SELF-RECONFIGURABLE SOFTWARE-DEFINED NETWORKS BASED ON AUTOMATED PLANNING

ANGELA MARÍA RODRÍGUEZ VIVAS

Doctoral Thesis in Telematics Engineering

Advisor:
Oscar Mauricio Caicedo Rendón
Ph.D. in Computer Science

Co-Advisor:
Jéferson Campos Nobre
Ph.D. in Computer Science
Federal University of Rio Grande do Sul, Brazil

University of Cauca
Faculty of Electronic and Telecommunications Engineering
Department of Telematics
Line of Research in Advanced Services of Telecommunications
Popayan, December 2022

ANGELA MARÍA RODRÍGUEZ VIVAS

SELF-RECONFIGURABLE SOFTWARE-DEFINED NETWORKS BASED ON AUTOMATED PLANNING

Thesis presented to the Faculty of Electronic
and Telecommunications Engineering of the
University of Cauca to obtain the degree of

Doctor of Philosophy in:
Telematics Engineering

Advisor:
Prof. Dr. Oscar Mauricio Caicedo Rendón

Co-Advisor:
Prof. Dr. Jéferson Campos Nobre

Popayan
2022

To my two little ones, Jacobo and Jerónimo, whose births during my phd studies were my precious motivation to keep going during all those stressful moments.

Acknowledgements

I thank God, the faith I have in him is my strength in academics, and all aspects of my daily live.

I thank my beloved parents, Teresa and Nelson (RIP), they were a perfect complement in my training for life. Thanks to their dedication and affection, now I achieve this crucial step in my professional life.

I thank my husband, my unconditional partner, Jaime Alfredo, my in-laws, Alba Alicia and Alfredo Jose. I did the thesis but we all made the effort.

I thank my sisters Diana Marcela and Maria Cecilia, my niece Amaranta, my aunt and godmother Ana Cecilia. My valued women, each one with its character has given me important life lessons.

I am very grateful to my excellent advisor, Professor Oscar Mauricio Caicedo Rendon, his knowledge and dedication were invaluable factors for the successful development of my thesis.

Thanks to my professors at the Federal University of Rio Grande do Sul (UFRGS), Jéferson Campos Nobre and Lisandro Zambendetti Granvile, for welcoming me at the inf lab and guiding me during the first steps of this research.

I thank my great friend and colleague Jose Armando, who led me on the research way, and more than a colleague has been my mentor, always attentive to my concerns.

My thanks to professors and employees of the Department of Telematics of the University of Cauca, who helped me and gave me the opportunity of developing my doctoral studies.

Thanks to my folks in the 116 IPET room at the University of Cauca, for all those off-topic discussions, jokes, and laughs during the stressful working days.

Finally, I thank to the Colombian Ministry of Science Technology and Innovation that funded this thesis under Grant 727-2015.

Structured abstract

Background. 5G networks come with the promise to serve a wide range of use cases, each of which features a set of diverse and stringent requirements. By leveraging the potential of Software Defined Networks and Networks Functions Virtualization, a common infrastructure is sliced into multiple logical networks according to the diversified service demands, enabling the implementation of 5G network slicing. As any emerging networking technology, 5G network slicing imposes new challenges to the already complex network management. For instance, reconfiguration operations must be split between the Infrastructure Provider (InP) and the tenants leasing network slices. However, since tenants are not networking professionals, a bespoke reconfiguration system is required allowing them to timely reconfigure network slices meeting end-user demands, with minimal dependance on InP.
Aims. This thesis introduces an approach based on Automated Planning for achieving self-configuration in SDN. The main objective of the thesis is divided into three parts: *(i)* Design an approach based on automated planning for providing self-configuration in SDN, *(ii)* Present a reference implementation of the proposed approach, and *(iii)* Evaluate the proposed approach regarding time-response, time-planning, and network overhead.
Methods. This thesis proposes an approach based on Automated Planning for achieving self-configuration in SDN. Two main components are integrated for using embedded intelligence to make reconfiguration decisions and enforce decision-making governed by tenant *intents* . First, an approach named NORA, automatically transforms high-level network management policies expressed by the tenants in natural language (NL), to the *AP-goal* . For the operation of NORA, a dataset of NL network management policies has been constructed, and we resort to natural language processing to process them. Also, NORA merges the obtained *AP-goal* with initial network state (coming from Monitoring Analysis elements) and a *AP-domain* representing the network model trough *AP-problem* templates and generates an *AP-problem* in an AP language like PDDL or Strips. Second, an approach named ATRAP, with AP as the decision-making technique in a MAPE-K-based ACL to resolve the tenant-side network slice reconfiguration problem. Thanks to the design of a detailed model based on State Transition System for the tenant-side network slice reconfiguration problem, an AI-planner is able to compute reconfiguration plans on-the-fly to turn the network slicing from an -undesired- configuration state into a configuration state complying tenant *intents* .
Results. The evaluation of the thesis is twofold: NORA and ATRAP. The metrics for

evaluating NORA were precision, which allows measuring whether it produces precise translations, and processing time, which allows measuring its speed for generating planning problems. The NORA precision and processing time results position it as a promising solution to generate *AP-problems* needed to close the autonomic management loops that allow realizing self-driving networks. On the other hand, the metrics for evaluating ATRAP were planning time, which allows measuring the time taken by the AI-planner to compute a reconfiguration plan, and plan-length, which allows measuring the actions (*i.e.*, VNF migrations) composing a plan, *i.e.*, the reconfiguration cost given a plan. The ATRAP evaluation results revealed that the planning time is strongly committed to the dimensions of the networks involved in the reconfiguration, *i.e.*, the virtualized network and the SN. Conversely, the plan length is independent of the network size and is promising for the tenant since scarce reconfiguration actions are involved in each computed plan.

Conclusions. Traditional human-driven network management methods are inapplicable to emerging 5G network slicing due to the ever increasing diversity and dynamism of networking demands (regarding end-users, types of devices and services, etc.), which makes network reconfiguration increasingly complex. The introduction of autonomic reconfiguration approaches (*i.e.*, closed loop-based and governed by network management *intents*) is thus essential. Therefore, the proposed thesis exploiting AP as the reconfiguration decision making technique in a MAPE-K-based architecture governed by *intents* representing management policies defined by tenants is an attractive solution for achieving self-configurable SDNs.
Keywords: 5G networks, Autonomic Network Management, Network Management Intents, Automated Planning, Software Defined Networks.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

$5GNS$	5G Network Slice
ACL	Autonomic Control Loop
AI	Artificial Intelligence
AP	Automated Planning
CNL	Controlled Natural Language
DSL	Domain Service Language
HLR	High-Level Requirement
InP	Infrastructure Provider
$MAPE$	Monitoring-Analysis-Planning-Execution
ML	Machine Learning
NA	Network Analytics
NBI	NorthBound Interface
NFV	Network Functions Virtualization
NL	Natural Language
NLP	Natural Language Processing
NMS	Network Management System
$PBNM$	Policy-based Network Management
$PDDL$	Planning Domain Definition Language
SDN	Software Defined Networks
SLA	Service Level Agreement
SN	Substrate Network

Chapter 1

Introduction

1.1 Problem statement

Software-Defined Networking (SDN) emerged to face the need of flexible and dynamic paradigms to control and manage the (increasingly complex) communication networks. SDN refers to the ability of software programs to dynamically control individual network devices and the behavior of the network as a whole [1]. Regarding its programmable aptness, SDN has been pointed out as a tool to simplify classical management tasks (*i.e.*, FCAPS) [2, 3]. Nevertheless, a way to manage the specific functionalities of SDN planes (*i.e.*, Data, Control, and Application) was not considered in early definitions [4] [5]. In particular, network reconfiguration is one of the most usual management tasks carried out by network operators [6] due to changes after network deployment, like expansion or retraction of topology, and addition of devices and services. Such changes are triggered by dynamism in user and service requirements, business rules (*e.g.*, new provider for network devices and placement of middleboxes and load balancers), context conditions (*e.g.*, users mobility and addition of end users), and SDN programmability.

In SDN, network properties (*e.g.*, monitoring services) can be manipulated by software programs to modify network behavior in a much more flexible way than in conventional non-programmable scenarios [7]. However, such modifications (*e.g.*, addition/-modifications of controller modules) may turn the network into unstable states, because, despite its benefits, the SDN programmability does not guarantee compliance of management policies. It is an administrator task to reconfigure SDN on-the-fly to return it to a stable state. Such reconfiguration entails computation of proper configuration alternatives. For instance, if an SDN controller is added to the network leading to a two-size controllers cluster, the administrator has $2^n - 2$ possibilities to interconnect switches to controllers, where *n* is the number of switches. It is noteworthy that, if responses from centralized management systems or human operation are fairly slow, resulting network instability may expand quickly [6], impacting negatively on performance.

The use of the self-configuration property of the Autonomic Network Management (ANM) paradigm has been explored to overcome limits of manual intervention (*e.g.*, time-consuming, error-prone, and hard-to-scale) in network configuration [8] [9]. In ANM, the network status is continuously monitored and analyzed to detect abnormal behavior and turn high-level policies into executable management services with minute human intervention. This is usually achieved thanks to the operation of Autonomic Control Loops (ACLs). Self-configuration is the property of ANM with which network components configure and reconfigure themselves in a dynamic environment, steered by high-level policies [8]. Likewise, network components are able to install, set up, and integrate seamlessly into the network without (or minimally) disturbing its performance, with minimal human intervention [10] [11].

A well-known realization of ACLs is MAPE-K, the approach proposed by IBM [11]. MAPE-K loop chains components for Monitoring, Analyzing, Planning, and Executing, all of which rely on a Knowledge Repository (KR) orthogonal to the rest. The autonomic functions can use KR to retrieve and log knowledge data [12]. While the Monitoring and Analyzing components have vast ongoing works and proposals [13] [14] [15] [16], and the Executing component has implementation options like an off-the-shelf deployment system [17], the Planning component is yet to be developed for reconfigurations in SDN.

A solution to automatically calculate a set of configuration alternatives for SDN, complying management policies under different network conditions, could bring important benefits to SDN management such as reliability, efficiency, and reduction of capital expenditures/operating expenditures [18]. Furthermore, novel ANM trends like Cognitive Network Management (CNM) [12] and Knowledge-Defined Networking (KDN) [19] could be explored to complement the Planning component of the ACL.

To sum up, we propose automatic computation of (re)configuration tasks for SDN that could support administrator decisions during reconfiguration and provide recommendations while offering: (i) an increase in the reliability and efficiency of SDN operation, (ii) simplify configuration management of SDN planes; and (iii) reduce capital expenditures/operating expenditures (CAPEX/OPEX) for providers.

Some solutions have addressed self-configuration in SDN to enforce QoS settings on data plane devices [3] [20], but no solution is available to automatically compute a set of configuration alternatives for SDN, to be applied when the network experiences an unstable state. We argue that the automatic computation of configuration tasks, according to management policies could return the network into a stable state avoiding propagation of network instability and degradation of services.

Considering the need for an automated configuration planner for SDN to minimize the burden of manual operation, and increase reliability (SDN is vulnerable to misconfigurations) and efficiency (human actions are usually slower than machine), this thesis project focuses on solving the following research question:

Given an undesired network state, how to compute reconfiguration management actions on-the-fly to turn SDN into a desired state?

1.2 Hypothesis

To address the research question stated in Section 1.1, this thesis raises the following hypothesis: **An approach based on automated planning could compute SDN reconfiguration plans (ordered actions) on-the-fly to turn the network into a desired state.** During the development of this thesis and the study around this hypothesis, the main research question evolved into secondary research questions as follows.

- How to include the AP technique as a decision-maker for network reconfiguration, under the ANM paradigm?
- Can the AP based decision making for network reconfiguration be guided by high-level network management policies?

1.3 Objectives

1.3.1 General Objective

Introduce an approach based on automated planning for achieving self-configuration in SDN.

1.3.2 Specific Objectives

- Design an approach based on automated planning for providing self-configuration in SDN.
- Present a reference implementation of the proposed approach.
- Evaluate the proposed approach regarding time-response, time-planning, and network overhead.

1.4 Contributions

The present thesis contributes to the state-of-the-art on networks management proposing an approach for self-reconfigurable networks, comprising:

- A MAPE-K-based architecture that closes the Autonomic Control Loop (ACL) allowing autonomic reconfiguration of network slices from the tenant-side. The main characteristics of our architecture are:
  - AP is the AI technique for making reconfiguration decisions.
  - The business policies governing the ACL behavior are the *intents* representing high-level network management policies defined by tenants.
- A mechanism for obtaining *AP-goals* from network management policies.
- An approach for automatically generating *AP-problems* where the *AP-goal* comes directly from network management policies.
- A detailed model for the tenant-side network slice reconfiguration problem based on State-Transition System, which enables AP as a decision maker in the MAPE-K based architecture. Our model includes:
  - A representation for the Substrate Network (SN)
  - A representation for the tenant's virtualized network.
  - A representation of the instantaneous *configuration* of both (managed) networks.
  - A representation of the residual capacity of SN nodes and physical links given a *configuration*.
- A series of *AP-problems* that instantiates the system model for the tenant-side network slice reconfiguration problem in the Planning Domain Definition Language (PDDL). PDDL is a *de-facto* representation language for *AP-problems* in AP. Our *AP-problems* can be reused by researchers interested in exploring the benefits of AP as decision-maker for managing 5G network slices. Each *AP-problem* includes a PDDL description for:
  - The *AP-domain* , which generalizes the problem from its model.
  - The problem instance, which is a particular situation of the *AP-domain* .

1.5 Scientific production

1.5.1 Publications

The major contributions to the scientific community achieved with the realization of this thesis are listed below.

- "NORA: An Approach for Transforming Network Management Policies into Automated Planning Problems" [21]
  Sensor MDPI
  JCR Q1, SJR Q1, Publindex A1.
  March 4th, 2021

- "Model-Based Reinforcement Learning with Automated Planning for Network Management" [22]
  Sensor MDPI
  JCR Q1, SJR Q1, Publindex A1.
  August 22th, 2022
  Collaboration

- "Framework for Autonomic Management in Software Defined Networks" [23].
  32nd Network Management Research Group (NMRG), IETF 99
  July 20th, 2017
  Prague - Czech Republic

- "Bridging the Gap Between Network Management and Artificial Intelligence: a Natural Language Policies Translator", Extended Abstract and Poster.
  Latin America Students Workshop (LANCOMM19)[1], Brazilian Symposium on Computer Networks and Distributed Systems (SBRC2019)
  May 7th, 2019
  Gramado - Brazil

- "Autonomic Tenant-side Reconfiguration in 5G Network Slicing by exploiting Automated Planning"
  Submitted to: Computer Networks Journal.

1.5.2 Awards

During its execution, this thesis was awarded with the distinctions listed below.

- Internet Society Fellow to IETF 99[2].
  Travel grant awarded by: Internet Society (ISOC)
  July 15th - 22th, 2017
  Prague - Czech Republic

[1] http://sbrc2019.sbc.org.br/en/lancomm-student-workshop-2019/
[2] https://www.internetsociety.org/past-ngl-programs/past-fellowship-to-ietf/fellows/IETF-99

- One of the six best posters among all the submissions to LANCOMM 2019.
  Jury: Walter Willinger, Theophilus Benson and Justine Sherry
  May 7th, 2019
  Gramado - Brazil

Appendix A adds those scientific productions and awards in chronological order.

1.6 Methodology and organization

The research process that guided the development of this thesis is based on a typical scheme of the scientific method [24]. Figure 1.1 depicts the phases of the scientific research process: Problem Statement, Hypothesis Construction, Experimentation, Conclusion, and Publication. Problem Statement, for identifying and establishing the research question. Hypothesis Construction, for formulating the hypothesis and the associated fundamental questions. In addition, this phase aims to define and carry out the conceptual and technological approaches. Experimentation, for testing the hypothesis and analyzing the evaluation results. Conclusion, for outlining conclusions and future works. Note that Hypothesis Construction has feedback from Experimentation and Conclusion. Publication, for submitting and publishing papers for renowned conferences and journals. The writing of the dissertation document also belongs to this last phase.

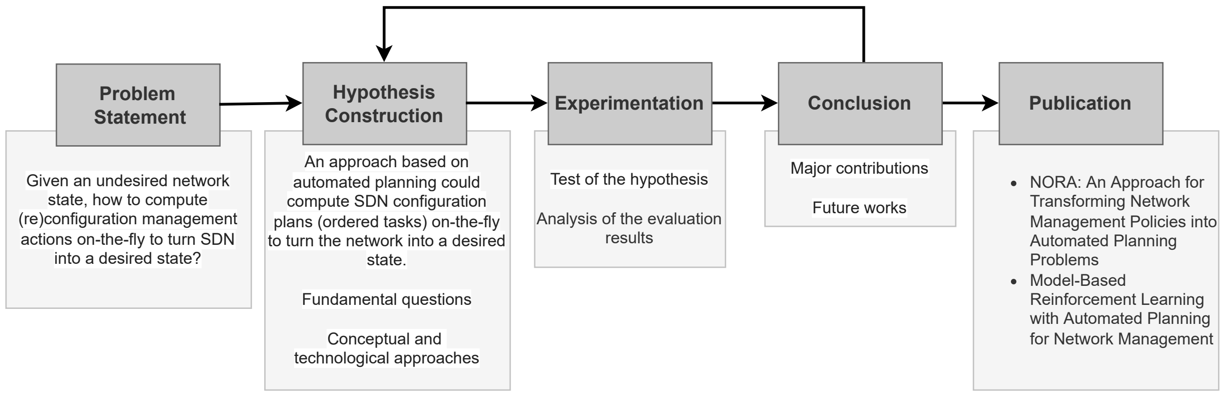

Problem Statement
Hypothesis Construction
Experimentation
Conclusion
Publication
Given an undesired network state, how to compute (re)configuration management actions on-the-fly to turn SDN into a desired state?
An approach based on automated planning could compute SDN configuration plans (ordered tasks) on-the-fly to turn the network into a desired state.
Fundamental questions
Conceptual and technological approaches
Test of the hypothesis
Analysis of the evaluation results
Major contributions
Future works
NORA: An Approach for Transforming Network Management Policies into Automated Planning Problems
Model-Based Reinforcement Learning with Automated Planning for Network Management

Figure 1.1: Thesis phases

The organization of this document reflects the phases of the methodology.

- This introductory Chapter presents the problem statement, raises the hypothesis, exposes the objectives of this thesis, summarizes the contributions, lists the scientific production, and describes the overall structure of this dissertation.

- **Chapter 2** reviews the main concepts and research related to 5G networks, Autonomic Network Management, Network Management Intents, Automated Planning, Software Defined Networks.
- **Chapter 3** introduces an approach for generating *AP-problems* where the *AP-goal* comes from the transformation of network management policies.
- **Chapter 4** presents an approach for tenant-oriented network slices reconfiguration based on a MAPE-K-based architecture and AP as a decision maker in the ACL.
- **Chapter 5** presents conclusions about the hypothesis and proposes future works.

Chapter 2

Background and State-of-the-Art

This chapter presents the central concepts of this thesis. The first section introduces a bottom-up description of the typical Software-Defined Networking (SDN) architecture. The second section provides a primer of the ANM concept, from its conception to its evolution dragged by emerging networking technologies and AI techniques. The third section introduces the evolved ANM concept, Zero-Touch Network Management. The fourth section presents an overview on Intents from the network management perspective. Sections fifth and sixth contextualize on 5G network slicing and its configuration and reconfiguration processes. The seventh section reviews the conceptual basis of Automated Planning and provides a landscape on its application to network management issues and section eighth bridges AP and reconfiguration of network slices into a single mathematical concept. Finally, sections nineth and tenth present the related work of this thesis and sum up the chapter.

2.1 Software Defined Networks

SDN offers a programmable architecture which initially included three horizontal planes: data, control, and application. The data plane incorporates network elements (NEs) that directly transport the customer traffic, handling incoming data flows along the forwarding paths computed and established in the control plane. NEs vary from simple resources (e.g., ports and memories) to composite devices (e.g., switches and routers). Data plane is also denominated Forwarding Plane [1,18]. In addition to traffic forwarding, the data plane performs data compression and dropping or changing of packets (e.g., alter packets header) which can be slightly seen as a subset of control and management functions [25]. Studies on SDN management expect NEs to include internal logic to autonomously respond to local events like usual network failures [18,25].

The control plane, usually implemented on a controller entity, centralizes the logic of the forwarding tasks [26]. The controller compiles decision policies from the user applications to instruct NEs on how to process and forward packets. The IRTF (Internet Research Task Force) SDN research group (SDNRG) [1] sums up the logic of the controller with the following functionalities: (i) topology discovery and maintenance, (ii) packet route selection and instantiation; and (iii) path failover mechanism. Latest research studies have identified issues related to the demarcation line between control and management operations in SDN [6]. For SDNRG, the control mostly refers to the device packet-handling capability, while the management typically refers to aspects of the overall device operation [1].

The application plane is, in short, where reside applications and services that define network behavior. SDN principles permit applications to specify to the control plane (via application-control interfaces), in a programmatic manner, the resources and behavior they require from the network, within the context of a business and policy agreement [25, 27]. Applications that directly support the operation of the data plane (such as routing processes within the control plane) are not considered part of the application plane [1].

Communication interfaces of SDN, *i.e.*, the SouthBound Interface (SBI) and the NorthBound Interface (NBI), were defined along with the planes. Through SBI (implemented with a vendor independent protocol), instructions from the control plane are enforced to the data plane [28]. Note that the data plane can send both solicited and unsolicited information. OpenFlow (OF) is a widespread open standard protocol for SBI, used by vendors and researchers [29]. NBI, in turn, is used for interactions between the application and control planes [30]. Over NBI, applications feed the control plane with information that contribute to the decision-making process, and the control plane provides an abstracted view of the network resources to the application plane, thanks to information and data models [27].

The architecture of SDN calls for tasks to ensure the correct operation and performance of the network in the long term, that is not to be performed by the planes themselves [6, 19]. For example, in the data plane, allocating resources to a particular client or enrolling each switch to its corresponding SDN controller. Defining the scope of a controller and configuring policies for its decisions, are issues of the control plane. Also, abnormal behavior of the controller is to be early solved given its critical role which performs most of its operations in a real-time regime. At the application plane, handling contracts and Service Levels Agreements (SLAs) of applications and creating and provisioning services are noticeable needs [6, 25].

Up to now, classical management (Faults, Configuration, Accounting, Performance, and Security - FCAPS [31]) seems to be under intensive research in SDN, but SDN-specific management is almost neglected. Recently, the SDN architecture was complemented with a vertical management plane to organize functions for the operation, administration, and maintenance (Figure 2.1) [18, 25, 27] of SDN planes. However, this plane counts considerable challenges as it is in the definition process yet. For instance,

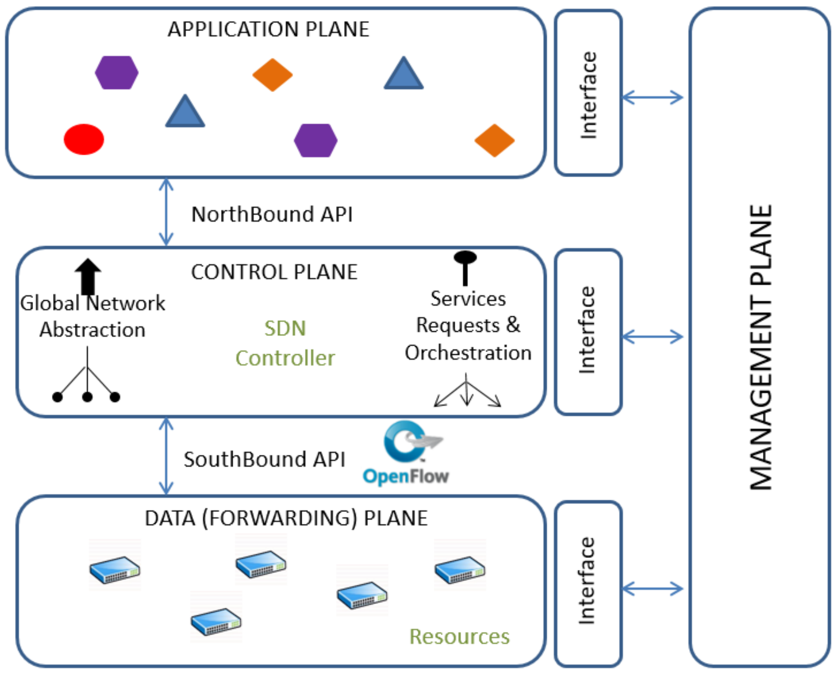


Figure 2.1: SDN architecture

the implementation of interfaces to the control and application planes rarely appeared in previous proposals [32]. These interfaces allow setting to be enforced in NEs and information to be retrieved back into the management plane, for example, to be reported to the Administrator [18]. Distribution of management decisions strengthens quick and local management operations while reducing the volume of management traffic. In SDN, local management should be executed by NEs like OpenFlow-enabled switches and controllers, expected to react in autonomic fashion. In this regard, centralization or distribution of SDN management is a current discussion, though a consensus points towards a hybrid approach [6]. Also, how to interoperate with third-party management services is to be solved in SDN. Such services could cover inventory services or policies and/or devices storage [18].

2.2 Autonomic Network Management

Autonomic Networking is a particular case of the Autonomic Computing described in 2001 by IBM [11]. ANM provides the network with capabilities to react to changes in user requirements or context conditions, returning to desired behavior with minute human intervention. In this sense, ANM calls for automated decisions for management actions [12] allowing the Administrator to focus more on business logic and less on low-level device configuration processes [8].

The goal in ANM is to provide self-management via self-CHOP properties. In general terms, self-CHOP includes: (i) self-configuration, where functions configure themselves based on self-knowledge and/or discovery [33], (ii) self-healing, that is, functions adapt on their own to changes in the environment and heal problems automatically [10], (iii) self-optimization, which means that autonomic functions automatically determine ways to optimize their behavior against a set of well-defined goals [8]; and (iv) self-protection, where functions automatically secure themselves against potential attacks [9].

In ANM, the network status is continuously monitored and compared with the desired bounds defined by high-level policies. If any event is detected in the network, the situation is analyzed to convert the policies into executable network management tasks (*e.g.*, topology reconfiguration). Finally, the network status is monitored again to check the effect of the executed action(s). Figure 2.2 depicts the alternative of IBM to solve this ACL behavior chaining components for: (i) Monitoring, to gather, filter and collate data as required, (ii) Analysis, to understand the data and determine if the managed element is acting as desired, (iii) Planning, which determines which actions should be taken to reconfigure the managed element; and (iv) Execution, to translate the planned actions into a set of configuration commands. These components rely on a Knowledge Repository (KR) [8] [9] orthogonal to the rest. The autonomic functions can use KR to retrieve and log knowledge data [12]. The loop is referred to as MAPE-K.

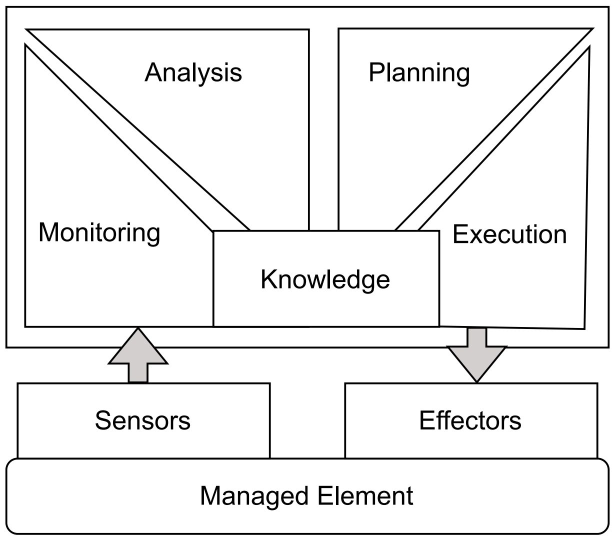


Figure 2.2: MAPE-K ACL

Beyond MAPE-K, different ACLs proposals have emerged from research projects. For example, FOCALE [34] is a blueprint architecture for orchestrating the behavior of heterogeneous and distributed computing resources. FOCALE stands for Foundation, Observation, Comparison, Action, and Learning Environment defining mainly two interworking loops, one for *Maintenance* when the gathered sensor data matches the desired network state, and another for *Adjustment* in the opposite case. CogMan [35], in turn, makes a fundamental change to FOCALE creating control loops based on the human cognitive model (CogMan will be extended on the Related Work Section).

A novel paradigm referred to as the KDN [19] operates by a control loop that exploits SDN and Network Analytics to provide automation, recommendation, optimization, validation and estimation. In the KDN operational loop, ML is the heart for the data analysis. Important challenges need still to be addressed in KDN as interdisciplinary efforts are combined therein. A similar proposal is the C-MAPE loop [12], where the original IBM MAPE is incorporated with cognition (through ML) at every component. For instance, the Monitor component should be able to determine the what, when and where to monitor.

Standardization bodies have also put efforts on ANM. ETSI has released several versions of the GANA architecture (Generic Autonomic Networking Architecture) fusing efforts from leading models (e.g., FOCALE and IBM MAPE) [36]. GANA counts some core concepts: (i) Managed Entities (MEs) at the bottom of the management and control hierarchy, means a managed resource that can be either a physical NE or some functional entity within a node/device, (ii) Decision Elements (DEs) in four levels of hierarchy (protocol, function, node, and network-level) are responsible for autonomic management, their behavior is called the self-properties; and (iii) the Knowledge Plane (KP) that enables advanced management and control intelligence thanks to a series of inner elements. The interaction of DEs from different levels is performed through the execution of an ACL.

Finally, the Autonomic Networking Integrated Model and Approach (ANIMA) IETF group is currently working on a Reference Model for Autonomic Networking [37] focused on autonomic nodes, that is, nodes interact with each other to form feedback loops and perform autonomic functions at node level [33]. ANIMA introduces two main architectural elements: (i) Autonomic Service Agents (ASAs) that is like the atomic part (*e.g.*, a piece of code) of an autonomic function; and (ii) a crosslayer called Autonomic Networking Infrastructure (ANI), the foundation for autonomic functions.

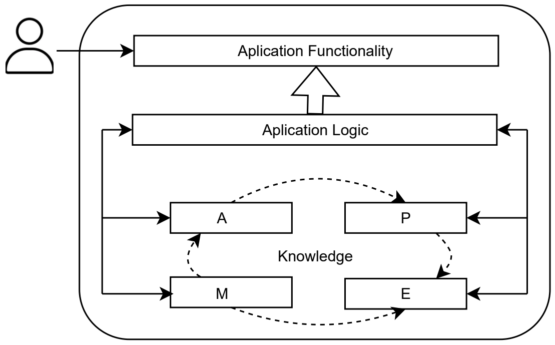
2.3 Zero-touch Network Management

With the pivotal deployment of 5G, not only does network usage and traffic increase significantly, but so does its technical, network, and operation complexity. For this reason, operators networks seek automation to reduce the cost of operations, time to service, and revenue of new and innovative services. Zero-Touch Network Management positions itself as an alternative to face these demands. These networks will use automation and AI/ML with the purpose of healing and adjusting themselves, based on the signals in the data they collect and analyze across all network activity [38]. The greatest achievement of automation in these new deployments is to enable autonomous networks which will be driven by high-level policies and rules; where the networks will be capable of self-configuration, self-monitoring, self-healing, and self-optimization without further human intervention. These objectives demand requirement of capacity, extremely low latency, high reliability, high expectations in customer experience and support for massive machine-to-machine communication. Zero-Touch Network Management requires a new horizontal and vertical architecture framework designed for closed-loop automation and optimized for artificial intelligence algorithms that ETSI ZSM has been working on since 2017 with the aim of accelerating the definition of the architecture and the required end-to-end solutions [39].

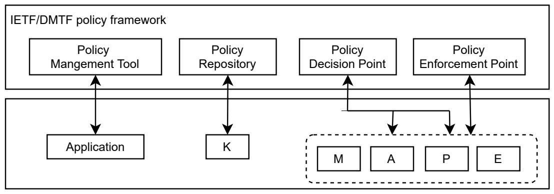
2.4 Network Management Intents

According to [40] policies can be classified into *Action Policy*, *Goal Policy*, and *Utility Function Policy*. An *Action Policy* dictates the action that the Network Management System (NMS) should take whenever the system is in a given state. Typically, a NMS based on *Action Policies* follows the structure IF(*Condition*) THEN(*Action*), where *Condition* specifies either a specific state or a set of possible states that all satisfy the given *Condition*. Note that the state that the NMS will reach taking the given action is not specified explicitly. Rather than specifying what to do in the current state *S*, a *Goal Policy* specifies how the NMS should behave when a single desired state *S*, or one or more criteria that characterize an entire set of desired states happen. *Goal Policies* provide only a binary state classification: 'desirable' and 'undesirable' [41]. A *Utility Function Policy* is an objective function that expresses each possible state's value. *Utility Function Policies* generalize *Goal Policies*.

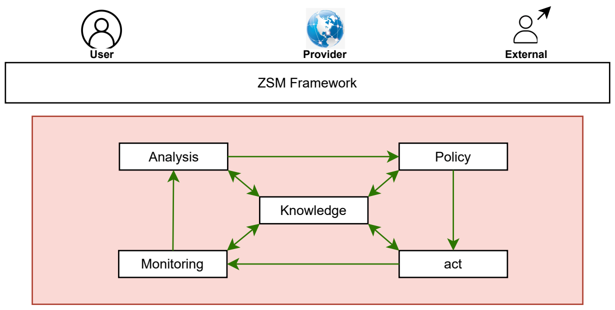
Network Management Intents are guidelines and constraints to network management [42] and represent the evolution of high-level network management policies, as a special case of policy. They represent service requirements, such as availability, response time, throughput, and security. A higher level intervention from the users is regarded as an *intent* [43]. The objective is to provide guidance in the form of network understandable input from the users. The required information from user to interact with the network through *intent* does not encompass low-level or configuration level information. An autonomous network is able to understand this *intent* from nodes, finally providing a configuration for the involved network functions. In this thesis we use the Intent concept defined by the IETF [44]: an *intent* is a set of operational goals (that a network is supposed to meet) and outcomes (that a network is supposed to deliver), expressed in a declarative manner without specifying how to achieve or implement them. Recent studies on network management *intents* [45, 46] have resulted in architectural proposals, as those shown in Figure 2.3, for defining the -close- relationship between *intents* , ANM and ZSM.



(a) Intents and MAPE-K



(b) Policy and MAPE-K



(c) Intents and ZSM

Figure 2.3: Intents in ANM and ZSM

2.5 5G Network Slices

5G networks will offer network services to a wide range of end-users including hand-held devices, self-driving cars, and e-health devices. The services offered under the use cases eMBB, URLLC, mMTC are characterized by different requirements regarding for example bandwidth and latency [47]. Simultaneously satisfying all service requirements is a key challenge for network operators. The 3rd Generation Partnership Project (3GPP) has therefore proposed network slicing, *i.e.*, , multiplexing logically independent networks on a substrate where each network is tailored for a particular use case [39].

The service provided by a network slice is thus defined by its Service Function Chain (SFC) [48]. For example, a service offered under the mMTC use case, including essential control and user plane VNFs, *i.e.*, Access and Mobility Management Function (AMF), Session Management Function (SMF) and User Plane Function (UPF), can be defined as $AMF_0 \rightarrow AMF_1 \rightarrow AMF_2 \rightarrow SMF_0 \rightarrow UPF_0$. Several AMFs shape this

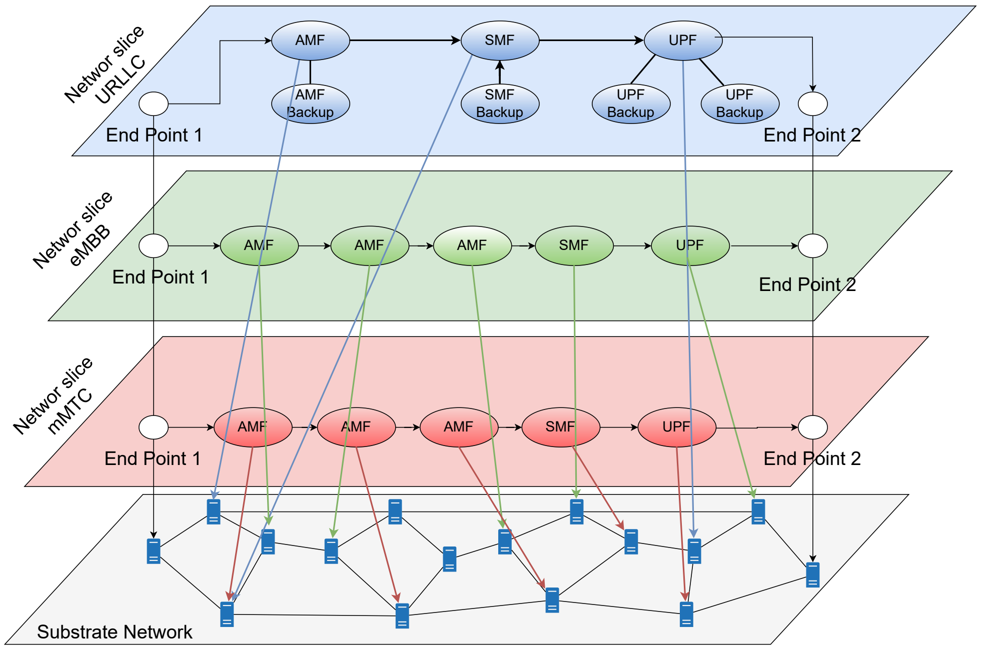


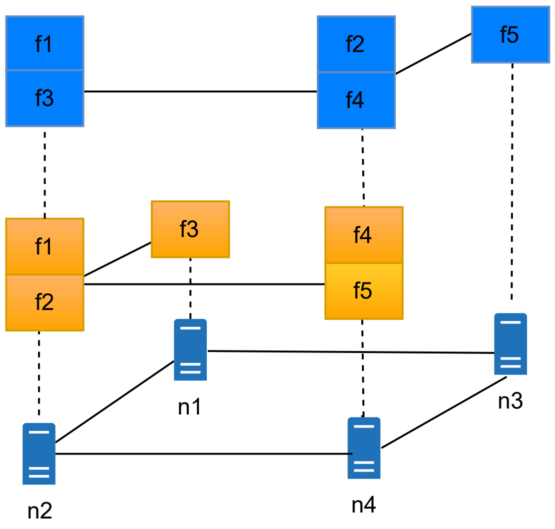
Figure 2.4: 5G network slicing

SFC because mMTC provides access to several types of devices [49]. A VNF can be instantiated in any general-purpose computing node, thus allowing the slices to scale and adapt with the demands. A VNF is instantiated as Container (VM or Docker) and the number of containers can be adjusted according to instantaneous network load. For example, an increase in the traffic load can trigger a firewall VNF to scale-out and instantiate additional containers, while a decrease in the traffic load can trigger it to scale-in and decommission idle containers.

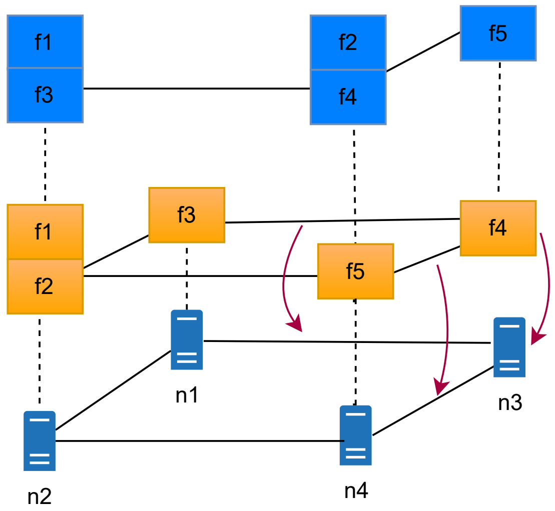
Figure 2.4 depicts the concept of network slicing for the three types of use cases most common in 5G. The URLLC network slice considers backups for AMF, SMF, and UPF that should be instantiated at nodes close to the end-user since URLLC must offer high reliability and low latency. The eMBB and mMTC graphs do not include backups for SMF or UPF as these service types do not have ultra-high reliability requirements. The mMTC network slice has more AMFs than the other types, as it must provide access to several types of device. The eMBB network slice has fewer AMFs than the mMTC network slice since the former attends fewer devices than the latter [49]. All VNFs composing the network slicing are instantiated over the shared resources of the SN.

2.6 Configuring and Reconfiguring Network Slices

A *configuration* for network slicing describes the instantaneous *mapping* of the virtual resources across all 5GNSs onto the shared SN [50]. Specifically, a network slicing *configuration* details the *mapping* of the VNFs and virtual links across all 5GNSs onto the computing nodes and physical links between the nodes [50]. Figure 2.5a exemplifies a *configuration* for two network slices, *i.e.*, NS_1 depicted as blue VNFs and NS_2 depicted as yellow VNFs. NS_1 has VNFs f1 and f3 mapped onto host node n2, VNFs f2 and f4 mapped onto host node n4, and VNF f5 mapped onto host node n3.



(a) A *configuration* for two network slices



(b) Migration of a VNF and its adjacent virtual links

Figure 2.5: Reconfiguring Network Slices

The scale-in and scale-out of VNFs cause the *configuration* of 5GNSs to change so that there may be a violation of tenant *intents*. For example, the network load imbalance set by the *configuration* shown in Figure 2.5a, may be violating tenant *intents* asking for evenly usage of SN resources. To remedy such undesired network *configuration*, 5GNSs must be reconfigured to *configurations* complying tenant *intents*. The process of reconfiguring network slices involves identifying a sequence of VNF and virtual links migrations that will turn the network slicing from the current -undesired- *configuration*, called source, to the desired *configuration*, called target [50]. Figure 2.5b exemplifies a step in the migration sequence during a 5GNS reconfiguration process. Performing 5GNS reconfiguration that adjusts the resource allocation for a 5GNS according to the variations of traffic demand is referred as the Network Slice Reconfiguration Problem (NSRP) [51].

2.7 Automated Planning

AP is a branch of AI which explores the process of using autonomous techniques to solve planning and scheduling problems [52]. An *AP-problem* is one in which we have some initial starting state (real-world situation), which we wish to turn into a desired goal state through the application of a *plan*. A *plan* in AP is an ordered set of possible actions automatically computed by an *AI-planner* [22].

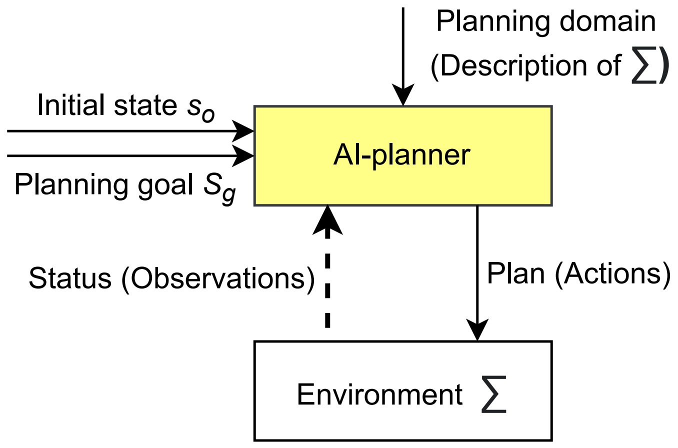


Figure 2.6: AP conceptual model

The conceptual model in classical AP is defined as a State Transition System (Figure 2.6). A *planning domain* models the environment as $\sum = (S, A, \gamma)$, where S, A, γ denote the finite set of states $S = \{s_1, s_2, s_3, ...\}$, the finite set of actions $A = \{a_1, a_2, ...\}$, and the state transition function $\gamma : S \times A \rightarrow 2^S$. The classical *AP-problem* is formally defined as $(\sum, s_0, S_g)$, where $s_0 \in S$ is the initial state and $S_g \subset S$ is the goal state or set of goal states (also referred to as *planning goal*). An AI-planner finds a sequence of transitions labeled with actions $[a_0, a_1, ..., a_p]$ that can be applied starting at s_0 resulting in a sequence of states $[s_0, s_1, ..., s_p]$ such that $s_p \in S_g$. The actions sequence $[a_0, a_1, ..., a_p]$ is the *plan* [52] [53] [54]. The practical representation language for *AP-domains* and *AP-problems* is PDDL [55] [56].

State-space planning is the *de-facto* search method of the AP community. AI-planners transform *AP-problem* descriptions (from PDDL) into a graph-search problem, and, by a heuristic function guiding the search, explore those vertices whose associated state s_i is reachable from s_o and get closer to S_g [57]. Over the years, AP research has been continuously attempting to solve complex *AP-problems* by evolving PDDL [58] and by trying heuristics that can reason effectively over diverse *AP-domains* (numeric [59], temporal [60], hybrid [58], etc.). Recently, it has been corroborated the feasibility of using AP to automate SDN management tasks and reduce the time required by network administrators to face network situations [61] [62]. However, in such approaches, the network administrator still has to manually describe the network problem in AP notation, which is difficult to interpret without prior knowledge. To realize ACLs based on AP, the network itself should create the *AP-problem* without human intervention, which is achieved in this thesis, and explained in Chapter 3.

2.8 State-space Planning for the Network Slice Reconfiguration Problem

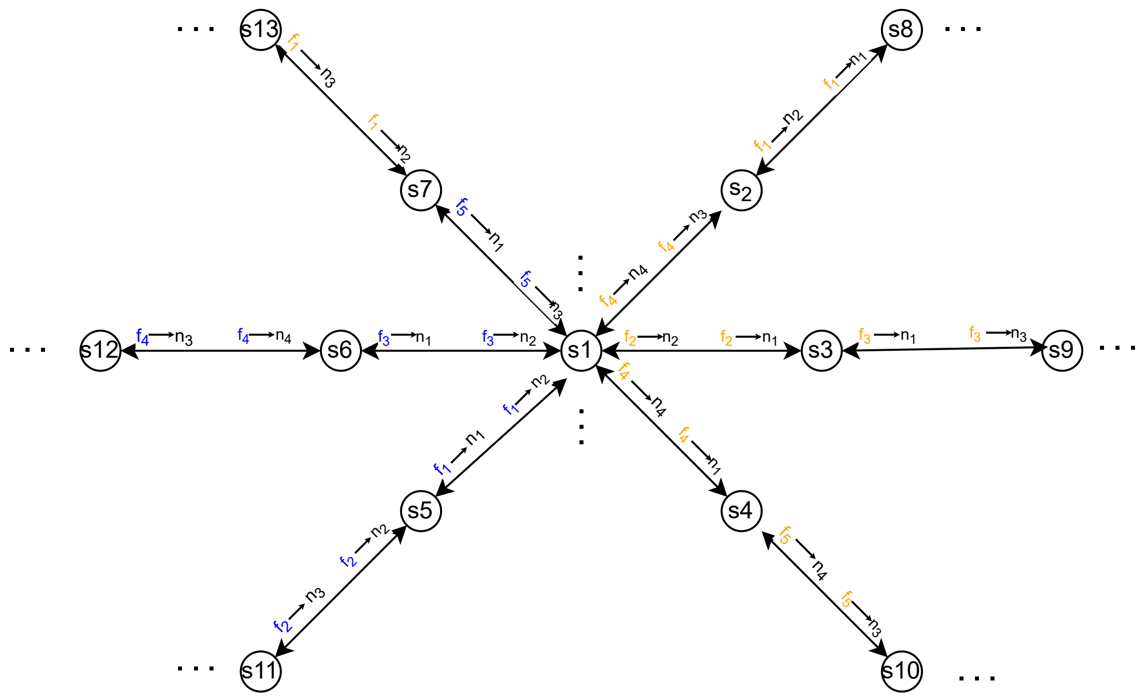


Figure 2.7: Migration Graph for NSRP

The state-space planning in the NSRP is the *migration graph* [50]. Given an *AP-problem* instance, a migration graph consists in a directed graph in which each vertex represents one of the possible *configurations*, *i.e.*, a unique *mapping* of the virtualized network onto the SN, and each edge represents a VNF migration. For a SN with m host nodes and a total of q VNFs across all network slices, a migration graph has m^q vertices, and each vertex has $q(m-1)$ incoming edges and $q(m-1)$ outgoing edges. Figure 2.7 shows an example of (a part of) the migration graph for the network slicing of Figure 2.5. Vertices labeled as s_1 and s_2 represent, respectively, the *configurations* given by Figures 2.5a and 2.5b. The edge from s_1 to s_2 indicates the state transition caused by applying action $a_x \in A$. a_x migrates f_4 of NS_2 from $n4$ to n_3 carrying the migration of its adjacent virtual links on the SFC.

2.9 Related Work

This Section presents the related work of this thesis divided into three major areas: approaches performing translation of high-level network management policies, proposals for delegating network slices management to tenants, and approaches for reconfiguring network slices.

2.9.1 Translation of High-Level Network Management Policies

Liu [63] proposed a mechanism to translate high-level objectives from the IT management domain into the goals of an *AP-problem* describing requirements for fault recovery. This mechanism uses rules to map fault expressions stated in the Domain Service Language (DSL) into the Planning Domain Definition Language (PDDL). However, the rules are attached to the IT management domain and, so their replication on the network management domain is hard. Other approaches [64] [65] transformed user requests expressed in NL into PDDL for facing telecommunications services issues. Despite the use of NL and its integration with AP, these approaches present several drawbacks. First, NLP's corpus is limited to requests related to the environmental early-warnings domain and, consequently, disregards models of *Goal Policies*. Second, the low-level configuration actions are specific for composing telecommunication services and leaves aside the network management tasks.

The works [66] [67] [68] introduced a framework to translate high-level policies expressed in CNL, into low-level flow rules for SDNs. The employed CNL follows a grammar of predefined regular expressions (*regexes*) representing terms of the network context such as "HTTP" and "FTP". These works use Inductive reasoning for analyzing policy objectives and abductive reasoning for determining if the network infrastructure can accommodate the reasoned objectives during translations. Although this approach provides helpful grammar contributions regarding classification of the network context terminology, the inductive and abductive reasoning processes led directly from high-level to low-level commands disregarding the possibility of an AI algorithm to interpret network management *regexes*.

Tuncer et al. [69] proposed an approach for the automatic decomposition of High-Level Requirements (HLRs) to network management operations. This approach relies on developing a NorthBound Interface (NBI), including mapping functionality, that associates technical HLRs to the network operator's services and functions that manage the network resources. This HLR-based approach performs the association through matching procedures to support operator-defined descriptors that encode distinct features and uniquely identify services and functions. In this approach, network administrators do not use NL and, consequently, they must fulfill the HLR format's attributes.

Jacobs et al. [70] introduced an approach to translate network administrators policies expressed in *NILE* (*i.e.,* an intermediate representation for network *intents*) into network configurations. This approach uses a recurrent neural sequence-to-sequence learning model to extract *intents* from NL and includes feedback from the network administrator for improving the learning process. Although this approach offers high accuracy in the translation process, it does not consider a closed-loop (requires feedback from network experts), pivotal for self-driving networks, and network administrators must learn *NILE*.

Riftadi and Kuipers introduced P4I/O [71], a framework that translates *Intents* into P4-programs using code-templates. Although the generated P4-programs offer excellent results for handling the network throughput, this framework requires that network administrators learn an extended version of *NILE* that adds custom actions for network tasks and does not support AI-based notations.

Widmer [72] proposed a state-machine-based refinement technique that uses a grammar for an Intent specification language and a parsing process to translate the *intents* to low-level blockchain selection policies abstracting underlying implementation details. This approach does not operate with policies expressed in NL nor explore AI-based notations.

This thesis differentiates from the above cited works as follows. Our transformation approach is from network management policies in pure natural language to the *AP-goal* , avoiding the user to learn special syntax, network commands, or structured languages. We transform the policies directly to the AI language, specifically an AP language.

2.9.2 Tenant-side 5G Network Slicing Management

The MANO-as-a-service (MANOaaS) paradigm [73] abstracts the ETSI MANO framework into customized and distributed per-tenant MANO instances. MANOaaS manages the tenants autonomy through the enforcement of management level agreements (MLA) that are negotiated between the tenant and the InP. MLA determines the scope of the delegated operations, while a central MANO entity maintains full administrative rights over the deployed MANO instances. During simulations, different number of tenants requested resources from the central MANO entity. The success resource request rate was quantified when the granted degree of autonomy per MANO instance was varied (*i.e.*, zero, partial, or full MLA autonomy). The instances getting full granted autonomy had the higher success rate. However, the operations delegated to tenants lack dedicated interfaces, which is a crucial feature to tenant-side management.

DASMO (Distributed Autonomous Slice Management and Orchestration) [74] combines the ETSI MANO framework, the OSS/BSS system, and an In-Slice Management (ISM) approach to deal with the scalability of network slicing management in a distributed manner. The ISM is part of each slice and uses autonomic/cognitive management mechanisms through Embedded Element Managers (EEM) and the Slice Manager (SM), allowing for local (*i.e.*, at the slice level) management decisions. The SM functional entities (*i.e.*, Tenant Oriented Functions, Autonomic Management Functions) implement the MAPE paradigm and an intent-based tenant oriented interface. The architectural concepts introduced in DASMO are essential to deal with issues of the tenant-side network slice management, however, prototyping and verification was not carried out.

Galis et al. adapted the IETF reference model for Autonomic Networking [33], into Autonomic Slice Networking [75] introducing the concept of Slice Element Managers (SEMs) to be located inside each NS. SEMs allow 3rd parties to dynamically customize the network characteristics within the constraints of resources allocated to the slice. Autonomic network functions (*i.e.*, naming, addressing, negotiation, synchronization, discovery and messaging) and signaling between SEMs are described, however, there is a long road ahead to deploy and asses SEMs performance because all descriptions are provided in a very high-level. Besides, the ACLs considered for SEMs are just to be defined but the work has not had continuity.

The 5G PPP SliceNet project [76] [77] aims to design, prototype and demonstrate an innovative, tenants-oriented, QoE-driven 5G network slicing framework focusing on cognitive network management. In the context of SliceNet, Spadaro et al. [78] propose a Cognition Plane embracing the MAPE-K approach for planning required (re-)configurations on the network infrastructure as well as deployment of new elements and functions on the configured services to remedy undesired situations. A tenant's feedback mechanism in included, allowing them to express their experience with the provisioned infrastructure. However, with this role, the tenant rather than having interference on the network (re-)configurations to fit his/her *intents*, he/she is a data source of the data acquisition system for the MAPE-K loop. Also, the interface for the tenants expressions is not specified.

The platform SFCLola [79] is proposed to handle SFC requests within a tenant's virtual network spanning multiple data centers (DCs) while minimizing support required from the InP. A Virtual Flow Forwarder (VFF) mechanism enforces chaining instructions within the tenant network of VMs without requiring access to the switches at the network infrastructure data plane. SFCLola was experimentally evaluated by sequentially sending a set of chain requests on a multi-DC infrastructure provided by the 5GINFIRE project [80]. Results demonstrated benefits in terms of computation time, resource consumption at VFF nodes, and an average error of 3.8% between the estimated latency and the measured end-to-end latency along the established chain. In SFCLola, the VFF exposes intent-based REST API to receive chaining rules, however, it is sole for the SFC network service and can not be easily extended to other tenant-side management operations like network (re-)configurations.

In the 5G NORMA project architecture [81] each tenant is provided its own t-MANO (tenant-MANO) by the c-MANO (central-MANO). Each t-MANO instance consists of its own NFVO, VNFM and VIM instances used by the tenant to manage and orchestrate his/her respective NSs with minimum dependence on the c-MANO. However, the degree of autonomy of the respective t-MANO stack instances will depend on the agreed SLA with the c-MANO. Likewise, the t-MANO stacks does not count autonomic/cogni-

Table 2.1: NSRP Related Work

Paper	Description	Approach for modeling the problem	Technique for making reconfiguration decisions	Tenant-oriented	Performance metrics
[82]	Hybrid model-data driven framework to proactively reconfigure network slices under demand uncertainty	Robust Optimization, Mixed Integer Linear Program and Mixed Integer Second Order Cone Program	State-of-the-art solver		Training loss, time to generate prediction intervals
[48]	Algorithm for avoiding unnecessary reconfigurations by predicting future traffic demands	Markov Decision Process	Deep Reinforcement Learning and Branching Dueling Q-network		Convergence properties, Long term resource consumption
[83]	Predictor-optimizer framework with the aim of minimizing the energy consumption	Robust Mixed Integer Programming	Linearization and Robust Optimization		Prediction intervals
[84]	Optimal and Fast Slice Reconfiguration solution aiming at obtaining high long-term revenue with low operation cost	Markov Renewal Process	Dueling Neural Network combined with Q-learning		Prediction performance, average long-term revenue, reconfiguration cost
[85]	Hybrid slice reconfiguration framework dealing with the cost incurred by frequent reconfigurations	Own and L1 norm to approximate the reconfiguration cost function	Fast Slice Reconfiguration and Dimensioning Slices with Reconfiguration		Reconfiguration ratio, fairness index, resource utilization ratio, reconfiguration cost and profit.
[86]	Load-balancing oriented deployment and reconfiguration for 5GNSs	Integer Linear Program	Own reactive strategy		Acepptance ratio, workloads of physical nodes and links.
[50]	Approach for finding a sequence of configurations from a source to a target configuration	Own model for a large scale 5G network	*Matryoshka* which uses: i) A search [31] optimizations ii) the divide-and-conquer approach iii) parallelization		Completion time, number of VM migrations
[87]	Inter-slice reconfiguration with aim of reducing network operational costs	Integer Linear Program	Make-before-break reactive scheme based on column generation (CG)		Execution time, Gains in network cost, accuracy of the CG models
ATRAP	Autonomic tenant-side reconfiguration of network slices	State-Transition System	Automated Planning	✓	Processing time, reconfiguration cost

tive characteristics neither high-level interfaces for tenant interactions.

The cited projects represent significant advances on providing to tenants an active role in the management of their network slices, nevertheless, those investigations did not focus on providing reconfiguration capabilities.

2.9.3 Reconfiguring Network Slices

Wei et al. [48] propose an Intelligent Network Slicing Reconfiguration Algorithm with aim of minimizing long-term resource consumption. They use Markov Decision Process (MDP) to model the long-term decision-making problem NSRP, and resort to DRL to solve it. To address the curse of dimensionality of the problem, they incorporate the Branching Dueling Q-network (BDQ) into DRL. Numerical results reveal that the proposed algorithm can avoid unnecessary reconfigurations by implicitly predicting future traffic demands. Later, these authors propose a predictor-optimizer framework that intelligently performs inter-slice reconfiguration with the aim of minimizing the energy consumption of serving 5GNSs [83]. This time, the NSRP is formulated as a Robust Mixed Integer Programming and solved with linearization and robust optimization. Numerical results demonstrate that the framework can flexibly achieve a trade-off between the robustness and the energy consumption.

With the goal of maximising long term revenue, Guan, Zhang and Leung [84] use a Markov Renewal Process to predict changes in the resource occupancy of 5GNSs and reserve resources for slices that obtain higher revenues at lower cost. They use deep dueling neural network combined with Q-Learning to choose for each slice whether to reconfigure it or not. According to simulation results, the work developed by Guan and collegues is effective in achieving long-term revenue for tenants.

The hybrid slice reconfiguration framework proposed by Wang et al. [85] deals with the cost incurred by frequent reconfigurations. Their framework consists of two schemes: a Fast Slice Reconfiguration that reconfigures flows for individual 5GNSs at the time scale of flow arrival/departure and a Dimensioning Slices with Reconfiguration that occasionally adjusts allocated resources according to the time-varying traffic demands. L_1 norm function is used to approximate the reconfiguration cost function. The framework is extended with a resource reservation mechanism to reduce potential reconfigurations in near future. Numerical results validate that the framework is effective in reducing reconfiguration overhead and achieving high profit for 5GNSs tenants.

The focus of the work performed by Lu et al. [86] is twofold: the *deployment* and the *reconfiguration* of 5GNSs. They formulate a load-balancing oriented 5GNSs deployment problem through Integer Linear Program (ILP) and a reactive strategy to accommodate a rejected 5GNS request by re-organizing the already-deployed 5GNSs. The 5GNS deployment algorithm is reutilized with slacked physical resources to find out the congested part of the network, due to which the 5GNS is rejected. These congested physical nodes and links are reconfigured by migrating VNFs and virtual links to re-balance the network. Simulations results show that resources utilization is improved by acommodating more 5GNSs in a dynamic environment.

Matryoshka is a divide-and-conquer approach created by Pozza et al. [50] for finding a sequence of configurations from a given source configuration to a desired target configuration. A dataset of pairs of source and target configurations in a large scale 5G network representing potential substrates and multiple network configurations was created and used for evaluating *Matryoshka*. Results show that *Matryoshka* is effective in finding good quality migration sequences compared to state-of-the-art A^* search [88].

Gausseran et al. [89] [87] propose to adaptively reconfigure 5GNSs based on column generation. The authors implement a *make-before-break* reactive scheme to perform inter-slice reconfiguration with aim of reducing network operational costs. Evaluation results show that this solution decreases the network cost without degrading the QoS as the network slices are not interrupted thanks to the *make-before-break* approach.

Table 2.1 summarizes related work on reconfiguration for 5G network slices. The projects cited in Table 2.1 represent significant advances in solving the NSRP, however, none of them focus on the tenant-side reconfiguration, which is a crucial feature on the 5G network slicing management landscape. Our automated-planning-based approach differentiates from previous related work as follows. First, a *reconfiguration plan* is automatically raised defining what VNFs have to be migrated and the particular order of such migrations for turning the network slicing from a source into a target configuration. Second, the target configuration is specified through a set of *planning goal* conditions coming from the tenant *intents*. Third, the output *reconfiguration plan* is known and readable to the tenant so that he/she could provide feedback prior to execution on the network.

2.10 Final remarks

This Chapter detailed the traditional three-plane architecture for deploying an SDN-based network as well as the later inclusion of the management plane in the architecture. Subsequently, the chapter provided the landscape on ANM and some of its variants as Self-driving networks and Cognitive Network Management. Also, the chapter introduced the concept of Network Management Intents and its relation with the autonomic control loops that allow to realize ANM. Subsequently, the chapter provides a conceptual overview of Automated Planning and its state-of-the-art model. The chapter introduced 5G network slicing along with its configuration and reconfiguration processes. Finally, this chapter exposed the literature review which is three-fold. From the Intents Translation approaches, we conclude that the cited approaches share the following shortcomings. First, they require policies described in a particular syntax (*e.g.,* CNL and Intents) that can be as hard to learn and interpret for network administrators as the AP notations are. Second, they assume a linear correspondence between high-level policies and network configuration tasks. In contrast, self-driving networks usually rely on AI algorithms to automatically and on the fly compute sequences of actions that carry out corrective and even preventive network configuration tasks to comply with high-level network management policies. In this sense, this thesis is a pioneer in transforming from high-level policies expressed in NL to AP notations in the network management domain. Regarding those approaches delegating network slice management to tenants, they lack automation and dedicated interfaces to tenants. Last, none of the projects solving the network slice reconfiguration problem is tenant-oriented. This thesis differentiates from previous related work as follows. First, Automated-Planning allows to define what VNF migration actions out of a potentially big one, need to take place, second, these actions can only happen in particular orders, of which there are many, third, the target configuration is specified through a set of goal conditions (coming from the tenant *intents*), and fourth, the output reconfiguration plan is known and readable to the tenant, so she/he could approve it through the implementation of a Tenant Portal.

Chapter 3

Transforming Network Management Policies into Automated Planning Problems

Networks' complexity and size are growing exponentially, making unfeasible their manual administration. The self-driving networks paradigm comes with the promise of accomplishing minimal or null human intervention [90] [91]. Realizing autonomic control loops (ACLs) for network management based on Artificial Intelligence (AI) techniques, like Automated Planning (AP) [61], Machine Learning (ML) [92], or their combination, is pivotal for achieving the self-driving networks' promise. Specifically, AP has been used in the networking domain to create autonomic solutions (or plans) formed by a set of primitive tasks that takes the network from an (troublesome) initial state to a desired state that satisfies network management policies. However, carrying out AP-based ACLs is complicated since network administrators, who are non-AI-experts, need to define network management policies as AP-goals in an AP notation, and combine them with the network status and network management tasks to obtain AP-problems. An AP-problem is a primary input for an AI planner to build up a solution plan.

In several domains like information technologies and telecommunications, diverse proposals [63–65] automatically translate policies expressed in natural language (NL) into AP-goals. Nevertheless, as these approaches use translation rules fitted to their domains, their adaptability to other ones is constrained. In the network management domain, some efforts based on *policies refinement* have been introduced to translate network management policies defined in Controlled Natural Language (CNL) [66] [67], Intents [93], or Requirement Formats [69] into Software-Defined Networks (SDN) flow rules [70] or P4 programs [71]. These policies' refinement-based approaches share some shortcomings. They require policies described in a particular syntax, such as CNL and Intents; overall, these syntaxes can be as hard to learn and interpret for network administrators as the AP notations. Also, they do not offer an interpretation bridge between management policies and AI notations, hindering AP-problems' realization and, consequently, the challenge of building up AP-based ACLs for network management remains unexplored.

This Chapter introduces a novel approach, called NORA, envisioned to generate *AP-problems* automatically; such a generation is fundamental to close autonomic management loops and, so also, to realize self-driving networks. The NORA's novelty lies in allowing the network administrator to express *Goal Policies* in NL and automatically transform them into AP-goals. NORA combines the AP-goals with the network status and network management tasks to generate *AP-problems* . NORA uses NLP as the translation technique and templates as the combination technique. To the best of our knowledge, we are pioneers in overcoming the interpretation gap between network management policies and *AP-problems* . Bridging this gap is fundamental for paving the self-driving network's realization since network administrators do not need to spend time learning new policy formats or AP-notations when building up ACLs. In this way, they can focus on their core tasks. We implemented a NORA's prototype and evaluated it using a *Goal Policies* dataset. Results show that NORA achieves high precision and spends a short-time on generating AP-problems. Consequently, we conclude that NORA is a promising solution to overcome barriers to using AP in self-driving networks.

The remainder of this chapter is organized as follows. Section 3.1 introduces the approach for transforming network management policies to automated planning problems. Section 3.2 presents a prototype that instantiates NORA and its evaluation. Finally, Section 3.3 contains some concluding remarks.

3.1 NORA

In this Section, we introduce how NORA operates at a high abstraction level. Also, we explain in detail the architecture and modules composing NORA.

3.1.1 High-level Operation

Figure 3.1 depicts NORA's architecture conceived to generate *AP-problems* automatically. This thesis argues that generating such problems is pivotal for achieving the self-driving network concept since they are essential to close autonomic management loops. Henceforth, we focus on presenting how NORA translates *Goal Policies* expressed in NL into planning goals and combines them with network status and network management tasks to generate AI-*AP-problems* . In turn, in this thesis, we assume an external Monitoring&Analysis module, like the one proposed in [94] [91], which provides the network status and a network model. The network model as described in [95] contains the management tasks necessary to realize the AP-based solutions.

A *Goal Policy* specifies either a single desired state σ, or one or more criteria that characterize a set of target states [40]. NORA operates with these policies because they are useful to express business goals without technical details as in the case of SLAs [66] [68]. We model a *Goal Policy* as either a 4-tuple or multiple 4-tuples $P =< Target, Metric, Condition, Threshold >$. In this model, $Target$ is a binary $< S|E >$, where *S* denotes a network service and *E* denotes an endpoint (*i.e.*, a network equipment or resource) or an end-user(s) (*e.g.*, researchers working on a specific laboratory of a University). $Metric$ denotes a network performance parameter measurable at services or endpoints. $Condition$ denotes a boolean comparison adjective statement.

$Threshold$ denotes boundaries for the metric values. Thus, for using NORA, the network administrator must express policies as follows: "Streaming traffic should receive bandwidth lower than 16 kbps", where $Target$: "Streaming traffic", $Metric$: "bandwidth", $Condition$: "lower than", and $Threshold$: "16 kbps". *Goal Policies* examples involving several network criteria, *i.e.*, more than one atomic policy are: "HTTP services should receive bandwidth higher than 100 kbps and delay lower than 300 ms" and "A network slice must all the time meet latency lower than 5 ms and packet loss rate under 10^{-4}". Note that, as in the last examples, *Goal Policies* can be decomposed in several tuples $< Target, Metric, Condition, Threshold >$, leading to *AP-problems* with several planning goals (see Figure 3.2b).

From a high-level perspective and according to Figure 3.1, NORA operates as follows. First, the network administrator expresses a management policy by following the NL-based *Goal Policy* model. Second, the *Lexer* decomposes the policy in representative terms for the network management domain. These terms can be a word or a phrase, from now on called *tokens*. For example, "Voice over IP" and "Video streaming" are *tokens* representing a network service. Third, the *Criteria Analyser* forms a set of structured criteria where each element reflects an atomic policy involved in the incoming *Goal Policy*. Fourth, the *Converter* maps each element of the criteria set to a particular goal notation required by an AP-planner. Fifth, NORA builds up the *AP-problem* by combining the obtained planning goals with the network status and management tasks. Figure 3.2 exemplifies the output of NORA at steps 4 and 5 in PDDL notation for a *Goal Policy* with a single (Figure 3.2a) and several goals (Figure 3.2b). In the next Subsections, we detail the NORA modules and how they interrelate to generate *AP-problems* from *Goal Policies*, network status, and network management tasks.

3.1.2 Lexer

This module receives management policies expressed by the network administrator in NL by following the *Goal Policy* model and extracts from them *tokens*. The *Lexer* builds up a matrix of *tokens* per each input policy as follows. First, it removes irrelevant terms to achieve faster

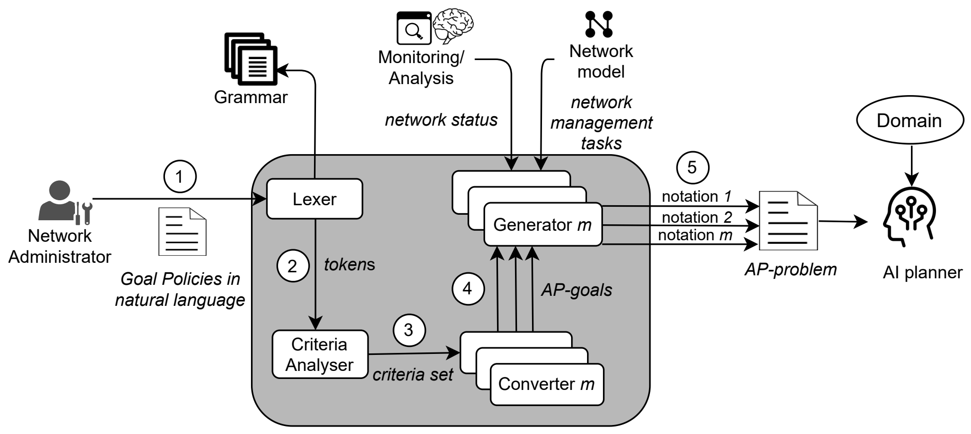
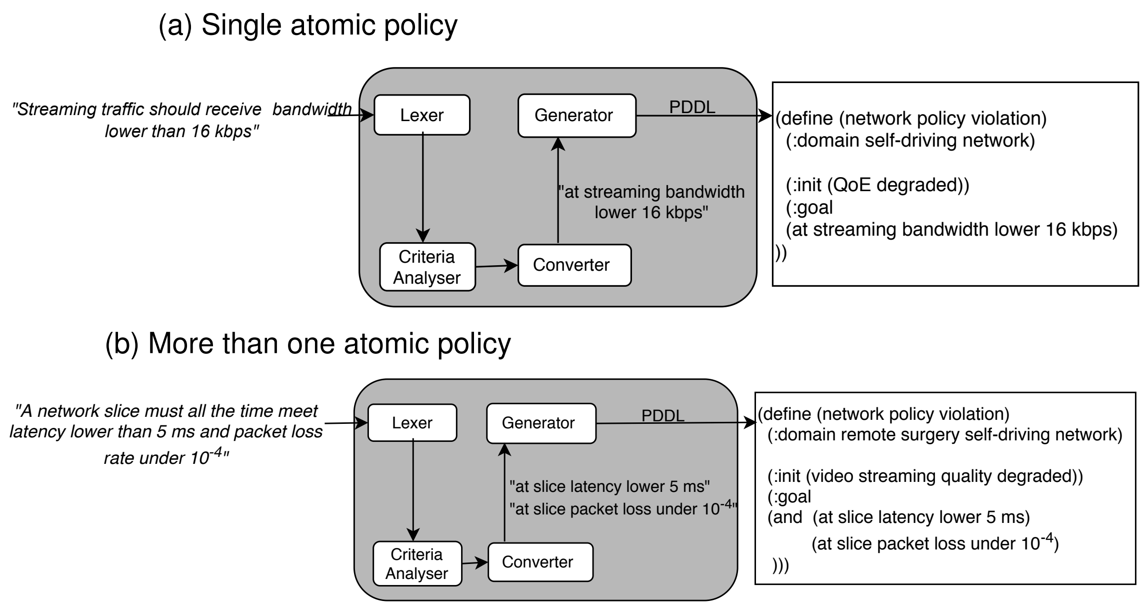


Figure 3.1: NORA Architecture


(a) Single atomic policy
"Streaming traffic should receive bandwidth lower than 16 kbps"
Lexer
Generator
PDDL
Criteria Analyser
Converter
"at streaming bandwidth lower 16 kbps"
(b) More than one atomic policy
"A network slice must all the time meet latency lower than 5 ms and packet loss rate under 10^-4"
Lexer
Generator
PDDL
Criteria Analyser
Converter
"at slice latency lower 5 ms"
"at slice packet loss under 10^-4"

```
(define (network policy violation)
  (:domain self-driving network)

  (:init (QoE degraded))
  (:goal
  (at streaming bandwidth lower 16 kbps)
))
```

```
(define (network policy violation)
  (:domain remote surgery self-driving network)

  (:init (video streaming quality degraded))
  (:goal
  (and  (at slice latency lower 5 ms)
          (at slice packet loss under 10^-4)
  )))
```

Figure 3.2: NORA - High-level operation

tokens identification. Let us suppose a network policy defined for a remote surgery scenario as P = "A network slice must all the time meet latency lower than 5 ms and packet loss rate under 10^{-4}". In P, the terms removed would be "a", "must", "all the time", and "meet".

Second, the *Lexer* performs stemming to reduce words composing terms; *e.g.*, in the raised policy "network slice" becomes "slice", "lower than" becomes "lower" and "packet loss rate" becomes "packet loss". Third, it carries out spell-checking to correct misspelled words or words damaged during stemming and compares remaining terms to expressions stored in a predefined domain grammar. Table 3.1 exemplifies our network management grammar based on [66]. Note that in Table 3.1 the Entity column corresponds to the 4-tuples defined for our policies model, *i.e.*, $Target, Metric, Condition, Threshold$, and the Expression column corresponds to terms of network management argot classified under each entity type. The *Connector* entity in the last row refer to expressions that allow us to determine whether an input policy includes several atomic policies, *i.e.*, it involves more than one tuple (see detail in subsection 3.1.3). Entities in the proposed grammar let categorize parts of an input policy instead of comparing with a set of specific policies; this offers flexibility to the extraction process.

Fourth, the *Lexer* marks as *tokens* the terms of the input policy matching grammar expressions and extracts their values and their positions in the original sentence. Figure 3.3 depicts the terms matched between the previous example policy *P* and the grammar (*i.e.*, "slice", "latency", "lower", "5 ms", "and", "packet loss", "under", "10^{-4}"), their corresponding entity type (*i.e., endpoint, metric, condition, threshold, connection, metric, condition, threshold*), and their start and end positions (*e.g.*, the term "slice" begins and ends at positions 11 and 15, respectively). A further 4-tuples format allows to structure data extracted per *token* as $t =< entityType, value, initialPosition, finalPosition >$. Note that from each input policy n

tokens can be marked, giving place to n tuples $t_1, t_2...t_n$. We defined a 4 x n matrix, called T, to store the n tuples representing *tokens* derived from query policies, *i.e.*, the rows of *T* are $t_1, t_2, ..., t_n$. As an example, rows $t_1, t_2...t_8$ in the $T(P)$ matrix in Figure 3.4 correspond to the eight tuples for the *tokens* marked in the policy P presented in Figure 3.3. Note that the data for the first *token* marked in P, *i.e.*, $t_1 =< endpoint, slice, 11, 15 >$, is the first row in $T(P)$ and so on. Fifth, the *Lexer* sends *T* to the *Criteria Analyser*.

Entity	**Expression**
Service	VoIP, Streaming, HTTP, FTP, SMTP, P2P...
Endpoint	gateway, database, slice, VM, CPU, client, user...
Metric	bandwidth, delay, throughput, jitter, load, latency, packet loss ...
Condition	more, high, higher, up, over, exceed, not under,...
	equal, like, even, same, similar,...
	less, lower, not exceed, down, below, under,...
Threshold-unit	ms, s, kbps, GB, GHz, %...
Connection	and, also, as well as, or...

Table 3.1: Network Management Grammar

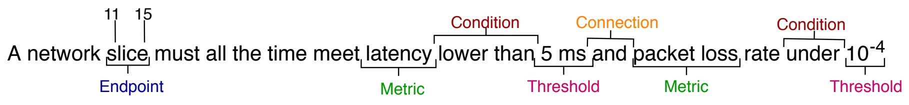

11
15
Condition
Connection
Condition
A network slice must all the time meet latency lower than 5 ms and packet loss rate under 10-4
Endpoint
Metric
Threshold
Metric
Threshold

Figure 3.3: Identification of entities in a *Goal Policy*

$$T(P) = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \\ t_7 \\ t_8 \end{bmatrix} = \begin{bmatrix} \text{endpoint} & \text{slice} & 11 & 15 \\ \text{metric} & \text{latency} & 40 & 46 \\ \text{condition} & \text{lower} & 48 & 52 \\ \text{threshold} & \text{5 ms} & 59 & 62 \\ \text{connection} & \text{and} & 64 & 66 \\ \text{metric} & \text{packet loss} & 68 & 78 \\ \text{condition} & \text{under} & 85 & 89 \\ \text{threshold} & 10^{-4} & 91 & 92 \end{bmatrix}$$

Figure 3.4: Matrix *T* for policy P

3.1.3 Criteria Analyzer

This module receives each T matrix computed by the *Lexer* and delivers a corresponding set of criteria involved in the *Goal Policy*; Figure 3.5a shows how each network criteria follows our policies tuples model. Thus, the *Criteria Analyzer* transforms every *T* matrix in a collection of network criteria: *i.e.*, $C = [c_1, c_2...c_k]$, where c_i is an atomic network management policy and k (*i.e.*, the size of C) represents the quantity of atomic policies contained in an input policy. The *tokens* of type *Connection* (Table 3.1) in a policy allows obtaining the k value (see Equation 3.1). For instance, in $T(P)$ (Figure 3.4) there is one *token* of type *Connection*, *i.e.*, t_5, which means that the raised policy *P* involves two atomic policies.

$$k = Connection \text{ in } T + 1; \tag{3.1}$$

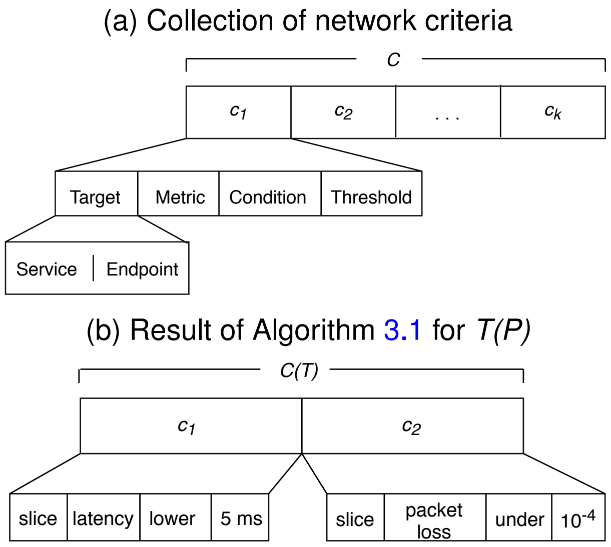


Figure 3.5: Criteria vs Goal Policy Model

Algorithm 3.1 transforms the matrix T into the set C. Initially, this algorithm counts the number of Connection *tokens* in T (line 1). Then, it calculates k (line 2) and creates a k-size string vector (line 3) (*e.g.,* in *P*, the values con=1 and k=2 - from Figure 3.4 and Equation 3.1- lead to $C = [c_1, c_2]$). Finally, it fulfills each c by performing a cycle with k iterations, each time completing a network criteria c_i conforming the set C (lines 4 to 14). In this cycle, Algorithm 3.1:

- Adds to c_1 the value of the *endpoint* or *service* with the minor $initialPosition$ in the matrix T (lines 5 and 6). In the example, t_1 is the $endpoint$ with the minor *initialPosition* (from Figure 3.4 *initialPosition(*t_1*)*=11); thus, at this step, $c_1 =$ "slice".

- Calculates proximity between *tokens* type *metric* and the previous selected *token* and adds the closest *metric* value to c_1 (lines 7 and 8). In the exemplified $T(P)$, this choice is t_2, hence, current c becomes $c_1 =$ "slice latency". The algorithm runs a similar process for *tokens* of type *condition* and *threshold* (lines 9 to 12). In this way, in our example, $c_1 =$ "slice latency lower 5 ms".

- Marks as "used" appended *tokens* (line 13). Note that they can be appended to more than one c_i when the quantity of a type of $token$ in T is less than k. For instance, observe that in $T(P)$ (Figure 3.4) t_1 is the only *token* of type *target* (*i.e.*, *service* or *endpoint*), thus, its value, *i.e.,* "slice", is appended to c_2 although it was earlier appended to c_1. On the other hand, tokens of the matrix T can be discarded of the resulting set C if there is a (already "used") $token$ of the same type closest to the previous element of the tuple.

Once Algorithm 3.1 ends up, the *Criteria Analyzer* sends the criteria set (C) to the *Converter*. In the example, the transformation of $T(P)$ after executing Algorithm 3.1 is $C = [c_1, c_2]$, where $c_1 =$"slice latency lower 5 ms" and $c_2 =$"slice packet loss under 10-4" (see Figure 3.5b).

Algorithm 3.1: Tokens Positions Comparison

```
1  $T$: Matrix of tokens.   $C = [c_1, c_2...c_k]$: Granular Goals Collection.   $con$ = tokens type
   Connection in $T$;
2  $k$ = $con$ + 1;
3  $C = [c_1, c_2...c_k]$;
4  for each $c$ in $C$ do
5      Search in $T$ the token of type $endpoint$ or $service$ with the minor initialPosition number;
6      c.append(endpoint|service.value);
7      Search in $T$ the closest token of type $metric$;
8      c.append(metric.value);
9      Search in $T$ the closest token of type $condition$;
10     c.append(constraint.value);
11     Search in $T$ the closest token of type $threshold$ ;
12     c.append($threshold$.value);
13     Mark in $T$ all already appended tokens
14 end
```

3.1.4 Converter

This module maps the elements of C (set of criteria: $c_1, c_2...c_k$) to a particular AP-goal notation (*e.g.,* PDDL) that is compatible with a specific planner (*e.g.,* Simple Hierarchical Ordered Planner [96] or Hierarchical Task Planner [97]). This planner can be used for closing the autonomic management loop. As there are multiple planning goal notations, the *Converter* defines a repository of m mapping functions (or converters) and a selection function that calls the appropriate mapping for the target *AP-problem* notation.

Listing 3.1: AI-planning goals in PDDL

```
at slice latency lower 5 ms
at slice packet loss under (10^{-4})
```

Listing 3.2: AI-planning goals in STRIPS

```
lower(slice, latency, 5 ms)
under(slice, packet loss, ( 10^{-4} ))
```

To exemplify the *Converter*, we overview its operation when using PDDL and STRIPS. In PDDL, the problem file reserves a piece of code for specifying goals. Listing 3.1 shows the syntax that the PDDL must generate. STRIPS defines problems using a boolean value function that allows describing the problem by logical conditions and specifies the problem goal as "things that we want to be true". Listing 3.2 shows that the STRIPS converter must generate the goal by placing the policy's $Condition$ followed by its remaining parameters (in brackets). This condition can be true or false.

3.1.5 Generator

From a general perspective, a planning-problem definition involves deciding what actions to execute given a goal and an initial state [90]. Thus, the primary entries for describing a problem in AI-planning based solutions are predicates defining an initial state, a problem goal, and (a reference to) a set of tasks (atomic or composed). NORA gets the problem goals automatically from *Goal Policies* expressed in NL. In turn, in NORA, the initial state corresponds to the network status obtained from an external Monitoring&Analysis module available in solutions running MAPE [94] or C-MAPE [91] ACLs. NORA assumes this loop delivers network status like "streaming quality degraded" and "QoE degraded". NORA retrieves the network management tasks from an external network model like the YANG-based and SDN-centered proposed in [95]). An example of a network management task is "scale up a VNF" or "scale down a VNF". We do not detail about *initial state* and *management tasks* because it is out of the scope of this thesis. We address them to get a closed ACL based on AI-planning techniques in future work. Summarizing, the *Generator* builds up an AI-*AP-problem* by combining a *Goal Policy* received from the *Converter*, the network status, and management tasks (along with their preconditions and effects). The implementation of this module follows a templates-based approach. Since NORA needs to build up problems into different AI-planning notations, similar to the *Converter*, m *Generators* are needed. Each one uses the corresponding template for the particular target notation.

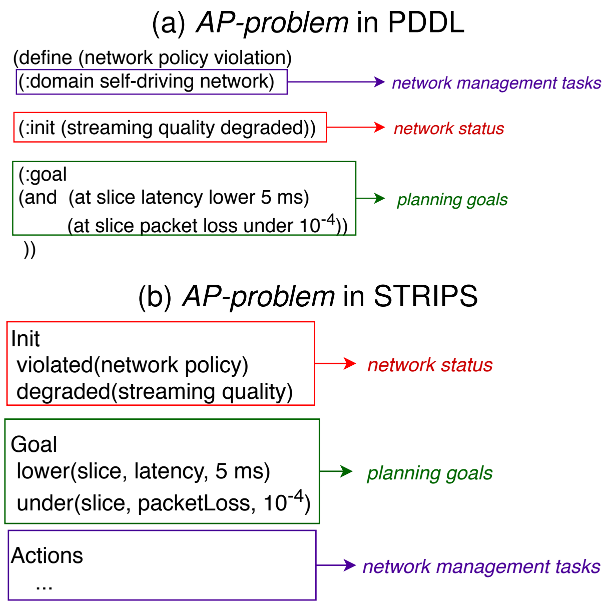

(a) AP-problem in PDDL
(define (network policy violation)
(:domain self-driving network)
network management tasks
(:init (streaming quality degraded))
network status
(:goal
(and (at slice latency lower 5 ms)
(at slice packet loss under 10^-4))
))
planning goals
(b) AP-problem in STRIPS
Init
violated(network policy)
degraded(streaming quality)
network status
Goal
lower(slice, latency, 5 ms)
under(slice, packetLoss, 10^-4)
planning goals
Actions
...
network management tasks

Figure 3.6: Example, *AP-problems*

Figure 3.6a shows an *AP-problem* described with the PDDL templates proposed in [98] [99]. In such templates, the problem attributes follow a schema-like representation including mainly: *i) name*, *i.e.*, a string used to identify the *AP-problem* - in the example "Network Policy Violation", *ii) domain* where *actions* (*i.e.*, network management tasks) are specified, *iii) initial state* is the network status; and *iv) goal state* containing one or several atomic goals that correspond to the *Goal Policy* in AP notation. Figure 3.6b shows an STRIPS-based *AP-problem* that includes the Sections: Init, Goal, and Actions. The Init Section corresponds to the network status. The Goal is the translated *Goal Policy*. The Actions are the network management tasks that the AP planner will use to achieve the Goal.

NORA sends the resulting *AP-problem* to the planner responsible for computing a management plan intended to obtain a closed network management ACL. An AP-based management plan is a sequence of management tasks that, once enforced in the underlying network, cause the network to go from the current status to another that meets the translated *Goal Policy*; recall, it is initially expressed in NL by the network administrator and translated to AP notation by NORA.

3.2 Evaluation

This evaluation aims to assess and discuss NORA's performance when generating *AP-problems* from *Goal Policies*, network status, and network management tasks. This Section initially introduces the prototype of NORA and the *Goal Policies* dataset used in the tests. This Section then describes the performance metrics assessed, namely *Precision* and *Processing Time*. Finally, this Section presents and discusses the NORA's evaluation results.

3.2.1 Prototype

Figure 3.7 depicts the prototype of NORA. The *Lexer* module was instantiated by using Rasa 1.10.0 [100], an ML-based NLP tool that allows understanding and manipulating NL for extracting *tokens* [101]. We used Rasa because a recent comparative study on NLP services' performance demonstrated that it overcomes similar tools, such as LUIS [102] and Lex [103], in terms of adaptability and customization thanks to its open-source nature [104]. The Linux Command Line is the user interface of NORA.

Figure 3.8 presents as example the *tokens* extracted by the Rasa-based *Lexer* when processing the *Goal Policy* P = "A network slice must all the time meet latency lower than 5 ms and packet loss rate under 10^{-4}". The data retrieved per policy are: *(i) end*, the position of the last character of the *token* in the policy, *(ii) entity*, the type of *token* according to the Grammar, *(iii) extractor*, an identifier for the ML-based engine used in the learning and extraction processes, *(iv) start*, the position of the first character of the *token* in the policy; and *(v) value*, the *token* itself as it appears in the policy.

We implemented the modules *Criteria Analyser*, *Converter*, and *Generator* as Python programs. These programs were integrated into the Rasa-based *Lexer* by inheriting from its *Action*

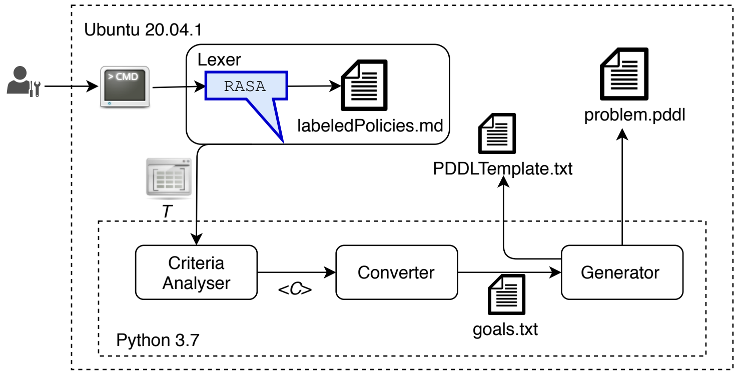


Figure 3.7: NORA Prototype

```
"project": "NORA"
ENTITIES
    {
        "end": 15,
        "entity": "endpoint",
        "extractor": "Mitie",
        "start": 11,
        "value": "slice"
    },
    {
        "end": 46,
        "entity": "metric",
        "extractor": "Mitie",
        "start": 40,
        "value": "latency"
    },
    {
        "end": 52,
        "entity": "condition",
        "extractor": "Mitie",
        "start": 48,
        "value": "lower"
    },
    {
        "end": 62,
        "entity": "threshold",
        "extractor": "Mitie",
        "start": 59,
        "value": "5 ms"
    },
    ...
]
```

Figure 3.8: Lexer Output

class [105]. Specifically, we developed a *Custom Action* that implements Algorithm 3.1 responsible for mapping *Goal Policies* into AP-goals in PDDL notation as in Listing 3.1 and generating PDDL-problems by filling out PDDL-problem templates [106]. These PDDL-problems stored in system files, jointly with a well-defined planning domain file, are enough input for executing an AI Planner, such as the STRIPS engine (Standford Research Institute Problem Solver) [107], responsible for automatically generating the corresponding management plan. For the sake of experimentation, we have used the PDDL notation for specifying the goals and problems of AP. However, it is noteworthy that the *Converter* and *Generator* modules can be implemented for NORA operates with other AP-notations (*e.g.,* STRIPS and Action Description Language).

3.2.2 Goal Policies Dataset and Lexer Tuning up

We created a *Goal Policies* dataset, available at our github repository[1], to tune up the NLP-based *Lexer* that allows NORA to learn how a network management policy is usually written and, so, to identify and extract *tokens* automatically; the NORA's *precision* heavily depends on the *Lexer* success. This dataset was built as follows. First, we collected 250 *Goal Policies* from 20 network management researchers. Second, we labeled each network management term of each policy with the corresponding entity type (*i.e., service*, *endpoint*, *metric*, *condition*, and *threshold*) according to our grammar (Table 3.1) and stored them in a plain text file ("labeledPolicies.md" in Figure 3.7) that Rasa is able to interpret as *training data*. For instance, the term "streaming" was labeled as *service*. Note that a term written in different ways -or with synonyms- like "P2P", "Peer to Peer" or "Peer-to-Peer" leads to the same label; in this example *service*. Third, we took the labeled policies as a base corpus for NORA and automatically generated further policies to obtain a dataset with 1000 *Goal Policies*. For this, we performed random combinations of terms for *services* or *endpoints*, *metrics*, *conditions* and *thresholds*, and added complementary expressions to complete phrases, *e.g.*, "On demand, network infrastructure must be configured for...", "... compared with other services...", "NORA, the network must ...".

We tuned up the *Lexer* module by using the cross-validation technique that allows using all available data for training and testing by splitting it into k number of groups [108] [109]; we used $k = 10$. Once tuned up, the NLP-based *Lexer* identified with high precision the *tokens*: $Service(92.6\%)$, $Metric(99.3\%)$, $Endpoint(93.2\%)$, and $Constraint(90\%)$. Conversely, this module identified with moderate precision (70%) the *tokens* of type $Threshold$; to increase this precision is necessary to add into the *Goal Policies* dataset more policies containing the label *Threshold*. Table 3.2 highlights in blue color as example some failures on the *Lexer's* operation, *i.e.*, *tokens* not identified or wrongly classified.

3.2.3 Performance Metrics

We evaluated NORA using the *Precision* and *Processing Time* performance metrics. *Precision* allows measuring whether NORA produces precise translations, meaning an *AP-problem*

[1]https://github.com/arodriguezvivas10/GoalPolicies

Tokens ground-truth labeled by experts (expected)					
ID	Service	Metric	Endpoint	Constraint	Threshold
1	- -	load	CPU / VM's	not exceed / not be under	80% / 20%
2	streaming	bandwidth	- -	higher than	20 Mbps
3	streaming	bandwidth	- -	higher than	20 Mbps
4	- -	latency	client B	less than	10 ms
5	download	- -	professors	no more than	1000000 MB per week

Lexer tokens identification					
ID	Service	Metric	Endpoint	Constraint	Threshold -unit
1	- -	load	CPU / - -	not exceed / not be under	80% / 20%
2	streaming	bandwidth	- -	higher than	20 Mbps
3	streaming	bandwidth	- -	higher than	20 Mbps
4	- -	latency	client	less than	10 ms
5	download	- -	professors	more than	–

Table 3.2: Lexer Tokens Identification vs Ground-Truth

generated by NORA includes the *Goal Policy*, network status, and network management tasks appropriate. High *Precision* is a mandatory requirement to push the NORA's adoption in self-driving networks. We measured $Precision$ as follows (see Figure 3.9). First, we created 1000 testing tuples $< gPol, gt >$ where $gPol$ represents an incoming *Goal Policy* and gt its corresponding *ground truth*. Each gt_i is the expected PDDL-problem file given the $gPol_i$ and assuming as known the network status and network management tasks. Second, we computed NORA precision by using Equation 3.2 where agr (agreement) is a boolean variable. agr is equal to 1 when the *AP-problem* generated by NORA (pf_i) matches the ground truth (gt_i) in terms of their textual content and syntax. Otherwise, agr is equal to 0. In turn, n refers to the total number of test query policies, and the summation A represents the overall NORA precision.

$$A = \sum_{i=1}^{n} \frac{agr_i}{n} \qquad agr = \begin{cases} 1 & \text{if } pp_i == gt_i \\ 0 & \text{in other case} \end{cases} \tag{3.2}$$

Processing Time allows measuring the speed of NORA for generating *AP-problems* . NORA's quickness is crucial when considering its adoption in self-driving networks because ACLs should address undesired network states (detected and triggered by Monitoring/Analysis modules) on-the-fly before the network instability expands and affects the Quality of Experience. We measured $ProcessingTime$ as the time elapsed since NORA receives a test query *Goal Policy* until it generates the corresponding *AP-problems* file, *i.e.*, from t_0 until t_f in Figure 3.9.

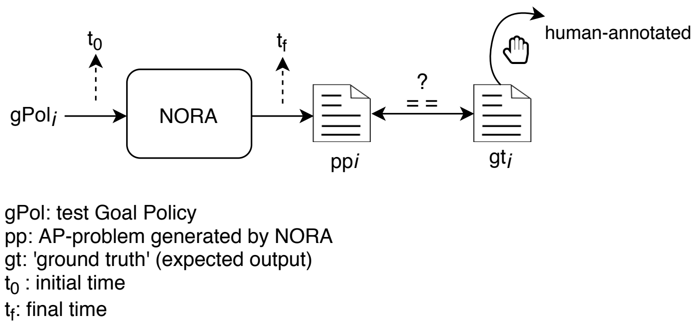


Figure 3.9: Test metrics

3.2.4 Results and Analysis

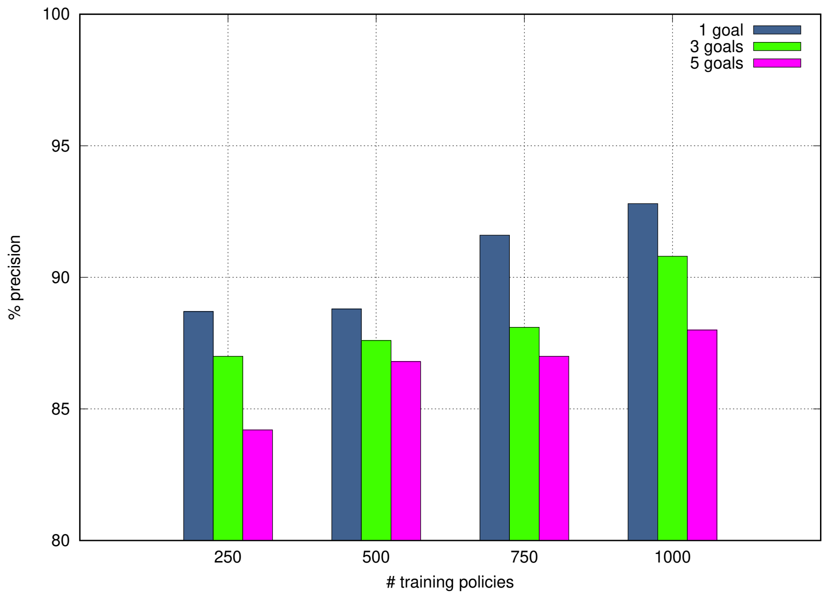


Figure 3.10: Precision vs Training Policies

Figure 3.10 depicts *Precision* values achieved by NORA as a function of the number of *training policies* and the granular goals involved in each test query *Goal Policy*. The NORA's *Precision* increases when the number of *training policies* rises; meaning that, as expected, a large *Goal Policy* dataset leads to improve the *Lexer* behavior regarding *tokens* identification. The *Precision* of NORA decreases when the number of granular goals expressed in the test query *Goal Policies* increases; meaning that complex policies hinder the NORA's behavior. In particular, NORA obtained the highest *Precision*, around 92.8%, with the dataset including 1000 *Goal Policies* and with a single granular goal per test query *Goal Policy*. NORA got the worst *Precision*, about 84.2%, with the dataset including 250 *Goal Policies* and with 5 granular goals per test query *Goal Policy*. The high-*Precision* obtained by NORA shows it is a promising solution to generate the *AP-problems* needed to close the autonomic management loops that allow realizing self-driving networks.

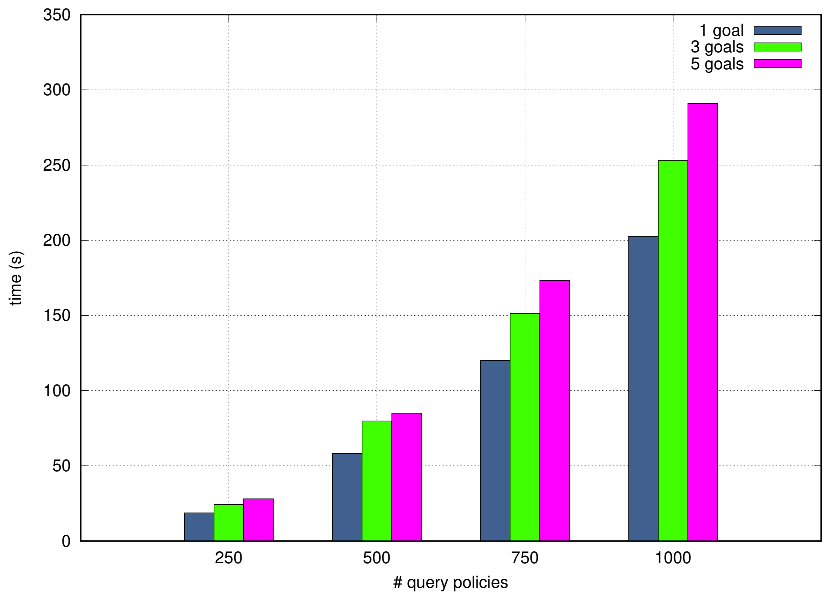


Figure 3.11: End-to-end Processing Time by Granular Goals

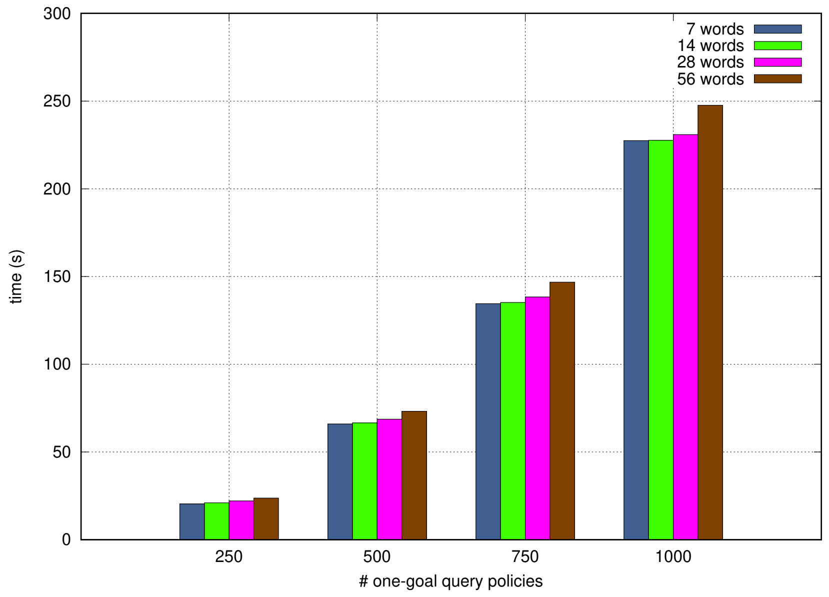


Figure 3.12: End-to-end Processing Time by Policy Length

Figures 3.11 and 3.12 depict the *Processing Time* as a function of the quantity of test query *Goal Policies* incoming one after another and the number of granular goals and words per tets query *Goal Policy*. The NORA's *Processing Time* increases when the number of incoming policies, goals, and words per policy grow up, although the last criteria, *i.e.*, words per policy, slightly alters the *Processing Time*. In particular, NORA obtained the worst *Processing Time*, around 290 seconds, when simultaneously translating 1000 policies with 5 goals each. NORA got the best *Processing Time*, about 20 seconds when simultaneously translating 250 policies with 1 goal per policy. The low-*Processing Time* obtained by NORA shows it allows closing the autonomic management loops quickly which is fundamental in the context of self-driving networks.

Since NORA is an *AP-problem* generation approach with no precedents in the networking domain, there is no conventional method to perform a direct comparison. Therefore, in the next lines, we compare NORA to HAUTO [65], a framework that includes an NLP-based module for transforming NL and environmental early warning information into PDDL problems. NORA achieved in average a *Precision* (92.8%) slightly lower than the obtained by HAUTO (94.4%). We can improve the NORA's *Precision* by increasing the dataset size and the grammar expressions; this is part of our next research steps. NORA when processing a *Goal Policy* including 5 goals got a *Processing Time* $(290$ milliseconds) equal to the achieved by HAUTO for analyzing a user requirement phrase and generating the PDDL problem file. This preliminary benchmark corroborates the NORA results are promising to put self-driving networks into reality.

3.3 Final Remarks

This Chapter introduced NORA, an approach that automatically generates *AP-problems* by transforming *Goal Policies* expressed in NL into AP-goals and combining them with both the network status and the network management tasks. The evaluation results showed that NORA achieves a precision near 92.8% and spends around 0.084 seconds on generating *AP-problems* , which evinces our approach is useful to overcome barriers to using AP to realize autonomic management scenarios that are pivotal for accomplishing self-driving networks.

We conceive NORA to operate with grammar and corpus defined in the English language since it is the most common language used in the network management area; for instance, network operators and equipment vendors usually specify Service Level Agreements (SLAs) and Command Line Interfaces (CLIs) in English. Therefore, we describe and evaluate the NORA's components using English sentences. However, it is remarkable that the NORA's architecture does not need changes to support *Goal Policies* specified in diverse languages.

NORA can be adapted to deal with *Goal Policies* expressed in diverse languages by following the next steps. First, redefining the network management grammar (Table 3.1) in the new language. Second, collecting *Goal Policies* in such a language; these policies are the basis for the corpus generation. Third, labelling the new-language *Goal Policies* corpus according to the grammar expressions. Fourth, tuning up the *Lexer* with the labelled policies (see Figure 3.7, *labeledPolicies.md* file).

Chapter 4

Tenant-oriented Reconfiguration of Network Slices based on Automated Planning

For tenants operating in 5G networks is fundamental to get the capability of reconfiguring the leased network slices since they need to react quickly to changing network conditions and end-users demands to meet Service Level Agreements (SLAs) [74]. Offering reconfiguration operations to tenants is an open research challenge [86] because, considering they can be non-expert networking professionals, such operations must be autonomous and governed by high-level network management policies.

Several conceptual investigations [73–75,78,79,81] proposed delegate network slicing management operations (*e.g.*, creation and deletion of slices [79]) to tenants. Nevertheless, those investigations did not focus on providing reconfiguration capabilities. Unlike the cited investigations, the works [48, 50, 83–87, 89] centered on addressing the Network Slicing Reconfiguration Problem (NSRP) [48] by using diverse techniques like Deep Reinforcement Learning (DRL), Integer Linear Programming (ILP), and heuristics. Those works faced NSRP from the InP perspective, so they disregarded the importance of providing reconfiguration capabilities to tenants. Furthermore, they did not consider Autonomous Control Loop (ACLs) neither high-level management policies governing their behavior, for instance, Monitoring-Analysis-Planning-Execution-Knowledge (MAPE-K), which are fundamental for realizing autonomic network (re)configuration.

In this chapter, we present ATRAP, an approach based on AP for facing NSRP from the tenant side; from now on called TsNSRP. ATRAP is pioneering using AP, an AI technique, for computing reconfiguration plans autonomously in a MAPE-K-based architecture. An ATRAP's plan aims to turn the network from a source *configuration* where one or more *intents* representing high-level network management policies are unmet into a target *configuration* in which the intents are met. As we represent a network slice as a SFC involving VNFs and virtual links connecting them, a reconfiguration plan involves migrating VNFs and virtual links into the SN in a particular order. Results revealed as ATRAP computes plans with few actions *i.e.*, migration of a VNF and its adjacent links), leading to low reconfiguration costs related to service disruption. Also, results showed as ATRAP's planning time depends on the size of SN and the tenant's virtualized network. Therefore, ATRAP is a promising solution for the tenant-side reconfiguration of network slices.

The contributions presented in this Chapter are:

- An architecture based on MAPE-K and AP for autonomic reconfiguration of network slices from the tenant-side and governed by *intents* representing tenant's high-level management policies.
- A system model for the tenant-side network slice reconfiguration problem based on State-Transition System, which enables AP as decision maker in the ATRAP's architecture.
- *AP-problems* that instantiate the system model in PDDL, which can be reused by researchers to explore the benefits of AP as decision maker for managing 5G network slices.

The reminder of this chapter is organized as follows. Section 4.1 introduces the MAPE-K based architecture of ATRAP. Section 4.2 presents the system model. Section 4.3 shows the evaluation process of ATRAP. Finally, we conclude the chapter in Section 4.4.

4.1 Architecture

Figure 4.1 depicts the proposed MAPE-K based ATRAP architecture for solving TsNSRP autonomously. The Monitoring & Analysis module checks the status (*i.e.*, current *configuration* and usage of SN resources) of the managed networks (*i.e.*, the SN and the tenant's virtualized network) to inform the Planning module about violation of *intents* defined by tenants. When one or more *intents* violations happen, the Planning module computes a reconfiguration plan to turn the network into a state where it meets the *intents* again. The calculated plan is enforced in the managed networks by the Execution module. In our architecture, the Knowledge Base is comprised by the Intents Repository and General Purpose Repository.

The architecture of ATRAP allows the reconfiguration of network slices to be carried out following autonomic network management principles [110] because the closed ACL limits the human intervention to the definition of high-level network management policies (*i.e.*, *intents* defined by the tenant) governing the ACL behaviour. In addition, the materialization of the Planning module with AP and the incorporation of NORA [21] in the ACL makes the reconfiguration process transparent to the ATRAP user, *i.e.*, the tenant owning network slices. Subsequently, we explain the individual modules operation.

The Monitoring & Analysis module creates the network status by periodically observing the current *configuration* of the managed networks and the level of usage of SN resources under such a *configuration*. For instance, at each *configuration* shown in Figure 2.5, the SN nodes and links have different levels of CPU and bandwidth usage. Monitoring & Analysis collects and compares such data to requirements involved in the *intents* previously recorded

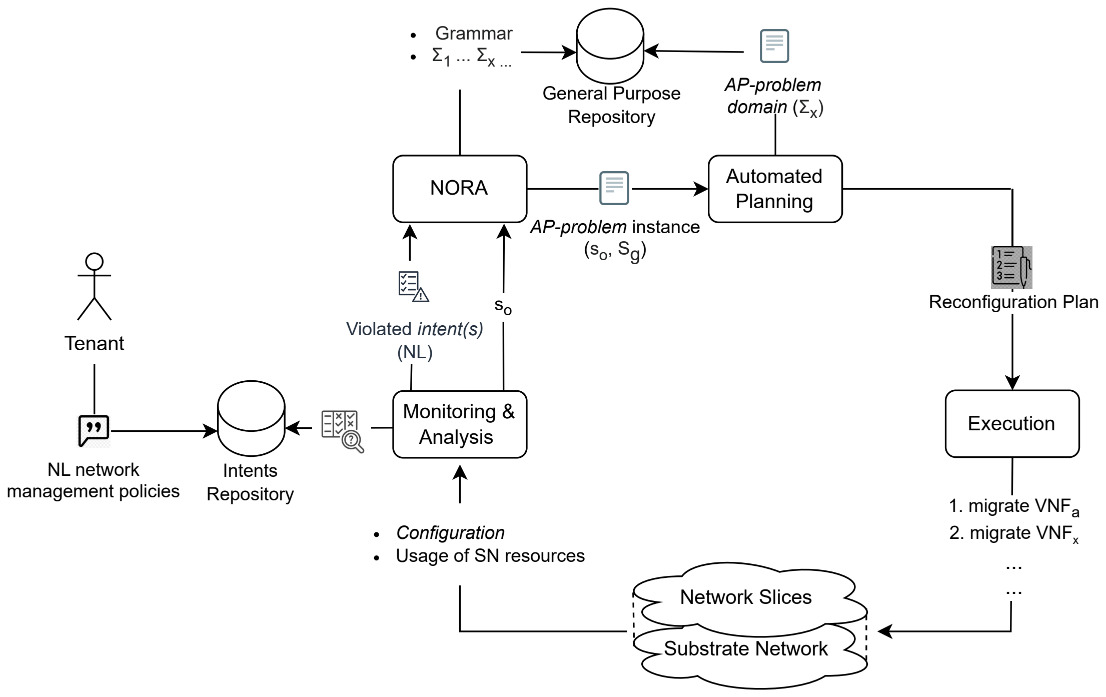


Figure 4.1: ATRAP Architecture

by the tenant in the Intents Repository. If the comparison reveals that one or more *intents* are being violated, a list of such violated *intents* (in NL) and a notification of undesired status s_o (*i.e.*, current *configuration* and level of usage of SN resources) are sent to NORA. The probing interval for data collection can be calculated with an intelligent monitoring approach like the proposed in [111]. The Intents Repository, introduced in Chapter 3, is a collection of high-level network management policies, expressed in NL, like *"No node in the substrate can be occupied in more than* 60% *of its total capacity"*.

The NORA module receives from Monitoring & Analysis, the undesired network status s_o and a list of *intents* defined by the tenant, violated at s_o. A General Purpose Repository is queried by NORA to get the Network Management Grammar (deep explanation on Section 3.1) and the available *AP-problem domains* $\sum_1 ... \sum_x$. The main task of NORA is to automatically transform the received *intents* (from NL) into a set of conditions defining the target state, which in AP is the *AP-goal* S_g. To fulfill this task, NORA uses natural language processing [64] to match input *intents* to Network Management Grammar expressions (Table 3.1). s_o, S_g, and the respective *AP-problem domain* $\sum_x$ are combined through *AP-problem* templates [106] to generate an *AP-problem* instance that will be processed by the Planning module. Readers interested on the *intents* transformation process, the *AP-problem* instances generation, and the construction of the dataset for the Intents Repository are referred to Chapter 3.

The Planning module is the core of the tenant-oriented reconfiguration of network slices. In ATRAP, this module is based on AP, which means that the *AP-problem* instance received from NORA, and the respective *AP-problem domain* $\sum_x$ queried from the General Purpose Repository are necessary and enough entries for its operation. In ATRAP, the *AP-problem* *domain* $\sum_x$ defines the reconfiguration operations, *i.e.*, possible VNF migrations, disclosed to tenants. This information could be derived from Management Level Agreements negotiated between InPs and tenants, like those proposed in [73]. In turn, the *AP-problem* instance defines the specific network situation at the time of monitoring, given by s_o and S_g. Planning is in charge of deciding which of the possible VNF migrations from $\sum_x$ will take place to achieve S_g starting from s_o, and the order of such migrations, through the calculation of a reconfiguration plan. *AP-problem domains*, *AP-problem* instances and the computed reconfiguration plans for TsNSRP are materialized in this thesis with PDDL, detailed in Sections 4.2 and 4.3.

The Execution module receives the reconfiguration plan from the Planning module and enforces it on the tenant's virtualized network. Once executed, the reconfiguration plan turns the managed networks from the source -undesired- state, given by s_o, where one or more *intents* are unmet, into a target state, given by S_g, where *intents* are met.

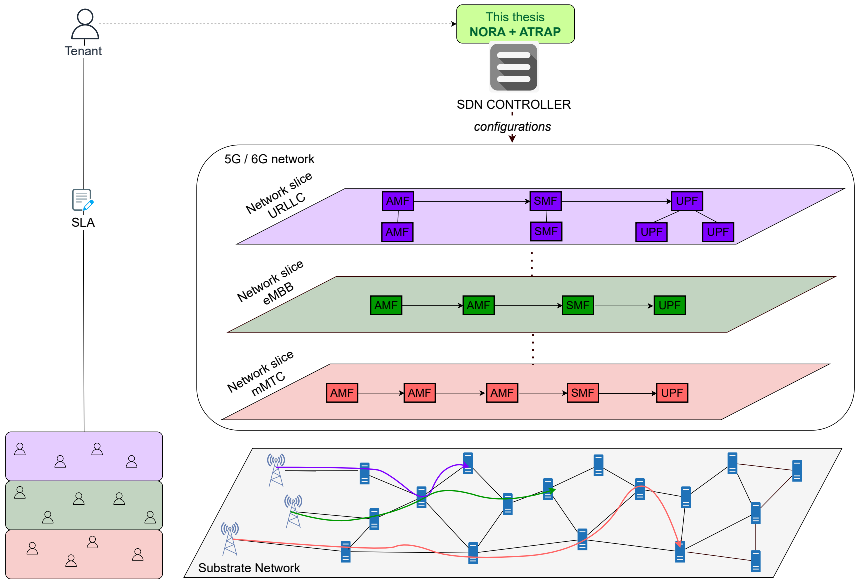


Figure 4.2: ATRAP and NORA in SDN

Figure 4.2 shows how could ATRAP and NORA be instantiated as an SDN controller module. Tenants interact with our self-reconfiguration system for SDN through a Tenant Portal that is a dedicated interface. The Tenant Portal allows the tenant to express their network mangement policies in NL, *i.e.*, the *intents* to be stored in the Intents Repository and further transformed by NORA. In the last step of the closed loop based ATRAP behaviour, Execution will enforce VNFs and virtual link migration (dictated by the reconfiguration plan complutmay by Planning) on network slices, changing the network *configuration* and the availability on individual resources on the SN.

4.2 Automated Planning for the Tenant-side Network Slice Reconfiguration Problem

In this Section we present the system model and the formulation of the TsNSRP. Also, we explain the generation of the PDDL *AP-domain* for the TsNSRP following prior formulations.

4.2.1 System Model

In ATRAP, the system model comprises four components as follows.

- A representation of the SN, *i.e.*, the computing nodes and the physical links connecting them.
- A representation of the virtualized network, *i.e.*, the VNFs and the virtual links in between for each SFC representing a network slice, which are dimensioned for the use cases the network is serving.
- A representation of the instantaneous *configuration* of both networks, *i.e.*, the *mapping* of the virtualized network onto the SN.
- A representation of the residual capacity of SN nodes and physical links given a *configuration*.

Let us consider SN as a labeled and weighted directed graph: $SN = \{N, P\}$ where N stands for the set of computing nodes, $N = \{n_1, n_2...n_m\}$, and P for the set of physical links, $P = \{(n_1, n_2), (n_1, n_3)...(n_l, n_m)\}$. The labels on the vertices give the processing node capacity, represented as CPU_{n_i} $\forall n_i \in N$. The weight on each edge gives the physical link bandwidth capacity, represented as $BW_{(n_i,n_j)}$ $\forall (n_i, n_j) \in P$. Every link $(n_i, n_j) \in P$ is bidirectional, and can be single or composed. (n_i, n_j) is single if the communication path between n_i and n_j is direct. (n_i, n_j) is composed if there are intermediate nodes, *e.g.*, n_x and n_z, for communicating n_i and n_j, *i.e.*, $(n_i, n_j) = (n_i, n_x) + (n_x, n_z) + (n_z, n_j)$. Either by single or composed physical links, communication must be guaranteed among the m nodes of the SN. Table 4.1 summarizes the notations used throughout the paper.

Let us assume the SFC of a network slice consists of h ordered VNFs linked to each other through $h-1$ bidirectional virtual links, denoted as $C : f_1 \to \cdots \to f_h$. The virtualized network is thus a collection of disjoint graphs $G = C_1 \oplus C_2 \oplus \cdots \oplus C_k$ where $C_1, C_2, ..., C_k$ are the k network slices being served simultaneously with its dedicated VNFs and virtual links. $G = \{F, V\}$ is a labeled and weighted directed graph where F stands for the set of VNFs accross the k network slices, $F = \{{f_1}^{C_1}, ..., {f_h}^{C_1}, ..., {f_1}^{C_k}, ..., {f_h}^{C_k}\}$, and V for the set of virtual links accross the k network slices, $V = \{(f_1, f_2)^{C_1}, ..., (f_{h-1}, f_h)^{C_1}, ..., (f_1, f_2)^{C_k}, ..., (f_{h-1}, f_h)^{C_k}\}$. Each $(f_{i-1}, f_i)^{C_j}$ links in bidirectional mode, ${f_{i-1}}^{C_j}$ with its successor in the SFC, ${f_i}^{C_j}$. The sizes of F and V are respectively $q = \sum_{j=1}^{k} h_{C_j}$ and $r = \sum_{j=1}^{k} (h-1)_{C_j}$. In G, the labels on the vertices and the weight on each edge, correspond to demands of processing and bandwidth, represented as ΔCPU_{f_i}, $\forall f_i \in F$ and $\Delta BW_{(f_{i-1},f_i)}$, $\forall (f_{i-1}, f_i) \in V$. To scale according to the 5G use case,

Symbol	**Description**
	SUBSTRATE NETWORK
$SN = \{N, P\}$	Set of computing nodes and physical links on the SN.
n_i	i-th node on SN, $n_i \in N$.
(n_i, n_j)	Physical link whose path communicates n_i and n_j, $(n_i, n_j) \in P$
m	Size of N, *i.e.*, number of computing nodes in SN.
z	Size of P, *i.e.*, number of physical links in SN.
CPU_{n_i}	Processing capacity of n_i.
$BW_{(n_i,n_j)}$	Bandwidth capacity of (n_i, n_j).
	VIRTUALIZED NETWORK
$G = \{F, V\}$	Set of VNFs and virtual links accross the virtualized network G .
C_j	j-th network slice composing G.
k	Number of network slices in G.
${f_i}^{C_j}$	i-th VNF of the network slice C_j, ${f_i}^{C_j} \in F$
$(f_{i-1}, f_i)^{C_j}$	Virtual link chaining ${f_{i-1}}^{C_j}$ and ${f_i}^{C_j}$, $(f_{i-1}, f_i)^{C_j} \in V$
h_{C_j}	Number of VNFs contained in C_j.
$(h-1)_{C_j}$	Number of virtual links contained in C_j.
q	Size of F, *i.e.*, number of VNFs in G
r	Size of V, *i.e.*, number of virtual links in G
$\Delta CPU_{{f_i}^{C_j}}$	Processing demand of VNF ${f_i}^{C_j}$.
$\Delta BW_{(f_{i-1},f_i)^{C_j}}$	Banwidth demand of virtual link $(f_{i-1}, f_i)^{C_j}$
$\epsilon({f_i}^{C_j})$	Scaling factor of ${f_i}^{C_j}$.
	CONFIGURATION AT TIME STEP t
$M(t)$	*Configuration* of the 5G network slicing (at time step t).
${f_i}^{C_j} \rightarrow n_i$	*Mapping* of ${f_i}^{C_j}$ onto n_i
$(f_{i-1}, f_i)^{C_j} \rightarrow (n_i, n_j)$	*Mapping* of $(f_{i-1}, f_i)^{C_j}$ onto (n_i, n_j)
$EP_{n_i}(t)$	Residual processing capacity of n_i.
$EP_{(n_i,n_j)}(t)$	Residual bandwidth capacity of (n_i, n_j).

Table 4.1: Notations

each ${f_i}^{C_j}$ is instantiated over one or more Virtual Machines or Dockers in the same computing node n_i. Thus, a scaling factor $\epsilon({f_i}^{C_j})$ is associated $\forall {f_i}^{C_j} \in F$ meaning how many containers are needed to instantiate ${f_i}^{C_j}$.

A *configuration* for network slicing details the instantaneous *mapping* of the virtual resources across the k network slices onto the SN resources [50]. Namely, at time step t, there is a mapping (embedding) of the q VNFs in F and the r virtual links in V onto a share of the elements of SN, $M(t) : G = \{F, V\} \rightarrow SN' = \{N', P'\}$, where $N' \subset N$ and $P' \subset P$. In turn, $M(t) = \{M_{C_1}(t), M_{C_2}(t), ..., M_{C_k}(t)\}$ and each $M_{C_j}(t)$ is split into VNFs *mapping* $M_{{f_i}^{C_j}}(t)$ and virtual links *mapping* $M_{(f_{i-1},f_i)^{C_j}}(t)$, as indicated in Equation 4.1. We call *embedding potential (EP)* [49] to the residual processing and bandwidth capacity of both, nodes and physical links of the SN given a *configuration M(t)*. They are respectively expressed in Equations 4.2 and 4.3. Note that if (n_i, n_j) is composed, *i.e.*, $(n_i, n_j) = (n_i, n_x) + (n_x, n_z) + (n_z, n_j)$, the use of resources in any of the single composing paths (n_i, n_x), (n_x, n_z), or (n_z, n_j) affects the value of $EP_{(n_i,n_j)}$.

$$M_{C_j}(t) : \begin{cases} M_{f_i{}^{C_j}}(t) : f_1{}^{C_j} \to n_x, ..., f_h{}^{C_j} \to n_z \\ M_{(f_{i-1},f_i)^{C_j}}(t) : (f_1, f_2)^{C_j} \to (n_x, n_y), ..., (f_{h-1}, f_h)^{C_j} \to (n_w, n_z) \end{cases} \tag{4.1}$$

$$EP_{n_i}(t) = CPU_{n_i} - \sum_{b=1}^{h_{C_a}} \sum_{a=1}^{k} \Delta CPU_{f_b{}^{C_a}} \times \epsilon(f_b{}^{C_a}), \ \Leftrightarrow M_{f_b{}^{C_a}}(t) : f_b{}^{C_a} \to n_i EP_{n_i}(t) \geq 0 \tag{4.2}$$

$$\text{EP}_{(n_i,n_j)}(t) = BW_{(n_i,n_j)} - \textstyle\sum_{b=1}^{h_{C_a}} \sum_{a=1}^{k} \Delta BW_{(f_{b-1},f_b)^{C_a}},$$

$$\Leftrightarrow M_{(f_{b-1},f_b)^{C_a}}(t) : (f_{b-1}, f_b)^{C_a} \to (n_i, n_j) EP_{(n_i,n_j)}(t) \geq 0 \tag{4.3}$$

4.2.2 Automated Planning Problem Formulation

In Section 2.7, we described the State Transition System based AP model as $\sum = (S, A, \gamma)$, where S, A, γ denote the finite set of states $S = \{s_1, s_2, s_3, ...\}$, the finite set of actions $A = \{a_1, a_2, ...\}$, and the state transition function $\gamma : S \times A \to 2^S$. In the present Section, we formulate the TsNSRP under such model.

State Space. The state space for TsNSRP is the set of all possible *configurations* for the managed networks, *i.e.,* the vertex of the *migration graph* exemplified in Figure 2.7. The search algorithm of the AI-planner traverses the *migration graph* to meet Sg conditions. Each node in the *migration graph* for TsNSRP determines: i) the current network *mapping* detailed by $M(t)$, and ii) the availability of resources in the SN given $M(t)$. Thus, in ATRAP we represent an state as $s(t) = \{M(t), EP_N(t), EP_P(t)\}$.

- $M(t)$ indicates the *mapping* of the k network slices being served at time t onto the SN as $M(t) = \{M_{C_1}(t), M_{C_2}(t), ..., M_{C_k}(t)\}$. Each $M_{C_j}(t)$ is split into VNFs mapping, and virtual links mapping as indicated in Equation 4.1.

- $EP_N(t)$ indicates the processing availability of the m nodes in SN at time step t as $EP_N(t) = \{EP_{n_1}(t), ..., EP_{n_m}(t)\}$. Values for EP_{n_i} are computed by Equation 4.2.

- $EP_P(t)$ indicates the bandwidth availability of the z physical links in SN at time step t as $EP_P(t) = \{EP_{(n_1,n_2)}(t), ..., EP_{(n_l,n_m)}(t)\}$. The value of z depends on m and on the specific SN topology. Values for $EP_{(n_i,n_j)}$ are computed by Equation 4.3.

Action Space. In Section 2.8 we introduced the *migration graph* representing all the possible states for the TsNSRP, *i.e.*, network *configurations* and availability of SN resources, and all the possible state transitions, *i.e.*, VNF migrations. In ATRAP, at state $s(t)$ the q VNFs in

F are candidates for being migrated among $m-1$ nodes in the SN. Thus, the action is an T-dimensional vector $A = \{a_1, a_2, ..., a_T\}$ where $T = q(m-1) \times 2$ (from Section 2.8).

$a_x \in A$ denotes the action of migrating ${f_i}^{C_j}$ from a source to a destination node. Namely, ${f_i}^{C_j} \rightarrow n_{source}$ at $s(t)$, is migrated to ${f_i}^{C_j} \rightarrow n_{destination}$ at $s(t+1)$ once a_x is applied. a_x carries the migration of $(f_{i-1}, f_i)^{C_j}$ and $(f_i, f_{i+1})^{C_j}$, adjacent virtual links of ${f_i}^{C_j}$, from source to destination physical links. Namely, $(f_{i-1}, f_i)^{C_j} \rightarrow (n_x, n_{source})$ and $(f_i, f_{i+1})^{C_j} \rightarrow (n_{source}, n_y)$ at $s(t)$ are respectively migrated to $(f_{i-1}, f_i)^{C_j} \rightarrow (n_x, n_{destination})$ and $(f_i, f_{i+1})^{C_j} \rightarrow (n_{destination}, n_y)$ at $s(t+1)$ once applied a_x. The reason for migrating adjacent virtual links of ${f_i}^{C_j}$ is to maintain communication between current locations of its neighbors VNFs in C_j, $f_{i-1}^{C_j} \rightarrow n_x$ and $f_{i+1}^{C_j} \rightarrow n_y$ (that are not been migrated) and the new location ${f_i}^{C_j} \rightarrow n_d$, since ${f_i}^{C_j}$ is being migrated from n_{source} to $n_{destination}$.

State Transition Function. a_x can be applied to transit the network from $s(t)$ to $s(t+1)$ only if conditions given by Equations 4.4, 4.5, and 4.6 are met. Condition given by Equation 4.4 indicates that ${f_i}^{C_j}$ can be migrated to n_d only if at time step t, n_d has an available amount of CPU at least equal to the amount of CPU demanded by ${f_i}^{C_j}$ multiplied by $\epsilon({f_i}^{C_j})$. $\epsilon({f_i}^{C_j})$ refers to how may containers are needed to instantiate ${f_i}^{C_j}$. In ATRAP, the $\epsilon({f_i}^{C_j})$ containers instantiating ${f_i}^{C_j}$ are mapped to the same physical node n_i. Likewise, conditions given by Equations 4.5 and 4.6 indicate that, to be considered as destination physical links, (n_x, n_d) and (n_d, n_y) must have, at time step t, an available amount of bandwidth at least equal to the amount of bandwidth demanded by the virtual links being migrated, *i.e.*, $(f_{i-1}, f_i)^{C_j}$ and $(f_i, f_{i+1})^{C_j}$.

$$\frac{EP_{n_d}(t)}{\Delta CPU_{{f_i}^{C_j}} \times \epsilon({f_i}^{C_j})} \geq 1 \tag{4.4}$$

$$\frac{EP_{(n_x,n_d)}(t)}{\Delta BW_{(f_{i-1},f_i)^{C_j}}} \geq 1 \tag{4.5}$$

$$\frac{EP_{(n_d,n_y)}(t)}{\Delta BW_{(f_i,f_{i+1})^{C_j}}} \geq 1 \tag{4.6}$$

where:

${f_i}^{C_j}$ - VNF to be migrated.

n_d - mapping node of ${f_i}^{C_j}$ at $s(t+1)$, *i.e.*, $M_{{f_i}^{C_j}}(t+1) : {f_i}^{C_j} \rightarrow n_d$

$f_{i-1}^{C_j}, f_{i+1}^{C_j}$ - previous and next VNFs of ${f_i}^{C_j}$ in the SFC, *i.e.*, $C_j : \cdots \rightarrow f_{i-1} \rightarrow f_i \rightarrow f_{i+1} \rightarrow \cdots$

n_x - mapping node of $f_{i-1}^{C_j}$ at states $s(t)$ and $s(t+1)$, *i.e.*, $M_{{f_{i-1}}^{C_j}}(t) : {f_{i-1}}^{C_j} \rightarrow n_x$ and $M_{{f_{i-1}}^{C_j}}(t+1) : {f_{i-1}}^{C_j} \rightarrow n_x$

n_y - mapping node of $f_{i+1}^{C_j}$ at states $s(t)$ and $s(t+1)$, *i.e.*, $M_{{f_{i+1}}^{C_j}}(t) : {f_{i+1}}^{C_j} \rightarrow n_y$ and $M_{{f_{i+1}}^{C_j}}(t+1) : {f_{i+1}}^{C_j} \rightarrow n_y$

4.2.3 Automated Planning Domain

As stated in Section 2.7, in AP, the *AP-domain* models the environment, *i.e.*, it defines the aspects of a problem that do not change regardless of what specific situation we are trying to solve [112]. In PDDL, such a model is defined trough *types*, *facts*, and *actions*. *Types* are for declaring entities of interest, *facts* are the properties of such entities in which we are interested; can be true/false or maintain a numeric value throughout the duration of the plan, *actions* allow state transitions. In this Section, we present the design of the *AP-domain* for this thesis, which allows the generalization in PDDL of the TsNSRP formulated in Sections 4.2.1 and 4.2.2.

Types. In PDDL, any entity involved in the problem situation is referred to as *types*. According to our system model (Section 4.2.1), the entities involved in the TsNSRP are the elements of $SN = \{N, P\}$ and $G = \{F, V\}$. We define four *types* for referring to elements of N, P, F, and V, respectively: *host-node*, *physical-link*, *vnf*, and *virtual-link*. Listing 4.1 shows the PDDL typing to this end. The code snippet in Listing 4.1 starts with the header that every PDDL *planning domain* must include, *i.e.*, our identification *name* "TsNSRP", and a *requirements* list, similar to import/include statements in programming languages. The *:fluents* requirement allows the inclusion of a *:function* block which represent numeric variables in the *planning domain*. The *:negative-preconditions* allows the use of *not* in preconditions. These requirements are applied in Listings 4.3,4.4 and 4.5, explained right away. A thorough explanation about *requirements* in PDDL can be found in [112].

Listing 4.1: Types for the TsNSRP *AP-domain*

```
(define (domain TsNSRP)
    (:requirements :fluents :negative-preconditions)
    (:types
        ; SUBSTRATE NETWORK
        host-node - object
        physical-link - object
        ; VIRTUALIZED NETWORK
        vnf - object
        virtual-link - object
    )
        ...
)
```

Facts. *facts* are properties associated to *types*, that are relevant in the environment model. In PDDL, boolean *facts* are referred to as *predicates* and numeric facts are referred to as *functions* (*a.k.a., numeric-fluents*). Listings 4.2 and 4.3 show our *facts* declaration for the *types* of Listing 4.1, following our system model.

In Listing 4.2, the lines commented as *"PHYSICAL LINKS"* indicate that every *physical-link type* forms a bidirectional communication path between n_i and n_j, *i.e.*, (n_i, n_j). The next code line indicates that each *vnf* has a successor on the SFC, *i.e.*, $f_1 \rightarrow \cdots \rightarrow f_h$. For f_h this predicate $next-vnf$ will not be instantiated because f_h does not have a successor (f_{h+1} does not exist). In the last two lines of Listing 4.2 we declare the *mapping* for each *vnf* and *virtual-link types* onto *host-node* and *physical-link types* alluding to our system model, $M(t) : G = \{F, V\} \rightarrow SN' = \{N', P'\}$, $N' \subset N$ and $P' \subset P$, and $M_{C_j}(t)$ splitted into $M_{{f_i}^{C_j}}(t)$ and $M_{(f_{i-1},f_i)^{C_j}}(t)$. Listing 4.3 shows the code snippet declaring numeric properties of the *types* in Listing 4.1. Line by line, this $functions$ section declares: $CPU_{n_i}, BW_{(n_i,n_j)}, \Delta CPU_{f_i}, \Delta BW_{(f_{i-1},f_i)}, \epsilon_{f_i}, EP_{n_i}, EP_{(n_i,n_j)}$.

Listing 4.2: Predicates for the TsNSRP *AP-domain*

```
(:predicates
    ; PHYSICAL LINKS
    (adjacent-nodes-i-j ?p-link-physical-link ?node-i ?node-j-host-node)
    (adjacent-nodes-j-i ?p-link-physical-link ?node-j ?node-i-host-node)
    ; SERVICE FUNCTION CHAIN
    (next-vnf ?vnf-to-migrate ?next-vnf - vnf)
    ; MAPPING: VIRTUAL -> PHYSICAL
    (vnf-mapped-to ?vnf - vnf ?h-node - host-node)
    (v-link-mapped-to ?v-link - virtual-link ?p-link - physical-link)
)
```

Listing 4.3: Functions for the TsNSRP *AP-domain*

```
(:functions
    ; SUBSTRATE NETWORK RESOURCES CAPACITY
    (cpu-capacity ?node - host-node)
    (bandwidth-capacity ?p-link - physical-link)
    ; VIRTUALIZED NETWORK RESOURCES DEMAND
    (required-cpu ?vnf - vnf)
    (required-bandwidth ?v-link - virtual-link)
    (scaling-factor ?vnf - vnf)
    ; AVAILABILITY OF RESOURCES ON SUBSTRATE NETWORK (EP)
    (embedding-cpu ?h-node - host-node)
    (embedding-bandwidth ?p-link - physical-link)
)
```

Actions. A PDDL *action* includes three Sections as follows. *Parameters* define which $types$ we are performing an action on. *Precondition*, a series of *facts* conjunctions and disjunctions which must be satisfied for the action to be applied. *Effect* define which *facts* should be set to true or false or change its numeric value if an action is applied. Listings 4.4 and 4.5 show our PDDL coding for $a_x \in A$ (from Section 4.2.2). The declared *parameters* call for: ${f_i}^{C_j}$, n_{source} (or n_s), $n_{destination}$ (or n_d), ${f_{i-1}}^{C_j}$, n_x, $(f_{i-1}, f_i)^{C_j}$, (n_x, n_s), (n_x, n_d), ${f_{i+1}}^{C_j}$, n_y, $(f_i, f_{i+1})^{C_j}$, (n_s, n_y), (n_d, n_y). In *precondition* we verify *mappings* at $s(t)$ (*i.e.*, prior to migration) and availability of resources in n_d, (n_x, n_d) and (n_d, n_y). That is, the conditions given by Equations 4.4, 4.5 and 4.6 are coded into PDDL in the *precondition* Section. In turn, in *effect* we update *mappings* at $s(t+1)$ (*i.e.*, after migration) and availability of resources in n_s, n_d, (n_x, n_s), (n_x, n_d), (n_s, n_y) and (n_d, n_y). That is, the values given by Equations 4.2 and 4.3 are updated by the PDDL coding in the *effect* Section.

Due to space issues we limit the *precondition* and *effect* codes shown in Listings 4.4 and 4.5 avoiding the code for the "next link to migrate", *i.e.*, for $(f_i, f_{i+1})^{C_j}$. Complete and functional *planning domain* following our TsNSRP model can be found in our github repository [1].

Listing 4.4: Action *migrate-vnf* for the TsNSRP *AP-domain*

```
(:action migrate-vnf
    :parameters(
        ?vnf-to-migrate - vnf
        ?from-node ?to-node - host-node
        ; PREVIOUS VNF ON SFC
        ?prev-vnf - vnf
        ?prev-vnf-node - host-node
        ?v-link1-to-migrate - virtual-link
        ?from-p-link1 ?to-p-link1 - physical-link
        ; NEXT VNF ON SFC
        ?next-vnf - vnf
        ?next-vnf-node - host-node
        ?v-link2-to-migrate - virtual-link
        ?from-p-link2 ?to-p-link2 - physical-link
    )
    :precondition(and
        ; VNF TO MIGRATE
        (vnf-mapped-to ?vnf-to-migrate ?from-node)
        (not(= ?from-node ?to-node))
        (>= (/(embedding-cpu ?to-node)
        (*(scaling-factor)(required-cpu ?vnf-to-migrate))) 1)
        ; PREVIOUS LINK TO MIGRATE
        (v-link-mapped-to ?v-link1-to-migrate ?from-p-link1)
        (not(= ?from-p-link1 ?to-p-link1))
        (next-vnf ?prev-vnf ?vnf-to-migrate)
        (vnf-mapped-to ?prev-vnf ?prev-vnf-node)
        (adjacent-nodes-i-j ?to-p-link1 ?prev-vnf-node ?to-node)
        (adjacent-nodes-j-i ?to-p-link1 ?to-node ?prev-vnf-node)
        (>= (/(embedding-bandwidth ?to-p-link1)
        (required-bandwidth ?v-link1-to-migrate)) 1)
        ; NEXT LINK TO MIGRATE
        ...
    )
    :effect(and
        ; VNF TO MIGRATE
        (vnf-mapped-to ?vnf-to-migrate ?to-node)
        (not (vnf-mapped-to ?vnf-to-migrate ?from-node))
        (decrease (embedding-cpu ?to-node)
        (*(scaling-factor)(required-cpu ?vnf-to-migrate)))
        (increase (embedding-cpu ?from-node)
```

[1] https://github.com/arodriguezvivas10/ATRAP-*AP-problems*

Listing 4.5: Action *migrate-vnf* for the TsNSRP *AP-domain*

```
            (*(scaling-factor)(required-cpu ?vnf-to-migrate)))
            ; PREVIOUS LINK TO MIGRATE
            (v-link-mapped-to ?V-link1-to-migrate ?to-p-link1)
            (not (v-link-mapped-to ?v-link1-to-migrate ?from-p-link1))
            (decrease (embedding-bandwidth ?to-p-link1)
            (required-bandwidth ?v-link1-to-migrate))
            (increase (embedding-bandwidth ?from-p-link1)
            (required-bandwidth ?v-link1-to-migrate))
            ; NEXT LINK TO MIGRATE
            ...
        )
    )
```

4.3 Evaluation

In this Section we evaluate ATRAP. Initially, we detail the setup for the experiments and introduce the evaluation metrics. Finally, we present and discuss the results obtained during experimentation.

4.3.1 Setup

The setup for our experiments involves: i) generation of *AP-problem* instances according to our *AP-domain* presented in Section 4.2.3; ii) tunning up of the test environment; and iii) definition of the metrics used for the evaluation.

Problem Instances

An *AP-problem* instance forms the other half of a *AP-problem* as $(\sum, s_0, S_g)$. In PDDL, the *AP-problem* instance solidifies *AP-domain* $(\sum)$ expressions for a particular situation into three distinct Sections as follows.

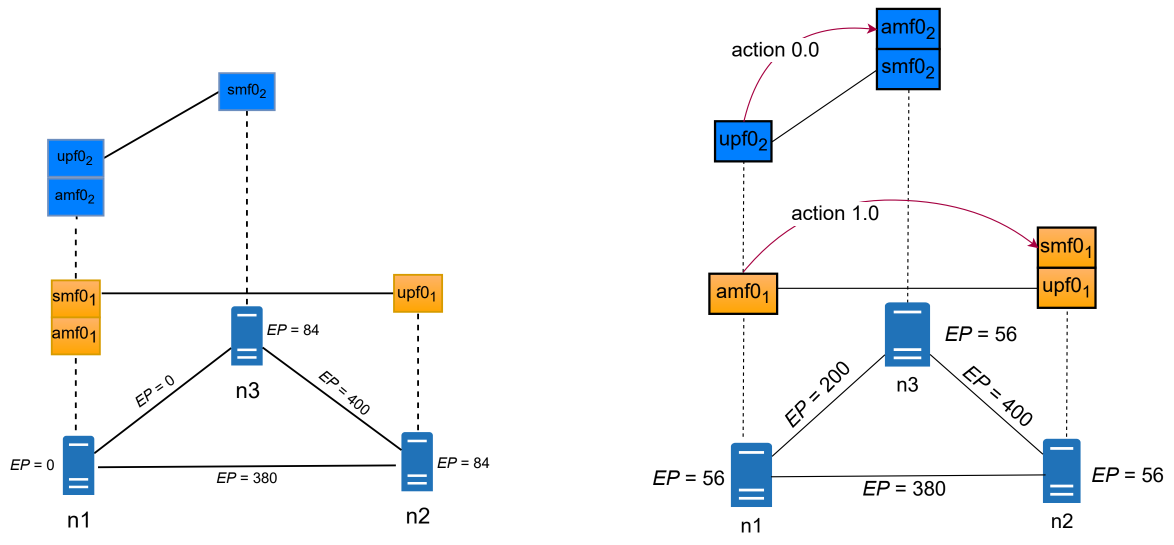
- *Objects* to define exactly what entities exist
- *Init* to declare what is true about each object and the value of their numeric properties at the initial state s_o.
- *Goal* to logically express *facts* as S_g that must be satisfied in order for a *plan* to be considered a solution

Parameter	Value and Units
m	3, 4, 5-node topology
k	2, 4, 6
h_{C_j}	3, 5
CPU_{n_i}	112 cores
$BW_{(n_i,n_j)}$	200, 400 Gbps
$\Delta CPU_{{f_i}^{C_j}}$	4.0 cores
$\Delta BW_{(f_{i-1},f_i)^{C_j}}$	URLLC:20 mMTC:200 Gbps
$\epsilon({f_i}^{C_j})$	scalar between 2 and 12
goal	cpu, bw , cpu+bw

Table 4.2: Experiments Setup

Listings 4.6, 4.7 and 4.8 present the coding for our smaller *AP-problem* instance exemplified in Figure 4.3a, where m=3, k=2 and h_{C_j}=3 (from Table 4.2). We generated a total of 28 TsNSRP instances by combining values of Table 4.2. We take the values of Table 4.2 from the research performed by Pozza *et al.* [50] at Nokia Bell Labs.

Listing 4.6 presents header and *objects*. We defined a structure for the identifier names of our instances alluding for the values of m, k, and h_{C_j}. In the example, "instance-3n-2s-3v" indicates 3 nodes, 2 network slices and 3 vnfs per network slice. In our instances, the k network slices are composed by the same number of VNFs, *i.e.*, $h_{C_1} = h_{C_2} = \cdots = h_{C_j}$. Also, the SN in all our instances follow a ring topology where $z = [(m/2)(m-1)] \times 2$. Next code line in Listing 4.6 points to the *AP-domain* $\sum_x$ that we are instantiating, *i.e.*, "TsNSRP" (from Listing 4.1). The *objects* Section instantiates the *types* defined in Listing 4.1 for the network exemplified in Figure 4.3a (m=3, z=3, k=2 and h_{C_j}=3). *inner-path* is a -ghost- physical link from node n_i to node n_i. At once, we instantiate two network slices with its dedicated VNFs and virtual links chaining them, C_1: $amf0_1 \rightarrow smf0_1 \rightarrow upf0_1$ and C_2: $amf0_2 \rightarrow smf0_2 \rightarrow upf0_2$.



(a) *Configuration* at s_o (b) *Configuration* where S_g is met

Figure 4.3: Example of an *AP-problem* instance and its solution

Listing 4.6: Objects for the TsNSRP instance

```
(define (problem instance-3n-2s-3v)
        (:domain TsNSRP)
        (:objects
                ; SUBSTRATE NETWORK
                ; Physical Nodes
                n1 - host-node
                n2 - host-node
                n3 - host-node
                ; Physical Links (all bidirectionals)
                n1-n2 - physical-link
                n1-n3 - physical-link
                n2-n3 - physical-link
                inner-path-n1 - physical-link
                inner-path-n2 - physical-link
                inner-path-n3 - physical-link
                ; Network Slice 1 -> URLLC
                amf0-ns1 - vnf
                smf0-ns1 - vnf
                upf0-ns1 - vnf
                amf0-smf0-ns1 - virtual-link
                smf0-upf0-ns1 - virtual-link
                ; Network Slice 2 -> mMTC
                amf0-ns2 - vnf
                smf0-ns2 - vnf
                upf0-ns2 - vnf
                amf0-smf0-ns2 - virtual-link
                smf0-upf0-ns2 - virtual-link
                )
    ...
)
```

Listing 4.7 instantiates s_o exemplified in Figure 4.3a. First two lines indicate bidirectionality of physical links (n_i, n_j). Next, we form C_1 (of type URLLC) by instantiating the $next - vnf$ $predicate$ declared in Listing 4.2, and we continue with its *configuration* $M_{C_1}(t)$ at s_o through the instantiation of the $vnf{-}mapped{-}to$ and $v{-}link{-}mapped{-}to\, predicates$ declared in Listing 4.2. This, and all *mappings* $M(t)$ thorough our problem instances were randomly selected with the criteria that the SN is 50% occupied at s_0. For this, $\epsilon(f_i)$ was adjusted for each problem instance with values from Table 4.2. Instantiation code for C_2 (of type mMTC) is cropped from Listing 4.7 due to space issues, but it is all similar to the C_1 instantiation. Complete *AP-problem* instances codes are available in our github repository [2]. Next code lines in Listing 4.7 instantiate each of the *functions* declared in Listing 4.3 with values from Table 4.2. In order to indicate negligent consumption of bandwidth for *inner-paths*, we assign them an extremely high EP value against no-*inner-paths*. *Init* code have also been cropped due to space issues.

[2]https://github.com/arodriguezvivas10/ATRAP-*AP-problems*

Listing 4.7: Initial state s_o for TsNSRP instance

```
(:init
        ; SUBSTRATE NETWORK
        (adjacent-nodes-i-j n1-n2 n1 n2)
        (adjacent-nodes-j-i n1-n2 n2 n1)
        ...
        ; VIRTUALIZED NETWORK
        ; NETWORK SLICE 1 -> URLLC (SERVICE FUNCTION CHAIN)
        (next-vnf amf0-ns1 smf0-ns1)
        (next-vnf smf0-ns1 upf0-ns1)
        ; vnfs mappings
        (vnf-mapped-to amf0-ns1 n1)
        (vnf-mapped-to smf0-ns1 n1)
        (vnf-mapped-to upf0-ns1 n2)
        ;virtual links mappings
        (v-link-mapped-to amf0-smf0-ns1 inner-path-n1)
        (v-link-mapped-to smf0-upf0-ns1 n1-n2)
        ; NETWORK SLICE 2 -> mMTC (SERVICE FUNCTION CHAIN)
        ...
        (=(cpu-capacity n1) 112) ; 4 CPUs x 28 cores
        ...
        (=(bandwidth-capacity n1-n2) 400) ; Gbps
        ...
        (=(required-cpu amf0-ns1) 4.0)
        ...
        (=(required-bandwidth amf0-smf0-ns1) 20) ; Gbps
        ...
        (=(required-bandwidth amf0-smf0-ns2) 200) ; Gbps
        ...
        (=(scaling-factor amf0-ns1) 7)
        ...
        ; AVAILABILITY OF RESOURCES IN SUBSTRATE NETWORK
        (=(embedding-cpu n1) 0)
        (=(embedding-cpu n2) 84)
        (=(embedding-cpu n3) 84)
        (= (embedding-bandwidth n1-n2) 380)
        (= (embedding-bandwidth n1-n3) 0)
        (= (embedding-bandwidth n2-n3) 400)
        (= (embedding-bandwidth inner-path-n1) 40000000)
        (= (embedding-bandwidth inner-path-n2) 40000000)
        (= (embedding-bandwidth inner-path-n3) 40000000)
)
```

Listing 4.8 shows the *AP-goal* for the exemplified *AP-problem* instance. The *intent* in our example is: *"No node in the substrate can be occupied in more than 60% of its total capacity"*. As explained in Section 4.1, this kind of *intents* are automatically transformed in this thesis from

NL into the PDDL *goal*. The *goal* code in Listing 4.8 is the cited *intent* after being processed by NORA [21], deeply explained in Chapter 3. For the sake of brevity we are exemplifying an *AP-goal* involving only cpu conditions, however, throughout our *AP-problem* instances the conditions involved in a *AP-goal* are threefold (from Table 4.2): cpu, bandwidth, cpu + bandwidth. For instance, a tenant interested at the same time in evenly usage of SN resources and saving bandwidth on physical links, expresses two different *intents*: *"No node in the substrate can be occupied in more than 60% of its total capacity"* and *"No path in the substrate can be occupied in more than 50% of its total capacity"*. Both (or more) *intents* are automatically processed by NORA and instantiated as *goal* in our *AP-problem* instances.

Listing 4.8: *AP-goal* S_g for TsNSRP instance

```
(:goal (and
    ; "No node in the substrate is occupied in more than 60% of
        its total capacity"
    (<(embedding-cpu n1)(*0.6(cpu-capacity n1)))
    (<(embedding-cpu n2)(*0.6(cpu-capacity n2)))
    (<(embedding-cpu n3)(*0.6(cpu-capacity n3)))
    )
)
```

Test Environment

Figure 4.4 shows the test environment of ATRAP, which was executed on an Ubuntu 22.04.1 LTS laptop with Intel Core i7-8565 CPU and 16 GB RAM. The coding of *AP-domain* and *AP-problem* instances were developed with PDDL 2.1 because it is the PDDL version that introduced the feature of numeric *facts*, *i.e.*, *functions* [113] (prior PDDL 1.2 supports just boolean *facts*, *i.e., predicates*). To make coding friendly, we used the free version of the Sublime Text [3] edition tool along with myPDDL [114], a modular toolkit for developing and manipulating PDDL files. We implemented the Planning module of the ATRAP architecture (Figure 4.1) with the ENHSP [4] (Expressive Numeric Heuristic Search Planner) AI-planner [59] [115] [116], which runs under the Java Runtime Environment 17.0.5. We chose ENHSP because it satisfies classical and numeric planning (*i.e.*, it supports PDDL 2.1). We worked with its latest available version to the date, ENHSP-20 [5].

Listing 4.9 shows the 2-actions reconfiguration plan computed by ENHSP-20 when its inputs were the *AP-domain* coded through Listings 4.1, 4.2, 4.3 and **??** and the *AP-problem* instance coded through Listings 4.6, 4.7 and 4.8. ENHSP-20 uses a variety of heuristics (*e.g.*, hadd, aibr, haddabs, landmarks) and search mechanisms (*e.g.*, wastar, idastar, dfsbnb) to process and transform those PDDL descriptions into a graph-search problem (like the *migration graph* of Figure 2.7) and come up with the output plan (displayed through the Linux command-line and as a plain text file). At each step of the reconfiguration plan shown in this Listing, a VNF ${f_i}^{C_j}$

[3]https://www.sublimetext.com/

[4]https://sites.google.com/view/enhsp/

[5]https://gitlab.com/enricos83/ENHSP-Public/-/tree/enhsp-20

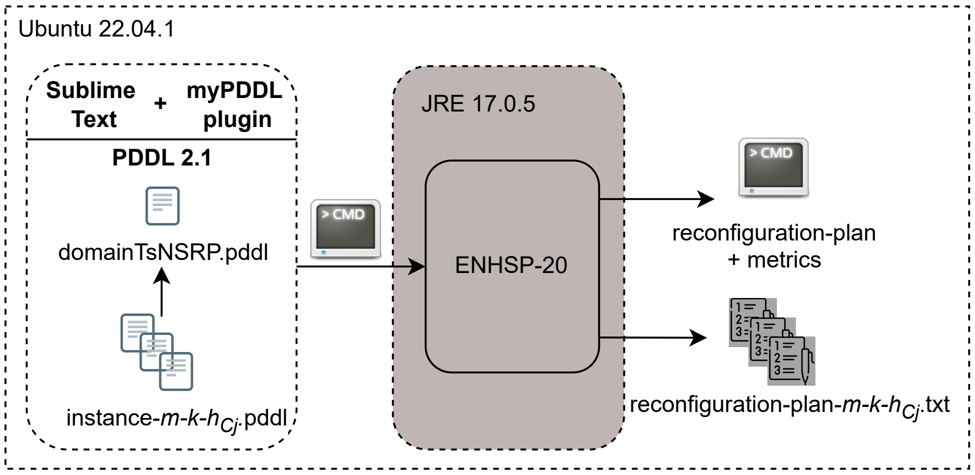

Ubuntu 22.04.1
Sublime Text + myPDDL plugin
PDDL 2.1
domainTsNSRP.pddl
instance-*m*-*k*-*h*$_{Cj}$.pddl
> CMD
JRE 17.0.5
ENHSP-20
> CMD
reconfiguration-plan + metrics
reconfiguration-plan-*m*-*k*-*h*$_{Cj}$.txt

Figure 4.4: Test Environment

is migrated along with its adjacent virtual links $(f_{i-1}, f_i)^{C_j}$ and $(f_i, f_{i+1})^{C_j}$. Once the reconfiguration plan is enforced on the tenant's virtualized network (by the Execution module of Figure 4.1), the managed networks turn from the source status given by the *configuration* depicted in Figure 4.3a, into a target status given by the *configuration* depicted in Figure 4.3b where the *AP-goal* of Listing 4.8 (coming from *intents* defined by the tenant) is met. A list of reconfiguration plans computed by ENHSP-20 for our *AP-problem* instances can be accessed in our github repository[6].

Listing 4.9: Reconfiguration Plan

```
0.0: (migrate-vnf amf0-ns2 n1 n3 smf0-ns2 n3 amf0-smf0-ns2 n1-n3
       inner-path-n3)
1.0: (migrate-vnf smf0-ns1 n1 n2 amf0-ns1 n1 amf0-smf0-ns1 inner-path-n1
      n1-n2 upf0-ns1 n2 smf0-upf0-ns1 n1-n2 inner-path-n2)
```

Metrics

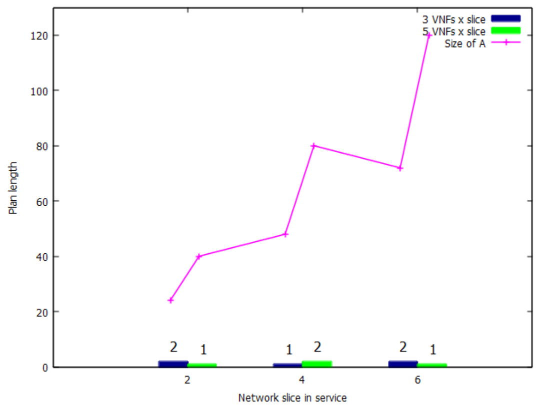
We evaluated ATRAP with two metrics: plan-length and planning time, both values provided by ENHSP after each execution. The plan-length is the number of actions composing a plan. In ATRAP, an action in the reconfiguration plan is the migration of a VNF and its adjacent virtual links (Listing 4.9). Considering that tenants must assume a cost for the service interruption caused by a VNF migration [85] [84], we take the plan-length as the reconfiguration cost. In turn, the planning time is the time used by ENHSP for coming up with the output plan. In ATRAP, this metric is of vital importance because remediation of undesired network status must be performed on-the-fly.

[6]https://github.com/arodriguezvivas10/ATRAP-*AP-problems*

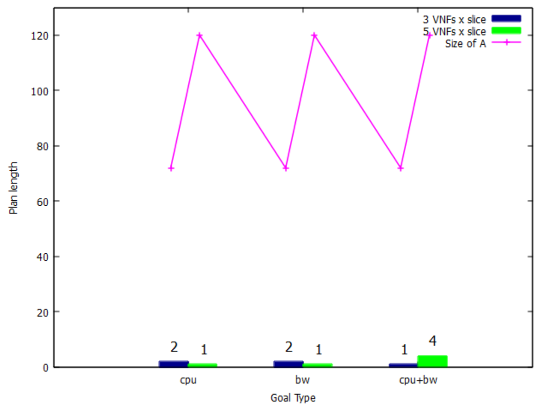
4.3.2 Results

Here, we present the evaluation results regarding first, plan-length (Figures 4.5 to 4.8), and second, planning time (Figures 4.9 to 4.12). All the tests were performed for network slices under the URLLC use case. We chose this use case because is the one with the stringent QoS requirements in 5G networks [117].

In Section 4.2.2 we defined the Action Space for the TsNSRP as a T-dimensional vector $A = \{a_1, a_2, ..., a_T\}$ where $T = q(m-1)\times 2$. The aim of Figures 4.5, 4.6 and 4.7 is to demonstrate how far is the length of each reconfiguration plan computed by ATRAP, from its corresponding T value, *i.e.*, the size of A (recall, A represents all possible VNF migrations, m is the total of nodes in the SN and q is the total of VNFs across all network slices). For instance, in Figure 4.5a the data for generating the pink line representing the size of A are (from left to right) m=3, q_1=6, q_2=10, q_3=12, q_4=20, q_5=18 and q_6=30 resulting in T_1=24, T_2=40, T_3=48, T_4=80, T_5=72 and T_6=120. Note that in this case the maximum of VNF migrations composing a reconfiguration plan is two. In Figure 4.5b, T values oscillate between 72 and 120 because here we have a fixed number of network slices in service, *i.e.*, k=6. In this case, the maximum of VNF migrations composing a reconfiguration plan is four.



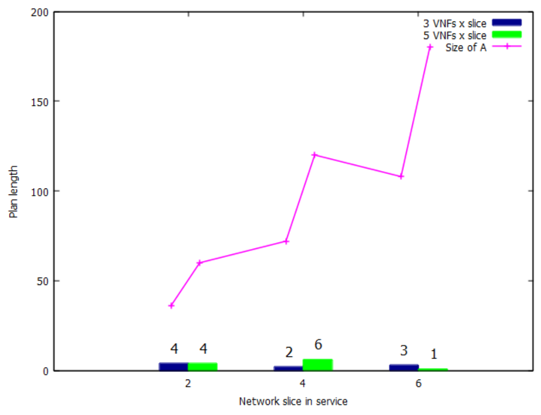
(a) Different networks slices in service (Goal=cpu)



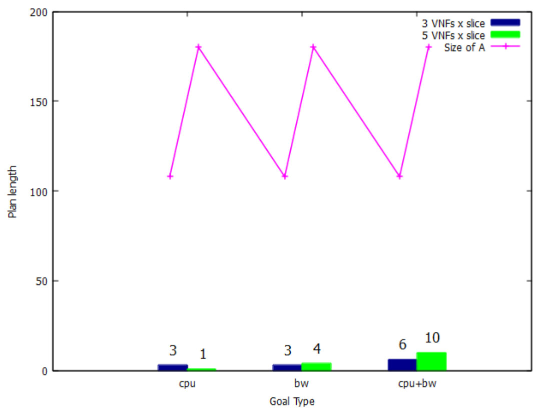
(b) Different types of goal (Network slices in service=6)

Figure 4.5: Plan-length for a 3-node SN

Results shown in Figures 4.6 and 4.7 are similar to those of Figure 4.5, for a 4-node and 5-node SN, respectively. The largest plan-length value throughout our evaluations is ten, from Figure 4.6b, which corresponds to 5.5% of the total *AP-domain* actions for its case, T=180. As stated in Section 4.3.1, the plan-length is directly associated with the reconfiguration cost assumed by the tenant, thus, the worst case is given for m=4, k=2 and h_C=3, shown in Figure 4.5a where the reconfiguration plan composed by 4 VNF migrations corresponds to the 11.11% of the total *AP-domain* actions for its case, T=36. Thereby, results regarding plan-length allow us to affirm that the reconfiguration cost that tenants must assume when executing an ATRAP's reconfiguration plan, is low with respect to the total number of possible VNF migrations declared as *AP-domain* actions, which cause service disruptions. Also, the results clarify that such a reconfiguration cost is independent of the quantity of network slices in service k, of the goal type (cpu, bandwidth, cpu+bandwidth), and of the SN size. The latter is corroborated with the results depicted in Figure 4.8.



(a) Different networks slices in service (Goal=cpu)



(b) Different types of goal (Network slices in service=6)

Figure 4.6: Plan-length for a 4-node SN

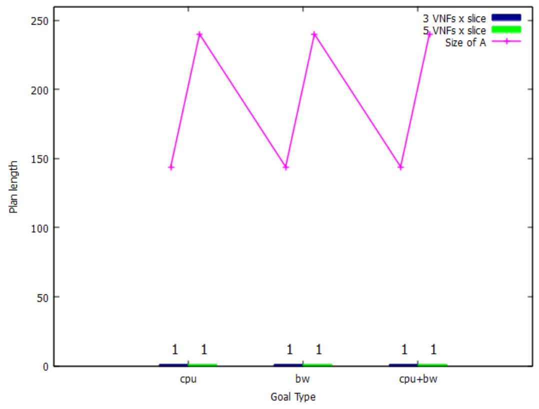
Unlike the plan-length of ATRAP's reconfiguration plans, the planning time in ATRAP is heavily influenced by the sizes of the SN and the tenant's virtualized network, specifically, by the parameters m, k and h_C. From Figures 4.9a, 4.10a and 4.11a we can clearly deduce that the planning time has an exponential growth as the SN has more nodes. Also, we can observe that the quantity of network slices in service k and the VNFs composing each slice h_C are parameters defining the planning time. Instead, the goal type has no influence on planning time which is revealed by results shown in Figures 4.9b, 4.10b and 4.11b. Finally, Figure 4.12 corroborates that the planning time is strongly commited to the dimensions of the SN and the virtualized network, with an inflection point at 4-node SN. Results throughout our tests vary between 2 seconds and 8 hours.

We allude the exponential increase of the planning time to the pure nature of AI-planners operation in which PDDL descriptions are first transformed into a graph-search problem where nodes represent states visited by the AI-planner. ENHSP builds this graph in an incremental-forward fashion, and is guided by a heuristic function to explore only those nodes whose associated state is reachable from the *init* and get the AI-planner closer to the *AP-goal* [118]. From our definition for *migration graph* in Section 2.8, the quantity of the vertex, representing the State Space in TsNSRP is m^q. The larger the graph, the larger the exploration time used by the AI-planner.

4.4 Final remarks

This Chapter presented ATRAP, an approach based on AP, MAPE-K, and high-level network management policies for addressing the problem of reconfiguring autonomously 5g network ch4-atrap/figures/PL5_a.png

(a) Different networks slices in service (Goal=cpu)



(b) Different types of goal (Network slices in service=6)

Figure 4.7: Plan-length for a 5-node SN

slices from the tenant perspective as a particularization of the network slicing reconfiguration problem faced in the literature in an in-provider way. AP allowed ATRAP to use high-level management policies to define when and what VNFs and their adjacent links into SFCs (representing slices) must migrate to bring the tenant's sliced network from a source configuration where intents are unmet to a target configuration (goal conditions) in which the tenant complies again with the intents. ATRAP included AP in the Planning phase of MAPE-K to reconfigure slices autonomously. Note that as ATRAP's reconfiguration plans are non-complex for understanding humans, tenants could, in the future, provide feedback to improve the reconfigurations. ATRAP computed plans with few actions (migrations of VNFs and their adjacent links), generating short service disruption and, consequently, low reconfiguration cost. The time spent by ATRAP to calculate the plans depends on the size of the substrate network and the tenant's slices. The results on planning time and reconfiguration cost showed as ATRAP is a promising solution for the in-tenant reconfiguration of network slices.

For future work, we intend to extend ATRAP to support 6G use cases. Also, we plan to combine AP with DRL to decrease ATRAP's planning time. Another interesting future work would be the creation of a tenant portal for providing reconfiguration reports.

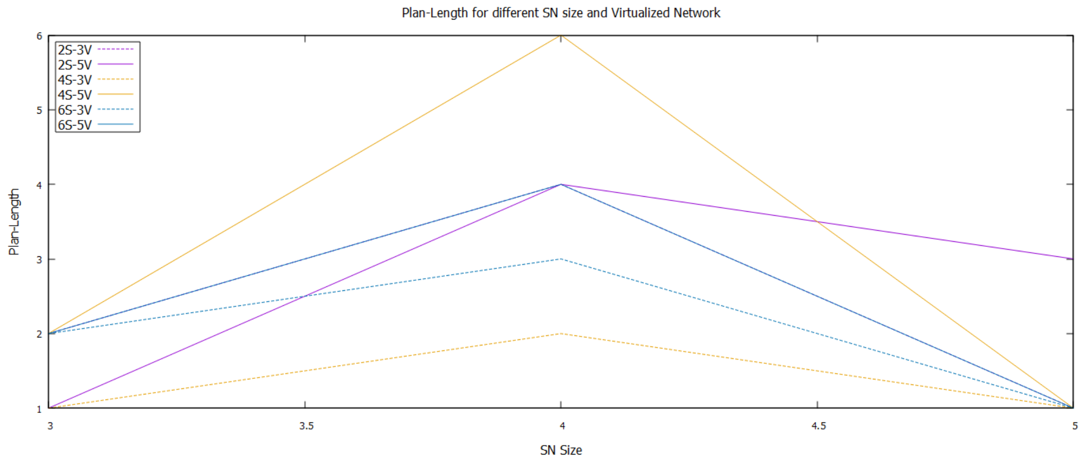
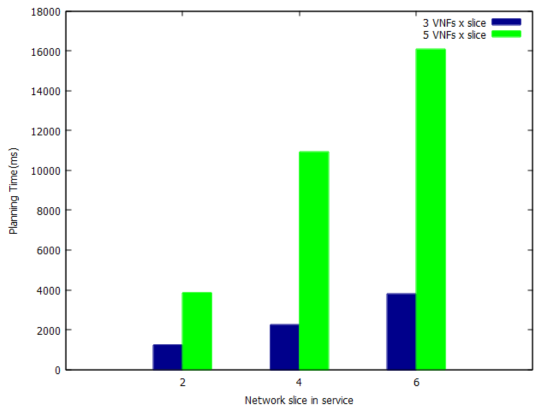
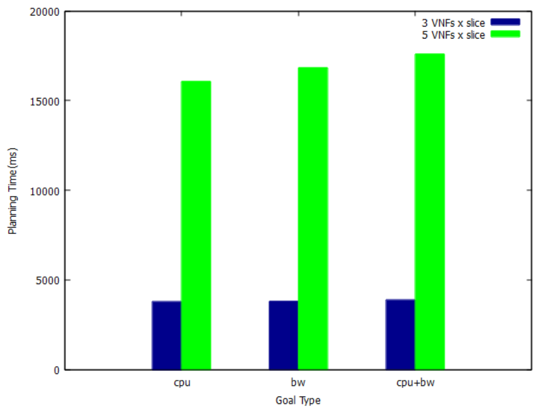


Figure 4.8: Plan-length for different sizes of SN and virtualized network (Goal=cpu)

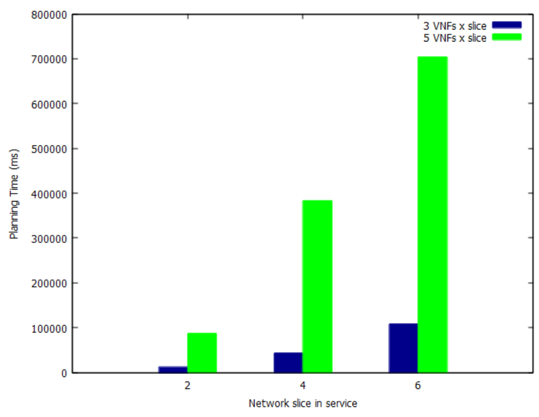


(a) Different networks slices in service (Goal=cpu)

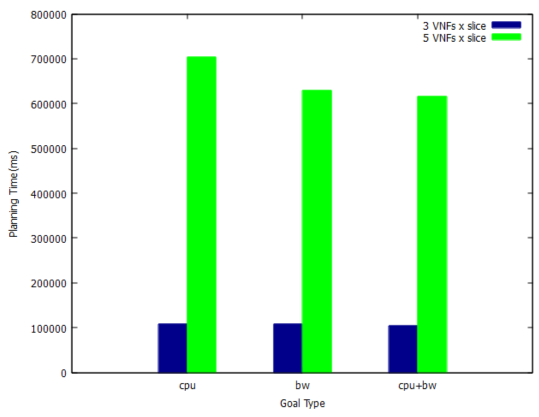


(b) Different types of goal (Network slices in service=6)

Figure 4.9: Planning time for a 3-node SN

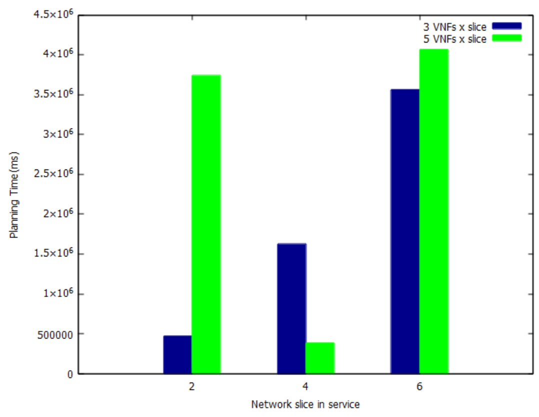


(a) Different networks slices in service (Goal=cpu)

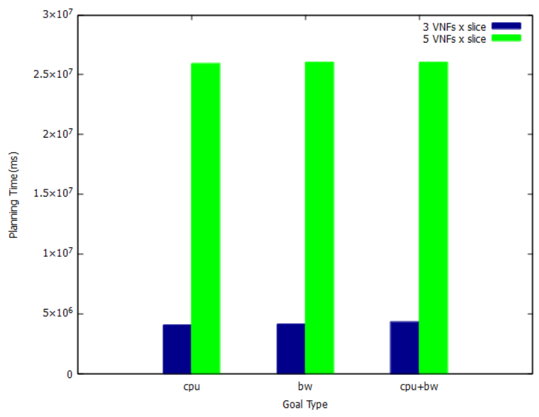


(b) Different types of goal (Network slices in service=6)

Figure 4.10: Planning time for a 4-node SN



(a) Different networks slices in service (Goal=cpu)



(b) Different types of goal (Network slices in service=6)

Figure 4.11: Planning time for a 5-node SN

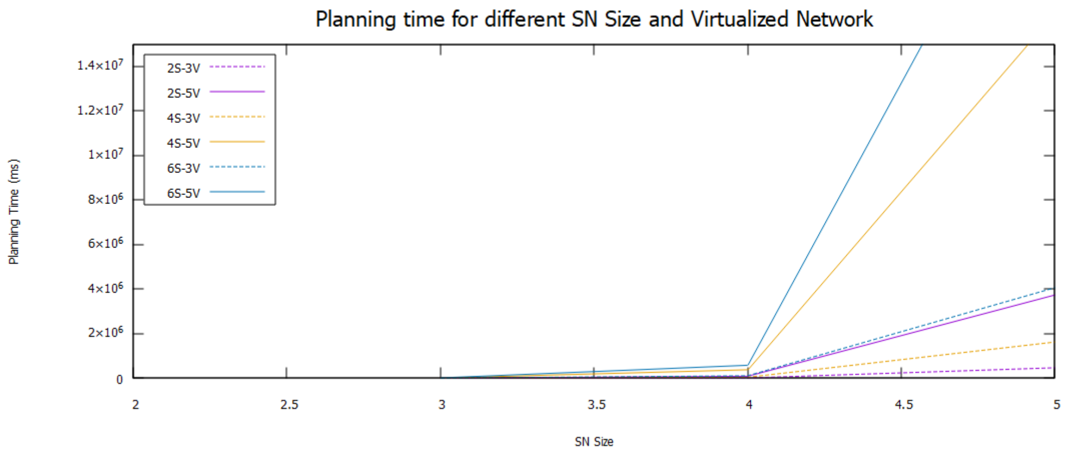


Figure 4.12: Planning time for different sizes of SN and virtualized network (Goal=cpu)

Chapter 5

Conclusions

This Chapter starts summarizing the research work carried out in this thesis. Then, it provides answers to the fundamental questions that guided the verification of the hypothesis defended in this thesis. The last Section outlines directions for future work.

This thesis presented the investigation carried out to verify the hypothesis: **an approach based on automated planning could compute SDN configuration plans (ordered tasks) on-the-fly to turn the network into a desired state.** Based on the hypothesis, this work proposed a reconfiguration approach that leverages natural language processing and AP to guide decision-making in an MAPE-K based autonomic management loop. Two major components form the proposed reconfiguration approach: NORA and ATRAP, which can be instantiated as an SDN controller module, as shown in Figure 4.2, and enforce configuration changes in the network slices with *e.g.*, OpenFlow [3] commands.

NORA is an interpreter from high-level network management policies to *AP-goal* of an *AP-problem* in an AP language like PDDL or Strips. A natural language dataset of network management policies was constructed, and we resorted to the natural language processing technique to transform them. Also, PDDL *AP-problem* templates were used during the transformation process. In addition, NORA merges the obtained *AP-goal* with an undesired network status and an specific reference to *AP-domain* to generate a complete *AP-problem* by using PDDL or Strips *AP-problem* templates. NORA was evaluated in terms of precision and processing time whose results let to conclude that NORA is a promising solution to overcome barriers to using AP in self-driving networks.

ATRAP is an approach for reconfiguring network slices based on AP. ATRAP allows the decision making of an autonomic management loop to be governed by the tenant intents. By using ATRAP, tenants are able to timely reconfigure their network slices with minimal dependance on InPs. A MAPE-K-based architecture was introduced for the operation of ATRAP. A detailed model was proposed for the tenant-side network slice reconfiguration problem including the SN, the tenant virtualized network, the *configuration* for the former networks and the *AP-domain* (based on State Transition System). Finally, a series of PDDL *AP-domains* and *AP-problem* instances were developed which can be reused by researchers interested in exploring the goodnesses (exposed in Section 4.1) of AP in the network management field. The evaluation and results of ATRAP allow to conclude that this thesis constitutes a reference point to researches interested in self-reconfigurable SDNs.

5.1 Answers to the fundamental questions

This Section reviews and answers the fundamental and secondary research questions guiding this thesis.

Given an undesired network state, how to compute (re)configuration management actions on-the-fly to turn SDN into a desired state?

Through the introduction of NORA and ATRAP, this thesis came up with a concrete answer to the research question: AP is a feasible technique for providing self-configuration in SDN. On the one hand, NORA was our primer in the research, initially conceived as a tool to bridge the interpretation gap between network management high-level policies and AI languages. The interesting results of NORA strengthened our research directions on extending its scope in order to achieve an autonomic management solution under a closed control loop. On the other hand, ATRAP effectively closes the autonomic management loop coping with the ANM goals: reduce human intervention in the loop. Thanks to the development of NORA and ATRAP, the road towards self-reconfigurable networks by exploiting the benefits of AP under the ANM paradigm is open and promising.

How to include the AP technique as a decision maker for network reconfiguration, under the ANM paradigm?

With the aim of answering this question we first did a flashback to the core principles of AP: i) AP allows to define what actions out of a potentially big one, need to take place, ii) these actions can only happen in particular orders, of which there are many, iii) the target configuration is specified through a set of goal conditions, iv) the output plan is known to the user. Also, considering the operation of AI-planners, which build a search graph from PDDL *AP-problem* and *AP-domain* descriptions, we guided our research on how a network reconfiguration problem could be embodied as such a search graph. That's how we got to reconfiguration of 5G network slicing. The migration of the VNF and virtual links composing network slices are the reconfiguration actions, of which there can be many depending on the offered services, quantity of end-users, use cases being served, and so on. Since in the 5G network slicing paradigm this is tenants who want to reconfigure network slices by themselves, they would provide the goal conditions as they own network slices demands.

The decision making role of AP in the tenant-oriented reconfiguration of network slices, demanded a closed ACL with regard of being autonomous. According to the state-of-the-art consulted around ANM and ZSM, we found a MAPE-K approach guided by high-level network management policies an adequate approach for setting up our main architecture, since the Planning module of MAPE-K is the one in charge of making decisions with embedded intelligence. That is how we materialized the Planning module with AP giving life to ATRAP, which relies on

NORA for the processing of the high-level network management policies, *i.e.*, the intens defined by tenant, guiding the ACL behaviour.

In short, this thesis demonstrated that AP, an yet unexplored AI technique in the network management field, can be leveraged to solve further network management problems because the *AP-domain* allows describing universal aspects of a problem. Our *AP-problem* files can serve as a starting point to model and solve more complex network reconfiguration problems.

Can the AP based decision making for network reconfiguration be guided by high-level network management policies?

Our research around high-level network management polices evolved along with the concept, *i.e.*, , from policies to *intents* , which is an special case of policy. Currently, the *intents* -based self-driving networking is a hot topic and involves ANM and ZSM paradigms (Figure 2.3). Our aim setting up this question was to bridge the interpretation gap between high-level network management polices or *intents* and AI languages, so that ACLs using embedded intelligence for network reconfiguration could be closed fulfilling the fundamentals of ANM and ZSM.

We found a goldmine in AP, since its operation is guided by goals, in turn represented as conditions. Thanks to NLP, we were able to automatically transform *intents* representing high-level network management policies defined by tenants, to the *AP-goals* . In addition we were able to merge obtained *AP-goals* with data coming from further modules of our architecture, *i.e.*, , the initial undesired network status detected by Monitoring & Analysis, and the *AP-domain* queried from the commom Knowledge Base, specifically, from a General Purpose Repository, producing PDDL *AP-problem* instances.

In short, through the joint operation of ATRAP and NORA, this thesis demonstrated that network management *intents* expressed in pure natural language can guide AP-based decision making for reconfiguration of network slices, providing to tenants the capability of timely reconfigure network slices with minimal dependence on InP, even with no prior knowledge on networking.

5.2 Future work

During the development of this thesis, we observed interesting opportunities for further research. These opportunities are outlined as follows.

- Extend ATRAP to support 6G use cases.
- Combine AP with DRL aiming to decrease ATRAP's planning time.
- Create a Tenant Portal for providing reconfiguration reports.

Bibliography

[1] E. Haleplidis, K. Pentikousis, S. Denazis, J. H. Salim, D. Meyer, and O. Koufopavlou, “Software-Defined Networking (SDN): Layers and Architecture Terminology.” RFC 7426, Jan. 2015.

[2] H. Kim and N. Feamster, “Improving network management with software defined networking,” *IEEE Communications Magazine*, vol. 51, pp. 114–119, February 2013.

[3] Y. Wang and I. Matta, “Sdn management layer: Design requirements and future direction,” in *IEEE 22nd International Conference on Network Protocols*, pp. 555–562, Oct 2014.

[4] N. Feamster, J. Rexford, and E. Zegura, “The road to sdn,” *ACM Queue*, vol. 11, pp. 20:20–20:40, Dec. 2013.

[5] K. Kirkpatrick, “Software-defined networking,” *Commun. ACM*, vol. 56, pp. 16–19, Sept. 2013.

[6] S. Kuklinski and P. Chemouil, “Network management challenges in Software-Defined Networks,” *IEICE Transactions on Communications*, vol. 97, no. 1, pp. 2–9, 2014.

[7] A. Schwabe, E. Rojas, and H. Karl, “Minimizing downtimes: Using dynamic reconfiguration and state management in sdn,” in *2017 IEEE Conference on Network Softwarization (NetSoft)*, pp. 1–5, July 2017.

[8] B. Jennings, S. Van Der Meer, S. Balasubramaniam, D. Botvich, M. Ó. Foghlú, W. Donnelly, and J. Strassner, “Towards autonomic management of communications networks,” *IEEE Communications Magazine*, vol. 45, no. 10, pp. 112–121, 2007.

[9] Z. Movahedi, M. Ayari, R. Langar, and G. Pujolle, “A survey of autonomic network architectures and evaluation criteria,” *IEEE Communications Surveys & Tutorials*, vol. 14, no. 2, pp. 464–490, 2012.

[10] N. Samaan and A. Karmouch, “Towards autonomic network management: an analysis of current and future research directions,” *IEEE Communications Surveys and Tutorials*, vol. 11, pp. 22–36, rd 2009.

[11] J. O. Kephart and D. M. Chess, “The vision of autonomic computing,” *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[12] S. Ayoubi, N. Limam, M. A. Salahuddin, N. Shahriar, R. Boutaba, F. Estrada-Solano, and O. M. Caicedo, "Machine learning for cognitive network management," *IEEE Communications Magazine*, vol. 56, no. 1, pp. 158–165, 2018.

[13] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "Payless: A low cost network monitoring framework for software defined networks," in *IEEE Network Operations and Management Symposium*, pp. 1–9, May 2014.

[14] N. L. M. van Adrichem, C. Doerr, and F. A. Kuipers, "Opennetmon: Network monitoring in openflow software-defined networks," in *IEEE Network Operations and Management Symposium*, pp. 1–8, May 2014.

[15] X. T. Phan and K. Fukuda, "Sdn-mon: Fine-grained traffic monitoring framework in software-defined networks," *Journal of Information Processing*, vol. 25, pp. 182–190, 2017.

[16] X. T. Phan, I. D. Martinez-Casanueva, and K. Fukuda, "Adaptive and distributed monitoring mechanism in software-defined networks," in *2017 13th International Conference on Network and Service Management (CNSM)*, pp. 1–5, Nov 2017.

[17] W. L. da Costa Cordeiro, G. S. Machado, F. G. Andreis, A. D. dos Santos, C. B. Both, L. P. Gaspary, L. Z. Granville, C. Bartolini, and D. Trastour, "Changeledge: Change design and planning in networked systems based on reuse of knowledge and automation," *Computer Networks*, vol. 53, no. 16, pp. 2782–2799, 2009.

[18] J. A. Wickboldt, W. P. D. Jesus, P. H. Isolani, C. B. Both, J. Rochol, and L. Z. Granville, "Software-defined networking: management requirements and challenges," *IEEE Communications Magazine*, vol. 53, pp. 278–285, January 2015.

[19] A. Mestres, A. Rodriguez-Natal, J. Carner, P. Barlet-Ros, E. Alarcón, M. Solé, V. Muntés-Mulero, D. Meyer, S. Barkai, M. J. Hibbett, G. Estrada, K. Ma'ruf, F. Coras, V. Ermagan, H. Latapie, C. Cassar, J. Evans, F. Maino, J. Walrand, and A. Cabellos, "Knowledge-defined networking," *SIGCOMM Comput. Commun. Rev.*, vol. 47, pp. 2–10, Sept. 2017.

[20] F. Volpato, M. P. D. Silva, A. L. Gonçalves, and M. A. R. Dantas, "An autonomic qos management architecture for software-defined networking environments," in *IEEE Symposium on Computers and Communications (ISCC)*, pp. 418–423, July 2017.

[21] A. Rodriguez-Vivas, O. M. Caicedo, A. Ordoñez, J. C. Nobre, and L. Z. Granville, "Nora: An approach for transforming network management policies into automated planning problems," *Sensors*, vol. 21, no. 5, p. 1790, 2021.

[22] A. Ordonez, O. M. Caicedo, W. Villota, A. Rodriguez-Vivas, and N. L. da Fonseca, "Model-based reinforcement learning with automated planning for network management," *Sensors*, vol. 22, no. 16, p. 6301, 2022.

[23] A. Rodríguez-Vivas, L. A. Eraso, J. C. Nobre, and O. M. C. Rendón, "Framework for autonomic management in software defined networks,"

[24] S. Crawford and L. Stucki, “Peer review and the changing research record,” *J. Am. Soc. Inf. Sci.*, vol. 41, pp. 223–228, Mar. 1990.

[25] O. N. Foundation, “Sdn architecture v1.0, technical reference tr-502,” tech. rep., 2014.

[26] D. B. Rawat and S. R. Reddy, “Software defined networking architecture, security and energy efficiency: A survey,” *IEEE Communications Surveys Tutorials*, vol. 19, pp. 325–346, Firstquarter 2017.

[27] I. T. Union, “Framework of software-defined networking, recommendation y.3300,” tech. rep., 2014.

[28] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-defined networking: A comprehensive survey,” *Proceedings of the IEEE*, vol. 103, pp. 14–76, Jan 2015.

[29] F. Estrada-Solano, A. Ordonez, L. Z. Granville, and O. M. C. Rendon, “A framework for sdn integrated management based on a cim model and a vertical management plane,” *Elsevier Computer Communications*, vol. 102, pp. 150 – 164, 2017.

[30] B. A. A. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka, and T. Turletti, “A survey of software-defined networking: Past, present, and future of programmable networks,” *IEEE Communications Surveys Tutorials*, vol. 16, pp. 1617–1634, Third 2014.

[31] “Management framework for open systems interconnection (osi) for ccitt applications,” september 1992.

[32] A. Voellmy, H. Kim, and N. Feamster, “Procera: A language for high-level reactive network control,” in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN ’12, (New York, NY, USA), pp. 43–48, ACM, 2012.

[33] M. H. Behringer, M. Pritikin, S. Bjarnason, A. Clemm, B. E. Carpenter, S. Jiang, and L. Ciavaglia, “Autonomic Networking: Definitions and Design Goals.” RFC 7575, June 2015.

[34] J. Strassner, N. Agoulmine, and E. Lehtihet, “Focale: A novel autonomic networking architecture,” 2006.

[35] S. Kim, J.-M. Kang, S.-s. Seo, and J. W.-K. Hong, “A cognitive model-based approach for autonomic fault management in openflow networks,” *ACM International Journal of Network Management*, vol. 23, no. 6, pp. 383–401, 2013.

[36] T. B. Meriem, R. Chaparadza, B. Radier, S. Soulhi, and A. P. López, “Gana-generic autonomic networking architecture,” 2016.

[37] M. H. Behringer, B. E. Carpenter, T. Eckert, L. Ciavaglia, P. Pierre, B. Liu, J. C. Nobre, and J. Strassner, “A Reference Model for Autonomic Networking,” Internet-Draft draft-ietf-anima-reference-model-05, Internet Engineering Task Force, Oct. 2017. Work in Progress.

[38] E. Coronado, R. Behravesh, T. Subramanya, A. Fernàndez-Fernàndez, M. S. Siddiqui, X. Costa-Pérez, and R. Riggio, "Zero touch management: A survey of network automation solutions for 5g and 6g networks," *IEEE Communications Surveys Tutorials*, vol. 24, no. 4, pp. 2535–2578, 2022.

[39] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2429–2453, 2018.

[40] J. O. Kephart and W. E. Walsh, "An artificial intelligence perspective on autonomic computing policies," in *Proceedings. Fifth IEEE International Workshop on Policies for Distributed Systems and Networks, 2004. POLICY 2004.*, pp. 3–12, June 2004.

[41] H. Derbel, N. Agoulmine, and M. Salaün, "Anema: Autonomic network management architecture to support self-configuration and self-optimization in ip networks," *Computer Networks*, vol. 53, no. 3, pp. 418 – 430, 2009.

[42] S. Lohmüller, *Cognitive Self-Organizing Network Management for Automated Configuration of Self-Optimization SON Functions*. PhD thesis, Universität Augsburg, 2019.

[43] K. Mehmood, K. Kralevska, and D. Palma, "Intent-driven autonomous network and service management in future networks: A structured literature review," *arXiv preprint arXiv:2108.04560*, 2021.

[44] A. Clemm, L. Ciavaglia, L. Z. Granville, and J. Tantsura, "Intent-Based Networking - Concepts and Definitions." RFC 9315, Oct. 2022.

[45] K. Dzeparoska, N. Beigi-Mohammadi, A. Tizghadam, and A. Leon-Garcia, "Towards a self-driving management system for the automated realization of intents," *IEEE Access*, vol. 9, pp. 159882–159907, 2021.

[46] N. F. S. de Sousa and C. E. Rothenberg, "Clara: Closed loop-based zero-touch network management framework," in *2021 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pp. 110–115, IEEE, 2021.

[47] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network slicing in 5g: Survey and challenges," *IEEE communications magazine*, vol. 55, no. 5, pp. 94–100, 2017.

[48] F. Wei, G. Feng, Y. Sun, Y. Wang, and Y.-C. Liang, "Dynamic network slice reconfiguration by exploiting deep reinforcement learning," in *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, 2020.

[49] W. F. Villota-Jacome, O. M. C. Rendon, and N. L. da Fonseca, "Admission control for 5g core network slicing based on deep reinforcement learning," *IEEE Systems Journal*, 2022.

[50] M. Pozza, P. K. Nicholson, D. F. Lugones, A. Rao, H. Flinck, and S. Tarkoma, "On reconfiguring 5g network slices," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 7, pp. 1542–1554, 2020.

[51] F. Wei, G. Feng, Y. Sun, Y. Wang, S. Qin, and Y.-C. Liang, "Network slice reconfiguration by exploiting deep reinforcement learning with large action space," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2197–2211, 2020.

[52] M. Ghallab, D. Nau, and P. Traverso, *Automated planning and acting*. Cambridge University Press, 2016.

[53] D. S. Nau, "Current trends in automated planning," *AI magazine*, vol. 28, no. 4, p. 43, 2007.

[54] S. Russel, "Artificial intelligence. a modern approach/russel s., norvig p," 2007.

[55] C. Aeronautiques, A. Howe, C. Knoblock, I. D. McDermott, A. Ram, M. Veloso, D. Weld, D. W. SRI, A. Barrett, D. Christianson, *et al.*, "Pddl| the planning domain definition language," *Technical Report, Tech. Rep.*, 1998.

[56] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "Pddl: The planning domain definition language, version 1.2," *Yale Center for Computational Vision and Control, Tech Report CVC TR98003/DCS TR1165*.

[57] J. Espasa, J. Coll, I. Miguel, and M. Villaret, "Towards lifted encodings for numeric planning in essence prime," in *CP 2019 Workshop on Constraint Modelling and Reformulation*, 2019.

[58] W. M. Piotrowski, M. Fox, D. Long, D. Magazzeni, and F. Mercorio, "Heuristic planning for pddl+ domains," in *Workshops at the Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[59] E. Scala, P. Haslum, S. Thiébaux, *et al.*, "Heuristics for numeric planning via subgoaling," 2016.

[60] A. Coles, A. Coles, M. Fox, and D. Long, "Temporal planning in domains with linear processes," in *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.

[61] M. A. Gironza-Ceron, W. F. Villota-Jacome, A. Ordonez, F. Estrada-Solano, and O. M. C. Rendon, "Sdn management based on hierarchical task network and network functions virtualization," in *2017 IEEE Symposium on Computers and Communications (ISCC)*, pp. 1360–1365, July 2017.

[62] W. Villota, M. Gironza, A. Ordoñez, and O. M. C. Rendon, "On the feasibility of using hierarchical task networks and network functions virtualization for managing software-defined networks," *IEEE Access*, vol. 6, pp. 38026–38040, 2018.

[63] F. Liu, *Supporting IT Service Fault Recovery with an Automated Planning Method*. PhD thesis, lmu, 2011.

[64] A. Ordoñez, J. C. Corrales, and P. Falcarin, “Natural language processing based services composition for environmental management,” pp. 497–502, July 2012.

[65] A. Ordonez, V. Alcázar, J. C. Corrales, and P. Falcarin, “Automated context aware composition of advanced telecom services for environmental early warnings,” *Expert Systems with Applications*, vol. 41, no. 13, pp. 5907 – 5916, 2014.

[66] C. C. Machado, J. A. Wickboldt, L. Z. Granville, and A. E. Schaeffer Filho, “Policy authoring for software-defined networking management.,” in *IM*, pp. 216–224, 2015.

[67] C. C. Machado, J. A. Wickboldt, L. Z. Granville, and A. Schaeffer-Filho, “Arkham: An advanced refinement toolkit for handling service level agreements in software-defined networking,” *Journal of Network and Computer Applications*, vol. 90, pp. 1 – 16, 2017.

[68] C. C. Machado, L. Z. Granville, A. Schaeffer-Filho, and J. A. Wickboldt, “Towards sla policy refinement for qos management in software-defined networking,” in *2014 IEEE 28th International Conference on Advanced Information Networking and Applications*, pp. 397–404, May 2014.

[69] D. Tuncer, M. Charalambides, G. Tangari, and G. Pavlou, “A northbound interface for software-based networks,” in *2018 14th International Conference on Network and Service Management (CNSM)*, pp. 99–107, Nov 2018.

[70] A. S. Jacobs, R. J. Pfitscher, R. A. Ferreira, and L. Z. Granville, “Refining network intents for self-driving networks,” in *Proceedings of the Afternoon Workshop on Self-Driving Networks*, SelfDN 2018, (New York, NY, USA), pp. 15–21, ACM, 2018.

[71] M. Riftadi and F. Kuipers, “P4i/o: Intent-based networking with p4,” in *2019 IEEE Conference on Network Softwarization (NetSoft)*, pp. 438–443, June 2019.

[72] P. Widmer and B. Stiller, *Design and Implementation of an Intent-based Blockchain Selection Framework*. PhD thesis, Master’s thesis, University of Zurich, 2020.

[73] F. Z. Yousaf, V. Sciancalepore, M. Liebsch, and X. Costa-Perez, “Manoaas: A multi-tenant nfv mano for 5g network slices,” *IEEE Communications Magazine*, vol. 57, no. 5, pp. 103–109, 2019.

[74] S. Kukliński and L. Tomaszewski, “Dasmo: A scalable approach to network slices management and orchestration,” in *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–6, 2018.

[75] A. Galis, K. Makhijani, D. Yu, and B. Liu, “Autonomic Slice Networking,” Internet-Draft draft-galis-anima-autonomic-slice-networking-05, Internet Engineering Task Force, Sept. 2018. Work in Progress.

[76] Q. Wang, J. Alcaraz-Calero, M. B. Weiss, A. Gavras, P. M. Neves, R. Cale, G. Bernini, G. Carrozzo, N. Ciulli, G. Celozzi, A. Ciriaco, A. Levin, D. Lorenz, K. Barabash, N. Nikaein, S. Spadaro, D. Morris, J. Chochliouros, Y. Agapiou, C. Patachia, M. Iordache, E. Oproiu, C. Lomba, A. C. Aleixo, A. Ro-Drigues, G. Hallissey, Z. Bozakov, K. Koutsopoulos, and P. Walsh, "Slicenet: End-to-end cognitive network slicing and slice management framework in virtualised multi-domain, multi-tenant 5g networks," in *2018 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, pp. 1–5, 2018.

[77] "Slicenet project." urlhttps://slicenet.eu/. Accessed 05-18-2021.

[78] S. Spadaro and K. Nagin, "Cognitive network slice management: The slicenet approach," *EURESCOM message*, vol. 1, no. 1, pp. 12–13, 2020.

[79] G. Cuffaro, F. Paganelli, and P. Cappanera, "Tenant-side management of service function chaining: Architecture, implementation and experiment on a future internet testbed," in *2019 IEEE Conference on Network Softwarization (NetSoft)*, pp. 124–132, 2019.

[80] A. Gavras, S. Denazis, C. Tranoris, H. Hrasnica, and M. B. Weiss, "Requirements and design of 5g experimental environments for vertical industry innovations," in *2017 Global Wireless Summit (GWS)*, pp. 165–169, 2017.

[81] "D3.3: 5g norma network architecture - final report," Sept. 2017.

[82] F. Wei, S. Qin, G. Feng, Y. Sun, J. Wang, and Y.-C. Liang, "Hybrid model-data driven network slice reconfiguration by exploiting prediction interval and robust optimization," *IEEE Transactions on Network and Service Management*, 2021.

[83] F. Wei, G. Feng, Y. Sun, Y. Wang, and S. Qin, "Proactive network slice reconfiguration by exploiting prediction interval and robust optimization," in *GLOBECOM 2020-2020 IEEE Global Communications Conference*, pp. 1–6, IEEE, 2020.

[84] W. Guan, H. Zhang, and V. C. Leung, "Slice reconfiguration based on demand prediction with dueling deep reinforcement learning," in *GLOBECOM 2020-2020 IEEE Global Communications Conference*, pp. 1–6, IEEE, 2020.

[85] G. Wang, G. Feng, T. Q. Quek, S. Qin, R. Wen, and W. Tan, "Reconfiguration in network slicing—optimizing the profit and performance," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 591–605, 2019.

[86] X. Lu, X. Wang, L. Pang, J. Liu, Q. Yang, and X. Song, "Deployment and reconfiguration for balanced 5g core network slices," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 104, no. 11, pp. 1629–1643, 2021.

[87] A. Gausseran, F. Giroire, B. Jaumard, and J. Moulierac, "Be scalable and rescue my slices during reconfiguration," *The Computer Journal*, vol. 64, no. 10, pp. 1584–1599, 2021.

[88] E. M. Dow and J. N. Matthews, “Wayfinder: parallel virtual machine reallocation through a* search,” *Memetic Computing*, vol. 8, no. 4, pp. 255–267, 2016.

[89] A. Gausseran, *Algorithmes d’optimisation pour le network slicing pour la 5G*. PhD thesis, Université Côte d’Azur, 2021.

[90] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: theory and practice*. Elsevier, 2004.

[91] S. Ayoubi, N. Limam, M. A. Salahuddin, N. Shahriar, R. Boutaba, F. Estrada-Solano, and O. M. Caicedo, “Machine learning for cognitive network management,” *IEEE Communications Magazine*, vol. 56, pp. 158–165, Jan 2018.

[92] A. Mestres, A. Rodriguez-Natal, J. Carner, P. Barlet-Ros, E. Alarcón, M. Solé, V. Muntés-Mulero, D. Meyer, S. Barkai, M. J. Hibbett, G. Estrada, K. Ma’ruf, F. Coras, V. Ermagan, H. Latapie, C. Cassar, J. Evans, F. Maino, J. Walrand, and A. Cabellos, “Knowledge-defined networking,” *SIGCOMM Comput. Commun. Rev.*, vol. 47, p. 2–10, sep 2017.

[93] M. Behringer, S. Bjarnason, S. Jiang, B. Carpenter, M. Pritikin, L. Ciavaglia, and A. Clemm, “Autonomic networking: Definitions and design goals,” 2015.

[94] D. M. Chess and J. O. Kephart, “The vision of autonomic computing,” *Computer*, vol. 36, pp. 41–50, 01 2003.

[95] A. I. Montoya-Munoz, D. M. Casas-Velasco, F. Estrada-Solano, A. Ordonez, and O. M. C. Rendon, “A yang model for a vertical sdn management plane,” in *2017 IEEE Colombian Conference on Communications and Computing (COLCOM)*, pp. 1–6, Aug 2017.

[96] D. Nau, T. . Au, O. Ilghami, U. Kuter, D. Wu, F. Yaman, H. Munoz-Avila, and J. W. Murdock, “Applications of shop and shop2,” *IEEE Intelligent Systems*, vol. 20, no. 2, pp. 34–41, 2005.

[97] K. Erol, *Hierarchical task network planning: formalization, analysis, and implementation*. PhD thesis, 1996.

[98] P. Gregory, “Pddl templating and custom reporting: Generating problems and processing plans,”

[99] V. Strobel and A. Kirsch, “Mypddl: Tools for efficiently creating pddl domains and problems,” in *Knowledge Engineering Tools and Techniques for AI Planning*, pp. 67–90, Springer, 2020.

[100] “Rasa.” https://www.rasa.com/. Accessed: 2020-05-08.

[101] G. G. Chowdhury, “Natural language processing,” *Annual review of information science and technology*, vol. 37, no. 1, pp. 51–89, 2003.

[102] “Luis nlu.” https://www.luis.ai/. Accessed: 2020-12-17.

[103] "Amazon lex." https://aws.amazon.com/es/lex/. Accessed: 2020-12-17.

[104] D. Braun, A. Hernandez-Mendez, F. Matthes, and M. Langen, "Evaluating natural language understanding services for conversational question answering systems," in *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, (Saarbrücken, Germany), pp. 174–185, Association for Computational Linguistics, aug 2017.

[105] "Rasaactions." https://rasa.com/docs/rasa/core/actions/#custom-actions. Accessed: 2020-05-08.

[106] "Pddltemplates." https://github.com/Pold87/myPDDL/blob/master/templates/problem-template.pddl. Accessed: 2020-05-21.

[107] "Stripsplanner." https://stripsfiddle.herokuapp.com/. Accessed: 2020-05-21.

[108] "How to make automated testing part of your rasa dev workflow." https://blog.rasa.com/rasa-automated-tests/. Accessed: 2020-10-04.

[109] "Rasa evaluating and testing." https://rasa.com/docs/rasa/testing-your-assistant. Accessed: 2020-10-04.

[110] I. Vaishnavi and L. Ciavaglia, "Challenges towards automation of live telco network management: Closed control loops," in *2020 16th International Conference on Network and Service Management (CNSM)*, pp. 1–5, IEEE, 2020.

[111] E. F. Castillo, O. M. C. Rendon, A. Ordonez, and L. Z. Granville, "Ipro: An approach for intelligent sdn monitoring," *Computer Networks*, vol. 170, p. 107108, 2020.

[112] A. G. et al., "Planning.wiki - the ai planning pddl wiki," 2022.

[113] M. Fox and D. Long, "Pddl2. 1: An extension to pddl for expressing temporal planning domains," *Journal of artificial intelligence research*, vol. 20, pp. 61–124, 2003.

[114] V. Strobel and A. Kirsch, "Planning in the wild: modeling tools for pddl," in *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*, pp. 273–284, Springer, 2014.

[115] E. Scala, P. Haslum, S. Thiébaux, and M. Ramirez, "Interval-based relaxation for general numeric planning," in *ECAI 2016*, pp. 655–663, IOS Press, 2016.

[116] E. Scala, P. Haslum, S. Thiébaux, and M. Ramirez, "Subgoaling techniques for satisficing and optimal numeric planning," *Journal of Artificial Intelligence Research*, vol. 68, pp. 691–752, 2020.

[117] A. Kaloxylos, "A survey and an analysis of network slicing in 5g networks," *IEEE Communications Standards Magazine*, vol. 2, no. 1, pp. 60–65, 2018.

[118] D. Li, E. Scala, P. Haslum, and S. Bogomolov, "Effect-abstraction based relaxation for linear numeric planning.," in *IJCAI*, pp. 4787–4793, 2018.

Appendix A

Scientific Production and Awards

The research work presented in this thesis was reported to the scientific community through paper submissions to renowned conferences and journals and it was awarded with two distinctions. The process of doing research, submitting paper, gathering feedback, and improving the work helped to achieve the maturity hereby presented. The list of published papers to date are listed below in chronological order. The published papers and awards certificates are available in the next pages.