i,w_j)\in D'} log(\sigma(\gamma_{ij} u_i^\top v_j))$$

Since the function to be maximized is defined by a sum, the stochastic gradient method is applied.

5.4.5 Training phase

In the learning phase, the deep neural networks are trained on tokens that were generated from the previous phase. The goal of this phase is to learn two models, one to recommend deployment descriptors given an initial description, the second one to complete an initial description with addiction information. We use CNN to learn the recommendation model and LSTM to learn the completion model (CNN and LSTM are described in ..)

5.4.5.1 Recommendation model for VNFDs

CNN is trained on a set of tokens extracted in the preprocessing phase. The learned model takes as input a partial deployment description, a partial VNFD in this case, and outputs an entire description file. The aim is to recommend a file that is close to the context of the initial description. We chose CNN for this task because of its ability to learn in an unsupervised manner the features of VNFDs.

The recommendation can be considered as a classification problem. This is because of the recommendation model chooses one outcome from a set of possible solutions. We choose to investigate two applications. In one application, the CNN model recommends a VNFD file directly as output. The VNFDs, in this case, could be segmented beforehand, such that the model learns to suggest a block of description that could be tuned or combined with other blocks by the service providers. In this application, the CNN classifies the input (set of tokens) to one of the files already existing in the VNFD catalog. Each VNFD is considered as a class. This application can be useful to identify which file is closely similar to an initial VNFD description independently of the performance goal or the type of the VNFD. On the other hand, the main shortcoming of this application is that the recommendation model needs to constantly relearn from the beginning at each time a VNFD is added or deleted from the repository.

In the second application, the recommendation model outputs a category of VNFDs. For example, this could the type of the VNFD, the performance goal that the VNFDs are satisfying, a VNFD of a NS, etc. It is up to the service provider or the domain expert to choose the criteria to which the VNFDs are grouped. This way, this application can help the orchestrator or the service provider to get an already existing

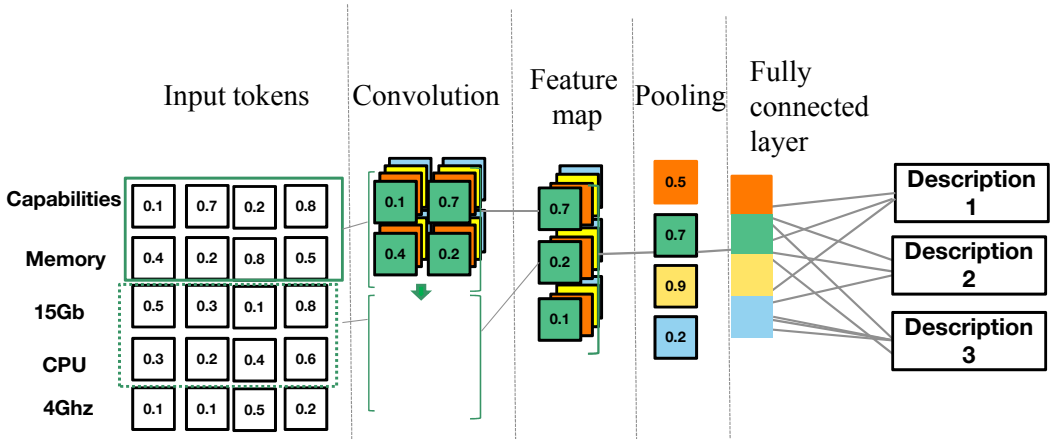


Figure 5.15: Convolutional Neural Network Architecture for learning a VNFDs recommendation model

category of VNFD to which an initial description is satisfying. The shortcoming of this application is that the VNFDs are usually defined without a formal strategy, it is, therefore, probable that the VNFDs of a category are very different from each other. It is challenging in that way to a good recommendation.

The two recommendation applications are following the same approach of learning. An overview of the CNN architecture for learning a recommendation model from VNFDs is depicted in Figure 5.15. The tokens are converted to their vector representation based on the look-up table, following one of the two vectorization methods: Appearance-based vectorization or token embeddings.

The vectors of the preparation phase are passed to the convolution layer to learn features from the descriptor files using multiple filter sizes. The most obvious features are selected in the pooling layer and concatenated into one neural vector. This neural vector is used in the fully connected layer to perform classification. We can summarize the CNN method into three processes: Token representation, feature extraction, and recommendation.

Token Representation

The input of the model is a fixed size vector of tokens $w = \{w_1, w_2, ..., w_D\}$, Where D is the input size. The tokens are initially converted to their numerical representation using the generated look-up table in the network data preparation phase. Thus, the original input is $x = \{x_1, x_2, ..., x_D\}$, where x_i is a vector representation of w_i. The model is iteratively trained with D sequence of tokens from the tokenized files. If a sequence is less than D, then it will be completed with empty characters.

Feature Extraction

This process aims to learn features from the descriptors and merge the best found into a single vector. This vector is passed afterward to the classification process. Features learning and extraction are accomplished by filters in the convolutional layer.

The input x is convoluted (dot multiplication operation) with multiple filters with different sizes to produce a feature score s_i. A filter is a matrix $W \times R$, such as W is the filter window size and R is the size of the token vector. $R = 1$ in the case of the highest appearance method. Each filter f_i is applied to multiple regions of the input and generates a list of feature scores. Each feature score s_j in the filter list is defined as:

$$s_j = ReLu(x_{j:j+W-1} \cdot f_i + b_s) \tag{5.7}$$

where $x_{j:j+W-1}$ is the region of the input to which the filter is applied, b_s is a bias, the operator "$\cdot$" denotes the convolution operation and $ReLU$ is a non linear function, $ReLu(x) = max(0, x)$. The training of filters is equivalent to learning a feature for the task of classifying the descriptor files.

The features that are extracted with each filter are assembled to form a feature map. After generating the feature maps for all the filters, a max-pooling operation is applied over each feature map to highlight the best resultant feature that is captured by each filter. This operation p_f consists of selecting a feature with maximum value from the features map.

$$p_f = max(s) = max(s_1, s_2, s_3, ..s_k) \tag{5.8}$$

The best features of each feature map are merged into a penultimate layer and are passed to a fully connected layer whose output is the probability distribution over all the classes (i.e.deployment descriptor files).

Recommendation

The fully connected layer takes the vector that contains all the max pooled features from all the filters to perform classification. We use that a Softmax function to make the classification. The output is $y \in \{D_1, D_2, ..., D_Y\}$, it could be pointer to description files, i.e. VNFDs or a category of VNFD, depending on the application type. Y could the number of files that are in the catalog or the maximum categories to be used for classification.

$$y = softmax(\gamma_i) == \frac{e(\gamma_i)}{\sum_{i'=1}^{Y} e(\gamma_{i'})} \tag{5.9}$$

where $\gamma_i = p(i|x, \theta)$, i is the number of the class $i \in \{1, 2, 3..Y\}$ and θ is the parameters of the model, including the weights and bias of the fully connected layer. The output is the class with the highest probability.

The parameters of the model are afterward updated using the random gradient descent method to maximize the optimization objective function $J(\theta)$

$$J(\theta) = \sum_{i'=1}^{Y} log\, p(\gamma^{(i)}|x^{(i)}, \theta) \tag{5.10}$$

5.4.6 Completion model for VNFDs

LSTM is used to learn a language model from the descriptor files so that it can be able to predict a sequence of tokens given an input description. As described previously (see section 5.2), LSTM is a recurrent neural network with a special type of memory cell that stores the context of the information. It can be seen as multiple copies of the same network, where each cell passes the information to its successor enabling the persistence of the information.

The model learns from a vectorized tokens of the preparation phase using multiple sequences extracted from the VNFDs. Similarly, to CNN, the first step of the model is to convert the tokens into their numerical representation using the lookup table in the first phase of the framework. LSTM is trained by considering at each iteration a fixed size sequence of tokens. The parameters are tuned during the learning to enable the model to predict with high accuracy the next token given a same sequence size of tokens. The model can be expressed as the following:

$$x(t+1) = F(x(t), x(t-1), ..., x(t-s)) \tag{5.11}$$

where t is the position of the token, $x(t+1)$ is the predicted token given a s sequence of tokens. The prediction is effectuated using :

$$y = softmax(w_i) = \frac{e(w_i)}{\sum_{i'=1}^{V} e(w_{i'})} \tag{5.12}$$

where w_i is a token from the vocabulary and V is the vocabulary size The learning is done by minimizing the log-loss function $J(\theta)$ using a simple Stochastic Gradient Descent (SGD), with respect to the parameters of the LSTM model (θ):

$$J(\theta)) = -log\, p(x_1) - \sum_{t=2}^{k} log\, p(x_t|x_{1:t-1}) \tag{5.13}$$

5.4.7 Execution phase

During this phase, the DNN models are used. To illustrate their usage, we consider example scenarios during design time and run time of an orchestrator. The design time is where the network and service descriptors are to be set and defined towards their deployment by the orchestrator. The Run time is where services and Network functions are already deployed and a change is needed to be done by the orchestrator

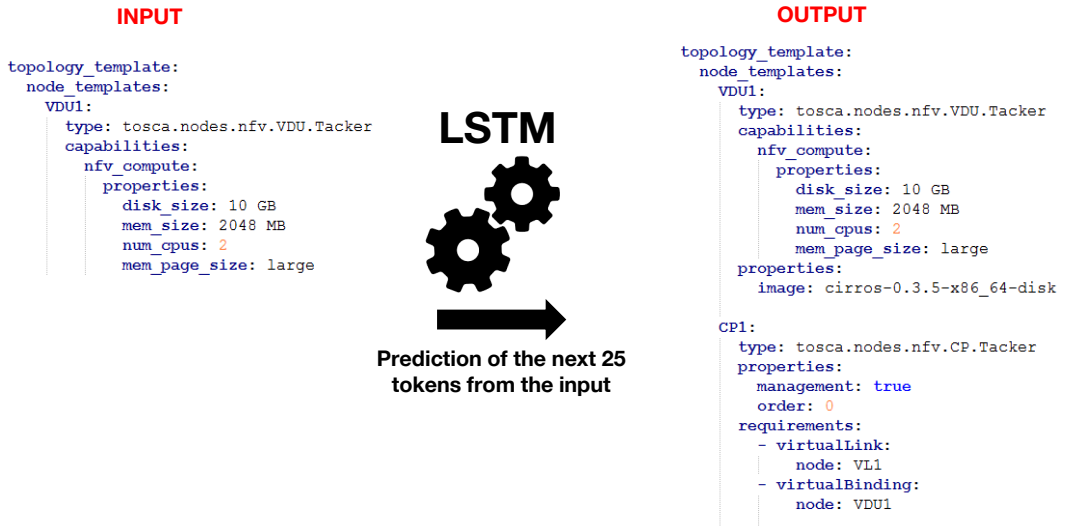


Figure 5.16: Using the LSTM method for deployment description completion

in response to faulty situations (Noisy VNFs, overloaded VNFs, etc.). We describe here below one scenario for the Design Time and one Scenario for the Run time:

Design Time: VNFD verification and Completion

In this scenario, the resource orchestrator needs to complete a given VNFD with additional information, like policy constraints, forwarding graph, etc. In this case, the DNN modules receive the incomplete VNFDs, compute the missing part with LSTM as illustrated in Fig. 5. The resource orchestrator then stores the newly completed VNFD in the catalog. In this scenario, the LSTM was used to complete the VNFD. Another variation would be to consider the case where there is a need to change completely several parts of the VNFD, here the LSTM is not enough as the sequential aspect is not sufficient to complete the whole VNFD. Hence here, CNN is invoked to compute the context of the remaining part within the VNFD and propose similar VNFD to complete it as shown in Fig.6.

Run time scenario: VNFD genration

The orchestrator is invoked to change the parameters within the Descriptors to handle faulty situations like resource conflict between virtual machines (e.g. Overloaded VNF). In this case, the orchestrator needs to change the VNFD and adapt it to a state of no conflict. To solve the problem, the orchestrator could specify a descriptor for the VNFs with resources and requirements that could potentially end the conflict. The proposed framework could be used to recommend deployment descriptors that satisfy the orchestrator demand.

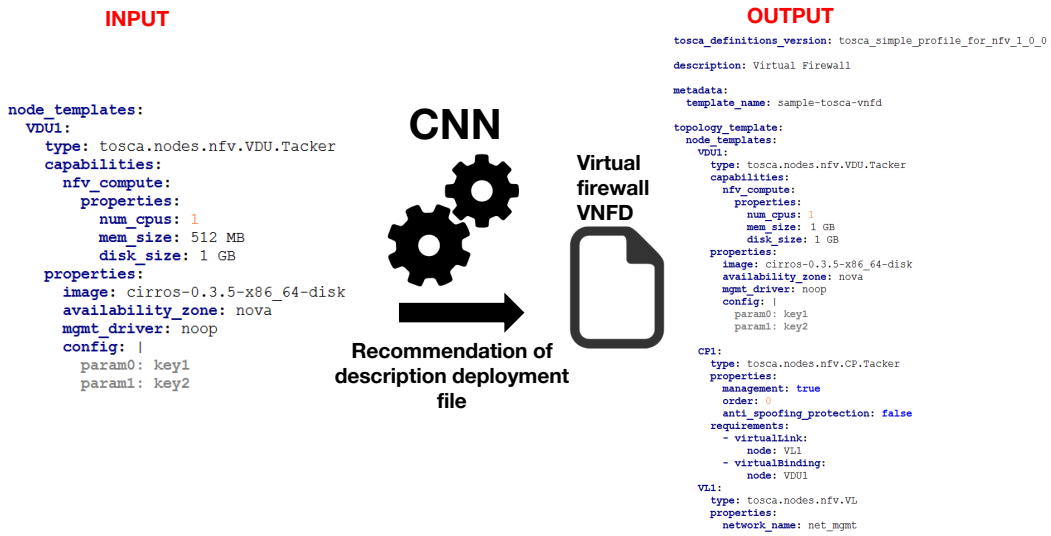


Figure 5.17: Using the CNN method for deployment description recommendation

5.5 Evaluation and Results

In this section, we evaluate the performance of our proposed deep learning framework. We focus more particularly our attention on the effectiveness of the two models: the recommendation model and the completion model. The recommendation model is evaluated based on the accuracy, precision, and recall of the recommendation. We evaluate two applications of the recommendation: recommendation of a VNFD file and the recommendation of a category of VNFD file.

In the case of the recommendation of VNFD files, CNN is trained on a set of VNFD blocks from the same VNFD file. The blocks are constructed randomly from different parts of the file to avoid having only a sequence of data in the same block. This way, CNN is trained to recognize patterns from the VNFD instead of only segments. The blocks are afterward divided into three sets, training, validation, and test sets. The last two sets are completely independent of the training set.

The recommendation of VNFD files model is initially trained on the training set to fit its parameters (e.g. weights of the CNN). The validation set is then used to tune the parameters of the model and to stop the training phase when for example there is an overfitting i.e. the classification error keeps increasing. Finally, the testing set is used to provide an unbiased evaluation of a final model fit on the training dataset.

In the case of recommendation on VNFD categories model, the evaluation is also carried on the three sets of training, validation, and testing. Nevertheless, the sets, in this case, are constituted by VNFDs grouped by categories.

The completion model is evaluated based on the accuracy of completing a given initial description. The LSTM is trained on sequences of tokens with the goal of

Table 5.1: Parameters of the CNN model

Parameters	Value
Train/Test %	80/20 %
Learning rate	0.001
Filter number	150
Batch size (tokens)	500

predicting the next token.

To test the effectiveness of the preparation phase methods, we compare the two methods, the appearance-based indexation method and token embeddings methods in two different scenarios. In the first scenario, the methods are applied to all the sets (training, validation, and testing), and in the second scenario, the methods are applied in the learning phase. This way, we can evaluate the impact of the preparation phase on the performance results.

We start by describing the data set and the experimental setup that we used to train the deep neural network models and then discuss the results of the experiments for each model.

5.5.1 Data set and experimental Setup

The experiments are performed on a data set constituted of a catalog of different VNFDs containing up to 500 VNFDs, defined in YAML files, and following the TOSCA for NFV template [2]. The resulting code corpus includes more than 1,500,000 tokens that generate a vocabulary of 95,000 unique tokens. We use %80 of the data set for training and 10% for validation and 10% for testing.

The skip-gram model is used to learn the token embedding the embedding in the preparation to learn vector representation for the tokens. The vector dimension is set to 128.

All the models are implemented in python using Tensorflow, a widely used open-source library for numerical computation and large-scale machine learning. The experiments are tested in a machine equipped with an Intel i7 CPU and 8 GB of RAM.

5.5.2 CNN for VNFD recommendation

We evaluate the performance of the CNN model based on the accuracy and the F measure. The latter is a harmonic mean of precision and recall. The evaluation is measured by varying two parameters, the input size of CNN and the number of filters. The input size indicates how much information CNN needs to take into consideration in order to be effective for making recommendations. The number of filters is important for CNN for learning the features.

Table 5.2: Evaluation of the CNN recommendation model for VNFD files with respect to the CNN input size

Vecotrization	Input size (Tokens)	Training_acc	Testing_acc	F measure
TE	20	0.20	0.24	0.54
TE	50	0.26	0.31	0.57
TE	100	0.42	0.44	0.51
TE	200	0.56	0.6	0.58
TE	500	0.78	0.84	0.67
ABI	20	0.24	0.21	0.39
ABI	50	0.25	0.29	0.41
ABI	100	0.29	0.39	0.44
ABI	200	0.38	0.52	0.43
ABI	500	0.41	0.69	0.48

Table 5.3: Evaluation of the CNN recommendation model for VNFD files with respect to the CNN number of filters

Vecotrization	Number of filters	Training_acc	Testing_acc	F measure
TE	50	0.51	0.6	0.51
TE	75	0.6	0.68	0.56
TE	100	0.65	0.71	0.59
TE	125	0.82	0.79	0.61
TE	150	0.7	0.84	0.67
ABI	50	0.45	0.54	0.34
ABI	75	0.51	0.59	0.31
ABI	100	0.55	0.62	0.39
ABI	125	0.52	0.65	0.4
ABI	150	0.53	0.69	0.48

Table 5.4: Evaluation of the CNN recommendation model for VNFD categories with respect to the CNN input size

Vecotrization	Input size (Tokens)	Training_acc	Testing_acc	F measure
TE	20	0.26	0.23	0.48
TE	50	0.32	0.46	0.54
TE	100	0.41	0.51	0.55
TE	200	0.57	0.62	0.61
TE	500	0.63	0.69	0.61
ABI	20	0.23	0.16	0.39
ABI	50	0.21	0.22	0.38
ABI	100	0.29	0.31	0.39
ABI	200	0.33	0.39	0.41
ABI	500	0.42	0.43	0.41

Table 5.5: Evaluation of the CNN recommendation categories for VNFD files with respect to the CNN number of filters

Vecotrization	Number of filters	Training_acc	Testing_acc	F measure
TE	50	0.48	0.5	0.44
TE	75	0.54	0.62	0.59
TE	100	0.55	0.65	0.61
TE	125	0.58	0.66	0.61
TE	150	0.63	0.69	0.61
ABI	50	0.31	0.34	0.31
ABI	75	0.33	0.36	0.32
ABI	100	0.39	0.40	0.39
ABI	125	0.41	0.40	0.39
ABI	150	0.42	0.43	0.41

Table 5.6: Parameters of the LSTM model

Parameters	Value
Train/Test %	80/20 %
Learning rate	0.001
sequence length	150
Batch size (tokens)	500

As mentioned previously, we consider in the evaluation of the two recommendation applications, recommendation model for VNFD files and recommendation model for VNFD categories. We also take into consideration the two methods of the preparation phase, token embedding (TS) and appearance-based indexation (ABI). The parameters that we considered for training the model are illustrated in Table 5.1. The performance results are indicated in Table 5.2, 5.3, 5.4 and 5.5.

From Table 5.2 and 5.4, we can notice that the input size of the description has an impact on the classification. That makes sense, because the more we add information about the description, the more the CNN can extract features and be able to classify it to the best descriptor file. We can also notice that the token embedding method gives a more accurate classification than the appearance-based indexation. This is due to the fact that embeddings enable CNN to choose the nearest possible descriptor given an input because of the vector representation that encodes the semantic of the tokens.

The performance results of CNN in the function of the number of filters for both recommendation applications are illustrated in Table 5.3 and 5.5. Each filter has a window size that is generated randomly following a uniform distribution between $[50, 500]$. We can notice that filters with multiple window sizes can capture the code features and achieve better classification results.

Overall, the performance of the recommendation model for VNFD files is better in our experiments than the recommendation model for VNFD categories. We can explain this observation by the fact that VNFDs are designed without a formal strategy and that different VNFD could be designed for the same performance goal. Therefore, the VNFDs that compose a given VNFD category could be very different and makes the extraction of features that could characterize it difficult.

5.5.3 LSTM for VNFD completion

The LSTM model settings are shown in Table 5.6 We evaluate here the accuracy of the model in function of the sequence length. The results in Table 5 show that LSTM performs better when it is trained on a longer sequence of data. That is actually one of the abilities of LSTM, to handle a long sequence of data and capture more information of the context to predict more accurately the next token.

We measure also the performance of LSTM for generating a sequence of tokens by

Table 5.7: Experiments on the LSTM model using different input sequence length

Sequence length	TE	ABI
20	0.4	0.39
35	0.46	0.41
50	0.51	0.49
65	0.71	0.61
80	0.79	0.68

Table 5.8: Experiments on the LSTM model using different output length

Output length	TE	ABI
5	0.87	0.79
10	0.82	0.71
30	0.78	0.68
70	0.6	0.42
180	0.43	0.21

Table 5.9: Experiments on the CNN model combined with LSTM using different input sizes

input size	TE	ABI
20	0.62	0.56
50	0.67	0.6
100	0.67	0.64
200	0.75	0.64
500	0.84	0.69

variating the length of the output. We fix here the input description to 500 tokens. Table 5.8 shows that the model is able to predict with high accuracy the sequence of tokens at the beginning of the prediction, then the performance decreases with the sequence size.

Combining LSTM with CNN can actually enhance the classification accuracy by completing the input size of CNN using the LSTM model. The results in Table 5.9 can demonstrate this performance.

5.6 Conclusion

To enable auto-configuration in software networks, network components are expected to automatically adapt their own configuration to unexpected situations to preserve the required network performance. For that matter, the configuration is expected to be generated automatically by management entities in software networks. We tackled in this chapter the second and third research question specified in chapter 1: How to automatically generate configurations to a targeted data model? How to automatically generate configurations to a targeted data model?

To address these challenges, we focused our attention on a use case in NFV. Service providers in NFV have to associate with VNFs deployment descriptors. The deployment descriptors are complex and tedious to design manually or with static automation, they contain numerous components that are dependent on each other. Also, there is no formal strategy to assist the creation of the deployment descriptors except the experience and best practices of the domain experts. This leads to a plethora of strategies for deployments.

We proposed in this chapter a framework based on deep neural networks that learn from a set of deployment descriptors (VNFDs), two models. A model that can recommend a VNFD given an initial description and a model that can complete VNFDs with additional information given an initial description. This could be used to augment an orchestrator with an ability to select, recommend, and complete NFV descriptors from an initial description. The promising results from our experiments suggest that the framework is a solution that could be used in practical scenarios to enhance the dynamicity of deploying VNFs. Our next step involves developing a descriptor generation engine that could be used to suggest a new description that fits the most initial requirements.

The performance of the recommendation shows also that it is hard to learn from VNFDs when they do not share common characteristics, this is the case for the recommendation of VNFD categories. To overcome this problem, we propose in the next chapter an alternative solution for learning from configuration files. This solution is model-driven, it aims at capturing all the configuration variations into a single representation that could be used for example to mine the dependency between the configuration elements and extract more implicitly the features from the configuration files.

The work in this chapter is published in C2, C3 and C5 (see chapter 1, section 1.4).

Chapter 6

Model-driven approach for configuration generation

Contents

6.1 Introduction

The design of configuration files in software networks usually follows a traditional process design where a service provider starts the design and continuously adds elements. This is known as a bottom-up design approach. Differently, in a top-down design approach, instead of starting from scratch, the design process starts with the "big picture" which is then reduced to the relevant parts in order to obtain the required solution. Such a big picture represents a reference configuration model which can be considered as an off-the-shelf solution that requires configuration before it can be used in a specific context. Following a top-down approach for creating configuration models allows to accelerate the design process by providing "plug and play" models that can be easily configured by service providers. It also improves the understandability of services' and functions' configuration by providing a generic solution that captures best practices in creating deployment descriptors in a specific domain.

In order to enable a top-down design approach, a language that supports the creation of customizable and adaptable configuration models is required. This chapter contributes to this area by focusing on VNFDs and proposing a configurable deployment descriptor model that allows to represent multiple existing deployment descriptor models into one customizable model. We focus our attention on the virtual network function deployment descriptor models (VNFDs) as they are the main component of NFV that compose each network service. Using configurable elements, the proposed configurable model allows representing the commonalities and differences between the consolidated VNFDs. The configurable elements need to be configured by the service provider in order to derive a VNFD. In this way, a configurable VNFD model allows to represent multiple alternatives in designing VNFDs for a specific network service. The configurable elements allow to explicitly show the commonalities and differences among different design decisions. It is noteworthy that the scope of the present contribution is to introduce a configurable modeling language of a VNFD. The construction of a configurable VNFD out of a collection of VNFDs is out of scope and is left for future work.

Concretely, we propose a graph-based representation of a VNFD model which is based on the ETSI NFV specification [38]. Therefore, our graph representation is generic and abstracts from the specific details of existing modeling languages (e.g.

TOSCA [2]). We then extend the proposed graph model with configurable elements and define their configuration constraints. The configuration constraints ensure a structurally correct configuration of the model.

Learning the configurable VNFD model is a challenging task. That is to find and group together similar elements in the set of VNFD files. It is challenging because the VNFDs could be designed by different service providers or domain experts and therefore, similar VNFD elements with similar functionalities could be labeled or structured differently. The configuration could also differ between the elements. To overcome this problem, we propose an algorithm based on machine learning to search and cluster similar VNFD elements based on the distance between the features of the elements. We define what are the features that characterize the elements and also the metrics that that measure the similarity between the elements.

The remainder of this chapter is organized as follows: Section 6.2 defines a representation for the VNF deployment descriptors (VNFDs). Section 6.3 presents our proposed configurable descriptor model. Section 6.4 describe our machine learning based approach to learn automatically the configurable model. Section 6.5 presents how can the configurable model be used to assist the desgin of VNFDs. Section 6.6 describes other applications of the configurable model. Section 6.6 presents the results of our experiments on our proposed solutions and section 6.8 concludes the chapter.

6.2 VNFD representation

As the structure of a VNFD can be mapped to a graph, we choose graph theory to represent a VNFD model as a tree-like structure. For sake of simplicity, we restrict our definition to the general elements that compose a VNFD regardless of any data model. We consider for that matter the ETSI NFV specification on the VNFD information model [38] (see chapter 5, section 5.3). We are considering the flowing VNFD elements under this specification: VNF Component (VNFC), Virtual deployment unit (VDU), Virtual storage resource (VS), Virtual memory resources(VM), Virtual Compute resources (VC), and connection point descriptors (CPD).

Figures. 6.2 and 6.3 illustrate two basic examples of a VNFD representation of a firewall and CPE respectively. Figure 6.1(a) shows the basic elements that compose the VNFD model representation. The nodes in the VNFD model represent the different component instances that compose the VNFD. Each component instance has one of the following types: *VNFC, VDU, VM, VS, VC, or CPD* and may have one or more attributes. The root node represents an instance of the type VNFD. In the example in Figure 6.2, the root node v_1 is an instance of type VNFD and has the attribute "name" which has the value "vFirewall". Graphically, we use ":" in the node label to indicate the component from which the instance is derived. For sake of simplicity, in our example, we define one attribute for each node and we indicate it between two parentheses"()" in the node.

The relations between the nodes can be one of the following three types: *compo-*

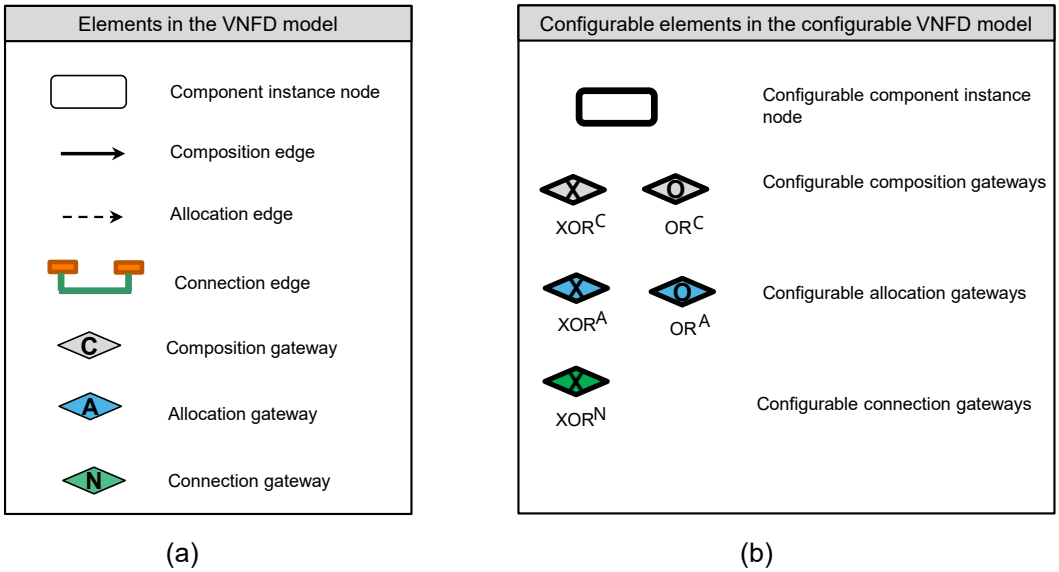


Figure 6.1: (a) graphical representation of the elements used in a VNFD model and (b) graphical representation of the configurable elements in a configurable VNFD model

sition, allocation or connection. A composition relation between two nodes v_1 and v_2 indicates that the instance v_1 is composed by the instance v_2. That means that v_2 is part of v_1 and cannot exist without it. The composition relation exists typically between (i) VNFDs and VNFs, (ii) VNFCs and VDUs, and (iii) VDUs and CPDs. Graphically, a 1:1 composition relation is represented by a solid line. For example, in Figure 6.2, instance v_4 has a single composition relation, to v_7. In case a 1:n composition relation exists between multiple nodes, we use a *composition gateway* to connect the nodes. Graphically, the composition gateway is represented by a grey diamond with the label "C". In our example, the instance of VNFC v_2 is composed of two instances: VDU v_3 and VDU v_4.

An allocation relation between two nodes v_1 and v_2 indicates that v_2 is a resource that needs to be allocated to v_1. The allocation relation can only exist between VDUs on the one hand and VS, VC, and VM on the other hand, i.e. only the virtual storage, virtual memory, and virtual component resources can be allocated to the VDUs. Graphically, the allocation relation is represented by a dashed line. As an example, in Figure 6.2, the VS v_5 is allocated to the VDU v_3. In case multiple resources are allocated to a VDU (i.e. 1:n relation), we use an *allocation gateway* to connect the nodes. Graphically, the gateway is represented by a blue diamond with the label "A". In our example, the VC v_8 and the VM v_9 are allocated to the VDU v_4.

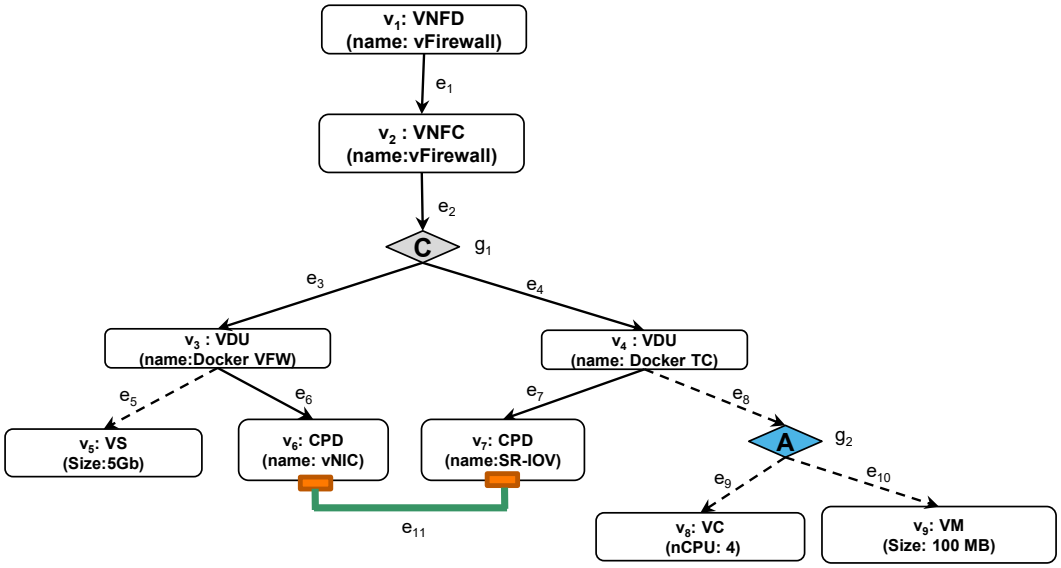


Figure 6.2: An example of a vFireWall VNFD represented graphically as a tree-like structure

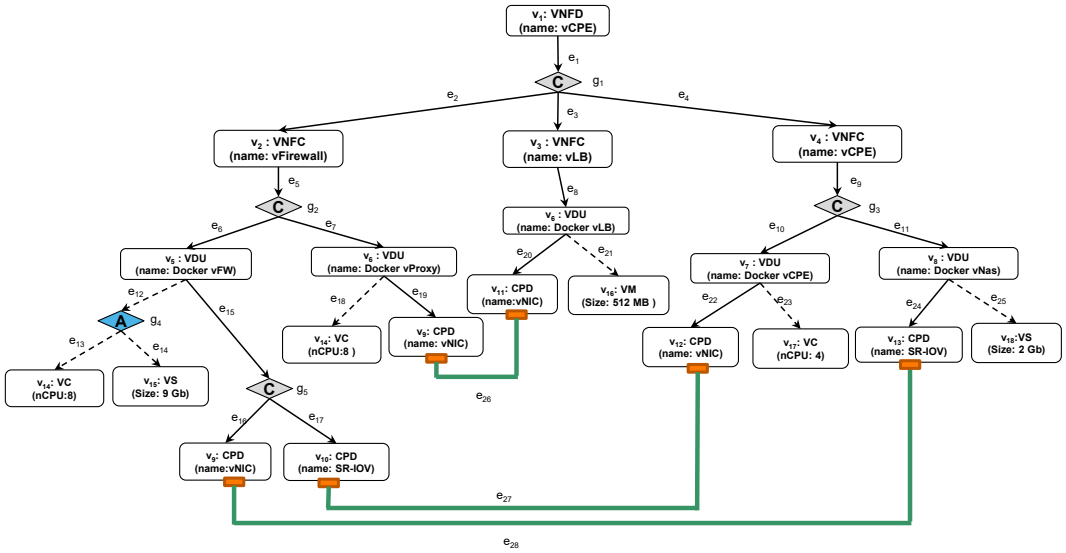


Figure 6.3: An example of a graphical representation of a vCPE VNFD that is described briefly in Figure5.7

A connection relation can exist between the component instances of type CPD

and is a bidirectional relation. It represents the virtual links that connect the CPDs of the internal VDUs together or to external VDUs of external VNFs. Graphically, a connection relation is represented by a solid green line. As an example, in Figure 6.2, the relation between the CPD v_6 and the CPD v_7 indicates that a virtual link connects the two connection points. We also introduce the *connection gateway* (as shown in Figure 6.1) that allows to connect multiple connection points together. The connection gateway is graphically represented by a green diamond with the label "N". Currently, the VNFD connection points can have only 1:1 relation. It is therefore improbable to have such a gateway. However, we will keep this gateway in our formal definition as it will be used later in the configurable descriptor model. Moreover, according to the ETSI NFV specification, a virtual link can have different attributes such as add some examples. Therefore, in our definition, we allow relations to have attributes, although this is not graphically shown in our example.

6.2.1 Formal Definition of a VNFD

In this section, we formalize the definition of a VNFD model as explained in the previous section. Let $\mathcal{U}_A$ be the universe of attributes' names and $\mathcal{U}_V$ be the universe of attributes' values. The definition of a VNFD model is given in Definition 6.2.1.

Definition 6.2.1 (VNFD model)**.** *A VNFD model is a tuple* $D = (V, \hat{V}, G, \hat{G}, E, \hat{E}, \lambda)$ *where:*

- $V = \{v_1, v_2, \ldots, v_n\}$ *is the set of nodes;*
- $\hat{V} : V \to \{VNFD, VNFC, VDU, VS, VM, VC, CPD\}$ *is a function that assigns a component type to a node* $v \in V$*;*
- G *is the set of gateways;*
- $\hat{G} : G \to \{C, A, N\}$ *is a function that assigns a type to a gateway* $g \in G$ *such that:*
  - $\hat{G}(g) = C$ *indicates a composition gateway,*
  - $\hat{G}(g) = A$ *indicates an allocation gateway and*
  - $\hat{G}(g) = N$ *indicates a connection gateway.*
- $E \subset (V \cup G) \times (V \cup G)$ *is the set of edges;*
- $\hat{E} : E \to \{composition, allocation, connection\}$ *is a function that assigns a type to an edge* $e \in E$*;*
- $\lambda : V \cup E \to \mathcal{P}(\mathcal{U}_A \times \mathcal{U}_V)$ *is a function that assigns for either a node* $v \in V$ *or an edge* $e \in E$ *a set of attributes and their corresponding values, i.e. for an element* $m \in V \cup E$, $\lambda(m) = \{(attr_i, value_i) \mid i \geq 1\} \in \mathcal{P}(\mathcal{U}_V \times \mathcal{U}_A)$*. We use the shorthand* $\lambda_{attr}(m) = value$ *to refer to the pair* $(attr, value) \in \lambda(m)$*.*

In our example in Figure 6.2, the definition of the VNFD model is as following: $V = \{v_1, \ldots, v_9\}$; $\hat{V}(v_1) = \mathit{VNFD}$, $\hat{V}(v_2) = \mathit{VNFC}$ and so on; $G = \{g_1, g_2\}$; $\hat{G}(g_1) = C$, $\hat{G}(g_2) = A$; $E = \{e_1, \ldots, e_{11}\}$; $\hat{E}(e_1) = \mathit{composition}$, $\hat{E}(e_5) = \mathit{allocation}$ and so on; $\lambda_{name}(v_1) = \mathit{vFirewall}$, $\lambda_{Size}(v_5) = 5\,GB$ and so on.

6.3 Configurable Deployment Descriptor Model

This section introduces the notion of a configurable VNFD model. The configurable representation of a VNFD aims to capture different predefined VNFD models into one customizable model. It is often the case that the VNFD components are repeatedly used with similar settings to deploy different VNFs with different aims. As an example, the VNFDs in Figure 6.2 and 6.3 both have a vFirewall VNFC, which are deployed on a VDU Docker vFirewall. However, the VDU in Figure 6.2 requires a VS of size 5GB while the VDU in Figure 6.3 requires a VS of size 9GB. In addition, this latter VDU requires a VC with 8 CPUs while the former does not. This example clearly shows that different VNFDs may share many commonalities while at the same time have some differences. Therefore, a configurable VNFD does not only allow to represent the commonalities and the differences into one model but can also explicitly depict the variations using *configurable elements*. These elements can be easily configured by a service provider to derive a VNFD model, referred to as VNFD variant.

In a configurable VNFD, the component instances and gateways can be configurable. A configurable element is graphically represented with a thick line. In addition to the elements shown in Figure 6.1(a), Figure 6.1(b) shows the configurable elements that can exist in a configurable VNFD. Figure 6.4 illustrates an example of a configurable VNFD that represents a consolidation of the VNFDs in Figure6.2 and 6.3. In the following, we explain in more details the configurable component instances (Section 6.3.1) and the configurable gateways (Section 6.3.2). Afterward, we formally define a configurable VNFD model (Section 6.3.3).

6.3.1 Configurable component instances

A configurable component instance can be configured in two ways. First, it can be configured to *ON* (i.e. the component instance is included in the VNFD variant that is derived from the configurable model) or *OFF* (i.e. the component instance is excluded from the VNFD variant). A component instance configured to OFF results in the deactivation of all the component instances that are linked to it. For example, in Figure 6.4, the component instances v_3 and v_4 are configurable. In case v_3 is configured to OFF, all the component instances v_3, v_8, v_{11} and v_{18} will be deactivated and later removed from the resulting VNFD variant.

Second, a configurable component instance can have configurable attributes. A configurable attribute is an attribute whose value can be adapted. Configurable

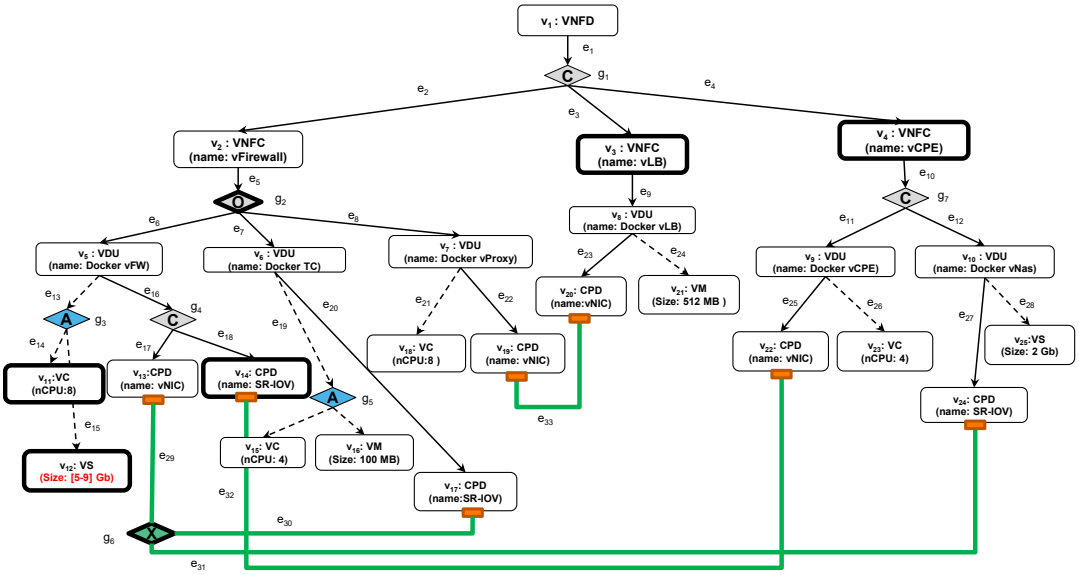


Figure 6.4: An example of a configurable deployment descriptor model that combines the VNFDs in Figs. 6.2 and 6.3

attributes are graphically represented in red color. In our example in Figure 6.4, the VS v_{16} has a configurable attribute "size" whose value is in the range $[5-9]GB$. This means that, during configuration, the size of the VS can be set to any number between $5GB$ and $9GB$. However, not all attributes can be configurable. In fact, some attributes such as the identifier of a component instance are considered as a characteristic that describes the component and therefore cannot be configurable. In our proposed model, we consider the attribute "name" as a characteristic of a component instance.

Table 6.1 shows the configurable elements and their possible configurations. As mentioned in the table, a configurable attribute value can be either expressed as a *range* (for numerical values) or as a *discrete set* (for numerical or textual values). A range describes the possible values that the configurable attribute could have. It is an interval of the minimal and maximal values that are recommended to be set. It can be configured to a single numerical value that exists in it. A discrete set is used mainly with textual values to describe the possible finite set of values an attribute can have. It can be configured to a single value that is picked from the set. For example, in Figure 6.4, the range is used for the configurable attribute "size" of the VS v_{12}. The discrete set can be used for instance if multiple images could be used to deploy a VDU. In this case, the attribute "image" of the VDU is configurable and its value can be defined as a discrete set that includes the different images.

Table 6.1: Configurable elements and their possible configurations

Configurable element			Configuration
Component instance			ON/OFF
	attributes' values	range	numerical value
		discrete set	numerical or textual value
Composition gateway		XOR	composition edge
		OR	omposition edge/gateway
Allocation gateway		XOR	allocation edge
		OR	allocation edge/gateway
Connection gateway		XOR	connection edge

6.3.2 Configurable gateways

The composition, allocation, and connection gateways in the VNFD model allow to model 1:n relations between the component instances. In a configurable model, a configurable gateway allows modeling a variable number of relations. That is, some of the gateway's relations can be deactivated and therefore excluded from the derived VNFD variant. In order to guide the configuration of a gateway, we introduce two operators, *OR* and *XOR*, that define the behavior of a configurable gateway and constrain its possible configurations. Table 6.1 shows the configurable gateways and their possible configurations.

A configurable OR composition gateway that connects a component instance v_1 to a set of component instances $v_2, \ldots, v_n$ indicates that v_1 can be composed of any combination of the component instances in the set $\{v_2, \ldots, v_n\}$. In case a 1:1 composition relation is selected, we say that the OR composition gateway is configured to a composition edge. In case a 1:n composition relation is selected, we say that the OR composition gateway is configured to a composition gateway. Graphically, a configurable OR composition gateway is represented with a thick grey diamond and a circle inside. For example, in Figure 6.4, g_1 is a configurable OR composition gateway. Therefore, the VNFC vFirewall can be composed of any combination of the three VDUs: Docker vFW, Docker TC and Docker vProxy. In case Docker vFW and Docker TC are selected, Docker vProxy with its allocated resources and CPDs are excluded and the configuration of the gateway becomes a composition gateway that connects vFirewall to Docker vFW and Docker TC.

A configurable XOR composition gateway indicates that at configuration time, a component instance can be composed of one and only one of the connected component instances. Therefore, the configuration of an XOR composition gateway is always a composition edge. Graphically, a configurable XOR composition gateway is represented with a thick grey diamond and a cross inside.

The behavior of the configurable allocation and connection gateways is similar to that of the composition gateway. Exceptionally, a configurable connection gate-

way does not have an OR operator since the connection relation is always a 1:1 relation. Graphically, the configurable allocation and connection gateways are represented similarly to the configurable composition gateway but with blue and green colors respectively.

In terms of behavior, the behavior of the XOR operator is included in that of the OR operator since an OR operator can be also configured to a 1:1 relation. However, we included the XOR operator to add more expressiveness to our configurable model. For example, the XOR operator in Figure6.3 can be replaced with an OR operator without changing the correctness of the representation.

6.3.3 Formal definition of a configurable VNFD

In this section, we provide a formal definition of a configurable VNFD. let $\mathcal{U}_A$ and $\mathcal{U}_V$ be the universes of attributes' names and values; $\mathcal{U}_V^c$ is the universe of configurable attributes' values which are defined in terms of range and discrete set as shown in Table 6.1.

Definition 6.3.1 (Configurable VNFD)**.** *A configurable VNFD is a tuple* $D^c = (V, \hat{V}, G, \hat{G}, E, \hat{E}, \lambda, V^c, V_a^c, G^c, \hat{G}^c, \lambda^c)$ *where:*

- $V, \hat{V}, G, E, \hat{E}, \lambda$ *are as specified in Definition 6.2.1;*
- $\hat{G} : G \backslash G^c \rightarrow \{C, A, N\}$ *is a function that assigns for a non-configurable gateway* $g \in G \setminus G^c$ *a type as specified in Definition 6.2.1;*
- $V_a^c \subseteq V^c \subseteq V$ *is the set of configurable nodes;* V_a^c *is the set of nodes whose attributes are configurable;*
- $G^c \subset G$ *is the set of configurable gateways;*
- $\hat{G}^c : G^c \rightarrow (\{C, A\} \times \{OR, XOR\}) \cup \{(N, XOR)\}$ *is a function that assigns a type for a configurable gateway* $g^c \in G^c$ *such that "C" indicates composition, "A" indicates allocation and "N" indicates connection; e.g.* $\hat{G}^c(g^c) = (C, OR)$ *indicates a configurable OR composition gateway.*
- $\lambda^c : V_a^c \rightarrow \mathcal{P}(\mathcal{U}_A \times \mathcal{U}_V^c)$ *is a function that assigns for a configurable node* $v^c \in V_a^c$ *a set of attributes whose corresponding values are configurable, i.e.* $\lambda^c(v^c) = \{(attr_i, value_i^c) \mid i \geq 1, attr \in \mathcal{U}_A \ \wedge \ value^c \in \mathcal{U}_V^c\}$. *We use the shorthand* $\lambda_{attr}^c(v^c) = value^c$ *to refer to the pair* $(attr, value^c) \in \lambda^c(v^c)$

In our example in Figure 6.4, the definition of the configurable VNFD is as following: $V = \{v_1, \ldots, v_{24}\}$; $\hat{V}(v_1) = VNFD$, $\hat{V}(v_3) = VNFC$ and so on; $G = \{g_1, \ldots, g_7\}$; $\hat{G}(g_1) = C$ and so on; $E = \{e_1, \ldots, e_{31}\}$; $\hat{E}(e_1) = composition$, $\hat{E}(e_{14}) = allocation$ and so on; $\lambda_{name}(v_2) = vFirewall$, $\lambda_{nCPU}(v_{11}) = 8$ and so on; $V^c = \{v_3, v_4, v_{11}, v_{12}, v_{14}\}$; $V_a^c = \{v_{12}\}$; $G^c = \{g_2, g_6\}$; $\hat{G}^c(g_2) = (C, OR)$, $\hat{G}^c(g_6) = (C, XOR)$; $\lambda_{size}^c(v_{12}) = [5 - 9]Gb$.

6.4 Learning the configurable model

In this section, we propose an approach based on machine learning that constructs automatically the configurable VNFD model. The approach is divided into three phases. In the first phase, each VNFD file in the data set is represented in a tree-like structure following the model defined in section 6.2. In the second phase, common nodes from the set of VNFD models are federated together, and in the last phase, the configurable VNFD model is constructed by adding the relation between all the nodes from all the VNFD models using the federated nodes and the configurable elements. We detail afterward each phase of the algorithm.

6.4.1 Transforming the VNFD files into a tree-like structure

The transformation of the VNFD files is a straightforward task as the VNFD is defined following a predefined data model. Once the data model of the VNFDs is identified, the VNFD model is constructed by representing each component instance in the file to a VNFD node, with respect to the type of the component. The relations between the component instances as represented as arcs between the nodes in the VNFD model. The gateways are added afterward when there are 1 to n relations between the nodes, with respect to the relation type. At the end of this phase, the set of VNFD files is transformed into a set of VNFD instances.

6.4.2 Federating common nodes in the set of VNFD instances

In the second phase, we propose an algorithm that searches automatically for common nodes in the set of VNFDs models that could be federated together. It is not obvious to recognize similar nodes in different VNFD models, where each VNFD could be designed by different domain experts or service providers. Therefore, nodes with similar functionality meaning could have different labels, different attributes, different configurations, and different relations to the other nodes.

Our algorithm cluster together similar nodes in the set of VNFD models based on four characteristics that define each node: label, type, attributes, and relations. Clearly, only nodes with similar type could be compared together. Thus, the node label, attributes, and relations are the node main features that will be used to measure the similarity between nodes.

Before discussing the algorithm, we define first the metrics that will be used by the algorithm.

6.4.2.1 Similarity metrics between the nodes

The similarity between the nodes is measured in terms of their features. Let $\ddot{V} = \{\ddot{v}_1, \ddot{v}_2, \ldots, \ddot{v}_{\ddot{N}}\}$, be the set that contains all the nodes that are extracted from the

VNFD data models, is the number of all the nodes. The features of each node $x \in \ddot{V}$ are formally defined as follows:

A node x has a type denoted $x.type$, a label denoted $x.label$ and a set of attributes denoted $x.attributes$. The node type is a predefined notation that is specified in the data model of the VNFD. The label of the node is a string that is set arbitrary by the service provider. The attributes are a set of key/value pairs that characterize the configuration of the node (component instance). To represent the node's attributes, we define $att_x = \{a_1, a_2, \cdot, a_A\}$, $x.attributes = att_x$ as the attributes vector of a node $x \in \ddot{V}$. att_x contains all the possible attributes that occur in the set of nodes $\ddot{V}$. The index i of $a_i \in Att_x$ represents a particular attribute key and its content is the value of that key. If an attribute $a_i \in Att_x$ does not exist in the node x, its value is then equal to 0. The qualitative attributes of the nodes are considered as categories. Each possible category is added as an individual attribute in att_x. It is equal to 1 if its value occurs in the node and equals 0 otherwise. The quantitative attributes are on the other hand normalized so that their values are restricted between $[0, 1]$. We use for that the basic normalization formula:

$$a_i^x = \frac{a_i^x - min_{\ddot{V}}(a_i)}{max_{\ddot{V}}(a_i) - min_{\ddot{V}}(a_i)} \tag{6.1}$$

where $a_i^x = a_i \in Att_x$, $max_{\ddot{V}}(a_i)$ and $min_{\ddot{V}}(a_i)$ are respectively the maximum and minimum values of a_i across all nodes attributes in $\ddot{V}$.

The distance between nodes

We define $dist(x, y), x, y \in \ddot{V}$, as the distance that measures how similar two nodes x and y are to each other. It is defined as follows:

$$dist(x, y) = (\omega_1 * sim_l) + (\omega_2 * sim_a) + (\omega_3 * sim_r) \tag{6.2}$$

where :

- $sim_l = sim_l(x.label, y.label)$ is the similarity between the labels of x and y
- $sim_a = sim_a(x.attributes, y.attributes)$ is the similarity between the attributes of x and y
- sim_r is a similarity that measures how much relations x and y have in common.
- ω_1, ω_2, ω_3 are weights between the similarities, such that $\sum_{i=1}^{3} \omega_i = 1$

The label similarity, sim_l, is a string metric that measures the syntactical similarity. We propose to use the Levenshtein distance [10] to compute the matching between two nodes labels. sim_l, is defined as follows :

$$sim_l(x.label, y.label) = \frac{LevenshteinDistance(x.label, y.label)}{max(|x.label|, |y.label|)} \tag{6.3}$$

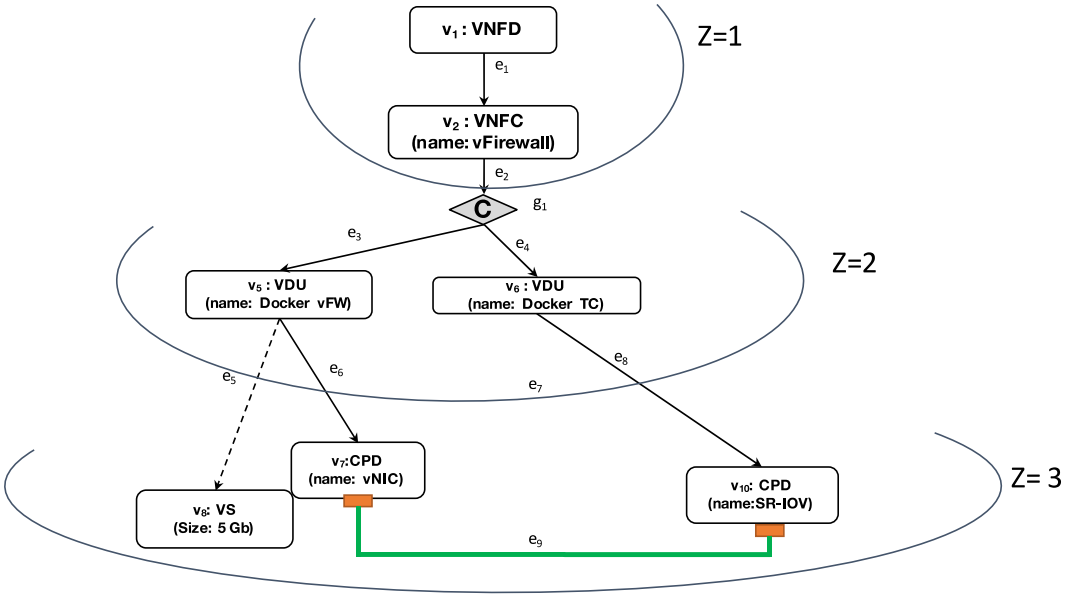


Figure 6.5: A VNFD model that shows the different relational levels between node v_1 and the other nodes

The similarity between the nodes attributes is the distance between their corresponding attribute vector. we propose to use for that the euclidean distance, sim_a is defined as follows :

$$sim_a(x.attributes, y.attributes) = \sqrt{\sum_{k=1}^{A}(x.a_k - y.a_k)^2} \tag{6.4}$$

The relational similarity, sim_r, between two nodes, measures the common elements that both nodes are in relation with. There are different levels of relations that could occur between the nodes in the same VNFD model. We divide this relation levels based on the reachability of the nodes. For example, the first level relation of node v_1 in Figure 6.5 are the nodes that are directly related to it i.e. v_2. The second level relation are the nodes v_5 and v_6. It is important in the relational similarity nodes to account for the level of the nodes when comparing the relations. Nodes that are in lower relation levels are more relevant to account for their similarity than the farthest relation levels. Therefore, we define the relational similarity as following:

$$Sim_r(x, y) = \sum_{z=1}^{Z} \frac{\sum_{\gamma \in \Gamma(x,y)^z} \frac{Z+1-z}{Z} \times dist_r(\gamma)}{|\Gamma(x, y)^z|} \tag{6.5}$$

where :

- Z is the number of relational levels to be considered in the similarity.
- $\Gamma(x,y)^z$ is the set of all combination of nodes (x',y') in the relational level z, such that x' is a node related to x and y' is a node related to y'
- $dist_r(\gamma)$ is the distance between the nodes in the relational set that measures the similarity interms of the label and the attributes. It is defined in eq. 6.6

$$dist_r(x,y) = (\omega'_1 * sim_l) + (\omega'_2 * sim_a) \tag{6.6}$$

where ω'_1, ω'_2 are weights. $\omega'_i = \omega_i + \frac{\omega_3}{2}, i \in \{1,2\}$

6.4.2.2 Nodes clustering algorithm

The clustering algorithm groups the nodes with similar types together in clusters. Each cluster contains the nodes that are likely to be similar. The resulted set of clusters is afterward added in the configurable VNFD model, each cluster is a single node in the configurable VNFD model.

Given a set of nodes with similar types, we propose to use the k-medoids machine learning method to find k clusters that groups similar nodes together. Initially, the k-medoids method selects arbitrary k nodes from the nodes set as the center of k independent clusters. At each iteration, the method minimizes the distance between all nodes and the center of clusters by assigning each node to the closest cluster. The center of clusters is afterward reselected at as the closest node in the cluster to all the other nodes in the same cluster. Algorithm 3 summarizes the clustering algorithm.

Algorithm 3 Clustering algorithm

Input: $\ddot{V}^T$(set of nodes of type T), K (number of clusters), Max_I (maximum number of iterations)

Output: $Cl_1, Cl_1, \ldots, Cl_K$ clusters of nodes

`// Initialization`

23 I=0 (Initial iteration)

$Center_k = Select(x \in \ddot{V}^T)$ (choosing K unique nodes as cluster centroids)

while *(Nochangesinall* $Center_k$) **do**

`// Constructing the K clusters`

24 *For each* $x \in \ddot{V}^T$, *Add (x,* Cl_j*) such that* $dist(x, center_j) = Min(dist(x, center_k)), \forall k \in [1,K]$

`// Updating the clusters centroids`

25 *For each* $Cl_k, k \in [1,K], Center_k := x, x \in Cl_k$ *such that* $\sum_{y' \in CL_k} dist(x,y') <= min(\sum_{y' \in CL_k} dist(x',y')), \forall x' \in CL_k$

26 I++

27 **end**

6.4.3 Constructing the configurable VNFD model

The configurable model is constructed starting from the root node. The root is by default named ($VNFD^c$), which is the grouping of all the root nodes in the VNFD models set. Each node is linked to other nodes for which it has relations with. If the node is created in the second phase (federation of other similar nodes) then different relations could exists depending on which configuration is selected. Configurable operators are added to account for this variability. Given the resulted clusters $Cl = cl_1, cl_2, \ldots, cl_K$ from the second phase. We define for each cluster $cl_i \in Cl$ three sets for each relation type that this cluster could to other clusters of nodes. $N^-_{i,r}, N^+_{i,r}, N^*_{i,r}$, where i is the cluster index and r is the relation type (composition, allocation or connection). The three sets contain the nodes or clusters of nodes that the cluster cl_i is in relation with (relation type = r).

Given that cluster cl_i contains nodes that are likely to be similar. We use the three sets for each relation type to define the configurable operators that we will assign to the cluster in the configurable model.

- The set $N^+_{i,r}$, contains nodes from other clusters that are in relation with more than one node in cl_i but not with all the nodes.
- $N^-_{i,r}$, contains nodes from other clusters that are in relation with only one node in cl_i
- $N^*_{i,r}$, contains nodes from other clusters that are in relation with all the nodes in cl_i.

The configurable gateways are added to the cluster cl_i relations depending on the sets $N^-_{i,r}, N^+_{i,r}, N^*_{i,r}$.

- A configurable XOR gateway is added between the cluster relations with the nodes in $N^-_{i,r}$, with respect to the relation type r.
- A configurable OR gateway is added between the cluster relations with the nodes in $N^+_{i,r}$.
- The relation with the nodes in $N^*_{i,r}$ are linked using a simple gateway or with a direct edge if this are only one node in $N^*_{i,r}$.

For example, suppose that our cluster contains the elements $cl_i = \{a, b, c\}$. and that $N^+_{i,r} = \{d, e\}$, $N^-_{i,r} = \{f, g\}$ and $N^*_{i,r} = \{h\}$. Then, the configurable model will have a node cl_i that have relations with type r to nodes d, e, f, g, and h. An OR configurable operator will be added before the relation with d, e. An XOR configurable operator will be added before the relation with f, g.

6.5 Application of the configurable model: Configuration Guidance Model

In this section, we propose an approach for supporting the deployment descriptor design and generation. As stated previously, the deployment descriptors are usually created manually or with static automation. That is certainly a cumbersome, time-consuming, and error-prone approach. The design of deployment descriptors is based only on the knowledge of the domain expert. There is no formal strategy to assist the design except experience and best practices. Moreover, NFV is a faster-growing area where the specifications are constantly improved to boost progress and ensure an agile response to the evolving industry needs. The deployment descriptors are consequently continuously evolving, which constrains the domain expert to always account for new updates to make the configurations. There is therefore a need for easing this labor-intensive task of designing the deployment descriptors.

Our approach for supporting deployment descriptors design helps domain experts to make correct configuration based on previous service deployments. The approach learns automatically from the experience of service deployment to extract useful and implicit knowledge for configuring deployment descriptors. It is certainly more efficient to capitalize on past deployment descriptors as many network functions and services share common configurations and are often reused and adapted by different entities for different contexts. For example, a VNF could be used to compose different services or used as a component for other VNFs. For example, as illustrated in Figure 6.2 and Fig6.3, the vFirewall VDU is used in both VNFDs. The approach advises domain experts on how to configure the deployment descriptors given their needs and enables them to customize the configurations and adapt them to their context.

The approach derives configuration guidelines from a guidance model. The guidelines capture the dependencies and the relations between the configuration choices of VNFD elements and can be used to generate automatically the deployment descriptors. They are extracted by mining the configurable deployment descriptor model and a repository of VNFD models.

We start by defining the configuration guidance model. The model recommends guidelines for the domain expert on the best configuration choices given the current context. Afterward, we show how our approach can extract the set of configuration guidelines and use them to assist the creation of deployment descriptors.

6.5.1 Configuration guidance model

Configuration guidelines are instructions that indicate which configuration to choose given a configuration context. The context is a set of already made configurations in the deployment descriptor. The guidelines could be modeled basically by if-then rules that describe a configuration for each possible context. Each rule, therefore, indicates a decision that should be made for a VNFD component instance given an

Table 6.2: A sample of guidelines extracted from the configurable deployment descriptor model presented in Figure6.4

g_1	if (v_1 = "vFirewall") then ($c_1 = \emptyset$)
g_2	if (v_1 = "vFirewall") then ($c_2 = v_6$)
g_3	if (v_1 = "vCPE") then ($c_1 = v_3, v_4$)
g_4	if (v_1 = "vCPE") then ($c_2 = v_7$)
g_5	if ($c_2 = v_6$) then ($c_3 = v_{11}$)
g_6	if ($c_2 = v_7$) then ($c_3 = v_{12}, v_{11}$)
g_7	if ($c_2 = v_7$) then ($c_5 = v_{19}$)
g_8	if ($c_2 = v_7$) then ($c_4 = e_{22}$)

already made configuration decision.

For example, we can extract the guidelines described in tab 6.2 from the configurable descriptor model in Figure 6.4 based on the VNFD examples in Figure6.2 and Figure6.3. The guideline g_8 for instance indicate that the connection point v_{11} should be connected to v_{13} when the VDU v_6 is selected in the VNFD.

The problem that arises from such representation of guidelines is that manually identifying all guidelines from a repository of deployment descriptors is a complex task, especially when there are large interdependencies between the VNFD configuration choices. A large number of interdependencies could result in a high number of guidelines, which may be confusing for the VNFD designer, given that the guidelines are interconnected and need to be ordered when used.

To overcome these challenges, we propose a guidance model that can be used to derive the guidelines. It is a tree-like structure that is inspired by the work in [12]. This structure representation allows representing the hierarchical ordering of the configurable operators in a parent-child fashion, where the parent configurable operator needs to be configured before its child element. Figure6.6 depicts an example of our proposed guidance model.

The configurable operators are graphically represented by circles. The configuration choices for each configurable operator are included in the representation, they are graphically represented with a rectangle and attached to the configurable operators with dotted lines. Each configuration choice has a probability of selection, that is labeled on the dotted line in the graphical representation. The probability has an impact on the decision making of VNFD designers. In Figure6.6 for example, the configurable operator c_1 has two configuration choices; either to choose two VNFCs, v_3 and v_4, or to choose no VNFC, with a probability of 0.5 and 0.5, respectively. This means that there is 50% chance for selecting either one of the two choices.

Configuration guidelines are modeled directly in the guidance model with cross-tree relations. A cross tree relation between two configurations indicates their dependency. Guidelines are represented graphically with a directed arrow between the configuration choices. The direction of the arrow indicates the configuration choice

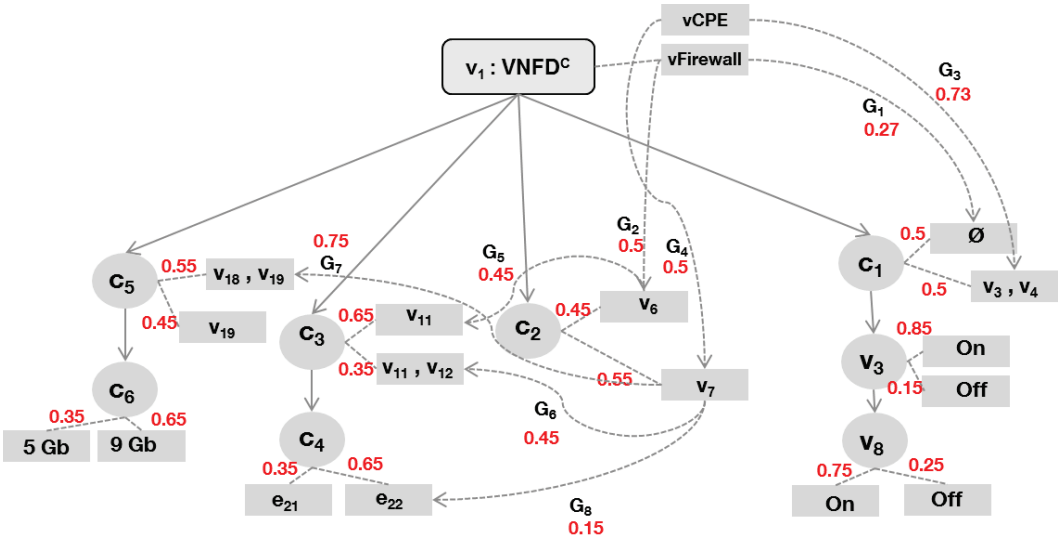


Figure 6.6: Representation of a configuration guidance model that is derived from the configurable deployment descriptor model in Figure6.4

that is a resulted consequence from the origin configuration choice. Each guideline is labeled with a probability of certainty that expresses the validity of the guideline. Once the guidelines are extracted, their validity could be used to determine which of the guidelines are more relevant. Hence, the VNFD designer could define a threshold of validity to simplify the usage of the guidance model. The guidelines in Tab.6.2 are illustrated in the configuration guidance model (Figure6.6). For example, the guideline g_6 indicates that if the VDU v_7 is selected, then the connection point v_{11} is likely to be connected to the connection point v_{13} with a probability of 0.45.

The dependency between the configuration guidelines is an important aspect to be considered. There could be three types of dependency relations between the guidelines: causality, concurrency, and exclusiveness. The causality relation indicates that a guideline is applied only after previous guidelines. For example, in the configuration guidance model in Figure 6.6, the guideline G_5 can only be executed after the guideline G_2. The concurrency indicates that the guidelines could be applied simultaneously, like for example in Figure 6.6, the guidelines G_6 and G_2 could be executed simultaneously. The exclusiveness indicates that only one of the guidelines could be applied. For example, G_6 and G_8 in Figure 6.6. To capture and represent the dependency relations into the guidance model, we use a Petri Net model where each transition represents a configuration guideline and the flow relation between the guidelines relates to their dependencies. A Petri net representation of the configuration guidelines in Tab. 6.2 are illustrated in Figure6.7. Each trace in the Petri net

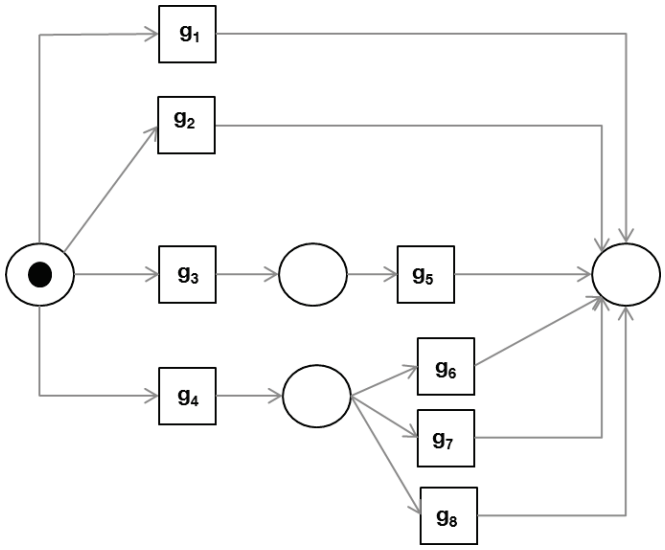


Figure 6.7: A Petri net example that illustrates the dependencies between the configuration guidelines in 6.2

is a possible application sequence of the configuration guidelines. For example, the guidelines g_4 and g_6, need to be applied sequentially.

The formal definition of the guidance model is given in definition 6.5.1

Definition 6.5.1 (Guidance model)**.** *A Guidance model is defined as* $\mathcal{G}_{\mathcal{M}} = (\mathcal{T}, \mathcal{C}, \mathcal{F}_{\mathcal{C}}, \mathcal{G}, \mathcal{P}_{\mathcal{C}}, \mathcal{P}_{\mathcal{G}}, \mathcal{O})$ *where:*

- $\mathcal{T} = (C', E')$*, is a directed graph, where* $C' \subseteq C$ *is the set of the configurable operators in the configurable descriptor model.* $E' = C' \times C'$ *is the set of the edges*
- $\mathcal{C}$*, is the set of the configuration choices that a configurable operators can have.*
- $\mathcal{F}_{\mathcal{C}} : N \rightarrow P(\mathcal{C})$*, is a function that maps a configurable operators to its valid configuration choices.*
- $\mathcal{G}$*, is the set of the configuration guidelines that could be derived from the configurable descriptor model.*
- $\mathcal{P}_{\mathcal{C}} : \mathcal{C} \rightarrow \mathcal{D}$ *, is a function that assigns for each configuration choice in* $\mathcal{C}$ *a probability of occurrence.*
- $\mathcal{P}_{\mathcal{G}} : \mathcal{G} \rightarrow \mathcal{D}$ *, is a function that assigns for each configuration guideline in* $\mathcal{G}$ *a probability of validity*

- $\mathcal{O} = (P, \mathcal{G}, \mathcal{R})$, *is the Petri net model that formalize the dependencies between the guidelines, where:* P, *is the set of the set of places.* $\mathcal{G}$ *is the set of guidelines and* $\mathcal{R}$ *is the set of guidelines relations.*

6.5.2 Dynamic guidance model extraction

We present here an approach that automatically extracts a guidance model from a repository of deployment descriptors (VNFDs in our case). This repository may contain deployment descriptors that were used to construct the configurable deployment descriptor, variants of deployment descriptors that were extracted from the configurable model or newly added deployment descriptors.

The approach takes as input a configurable model (D^c) and a set of deployment descriptor models ($D_i \in \mathcal{D}$), and outputs a guidance model. The approach is divided into four steps: In the first step, the configuration choices for each configurable operator in D^c are extracted from the repository of deployment descriptors $\mathcal{D}$. In the second step, the configuration guidelines are derived from the extracted configuration choices using association rule mining techniques. In the third step, the tree-like structure is extracted and in the last step, the dependencies relations between the configuration guidelines are formalized.

6.5.2.1 Step 1: Configuration choices extraction

In this step, we identify the set of VNFD configurable elements from the repository of deployment descriptors $\mathcal{D}$ and then construct a configuration matrix. The configuration matrix denoted M^c, contains the configuration choices that are present in each deployment descriptor, and for each configurable element in D^c. It is a $p \times q$ matrix, where p is the number of deployment descriptors in $\mathcal{D}$ and q is the number of configurable VNFD elements in D^c.

To identify the configuration choices at each deployment descriptor $i \in \mathcal{D}$, we first map the VNFD elements to their most similar elements in the configurable model. This is done by measuring their structural similarity. We consider here also the similarity metrics used in the learning of the configurable model.

Given $\dot{V}_i$ a set of VNFD elements in a VNFD model $i \in \mathcal{D}$ and $\dot{V}^g$, the set of configurable elements in the configurable model. We measure the similarity between nodes in $\dot{V}_i$ and $\dot{V}^g$ that have a similar type. Given two elements x and y that have similar type and are included in $\dot{V}_i$ and $\dot{V}^g$, respectively, we use the similarity distance ($dist(x, y)$) that is defined in equation (2) for that purpose, except that the relational similarity Sim_r in $dist(x, y)$ is replaced by a different relational metric. In this situation, instead of measuring the common elements that both nodes are in relation with, we are more interested in only measuring if the elements that in relation with the node x are also in relation with the node y. The goal is only to measure if the structure of x is similar to y. Let $\dot{R}_x$ and $\dot{R}_y$. the set of nodes that x and y in relation with, respectively. We define therefore Sim_s as follows:

$$\mathrm{Sim}_s = \frac{\sum_{x' \in \dot{R}_x} max_{\forall y' \in \dot{R}_y}(Sim_l(x',y') + Sim_a(x',y'))}{|\dot{R}_x|} \tag{6.7}$$

The new similarity distance $(dist(x,y)')$ is therefore :

$$\mathrm{dist(x,y)'} = (\omega_1 * Sim_l) + (\omega_2 * Sim_a) + (\omega_3 * Sim_s) \tag{6.8}$$

Note that the weight $((\omega_3)$ of the relational similarity (Sim_s) needs to be small relative to the label similarity (Sim_l) and the attribute similarity (Sim_a). That is because the nodes in the deployment descriptors don't need to have similar relation with other nodes as within the configurable model. The deployment descriptor could implement different variants of configuration choices.

6.5.2.2 Step 2: Guidelines derivation

So far, we extracted for each configurable element in D^c, the set of its configurations from each descriptor variant $D_i \in \mathcal{D}$. The extracted configurations are used as input to derive the configuration guidelines using association rule mining techniques. Association rule mining [8] is one of the most important techniques of data mining. It aims to find rules for predicting the occurrence of an item based on the occurrences of other items in a transactional database or other repositories. It has been first applied to the marketing domain for predicting the items that are frequently purchased together. Thereafter, it manifested its power and usefulness in other areas such as web mining [41] and recommender systems [68]. The Apriori algorithm [9] is one of the earliest and relevant proposed algorithms for extracting association rules. We also use the Apriori algorithm in our approach to derive the configuration guidelines.

Taking the configuration matrix as input, the Apriori algorithm proceeds in two phases. In the first phase, the set of frequent *correlated configurations* (i.e. configurations that often appear together in the same row in the configuration matrix) are discovered according to a frequency threshold. The Apriori algorithm uses the monotonicity property that all subsets of a frequent correlated set are also frequent. Therefore, Apriori starts by selecting the frequent single configurations, then generates the candidate pairs of configurations (i.e. correlated sets of two configurations) from the frequent singles and so on, until it finds all possible correlated configurations according to the frequency metric. It uses *Support*, a well-known metric to compute the frequency of a set of correlated configurations. The support is defined as the fraction of VNFD models in which the correlated configurations appear together. Let $\mathcal{C} = \{Conf_i : 1 \leq i \leq k\}$ be a set of k-correlated configurations. The support is computed as:

$$Sup = \frac{|\mathcal{D}_\mathcal{C}|}{|\mathcal{D}|} \tag{6.9}$$

where $|\mathcal{D}_\mathcal{C}|$ is the number of VNFD models in $\mathcal{D}$ that contain the configurations in $\mathcal{C}$ and $|\mathcal{D}|$ is the number of descriptor variants in $\mathcal{D}$. Support is equal 1 if all the

VNFD models in the repository contain the correlated configurations. Support is equal 0 if none of the VNF models contain the corresponding configurations together. A set of correlated configurations is frequent if its support is above a given threshold of $minSupp$.

In the second step of the algorithm, the set of relevant configuration guidelines in the form of Left Hand Side (LHS) → Right Hand Side (RHS) is derived from the frequently correlated configurations. To keep only relevant guidelines, the *confidence* metric is computed to evaluate the probability of occurrence of a guideline. The confidence of a configuration guideline $G : LHS \rightarrow RHS$ is defined as the probability of occurrence of the configurations in the right-hand side RHS given that the configurations in the left-hand side LHS are selected. It is computed as:

$$C = \frac{Sup_{(RHS \cup LHS)}}{Sup_{RHS}} \tag{6.10}$$

where $Sup_{(RHS \cup LHS)}$ is the support of the configurations in the right-hand and left-hand sides of G and Sup_{RHS} is the support of the configurations in the right-hand side. Confidence is equal to 1 if whenever the configurations in the right-hand side are selected, then the configurations on the left-hand side are also selected. A configuration guideline is relevant if its confidence is above a given confidence threshold of $minConf$.

6.5.2.3 Step 3: Tree-like structure extraction

The tree hierarchy $\mathcal{T}^c$ of the configuration guidance model $\mathcal{G}_{\mathcal{M}}$ consists of parent-child relations between the configurable elements. An element n_1^c is a candidate parent of a child element n_2^c if the configuration of n_2^c highly depends on that of n_1^c. The dependencies relations between the configurable elements can be derived from their configuration choices. It is very straight forward to derive the hierarchy of the deployment descriptors. This is because a deployment descriptor follows a predefined data model that indicates the structure of the VNFD elements. Therefore, in our case, tree hierarchy is derived following the ETSI NFV specification.

6.5.2.4 Step 4: Guidelines dependencies formalization

We aim here to formalize the dependencies relations between the configuration guidelines to assist the service providers incrementally applying them consistently and validly. Let $\mathcal{G} = \{G_i : i \geq 1\}$ be the set of the extracted configuration guidelines for a configurable VNFD mode $D^c = (V, \hat{V}, G, \hat{G}, E, \hat{E}, \lambda, V^c, V_a^c, G^c, \hat{G}^c, \lambda^c)$. We propose an approach based on the Theory of Regions to derive the configuration system. To do so, we first generate a transition system from the configuration guidelines in $\mathcal{G}$. Then, we use the Theory of Regions to synthesize a Petri-net. The transition system explicitly shows the deployment descriptor configuration states resulting from the application of the configuration guidelines and all possible transitions between them. It

contains the configuration states, i.e. the states in which a deployment descriptor file can be as a result of applying the configuration guidelines and the transitions between the states labelled with the configuration guidelines index.

A descriptor configuration state represents the set of selected elements' configuration at an instant t. It is defined as a m-dimensional vector, where m is the number of configurable elements in D^c and each entry in the vector represents a selected configuration of one configurable element. An entry is set to "-" if no configuration is selected for the corresponding configurable element. A configuration guideline G. : LHS RHS can be triggered if there exists a state, called pre-configuration state, in which the configurations in the LHS part are selected. If triggered, a new configuration state, called post-configuration state, in which the configurations in the RHS are added to the already selected configurations is resulted. Having the transition system as input, we use the Theory of Regions [64] in order to derive a configuration system represented as a Petri net.

6.6 Other applications of the VNFD Configurable model

In this section, we give some future perspectives on other application cases of the configurable model. As an illustration, we show the applicability of the configurable model to (i) generate automatically variant of deployment descriptors in section6.6.1, (ii) mine the dependencies between the configuration elements in section6.6.2 and (iii) uniformize the representation of deployment descriptors in section6.6.3.

6.6.1 Deployment descriptor variant generation

In the research area, there is a need for a large catalog of deployment descriptors to conduct research projects related to mining and learning from the configuration files. Unfortunately, there is no standard public data set of VNFDs or NSDs. Most of the available public repositories of the descriptors contain demo templates that indicate how to define the descriptors via examples. There are also different data models to define descriptors, such as Tosca and Yang. This makes it difficult to collect the scattered small data sets of descriptors into a unique repository. Moreover, many NFV organizations are reluctant to publish their own data, which accentuates the problem of collecting the descriptors.

The configurable deployment descriptor model could be used in this case as a tool for generating deployment descriptor variants and extend the catalog with more VNFDs. The configurable descriptor model exploits the variability captured from a valid data set to be able to generate more valid VNFDs. We illustrate this ability with an approach that uses the configurable operators to generate a new alternative of configuring a VNFD. We consider in this case the previous configurable descriptors model that annotates the VNFD components instances with their VNFD (see section

Table 6.3: Definition of the variation factors

Groupe	Function	Description
Composition Factors	$\text{add}_{comp}(c,d)$	adds one or more component instances from the available configuration choices.
	$\text{del}_{comp}(c,d)$	deletes one or more component instances. All the component instances that are related to the deleted component instances are also deleted.
	$\text{alt}_{comp}(c,d)$	Changes one of the component instances attached to the operator with one in the available configuration choices.
Allocation Factors	$\text{add}_{al}(c,d)$	add one or more ressources from the available from the available configuration choices.
	$\text{del}_{al}(c,d)$	Changes one of the ressources attached to the operator with one in the available configuration choices.
	$\text{alt}_{al}(c,d)$	Changes the attributs of a ressources with an other one from the available configuration choices. Change a ressource with an other one from the available configuration choices.
Connection Factors	$\text{add}_{con}(c,d)$	adds one or more connection points from the available configuration choices. Requirement: The new connection point needs to be connected a valid connection point.
	$\text{del}_{con}(c,d)$	deletes a connection point and its associated connection (virtual link)
	$\text{alt}_{con}(c,d)$	changes a connection point with an other one from the available configuration choices. Requirement: The new connection point needs to be connected a valid connection point.

6.2). The generation of the VNFD variants is performed through variating existing VNFDs. This helps the generation of a valid VNFD structure.

Given a configurable deployment descriptor model annotated with the VNFDs for each component instance $G = (V, E, C, \hat{V}, \hat{E}, \hat{C}, \mathcal{D}, f_{\mathcal{D}})$. We introduce the variation factors that we use to create new VNFDs. It is a set $\mathcal{A}$ of factors, each factor $a \in A$ is a function responsible for altering the VNFD in a specific way. We define the set of factors in tab.6.3. The factors are classified in three categories similarly to operator categories in the configurable model: composition, allocation and connection. For example, the composition factor, $add_{comp}(c,d)$ takes as input a configurable operator c and adds one or more component instances in the available configuration choices to the VNFD d.

One approach for the generation of multiple variations of VNFDs from a given VNFD is through employing randomly the variability factors that are defined in Tab.6.3 at each configurable operator in the configurable deployment descriptor model. We first select the VNFD from $\mathcal{D}$. Each configurable operator in the structure of the VNFD can be potentially used by one of the variability factors to create an alternative configuration. The category of the variability factor needs to be compatible with the category of the configurable operator. To avoid a large deviation from the original VNFD, we use a variation rate parameter that can regulate the use of the variability factors. This parameter is defined manually by the user. It indicates how much changes could be permitted in the new version of the VNFD. As an example, Fig.6.8 illustrates a variable version of the vFirewall VNFD that is defined in Fig. 6.2. To generate this variable version of the VNFD, the variability factor $add_{comp}(c_2, vFirewall)$ is used in configurable operator c_2 to include the VDU v_7. The resulted VNFD is a vFirewall VNFD with an additional VDU (Tocker TC).

6.6.2 Dependency mining

Mining the deployment descriptors to learn their characteristics is an important aspect that is unfortunately yet neglected. Deployment descriptors are designed based solely on the domain expert judgment. The expert chooses based on his expertise a good

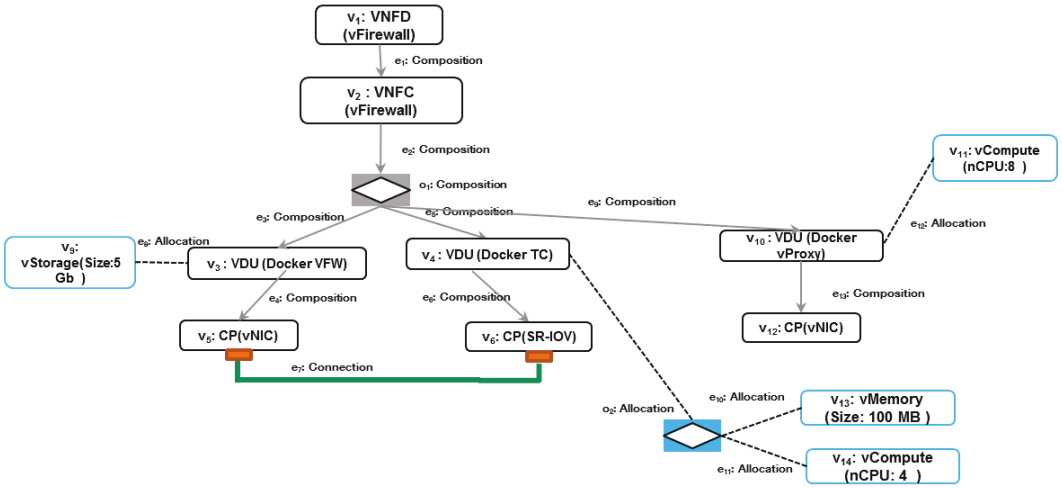


Figure 6.8: An example of a variable VNFD generated from the vFirewall VNFD that is defined in Fig.6.2

configuration for deploying a network function. It is therefore hard to tell what can be the best deployment configuration for a given network function as different domain experts can have different views on that matter. Other than that, learning the characteristics of the descriptors can be beneficial in many use cases. It can be helpful to extract features from the descriptors to enable artificial intelligence algorithms such as deep learning techniques to learn from the descriptors. Deep learning techniques could then be used to learn to classify deployment descriptors to their network function, to detect abnormal configurations, to cluster the deployment descriptor to a given network requirement, etc. We propose a basic approach inspired by process mining integration [13] that illustrates how the configurable descriptor model can be used to extract characteristics from the deployment descriptors. It is an approach that captures the dependencies between VNFDs and measures their similarity.

The configurable deployment model in this case is customized to account for the relation between the VNFDs and the components that compose them in the representation. Each VNFD component instance is annotated with the corresponding VNFDs that include it. The configurable deployment model is therefore defined as : $G = (V, E, C, \hat{V}, \hat{E}, \hat{C}, \mathcal{D}, f_{\mathcal{D}})$, where $\mathcal{D}, f_{DE}$ are additional elements in the graph. DE represents the set of VNFDs that are used in the configurable deployment model and $f_{\mathcal{D}} : V \rightarrow \mathcal{D}$, is a function that returns the VNFDs related to a component instance in V. Fig. 6.9 illustrates an excerpt of the configurable model with component instances

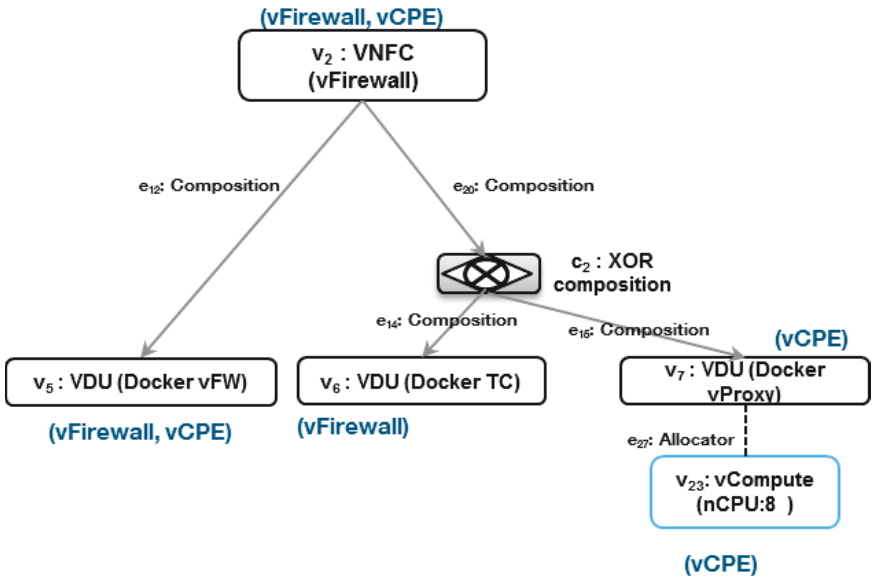


Figure 6.9: An excerpt of the configurable descriptor model that annotate the component instances with their corresponding VNFDs

annotated with their VNFDs. Notice that it could be other notations in the graph to account for different characteristics like the VNFD vendor, the version of the VNFD, the requirement that the VNFD is satisfying, etc. The goal in this example is to measure the similarity between the VNFDs in order to extract the common elements that can characterize them. We construct the set of VNFDs $\mathcal{D} = \{d_1, d_2, \ldots, d_D\}$ from the configurable descriptor model, where each element of $\mathcal{D}$ correspond to a VNFD and contains a set component instances that compose it.

We define a comparability measure (δ-comparability) that measures the ratio of common elements between VNFDs. We use the comparability measure to select the VNFDs that have a high comparable ratio above a predefined threshold. This measure allows us to preselect VNFDs that are potentially closer to each other in terms of common elements. Two VNFDs (d_1 and d_2) are δ-comparable for a threshold $\delta = 0.5$ if : $\frac{|d_1 \cap d_2|}{|d_1 \cup d_2|} \geq 0.5$.

After pre-selecting the set of VNFDs, we measure their similarity by taking into consideration also the relation between the VNFD component instances. We extract a numerical representation for each VNFD in order to compare and rank the similarity in a Euclidian distance metric space. We represent each VNFD in a matrix that captures the precedence dependencies between the VNFD component instances. It is a n-by-n matrix where n is the number of components that compose the VNFD. The value of the matrix cell is either 1, if there is a precedence dependency between the VNFD component instances. The dependency can be any type of relation between

the component instances (composition, allocation or connection) and the relation could be direct or indirect (via configurable operators). For example, L, Let M be the matrix of the VNFD that is defined in Fig. 6.2 and included in the configurable descriptor model in Fig.6.4. The size of the matrix is $|M| = 9$.

$$M = \begin{bmatrix} & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9 \\ v_2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ v_3 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ v_4 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ v_5 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ v_6 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ v_7 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ v_8 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ v_9 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The matrix cell $M(v_1, v_2) = 1$ indicates that there is a precedence relation between v_1 and v_2.

Once the matrices of the pre-selected VNFDs are constructed, we normalize all the matrices by also including in each VNFD matrix the missing component instances that belong to the rest of the VNFD matrices. The added elements matrix cells are obviously set to 0. At the end, all the matrices of VNFDs have the same size (the number of all the component instances of the VNFDs). We then measure the similarity between two matrices in the set using the dependency difference metric.

The dependency difference metric S as defined in 15 counts the number of discrepancies between two matrices. It is the trace of the difference between two matrices. Let M_1 and M_2 two normalized matrices. $S(M_1, M_2) = tr[(M_1 - M_2) \times (M_1 - M_2)^T]$ where $tr[.]$ is the trace (sum of the diagonal elements)of a matrix.

Using the dependency difference metric and the δ-comparability on the configurable deployment descriptor model can help to analyze the characteristics of the VNFDs and identify the relation between the VNFDs.

6.6.3 Uniform representation of the deployment descriptors

With the advent of NFV, new network services and functions are continuously emerging. Despite some efforts to find a consensus on a common information model to use for modeling the deployment descriptors like the ETSI NFV initiative, there is still an absence of a common data model for the deployment descriptors, which resulted on a diversity of solutions, such as: TOSCA, YANG,Hot, etc. This diversity obliges service designers to constantly adapt the descriptors to the heterogeneous NFV platforms in order to enable the integration of the network services/functions. Therefore, designing deployment descriptors for the network services/functions in such a highly

dynamic environment becomes a highly complex, time-consuming, and tedious task. The configurable deployment descriptor mode could be used as a common representation of the VNF component instances. It federates heterogeneous VNFDs into a single representation. It encompasses the knowledge and expertise from different sources. Moreover, the descriptor designer can, therefore, use this presentation to define the VNFD in various data models by mapping each VNFD component instance in the configurable to its desired model. This is enabled through predefined APIs that allow the translation between the different data models via the configurable model.

6.7 Evaluation and results

In this section, we conduct some experiments to evaluate our approach of using the VNFD configurable model to generate the configuration guidance model. We first start by defining the environment settings of our experiments. Then, we assess the complexity of our configurable VNFD model to show its effectiveness. The complexity is an important measure for the VNFD representation. A complex representation impacts the understandability of the VNFD and consequently increases the chances of errors. Afterward, we evaluate the performance of our proposed approach to learn the configurable VNFD model. More precisely, we focus on evaluating the efficiency of the clustering algorithm. Finally, we evaluate the configuration guidance model that is extracted from both the configurable model and the set of deployment descriptors. We conduct two experiments. In the first experiment, we evaluate the quality of the recommended configuration guidelines in terms of completeness and complexity. In the second experiment, we evaluate the accuracy of the extracted configuration guidance models by computing the Precision and Recall values.

6.7.1 Environment settings

In our experiment, we consider approximately 100 VNFDs collected from various organizations and private telco vendors. The data set include several variations of VNFDs for different VNFs such as Mobility Management Entity (MME), virtual Customer Premises Equipment (vCPE), virtualized Subscriber Data Management (vSDM), etc. The VNFDs are YAML files that follow the TOSCA data model.

We implemented a python program to construct the VNFD models from the data set. The number of nodes extracted, with respect to their type, are : 65 VNFC, 453 VDUs , 453 VCs, 421 VMs, 357 VSs, 520 CPs.

6.7.2 Complexity of the configurable deployment description model

To evaluate the complexity of our proposed solution, we consider a scenario in which a service provider wants to design a new vCPE VNF. The only information that he can rely on to have insights about the deployment configuration of the VNF is the

Table 6.4: Structural Complexity metrics for the two considered representation

Complexity metric	Multiple VNFD representations	Configurable deployment description model
CFC	13	19
ACD	2.3	3.1
CNC	0.7	0.56
Density	0.15	0.04

repository of the 30 VNFDs. This repository contains some VNFDs designed by the same service provider but for different VNFs and variation of VNFDs for the same vCPE VNF but designed by different service providers. In order to capitalize on the knowledge provided by the 30 VNFDs, we evaluate the complexity of using a representation given by the configurable descriptor model, against the complexity of using 30 VNFD models.

We assess the structural complexity of the two representations by computing the well-known complexity metrics: CFC (Control Flow Complexity), ACD (Average Connector Degree), CNC (Coefficient of Network Connectivity) and density. CFC metric [98] evaluates the complexity of the representation with respect of the configurable elements (OR, XOR, AND). It computes for each operator the number of states (VNFD component instances) that can be reached from one of the three split constructs. The measure is based on the relationships between mental discriminations needed to understand a split construct and its effects. ACD metric [22] calculate the number of nodes that gateways have as average. CNC metric [69] gives the ratio of edges to nodes and the density metric [95] relates the number of edges to the number of maximum edges that can exist among nodes. The obtained values for two representation are summarized in Table6.4.

The metrics of the multiple VNFD representations are obtained by summing all the metrics resulted from each VNFD representation individually. By comparing the metrics' values of the two representations (see Table6.4), we notice that the CFC of our configurable model is very high compared to the multiple VNFD representations. This can be explained by the fact that when similar VNFD elements are merged in the configurable model, more possibilities of relations with other VNFD elements are generated, especially in the case when the configurable operator "OR" is used. Thus, augmenting the structure of the configurable model with more relations. Also, we notice that density in the case of the configurable model is very low compared to the multiple VNFD representations. This is because when the VNFDs are less similar or similar VNFD elements have different relations to other VNFD elements results into more configurable operators injected in the configurable model. This leads to the increase of the number of relations that may exist in the configurable model while the number of available relations remain slightly the same as the multiple VNFD representations. Consequently, the density of the configurable model decreases significantly compared the multiple VNFD representations. The CNC of the configurable

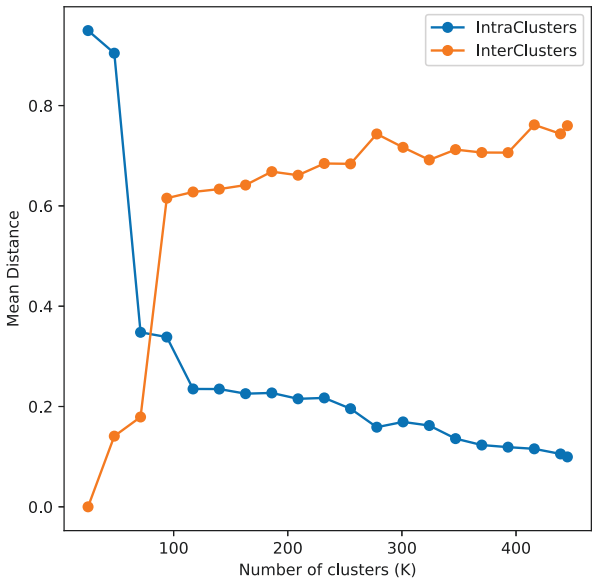


Figure 6.10: Inter-cluster and Intra-cluster mean similarity distance in terms of different cluster numbers, VDU nodes

model is slightly lower than the multiple VNFD representations. An increase in the CNC means that there exists a high number of relations between a relatively smaller number of nodes. The lower density value of the configurable model show that it has a reasonable complexity. This can be explained by the fact that the configurable descriptor model contains less duplication of the components instances and is more expressive.

6.7.3 Learning the configurable model

We evaluate the performance of the clustering algorithm by measuring the efficiency of the clusters into grouping similar nodes. We compute for that the mean similarity distance between the nodes inside the clusters (intra-clusters similarity) and the distance between the clusters (inter-clusters similarity) . The intra-cluster similarity measures how good the clusters group similar nodes and the inter-cluster similarity measures how isolated the clusters are from each other.

The clustering algorithm is applied separately for each node type in the VNFD model. The number of clusters to use is determined empirically. In Figure6.10, we measure the performance of the algorithm in terms of different numbers of clusters. We considered in that experiment the clustering of the VDU nodes. We notice that the

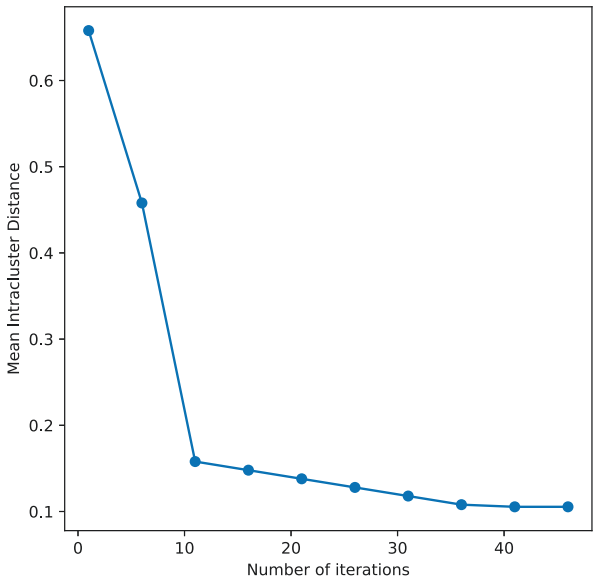


Figure 6.11: Intra-cluster mean similarity distance per iteration, VDU nodes, k=200

distance intra-clusters decrease when more clusters are considered and the distance inter-cluster increases with the number of clusters. This is because the algorithm isolates better the similar nodes in a large data set (450 nodes) when more clusters are used. However, the similarity distance is relatively stable after a while (when 130 clusters are used). This happens when an efficient number of clusters is reached. An additional number of clusters will not improve significantly the performance in this case.

The algorithm converges relatively faster to a minimal mean intra-cluster similarity. Figure6.11, shows the performance of the algorithm after a number of iterations. The experiment is also conducted on the VDU nodes. The number of clusters is fixed to 200. Initially, the performance increases significantly until it stabilizes after 10 iterations. This is due to the random initialization of the clusters' centroids.

The similarity distance between the nodes is impacted by the weights ω_1,ω_2 and ω_3 of sim_l, sim_a and sim_r respectively. In Figure 6.12 shows the performance of the algorithm in terms of different relational weight values (ω_3). The other weights (ω_1 and (ω_2) are fixed to $(1 - \omega_3)/2$). We considered in this experiment the clustering of the VDU nodes and the VNFC nodes. We notice that the performance decreases when the relational weight is small and increases otherwise. This is because the VDU and the VNFDC nodes have few attributes and many relations to the other nodes. Thus the relational weight is very important to be considered. We evaluated also the

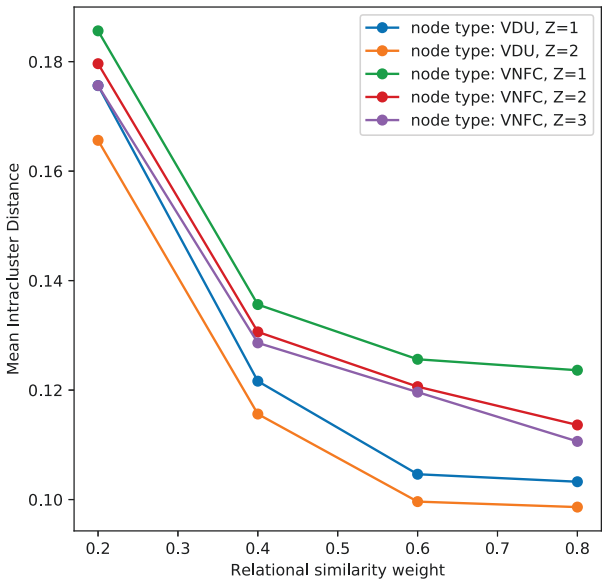


Figure 6.12: Intra-cluster mean similarity distance in terms of different relation weights

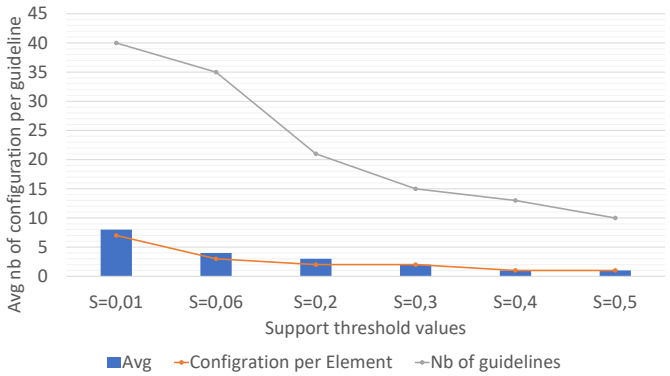


Figure 6.13: Number of guidelines, number of configurations per guideline and per element for different minimum support thresholds and a minimum confidence threshold C = 0:8

impact of the zone Z parameter. It indicates the level of the relations to be considered. We notice that this parameter can also enhance considerably the performance of the algorithm

6.7.4 Configuration guidance model

In this experiment, we evaluate the quality and accuracy of the configuration guidelines that are recommended from the configuration guidance model. The quality of the configuration guidelines is evaluated in terms of completeness and complexity, whereas the accuracy is evaluated in terms of the obtained and the expected configuration guidelines. The expected configuration guidelines are constructed manually with the help of several domain experts of service deployments. We asked them to provide us with the most relevant configurations from the VNFD model repository, we considered the configuration guidelines that are mostly agreed on by the experts.

6.7.4.1 Quality of configuration guidelines

The completeness and complexity of the extracted configuration guidelines are computed based on (1) the number of extracted configuration guidelines, (2) the number of configuration choices per guideline and (3) the percentage of extracted configurations per configurable element with different support threshold values and a confidence threshold C = 0:8. The complexity is expressed in terms of the number of extracted guidelines. The higher the number of extracted guidelines, the more the complexity increases. The completeness on the other hand is expressed in terms of the number of configurations per guideline and the percentage of extracted configurations per configurable element. A high number of configurations per guideline means that the guidelines cover more association between the configurations of all configurable

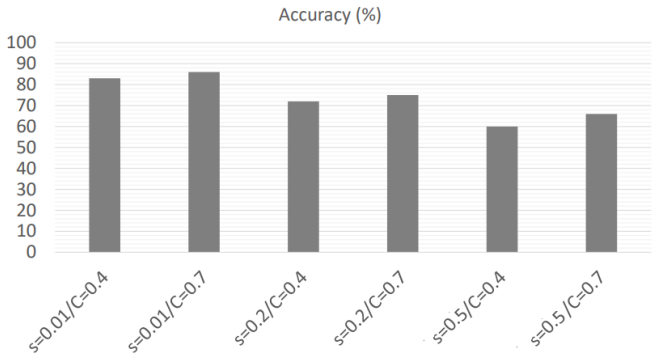


Figure 6.14: Accuracy of the generated guidelines for different support and confidence thresholds

elements in the deployment descriptors model. Also, a high percentage of retrieved configurations per element means that the guidelines cover most of the configuration choices.

The results are shown in Figure 6.13. , the average number of configurations per guideline and the percentage of extracted configurations per configurable element is depicted. The results show that, low support values (S = 0:01 and S = 0:06) record a high number of extracted guidelines and a high percentage of retrieved configurations per configurable element. This leads to the conclusion that a compromise needs to be found between the complexity and completeness. This is because the complexity of the configuration guidelines is positively correlated with their completeness (i.e. when the complexity increases the completeness increases) while both the completeness and complexity are negatively correlated with the support threshold value.

6.7.4.2 Accuracy of configuration guidelines

In this last experiment, we calculate the accuracy of the configuration guidance model for different support and confidence threshold values. For that, we use the set of relevant guidelines generated from the guidance model and compare them to the set of guidelines that are recommended manually by domain experts.

The results are depicted in Figure 6.14. We notice that the guidance model match almost perfectly the guidelines recommended by the domain experts. We also notice that accuracy depends more on the minimal support threshold rather than the minimum confidence threshold. This can be explained by the fact that the support threshold value determines the number of retrieved configurations choices and therefore the generated guidelines.

6.8 Conclusion

In this chapter, we proposed an alternative approach to the deep neural network approach that was proposed in the previous chapter. The approach proposed in this chapter is model-driven and it aims also at addressing the third research question (see chapter 1) on how to automatically generate coherent configurations in software networks. The model-driven approach aims at assisting service providers in NFV on designing the deployment descriptors with fewer efforts. Indeed, the design of deployment descriptors in NFV usually follows a traditional process design where a service provider starts the design and continuously adds elements in a bottom-up design fashion.

The model-driven approach is a configurable deployment descriptor model that allows to represent multiple existing deployment descriptor models into one customizable model. We focus our attention on the virtual network function deployment descriptor models as they are the main component of NFV that compose each network service. The configurable model helps the deployment descriptor design process by offering a model that captures various variabilities on the VNFD element configurations that can be used by service providers in an interactive way. The service providers could use the configurable model to select relevant parts of configuration in order to obtain the required solution.

The proposed configurable model is a graph-based representation of a VNFD model extended with configurable elements and configuration constraints. The configuration constraints ensure a structurally correct configuration of the model. The configurable elements help service providers to select the suited configuration from a set of possibilities, in an interactive way, to derive VNFDs. We proposed also in this chapter an algorithm based on machine learning to learn the configurable model from a set of deployment descriptors. The algorithm automatically clusters similar elements from the deployment descriptors based on a defined similarity distance metric.

We showed how can the configurable model be used to assist the design and generation of deployment descriptors. We proposed for that matter an approach that derives configuration guidelines from a guidance model. The guidelines capture the dependencies and the relations between the configuration choices of VNFD elements and can be used to generate automatically the deployment descriptors. They are extracted by mining the configurable deployment descriptor model and a repository of VNFD models

The experiments that we conducted showed that the configurable model has better complexity than using directly the descriptors from the catalog with separate representations. Our learning algorithm showed also a good performance for clustering similar descriptor elements. These results motivate us further to implement a tool that can be used by service providers to generate deployment descriptors.

The work in this chapter is published in C4,J1 and J2 (see chapter 1, section 1.4).

Chapter 7

Conclusion

In this thesis, we investigated the problem of auto-configuration in software networks. Auto-configuration is an essential capability that allows the networks to adapt and with less human intervention to new changes in the network by configuring new components seamlessly or modifying the parameters in the network according to the overall global state.

We identified three challenges that are important to be overcome to enable the auto-configuration capability. The first challenge is the heterogeneity between the configuration data models. The problem is that there is yet no consensus between network vendors and organizations on a common data model to define the configurations, which resulted in a plethora of solutions used by different network components. The cooperation between the network components is therefore not possible.

We investigated for this challenge a solution that handles automatically the heterogeneity between configuration data models. We focused on the use case of an SDN architecture where multiple heterogenous controllers are deployed. The SDN controllers in that case have to cooperate to share a global network view that allows them to localize network devices. The problem is that the SDN controllers may expose their functionalities (via APIs) using different data models. As a solution, we propose a semantic-based framework that automatically maps configuration elements from heterogeneous data models

The semantic framework is used as a mediation layer between the heterogeneous controllers. It builds an ontology from the configuration files of SDN controllers. Each ontology represents its concepts and the relation between them. The global network view is built by mapping similar concepts from the ontologies using our proposed algorithms. The semantic framework is adapted to two scenarios: (i) A centralized scenario where the SDN controllers expose their network views to a centralized entity that builds the network view, and (ii) a distributed scenario, where SDN controllers exchange in peer to peer their local network views and build locally the global view.

In the centralized scenario, we built a global ontology model that represents the domain knowledge of SDN with OWL. This global ontology describes the concepts that constitute an SDN architecture and it is used to map similar concepts of the controller's local ontologies. Whereas in the distributed scenario, the controller's ontologies are directly mapped to each other.

We evaluated the performance of our framework in both distributed and centralized scenarios, in terms of the matching accuracy of our mapping algorithms and in terms of the execution time. We tested the performance over different network topologies in a heterogenous multi-controller SDN architecture composed by controllers like ODL and ONOS. The evaluation shows that our framework is accurate and has a low computational execution time. However, the shortcomings of this framework are that it is dependent heavily on a predefined ontology in the centralized scenario and dependent on a predefined lexicon in the distributed scenario.

The second challenge that we identified for enabling auto-configuration is the automatic generation of configurations. The network has to automatically configure network components without human intervention to adapt to unexpected situations. The problem is that there is no formal strategy on how to choose the best configuration for each given situation. We could find different configurations made by network administrators for the same network performance goal. This situation makes it difficult to learn automatically patterns of the best configurations to take at each situation. Moreover, the configuration files in software networks are often designed and created manually. Which is a highly complex, time-consuming, and tedious task.

We investigated two approaches to overcome this problem and we have chosen an NFV architecture to implement and test our proposed approaches. In NFV, service providers have to associate with their virtual network functionalities description files before the onboarding of the VNFs. The description files indicate the deployment and operational behavior of the functionalities in terms of connectivity and resource requirements. The descriptor files are large files that are designed manually by the service providers, which is complex, tedious, and error-prone

Our first approach to the challenge of automatic configuration generation is based on deep neural networks. This approach learns from previously made VNF descriptor file models that could recommend and complete configurations. The approach is concretely a learning framework based on neural network architectures that is divided into three phases: the preparation phase, learning phase, and model tuning phase. In the preparation phase, we process the descriptors files into a format that is suited for the neural network architectures. We used a word embedding approach to representing the data that is extracted from the descriptor files based on its semantics. In the learning phase: we proposed two neural network architectures: A Convolutional neural network architecture to learn a recommendation model for the descriptors and Long short term memory architecture to learn a completion model for the descriptors. We evaluated afterward the generated models (recommendation and completion) in terms of their accuracy of prediction. The results are promising and suggest that the framework is a solution that could be used in practical scenarios to enhance the dynamicity of deploying VNFs. The shortcoming of this approach is that it is dependent on a large catalog of VNFD descriptors in order to learn an accurate model.

Our Second approach is a model-driven approach that assists service providers to design and generate descriptor files. The approach is a configurable model that merges

similar component elements from different descriptor files into a single model. This model captures and learns all the configuration variabilities from the descriptors. The model is a tree-structured graph that represents the component elements that appear in the descriptors along with configurable connectors that capture variable structures of configurations. We formalized a configurable deployment descriptor model that capitalizes on a catalog of descriptors and proposed an algorithm based on machine learning (K-medoids) to search and cluster similar elements from different descriptors and construct the configurable model. We also proposed an approach that extracts useful and implicit knowledge from the configurable model. The approach derives configuration guidelines by mining the configurable deployment descriptor model and a repository of VNFD models. The guidelines capture the dependencies and the relations between the configuration of VNFD elements. The experiments showed that the configurable model has better complexity than using directly the descriptors from the catalog with separate representations. Our learning algorithm also showed also good performance for clustering similar descriptor elements.

The third challenge that we identified for enabling auto-configuration in software networks is the propagation of configuration. It is important to identify the relations between configurations of network components. A single configuration could require a chain reaction of other changes in network components to keep the network state coherent. As future works, we intend to overcome this challenge and investigate approaches that model automatically the dependencies between the configurations in software network.

We also intend as future works to enhance the contributions made in this thesis. Regarding our semantic framework for handling the heterogeneity between configuration data models, we highlighted that using the ontologies as a way to represent and merge the knowledge is an efficient solution. However, relying solely on predefined sources of information like a global ontology or a lexicon to map the heterogeneous data models is not a viable solution. The efficiency of the mapping in this case relies heavily on the predefined sources of information. We need therefore to investigate other ways to map the ontologies together without the need of relying on predefined sources of information. We intend for this matter to investigate solutions based on machine learning and deep learning to map the ontologies together. That is because these types of solutions have proven to be efficient in the literature to learn various tasks in an unsupervised manner.

Regarding our learning framework for automating the configuration generation, we have identified several aspects that need to be improved. The first improvement that has to be made is to consider the behavioral aspect in the VNF descriptors and generalize this approach to all the configuration files in software networks. We considered in this work only the structural aspect of the VNF descriptors. The behavior aspect includes the scripts of the VNF descriptors that are used by the service providers to define the policies of the VNF descriptors and the operations related to handling their life cycle. This aspect adds more complexity to the learning

framework. This is because the scripts are written in a different language than the data model used to define the VNF descriptors and will require the leaning framework to be trained on different aspects at the same time in order for it to generate a coherent VNF descriptor. We also intend to develop a descriptor engine that could be used in real life by service providers or NFV platforms to generate and recommend VNF descriptors.

Finally, regarding our model-driven approach for assisting the deployment descriptor generation. We are working on developing a tool that learns from a set of VNF descriptors the configurable model. The tool could be interactively used by service providers to generate the VNF descriptors. We also aim to incorporate in this tool methods that can use the configurable model to learn the dependencies between the elements of the VNF descriptors, generate different variations of the VNF descriptors, learn and detect errors in the VNF descriptors and automatically generate the VNF descriptors. Similarly the learning framework, we also aim to generalize this approach to all the configuration files in software networks and also take into consideration the behavioral aspect in the configuration files in the configurable model.

Bibliography

[1] common information model onf cim. https://www.opennetworking.org/news-and-events/blog/common-information-model, accessed: 2017-12-05

[2] Topology and orchestration specification for cloud applications version 1.0. http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html (January 2013)

[3] Data-efficient performance learning for configurable systems (2017)

[4] (April 2020), https://qmonnet.github.io/whirl-offload/2016/07/08/introduction-to-sdn/

[5] (April 2020), http://yuba.stanford.edu//

[6] (April 2020), https://qmonnet.github.io/whirl-offload/2016/07/08/introduction-to-sdn/

[7] Afia, A.E., Sarhani, M.: Performance prediction using support vector machine for the configuration of optimization algorithms. In: 2017 3rd International Conference of Cloud Computing Technologies and Applications (CloudTech). pp. 1–7 (2017)

[8] Agrawal, R., Imielinski, T., Swami, A.N.: Mining association rules between sets of items in large databases. In: ACM SIGMOD'93. pp. 207–216

[9] Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Proceedings of the 20th International Conference on Very Large Data Bases, pp. 487–499. VLDB '94 (1994)

[10] Alipourfard, O., Liu, H.H., Chen, J., Venkataraman, S., Yu, M., Zhang, M.: Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In: 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17). pp. 469–482. USENIX Association, Boston, MA (Mar 2017), https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/alipourfard

[11] Assy, N.: Automated support of the variability in configurable process models. Ph.D. thesis, Université Paris-Saclay (Sep 2015)

[12] Assy, N., Chan, N.N., Gaaloul, W.: An automated approach for assisting the design of configurable process models. IEEE Trans. Services Computing 8(6), 874–888 (2015)

[13] Bae, J., Liu, L., Caverlee, J., Rouse, W.B.: Process mining, discovery, and integration using distance measures. In: 2006 IEEE Int. Conf. on Web Services (ICWS'06). pp. 479–488 (Sep 2006)

[14] Bao, L., Liu, X., Xu, Z., Fang, B.: Autoconfig: Automatic configuration tuning for distributed message systems. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. p. 29–40. ASE 2018, Association for Computing Machinery, New York, NY, USA (2018), https://doi.org/10.1145/3238147.3238175

[15] Benamrane, F., mamoun], M.B., Benaini, R.: An east-west interface for distributed sdn control plane: Implementation and evaluation. Computers Electrical Engineering 57, 162 – 175 (2017), http://www.sciencedirect.com/science/article/pii/S0045790616302798

[16] Berde, P., Gerola, M., Hart, J., Higuchi, Y., Kobayashi, M., Koide, T., Lantz, B., O'Connor, B., Radoslavov, P., Snow, W., Parulkar, G.: Onos: Towards an open, distributed sdn os. In: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking. pp. 1–6. HotSDN '14, ACM, New York, NY, USA (2014), http://doi.acm.org/10.1145/2620728.2620744

[17] Blial, O., Mamoun, M.B., Benaini, R.: An overview on sdn architectures with multiple controllers. In: Journal of Computer Networks and Communications (2016)

[18] Boden, B., Ester, M., Seidl, T.: Density-based subspace clustering in heterogeneous networks. In: Calders, T., Esposito, F., Hüllermeier, E., Meo, R. (eds.) Machine Learning and Knowledge Discovery in Databases. pp. 149–164. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)

[19] Bohring, H., Auer, S.: Mapping xml to owl ontologies. In: Leipziger Informatik-Tage, volume 72 of LNI. pp. 147–156. GI (2005)

[20] Cao, B., Kong, X., Yu, P.S.: Collective prediction of multiple types of links in heterogeneous information networks. In: 2014 IEEE International Conference on Data Mining. pp. 50–59 (2014)

[21] Cao, B., Kong, X., Yu, P.S.: Collective prediction of multiple types of links in heterogeneous information networks. In: 2014 IEEE International Conference on Data Mining. pp. 50–59 (2014)

[22] Cardoso, J., Mendling, J., Neumann, G., Reijers, H.A.: A discourse on complexity of process models. In: Proc. Int. Conf. on Business Process Management Workshops. pp. 117–128. BPM'06, Springer (2006)

[23] Chachada, S., Kuo, C..J.: Environmental sound recognition: A survey. In: 2013 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference. pp. 1–9 (2013)

[24] Cheatham, M., Pesquita, C.: Semantic Data Integration, pp. 263–305. Springer International Publishing, Cham (2017), https://doi.org/10.1007/978-3-319-49340-4_8

[25] Chen, J., Gao, H., Wu, Z., Li, D.: Tag co-occurrence relationship prediction in heterogeneous information networks. In: 2013 International Conference on Parallel and Distributed Systems. pp. 528–533 (2013)

[26] Chen, S., Liu, Y., Gorton, I., Liu, A.: Performance prediction of component-based applications. Journal of Systems and Software 74(1), 35 – 43 (2005), http://www.sciencedirect.com/science/article/pii/S0164121203003200, automated Component-Based Software Engineering

[27] Cooklev, T.: Making software-defined networks semantic. In: 2015 12th International Joint Conference on e-Business and Telecommunications (ICETE). vol. 06, pp. 48–52 (July 2015)

[28] Dabre, R., Chu, C., Kunchukuttan, A.: A comprehensive survey of multilingual neural machine translation (2020)

[29] Dargahi, T., Caponi, A., Ambrosin, M., Bianchi, G., Conti, M.: A survey on the security of stateful sdn data planes. IEEE Communications Surveys Tutorials 19(3), 1701–1725 (2017)

[30] Ding, Y., Ansel, J., Veeramachaneni, K., Shen, X., O'Reilly, U.M., Amarasinghe, S.: Autotuning algorithmic choice for input sensitivity. In: Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation. p. 379–390. PLDI '15, Association for Computing Machinery, New York, NY, USA (2015), https://doi.org/10.1145/2737924.2737969

[31] Dixit, A., Hao, F., Mukherjee, S., Lakshman, T., Kompella, R.: Towards an elastic distributed sdn controller. SIGCOMM Comput. Commun. Rev. 43(4), 7–12 (Aug 2013), http://doi.acm.org/10.1145/2534169.2491193

[32] Doan, A., Madhavan, J., Domingos, P., Halevy, A.: Learning to map between ontologies on the semantic web. In: Proceedings of the 11th International Conference on World Wide Web. p. 662–673. WWW '02, Association for Computing Machinery, New York, NY, USA (2002), https://doi.org/10.1145/511446.511532

[33] Duarte, F., Gil, R., Romano, P., Lopes, A., Rodrigues, L.: Learning non-deterministic impact models for adaptation. In: 2018 IEEE/ACM 13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS). pp. 196–205 (2018)

[34] Duarte, F., Gil, R., Romano, P., Lopes, A., Rodrigues, L.: Learning non-deterministic impact models for adaptation. In: 2018 IEEE/ACM 13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS). pp. 196–205 (2018)

[35] Esposito, F., Fanizzi, N., d'Amato, C.: Recovering uncertain mappings through structural validation and aggregation with the moto system. In: Proceedings of the 2010 ACM Symposium on Applied Computing. p. 1428–1432. SAC '10, Association for Computing Machinery, New York, NY, USA (2010), https://doi.org/10.1145/1774088.1774390

[36] Essayeh, A., Abed, M.: Towards ontology matching based system through terminological, structural and semantic level. Procedia Computer Science 60, 403 – 412 (2015), http://www.sciencedirect.com/science/article/pii/S1877050915022814, knowledge-Based and Intelligent Information Engineering Systems 19th Annual Conference, KES-2015, Singapore, September 2015 Proceedings

[37] Essayeh, A., Abed, M.: Towards ontology matching based system through terminological, structural and semantic level. Procedia Computer Science 60, 403 – 412 (2015), http://www.sciencedirect.com/science/article/pii/S1877050915022814, knowledge-Based and Intelligent Information Engineering Systems 19th Annual Conference, KES-2015, Singapore, September 2015 Proceedings

[38] ETSI: Gs nfv-ifa 011. https://standards.globalspec.com/std/13271186/gs-nfv-ifa-011 (2019)

[39] ETSI: Gs nfv-man 001 (apr 2020), https://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf

[40] Foundation, O.N.: (April 2020), https://www.opennetworking.org/software-defined-standards/specifications/

[41] Fu, X., Budzik, J., Hammond, K.J.: Mining Navigation History for Recommendation. In: IUI '00, pp. 106–112 (2000)

[42] Ghosh, S., Kristensson, P.O.: Neural networks for text correction and completion in keyboard decoding. CoRR abs/1709.06429 (2017), http://arxiv.org/abs/1709.06429

[43] Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., Shenker, S.: Nox: Towards an operating system for networks. SIGCOMM Comput. Commun. Rev. 38(3), 105–110 (Jul 2008), http://doi.acm.org/10.1145/1384609.1384625

[44] Guo, J., Czarnecki, K., Apely, S., Siegmundy, N., Wasowski, A.: Variability-aware performance prediction: A statistical learning approach. In: Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering. p. 301–311. ASE'13, IEEE Press (2013), https://doi.org/10.1109/ASE.2013.6693089

[45] Hacherouf, M., Bahloul, S.N., Cruz, C.: Transforming xml documents to owl ontologies: A survey. J. Inf. Sci. 41(2), 242–259 (Apr 2015), http://dx.doi.org/10.1177/0165551514565972

[46] Hajmoosaei, A., Abdul-Kareem, S.: An approach for mapping of domain-based local ontologies. In: 2008 International Conference on Complex, Intelligent and Software Intensive Systems. pp. 865–870 (March 2008)

[47] Hassas Yeganeh, S., Ganjali, Y.: Kandoo: A framework for efficient and scalable offloading of control applications. In: Proceedings of the First Workshop on Hot Topics in Software Defined Networks. pp. 19–24. HotSDN '12, ACM, New York, NY, USA (2012), http://doi.acm.org/10.1145/2342441.2342446

[48] He, J., Bailey, J., Zhang, R.: Exploiting transitive similarity and temporal dynamics for similarity search in heterogeneous information networks. In: Bhowmick, S.S., Dyreson, C.E., Jensen, C.S., Lee, M.L., Muliantara, A., Thalheim, B. (eds.) Database Systems for Advanced Applications. pp. 141–155. Springer International Publishing, Cham (2014)

[49] Helebrandt, P., Kotuliak, I.: Novel sdn multi-domain architecture. In: 2014 IEEE 12th IEEE International Conference on Emerging eLearning Technologies and Applications (ICETA). pp. 139–143 (Dec 2014)

[50] Hou U., L., Yao, K., Mak, H.F.: Pathsimext: Revisiting pathsim in heterogeneous information networks. In: Li, F., Li, G., Hwang, S.w., Yao, B., Zhang, Z. (eds.) Web-Age Information Management. pp. 38–42. Springer International Publishing, Cham (2014)

[51] Huang, H., Zubiaga, A., Ji, H., Deng, H., Wang, D., Le, H., Abdelzaher, T., Han, J., Leung, A., Hancock, J., Voss, C.: Tweet ranking based on heterogeneous networks. In: Proceedings of COLING 2012. pp. 1239–1256. The COLING 2012 Organizing Committee, Mumbai, India (Dec 2012), https://www.aclweb.org/anthology/C12-1076

[52] Iovanna, P., Ubaldi, F.: Sdn solutions for 5g transport networks. In: 2015 International Conference on Photonics in Switching (PS). pp. 297–299 (2015)

[53] Jacob, Y., Denoyer, L., Gallinari, P.: Learning latent representations of nodes for classifying in heterogeneous social networks. In: Proceedings of the 7th ACM International Conference on Web Search and Data Mining. p. 373–382. WSDM '14, Association for Computing Machinery, New York, NY, USA (2014), https://doi.org/10.1145/2556195.2556225

[54] Jamshidi, P., Siegmund, N., Velez, M., Kästner, C., Patel, A., Agarwal, Y.: Transfer learning for performance modeling of configurable systems: An exploratory analysis (2017)

[55] Jamshidi, P., Velez, M., Kästner, C., Siegmund, N.: Learning to sample: Exploiting similarities across environments to learn performance models for configurable systems. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. p. 71–82. ESEC/FSE 2018, Association for Computing Machinery, New York, NY, USA (2018), https://doi.org/10.1145/3236024.3236074

[56] Ji, M., Sun, Y., Danilevsky, M., Han, J., Gao, J.: Graph regularized transductive classification on heterogeneous information networks. In: Balcázar, J.L., Bonchi, F., Gionis, A., Sebag, M. (eds.) Machine Learning and Knowledge Discovery in Databases. pp. 570–586. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)

[57] Jmila, H., Khedher, M.I., El Yacoubi, M.A.: Estimating vnf resource requirements using machine learning techniques. In: Neural Information Processing. pp. 883–892. Springer, Cham (2017)

[58] Katsalis, K., Nikaein, N., Edmonds, A.: Multi-domain orchestration for nfv: Challenges and research directions. In: 15th Inter. Conf. on Ubiquitous Comp. and Commun. pp. 189–195 (Dec 2016)

[59] Kim, Y.: Convolutional neural networks for sentence classification. CoRR abs/1408.5882 (2014), http://arxiv.org/abs/1408.5882

[60] Kleinberg, J.M.: Authoritative sources in a hyperlinked environment. J. ACM 46(5), 604–632 (Sep 1999), https://doi.org/10.1145/324133.324140

[61] Kolesnikov, S., Siegmund, N., Kästner, C., Grebhahn, A., Apel, S.: Tradeoffs in modeling performance of highly configurable software systems. Software & Systems Modeling 18(3), 2265–2283 (Jun 2019), https://doi.org/10.1007/s10270-018-0662-9

[62] Koponen, T., Casado, M., Gude, N., Stribling, J., Poutievski, L., Zhu, M., Ramanathan, R., Iwata, Y., Inoue, H., Hama, T., Shenker, S.: Onix: A distributed control platform for large-scale production networks. In: Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation. pp. 351–364. OSDI'10, USENIX Association, Berkeley, CA, USA (2010), http://dl.acm.org/citation.cfm?id=1924943.1924968

[63] Kreutz, D., Ramos, F.M.V., Veríssimo, P.E., Rothenberg, C.E., Azodolmolky, S., Uhlig, S.: Software-defined networking: A comprehensive survey. Proceedings of the IEEE 103(1), 14–76 (2015)

[64] Lee, H., Lee, B., Park, K., Elmasri, R.: Fusion techniques for reliable information: A survey

[65] Li, C., Sun, J., Xiong, Y., Zheng, G.: An efficient drug-target interaction mining algorithm in heterogeneous biological networks. In: Peng, W.C., Wang, H., Bailey, J., Tseng, V.S., Ho, T.B., Zhou, Z.H., Chen, A.L. (eds.) Trends and Applications in Knowledge Discovery and Data Mining. pp. 65–76. Springer International Publishing, Cham (2014)

[66] Li, Y., Chen, M.: Software-defined network function virtualization: A survey. IEEE Access 3, 2542–2553 (2015)

[67] Lin, P., Bi, J., Wang, Y.: East-west bridge for sdn network peering. In: Su, J., Zhao, B., Sun, Z., Wang, X., Wang, F., Xu, K. (eds.) Frontiers in Internet Technologies. pp. 170–181. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)

[68] Lin, W., Alvarez, S.A., Ruiz, C.: Collaborative recommendation via adaptive association rule mining. In: Data Mining and Knowledge Discovery (2000)

[69] List, B., Korherr, B.: An evaluation of conceptual business process modelling languages. In: Proc. ACM Symp. on Applied Computing. pp. 1532–1539. SAC '06, ACM (2006)

[70] Liu, H.: Conditioning lstm decoder and bi-directional attention based question answering system (2019)

[71] Llorens-Carrodeguas, A., Cervelló-Pastor, C., Leyva-Pupo, I.: A data distribution service in a hierarchical sdn architecture: Implementation and evaluation. 2019 28th International Conference on Computer Communication and Networks (ICCCN) pp. 1–9 (2019)

[72] Luo, C., Guan, R., Wang, Z., Lin, C.: Hetpathmine: A novel transductive classification algorithm on heterogeneous information networks. In: de Rijke, M., Kenter, T., de Vries, A.P., Zhai, C., de Jong, F., Radinsky, K., Hofmann, K.

(eds.) Advances in Information Retrieval. pp. 210–221. Springer International Publishing, Cham (2014)

[73] Luo, C., Pang, W., Wang, Z.: Semi-supervised clustering on heterogeneous information networks. In: Tseng, V.S., Ho, T.B., Zhou, Z.H., Chen, A.L.P., Kao, H.Y. (eds.) Advances in Knowledge Discovery and Data Mining. pp. 548–559. Springer International Publishing, Cham (2014)

[74] Makaya, C., Freimuth, D.: Automated virtual network functions onboarding. In: IEEE Conf. on Network Function Virtualization and Software Defined Networks. pp. 206–211 (Nov 2016)

[75] Martinez, A., Yannuzzi, M., de Vergara, J.E.L., Serral-Gracià, R., Ramírez, W.: An ontology-based information extraction system for bridging the configuration gap in hybrid sdn environments. In: 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM). pp. 441–449 (May 2015)

[76] Meng, X., Shi, C., Li, Y., Zhang, L., Wu, B.: Relevance measure in large-scale heterogeneous networks. In: Chen, L., Jia, Y., Sellis, T., Liu, G. (eds.) Web Technologies and Applications. pp. 636–643. Springer International Publishing, Cham (2014)

[77] Mijumbi, R., Serrat, J., Gorricho, J., Bouten, N., De Turck, F., Boutaba, R.: Network function virtualization: State-of-the-art and research challenges. IEEE Communications Surveys Tutorials 18(1), 236–262 (2016)

[78] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2. p. 3111–3119. NIPS'13, Curran Associates Inc., Red Hook, NY, USA (2013)

[79] Movahedi, Z., Ayari, M., Langar, R., Pujolle, G.: A survey of autonomic network architectures and evaluation criteria. IEEE Communications Surveys Tutorials 14(2), 464–490 (2012)

[80] Nguyen, T., Yoo, M.: A vnf descriptor generator for tacker-based nfv management and orchestration. In: In Proc. Int. Conf. on Information and Communication Technology Convergence (ICTC). pp. 260–262 (Oct 2018)

[81] Nguyen, T.T.A., Conrad, S.: Ontology matching using multiple similarity measures. In: Proceedings of the International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management. p. 603–611. IC3K 2015, SCITEPRESS - Science and Technology Publications, Lda, Setubal, PRT (2015), https://doi.org/10.5220/0005615606030611

[82] Odarchenko, R., Tkalich, O., Konakhovych, G., Abakumova, A.: Evaluation of sdn network scalability with different management level structure. In: 2016 Third International Scientific-Practical Conference Problems of Infocommunications Science and Technology (PIC S T). pp. 128–131 (2016)

[83] Otter, D.W., Medina, J.R., Kalita, J.K.: A survey of the usages of deep learning in natural language processing. CoRR abs/1807.10854 (2018), http://arxiv.org/abs/1807.10854

[84] Ouali, I., Ghozzi, F., Taktak, R., Sassi], M.S.H.: Ontology alignment using stable matching. Procedia Computer Science 159, 746 – 755 (2019), http://www.sciencedirect.com/science/article/pii/S1877050919314164, knowledge-Based and Intelligent Information Engineering Systems: Proceedings of the 23rd International Conference KES2019

[85] Ouyang, X., Zhang, X., Ma, D., Agam, G.: Generating image sequence from description with lstm conditional gan (2018)

[86] Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab (November 1999), http://ilpubs.stanford.edu:8090/422/, previous number = SIDL-WP-1999-0120

[87] Parundekar, R., Knoblock, C.A., Ambite, J.L.: Linking and building ontologies of linked data. In: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (eds.) The Semantic Web – ISWC 2010. pp. 598–614. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)

[88] Parundekar, R., Knoblock, C.A., Ambite, J.L.: Discovering concept coverings in ontologies of linked data sources. In: Cudré-Mauroux, P., Heflin, J., Sirin, E., Tudorache, T., Euzenat, J., Hauswirth, M., Parreira, J.X., Hendler, J., Schreiber, G., Bernstein, A., Blomqvist, E. (eds.) The Semantic Web – ISWC 2012. pp. 427–443. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)

[89] Pereira, J.A., Martin, H., Acher, M., Jézéquel, J.M., Botterweck, G., Ventresque, A.: Learning software configuration spaces: A systematic literature review (2019)

[90] Phemius, K., Bouet, M., Leguay, J.: Disco: Distributed multi-domain sdn controllers. In: 2014 IEEE Network Operations and Management Symposium (NOMS). pp. 1–4 (May 2014)

[91] Popescul, A., Popescul, R., Ungar, L.H.: Statistical relational learning for link prediction (2003)

[92] Qi, H., Li, K.: Management System of Heterogeneous Software-Defined Networking Controllers, pp. 57–65. Springer International Publishing, Cham (2016), https://doi.org/10.1007/978-3-319-33135-5_4

[93] Recker, J., Rosemann, M., van der Aalst, W.M.P., Mendling, J.: On the syntax of reference model configuration - transforming the C-EPC into lawful EPC models. In: Business Process Management Workshops, BPM 2005 International Workshops, BPI, BPD, ENEI, BPRM, WSCOBPM, BPS, Nancy, France, September 5, 2005, Revised Selected Papers. pp. 497–511 (2005)

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

[95] Reijers, H.A., Vanderfeesten, I.T.P.: Cohesion and coupling metrics for workflow process design. In: Business Process Management. pp. 290–305. Springer (2004)

[96] Ren, P., Wang, X., Zhao, B., Wu, C., Sun, H.: Opensrn: A software-defined semantic routing network architecture. In: 2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). pp. 101–102 (April 2015)

[97] Riccobene, V., McGrath, M.J., Kourtis, M., Xilouris, G., Koumaras, H.: Automated generation of vnf deployment rules using infrastructure affinity characterization. In: IEEE NetSoft Conf. and Workshops. pp. 226–233 (June 2016)

[98] Rolón, E., Cardoso, J., García, F., Ruiz, F., Piattini, M.: Analysis and validation of control-flow complexity measures with bpmn process models. In: Halpin, T., Krogstie, J., Nurcan, S., Proper, E., Schmidt, R., Soffer, P., Ukor, R. (eds.) Enterprise, Business-Process and Information Systems Modeling. pp. 58–70. Springer (2009)

[99] Rosemann, M., van der Aalst, W.M.P.: A configurable reference modelling language. Inf. Syst. 32(1), 1–23 (2007)

[100] Santra, B., Mukherjee, D.P.: A comprehensive survey on computer vision based approaches for automatic identification of products in retail store. Image and Vision Computing 86, 45 – 63 (2019), http://www.sciencedirect.com/science/article/pii/S0262885619300277

[101] Shi, C., Kong, X., Huang, Y., S. Yu, P., Wu, B.: Hetesim: A general framework for relevance measure in heterogeneous networks. IEEE Transactions on Knowledge and Data Engineering 26(10), 2479–2492 (2014)

[102] Shi, C., Kong, X., Huang, Y., Yu, P.S., Wu, B.: Hetesim: A general framework for relevance measure in heterogeneous networks (2013)

[103] Shi, C., Li, Y., Zhang, J., Sun, Y., Yu, P.S.: A survey of heterogeneous information network analysis (2015)

[104] de Sousa, N.F.S., Perez, D.A.L., Rosa, R.V., Santos, M.A.S., Rothenberg, C.E.: Network service orchestration: A survey. arXiv:1803.06596 (2018)

[105] de Sousa, N.F.S., Perez, D.A.L., Rosa, R.V., Santos, M.A.S., Rothenberg, C.E.: Network service orchestration: A survey. CoRR abs/1803.06596 (2018)

[106] Staar, B., Lütjen, M., Freitag, M.: Anomaly detection with convolutional neural networks for industrial surface inspection. Procedia CIRP 79, 484 – 489 (2019), http://www.sciencedirect.com/science/article/pii/S2212827119302409, 12th CIRP Conference on Intelligent Computation in Manufacturing Engineering, 18-20 July 2018, Gulf of Naples, Italy

[107] Sun, Y., Barber, R., Gupta, M., Aggarwal, C.C., Han, J.: Co-author relationship prediction in heterogeneous bibliographic networks. In: 2011 International Conference on Advances in Social Networks Analysis and Mining. pp. 121–128 (2011)

[108] Sun, Y., Aggarwal, C.C., Han, J.: Relation strength-aware clustering of heterogeneous information networks with incomplete attributes. Proc. VLDB Endow. 5(5), 394–405 (Jan 2012), https://doi.org/10.14778/2140436.2140437

[109] Sun, Y., Han, J., Aggarwal, C.C., Chawla, N.V.: When will it happen? relationship prediction in heterogeneous information networks. In: Proceedings of the Fifth ACM International Conference on Web Search and Data Mining. p. 663–672. WSDM '12, Association for Computing Machinery, New York, NY, USA (2012), https://doi.org/10.1145/2124295.2124373

[110] Sun, Y., Han, J., Yan, X., Yu, P.S., Wu, T.: Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. Proc. VLDB Endow. 4, 992–1003 (2011)

[111] The OpenDaylight Project, Inc.: OpenDaylight - Technical Overview (2013), http://www.opendaylight.org/project/technical-overview

[112] Tootoonchian, A., Ganjali, Y.: Hyperflow: A distributed control plane for openflow. In: Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking. pp. 3–3. INM/WREN'10, USENIX Association, Berkeley, CA, USA (2010), http://dl.acm.org/citation.cfm?id=1863133.1863136

[113] Tsai, M.H., Aggarwal, C., Huang, T.: Ranking in heterogeneous social media. In: Proceedings of the 7th ACM International Conference on Web Search and Data Mining. p. 613–622. WSDM '14, Association for Computing Machinery, New York, NY, USA (2014), https://doi.org/10.1145/2556195.2556254

[114] Tsou, T., Aranda, P.A., Xie, H., Sidi, R., Yin, H., López, D.: Sdni: A message exchange protocol for software defined networks (sdns) across multiple domains (2012)

[115] Venugopalan, S., Rohrbach, M., Donahue, J., Mooney, R.J., Darrell, T., Saenko, K.: Sequence to sequence - video to text. CoRR abs/1505.00487 (2015), http://arxiv.org/abs/1505.00487

[116] Wang, C., Song, Y., El-Kishky, A., Roth, D., Zhang, M., Han, J.: Incorporating world knowledge to document clustering via heterogeneous information networks. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. p. 1215–1224. KDD '15, Association for Computing Machinery, New York, NY, USA (2015), https://doi.org/10.1145/2783258.2783374

[117] Wang, Q., Peng, Z., Wang, S., Yu, P.S., Li, Q., Hong, X.: clutm: Content and link integrated topic model on heterogeneous information networks. In: Dong, X.L., Yu, X., Li, J., Sun, Y. (eds.) Web-Age Information Management. pp. 207–218. Springer International Publishing, Cham (2015)

[118] Wang, W., Qi, Q., Gong, X., Hu, Y., Que, X.: Autonomic qos management mechanism in software defined network. China Communications 11(7), 13–23 (2014)

[119] Yu, H., Li, K., Qi, H., Li, W., Tao, X.: Zebra: An east-west control framework for sdn controllers. In: 2015 44th International Conference on Parallel Processing. pp. 610–618 (Sept 2015)

[120] Yu, X., Gu, Q., Zhou, M., Han, J.: Citation Prediction in Heterogeneous Bibliographic Networks, pp. 1119–1130. https://locus.siam.org/doi/abs/10.1137/1.9781611972825.96

[121] Zhang, J., Kong, X., Luo, R.J., Chang, Y., Yu, P.S.: Ncr: A scalable network-based approach to co-ranking in question-and-answer sites. In: Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management. p. 709–718. CIKM '14, Association for Computing Machinery, New York, NY, USA (2014), https://doi.org/10.1145/2661829.2661978

[122] Zhang, S., Yao, L., Sun, A., Tay, Y.: Deep learning based recommender system. ACM Computing Surveys 52(1), 1–38 (Feb 2019), http://dx.doi.org/10.1145/3285029

[123] Zhang, Y., Cui, L., Wang, W., Zhang, Y.: A survey on software defined networking with multiple controllers. Journal of Network and Computer Applications 103, 101 – 118 (2018)