UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

FREDERICO SCHARDONG

Taming NFV Orchestration using Decentralised Cognitive Components

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Computer Science

Advisor: Prof. Dr. Ingrid Oliveira de Nunes
Coadvisor: Prof. Dr. Alberto Egon
Schaeffer-Filho

Porto Alegre
November 2018

CIP — CATALOGING-IN-PUBLICATION

Schardong, Frederico

Taming NFV Orchestration using Decentralised Cognitive Components / Frederico Schardong. – Porto Alegre: PPGC da UFRGS, 2018.

81 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2018. Advisor: Ingrid Oliveira de Nunes; Coadvisor: Alberto Egon Schaeffer-Filho.

1. Network Functions Virtualisation. 2. Multi-agent Systems. 3. BDI Architecture. I. Nunes, Ingrid Oliveira de. II. Schaeffer-Filho, Alberto Egon. III. Título.

“There are others which are likely to startle us out of our complacency, and perhaps ultimately to drive us out of all the hypotheses in which we have hitherto found refuge into that state of thoroughly conscious ignorance which is the prelude to every real advance in knowledge.”

— JAMES CLERK MAXWELL

ACKNOWLEDGMENTS

I would like to especially thank Professors Ingrid Oliveira de Nunes and Alberto Egon Schaeffer-Filho for their guidance and patience. Through their encouragement and professionalism, I have learned numerous lessons about academia and science.

I would like to thanks the members of the examining committee, Marcelo Caggiani Luizelli, Felipe Meneguzzi and Weverton Cordeiro for their time reading my dissertation and giving valuable feedback. Also, I thank my colleagues and friends João, Vanius, Jhonny and Fernando of the Prosoft research group from Universidade Federal do Rio Grande do Sul (UFRGS) for the immeasurably valuable conversations, discussions and guidance. I thank my fried Matheus F. Fröhlich for sharing ideas, time and the path of pursuing a master degree. Thanks to all the staff from UFRGS, especially from Instituto de Informática (INF). Also, thanks to CNPq and CAPES for financially supporting this research.

Finally, I want to thank my parents Valnir and Vera, brothers Gustavo and Marcelo, and girlfriend Geovana, whose unconditional support, understanding and encouragement were of paramount importance for the conclusion of this dissertation.

ABSTRACT

Network Functions Virtualisation (NFV) decouples network functions from physical devices, simplifying the deployment of new services. Typical network functions, like firewalls, traffic accelerators, intrusion detection systems and intrusion prevention systems, are traditionally performed by proprietary physical appliances, which must be manually installed by network operators. Their deployment is challenging because they have specific chaining requirements. As opposed to traditional physical appliances, virtual network functions (VNFs) can be dynamically deployed and reconfigured on demand, posing strict management challenges to networked systems. The selection of the most appropriate VNFs to achieve a particular objective, the decision on where to deploy these VNFs and through which paths they will communicate are the responsibilities of an NFV orchestrator. In this dissertation, we propose to orchestrate VNFs using interacting cognitive components structured with the belief-desire-intention (BDI) architecture, leading to emergent solutions to address network challenges. The BDI architecture includes a reasoning cycle, which provides agents with rational behaviour, allowing agents to deal with different scenarios in which flexible and intelligent behaviour is needed. We extend the NFV architecture, replacing its centralised orchestrator with BDI agents. Our proposal includes a reverse auction protocol and a novel bidding heuristic that allow agents to make decisions regarding the orchestration tasks. Finally, we provide a testbed that integrates a platform for developing BDI agents with a network emulator, allowing agents to control VNFs and perceive the network. This testbed is used to implement VNFs and empirically evaluate our theoretical model in a distributed denial-of-service (DDoS) attack. The evaluation results show that a solution to the DDoS attack emerges through the negotiation of agents, successfully mitigating the attack.

Keywords: Network Functions Virtualisation. Multi-agent Systems. BDI Architecture.

Orquestrador NFV descentralizado baseado em raciocínio BDI

RESUMO

Network Functions Virtualisation (NFV) separa as funções de rede dos dispositivos físicos, simplificando a implantação de novos serviços. As típicas funções de rede, como firewalls, aceleradores de tráfego, sistemas de detecção de intrusão e sistemas de prevenção de intrusões, são tradicionalmente realizadas por equipamentos físicos proprietários, que devem ser instalados manualmente pelos operadores de rede. A implantação de equipamentos físicos é desafiadora porque eles têm requisitos específicos de encadeamento e ordenação. Ao contrário dos equipamentos físicos tradicionais, as funções de rede virtuais (VNFs) podem ser dinamicamente implementadas e reconfiguradas sob demanda, colocando desafios de gerenciamento rigorosos aos sistemas em rede. A seleção das VNFs mais apropriadas para atingir um objetivo específico e a decisão sobre onde implantar essas VNFs e por quais caminhos elas se comunicarão são responsabilidades de um orquestrador de NFV. Nesta dissertação, propomos orquestrar VNFs usando componentes cognitivos interativos estruturados com a arquitetura belief-desire-intention (BDI), levando a soluções emergentes para enfrentar os desafios da rede. A arquitetura BDI inclui um ciclo de raciocínio que fornece aos agentes um comportamento racional, permitindo que lidem com diferentes cenários nos quais o comportamento flexível e inteligente é necessário. Estendemos a arquitetura NFV substituindo seu orquestrador centralizado por agentes BDI. Nossa proposta inclui um protocolo de leilão reverso e uma nova heurística de licitação que permite que os agentes tomem decisões sobre as tarefas de orquestração. Por fim, nós fornecemos uma plataforma de testes que integra uma plataforma para o desenvolvimento de agentes BDI com um emulador de rede, permitindo que os agentes controlem as VNFs e percebam a rede. Essa plataforma de testes é usada para implementar VNFs e avaliar empiricamente nosso modelo teórico em um ataque de negação de serviço distribuído. Os resultados da avaliação mostram que uma solução para o ataque DDoS surge através da negociação de agentes, mitigando com sucesso o ataque.

Palavras-chave: Network Functions Virtualisation, Sistemas Multi-Agente, Arquitetura BDI.

LIST OF ABBREVIATIONS AND ACRONYMS

IaaS	Infrastructure as a Service
NaaS	Network as a Service
CAPEX	CAPital EXpenditure
OPEX	OPerating EXpenditure
NFV	Network Functions Virtualisation
NS	Network Services
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
NAT	Network Address Translation
VNF	Virtual Network Function
SLA	Service-Level Agreement
BDI	Belief-Desire-Intention
ETSI	European Telecommunications Standards Institute
DDoS	Distributed Denial-of-Service
NFVI-PoP	Network Functions Virtualisation Infrastructure Point of Presence
FIPA	Foundation of Intelligent Physical Agents
ACL	Agent Communication Language
NOS	Network Operating System
REST	REpresentational State Transfer
API	Application Programming Interface
IP	Internet Protocol
TSP	Telecommunications Service Provider
ETSI ISG NFV	ETSI Industry Specification Group for NFV
MANO	MANagement and Orchestration

NFVI	Network Functions Virtualisation Infrastructure
NF	Network Function
DHCP	Dynamic Host Configuration Protocol
PNF	Physical Network Function
OSS	Operations Support System
BSS	Business Support Systems
FCAPS	Fault, Configuration, Accounting, Performance, Security
VNFM	VNF Manager
NFVO	NFV Orchestrator
VIM	Virtualised Infrastructure Manager
VNFD	Virtualised Network Function Descriptor
VNF-FG	VNF Forwarding Graph
SDN	Software-Defined Networking
SFC	Service Function Chain
CDN	Content Delivery Network
QoS	Quality of Service
KQML	Knowledge Query and Manipulation Language
CDPS	Cooperative Distributed Problem Solving
CNET	Contract Net
DPI	Deep Packet Inspector
CFP	Call For Proposal

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

Over the years, the complexity and size of networks have drastically increased and so did the requirement for more flexible management. Physical appliances (also known as middleboxes) are proprietary highly specialised products that require specific chaining, physical installation, and their functionality cannot be easily changed. Moreover, the increasing demand for more diverse and short-lived networks to handle high data rates, such as in infrastructure as a service (IaaS) and network as a service (NaaS) (MIJUMBI et al., 2016), requires network operators to deploy rapidly and operate complex network equipments, leading to high capital expenditure (CAPEX) and operating expenditure (OPEX) (VERBRUGGE et al., 2006).

To address these issues, network functions virtualisation (NFV) has been proposed as a way to decouple network services (NS) from physical devices through virtualisation (ETSI ISG NFV, 2013; HERRERA; BOTERO, 2016). There is a broad set of services that has been traditionally performed by middleboxes—*e.g.*, firewalls, intrusion detection systems (IDS), intrusion prevention systems (IPS), traffic shaping, network address translation (NAT), traffic accelerators, caches and proxies—that can be virtualised into cheap and easily deployable virtual network functions (VNFs) (MARTINS et al., 2014). Differently from middleboxes, multiple virtualised network functions can share a single physical machine, enabling the use of computational power that would be otherwise lost in proprietary middleboxes (LUIZELLI et al., 2017). The evolution of networks into software-based functions and services are concrete steps towards future networks.

However, VNFs need to be managed and composed in meaningful ways such that the desired functionalities are achieved. This process is called NFV orchestration. NFV orchestration can be decomposed into three core problems: (i) automatic *selection* of VNFs; (ii) VNF *placement* in the virtualised network; and (iii) *chaining* of VNFs (MIJUMBI et al., 2016). Typically, humans design a network forwarding graph (ETSI ISG NFV, 2014b) of VNFs, that is, decide how VNFs are chained. However, as network complexity increases and service-level agreement (SLA) requirements over on-demand networks become more strict, guaranteeing the orchestration of virtual nodes in real-time becomes vital for carriers and service providers. Nonetheless, NFV orchestration schemes proposed so far do not explore the benefits of autonomous components, therefore relying on humans to enforce the SLAs, which is often impractical (HUIN; JAUMARD; GIROIRE, 2017; YASREBI et al., 2015; CLAYMAN et al., 2014).

This dissertation explores the orchestration problem from the perspective of autonomous components. Our proposal includes extensions to the NFV architecture (ETSI ISG NFV, 2014b) to use the *belief-desire-intention* (BDI) architecture (RAO; GEORGEFF et al., 1995) to perform NFV orchestration. The BDI architecture provides intelligent agents with a robust and flexible behaviour while simplifies their development by separating agents' actions in the environment from their internal reasoning. A decentralised orchestrator composed of autonomous BDI agents can arguably improve the orchestration of virtualised resources as their emergent behaviour allow them to adapt to unforeseen scenarios at run time. Furthermore, we introduce an auction heuristic used by these autonomous agents to collectively attack the selection, placement and chaining problems.

The remainder of this chapter is organised as follows. Section 1.1 presents the research question addressed in this work and limitations of existing approaches and Section 1.2 gives an overview of our proposed solution and a list of the contributions of this dissertation.

1.1 Problem Statement and Limitations of Related Work

Centralised solutions to the selection, placement and chaining problems have been proposed. Nonetheless, they have natural drawbacks, for instance, inherited fault intolerance as a single component controls the allocation of resources in the network and might require to halt completely due to hardware or configuration changes. In a decentralised solution, if a component of the system fails, then others perceive and assume its functions, thus maintaining overall availability. One of the consequences of this characteristic is that the different parts that compose a decentralised system can be updated/changed without affecting the system. In addition, decentralised systems can easily scale to accommodate changes in the size of what they are trying to solve. Based on these issues, we state our research question below.

Research Question: How to address the selection, placement, and chaining problems using a decentralised approach solution?

Different solutions to the orchestration core problems have been proposed, ranging from the use of policies to machine learning techniques. We list the limitations of existing work next.

Existing approaches are limited to centralised orchestrators. By centralising the orchestrator, one has to guarantee it has global knowledge and capacity to solve the orchestration problems altogether. Moreover, a system made of a single component is not resilient. If it fails, there will be no other component to assume its functions. A decentralised solution, in contrast, address such limitations and can flexibly reorganise itself to handle failure.

Most of existing the approaches do not consider all the problems involved in orchestrating VNFs. Most of the existing approaches do not consider the selection, placement and chaining problems jointly. Some approaches focus on the placement of VNFs, while others propose methods to automate VNF chaining. More complete solutions introduce techniques to solve both the placement and selection problems. These research efforts have the selection of VNFs already made as part of the input, which prevents the orchestration of VNFs in real-time to handle new scenarios. Furthermore, lifecycle management of VNFs is poorly explored. As such, most efforts to automate the orchestration of VNFs consider only the initial deployment of VNFs. The automation of placement and chaining is a major step towards the creation of robust orchestrators. Nonetheless, it is crucial to have orchestrators that are capable of selecting VNFs to be deployed, dealing with dynamic changes in the network, and with the deployment, configuration, monitoring and displacement of VNFs. Combining the solution of all these problems into a centralised application might be unfeasible, one would have to ensure the solutions of these problems do not conflict and coordinate them at design time. Embedding the solution of those problems into autonomous software components that communicate and share information would naturally inherit the benefits of a decentralised architecture.

There is a lack of solutions that address the orchestration problems from a multi-agent perspective. Agent technology is a promising approach to provide a decentralised solution to the three core problems, which has not been explored yet in this context. The dynamicity and adaptiveness provided by decentralised autonomous agents can significantly benefit the management and orchestration of VNFs. The emergent behaviour of agents enables the system to solve issues not predicted at design time.

1.2 Proposed Solution and Overview of Contributions

As discussed in the previous section, there are many issues that must be addressed with the adoption of a centralised solution to solve NFV core problems, such as the computation of all the required information from VNFs in order to solve such orchestration problems. Thus, considering the research question presented in the previous section and the potential of decentralised NFV orchestrators, this dissertation proposes a decentralised approach to NFV orchestration using belief-desire-intention (BDI) reasoning, addressing the selection, placement and chaining problems through the interaction among autonomous software agents. In our model, each Network Functions Virtualisation Infrastructure Point of Presence (NFVI-PoP), *i.e.*, each location where a VNF can be deployed, is represented by a BDI agent, which posses human-like rationale provided by a reasoning cycle, is aware of its needs and can perceive its environment. Agents communicate and through an auction protocol attack the selection, placement and chaining problems collectively, enabling VNFs to deal with unforeseen situations, providing resilience and robustness to the orchestrated resources.

Finally, we evaluate our proposed solution in a DDoS attack scenario. Considering the adaptive nature of BDI agents, they shall be able to address unforeseen issues. Moreover, we propose the development of a decentralised NFV orchestrator composed of BDI agents, whose interactions allow them to attack the selection, placement and chaining problems. In summary, the main contributions of this dissertation are:

- **A decentralised NFV orchestrator**, which aggregates multiple autonomous agents in a multi-agent system where they interact, negotiate and make decisions, emerging into a decentralised NFV orchestrator.
- **A formal model** that describes both the physical and virtual networks as well as the BDI agents.
- **A bidding heuristic** that allow agents to attack the selection, placement and chaining problems.
- **An integrated testbed**, allowing agents in the BDI4JADE framework to perceive and control VNFs in Mininet.
- **An implementation of the proposed NFV orchestrator and reverse-auction heuristic** to empirically test and validate the proposed techniques.

The remaining of this dissertation is organised as follows. In Chapter 2, we present

the background and review related work. We introduce basic concepts related to NFV, autonomous agents and multi-agent systems as well as the BDI architecture and negotiation in multi-agent systems. In Chapter 3, we formalise the agent-related concepts and present our extensions to the NFV architecture, which has its centralised orchestrator replaced by autonomous agents. Chapter 4 details the auction heuristic used by the autonomous agents to negotiate and attack the selection, placement and chaining problems. From these definitions, Chapter 5 presents the implementation of a testbed where the theoretical concepts are evaluated in a DDoS scenario. VNFs are implemented to perform specific network functions that interact to mitigate the attack. Additionally, we review some important related work where automation to the orchestration of VNFs have been proposed, discussing their limitations in Chapter 6. Finally, conclusion and future work directions are presented in Chapter 7.

2 BACKGROUND

In this chapter, we present the fundamental concepts for understanding this research. It is divided into two topics: network infrastructure and autonomous agents. Firstly, we introduce the network infrastructure concepts of NFV. Secondly, we detail autonomous agents and multi-agent systems concepts as well as the BDI architecture and multi-agent negotiations.

2.1 Network Functions Virtualisation

In this section, we detail the NFV architecture, describing its components and providing an overview of their relationship. We begin by introducing the motivation behind NFV and its benefits. Then we detail the architectural components and describe the orchestration of physical and virtual infrastructure as well as the orchestration of network services (NS).

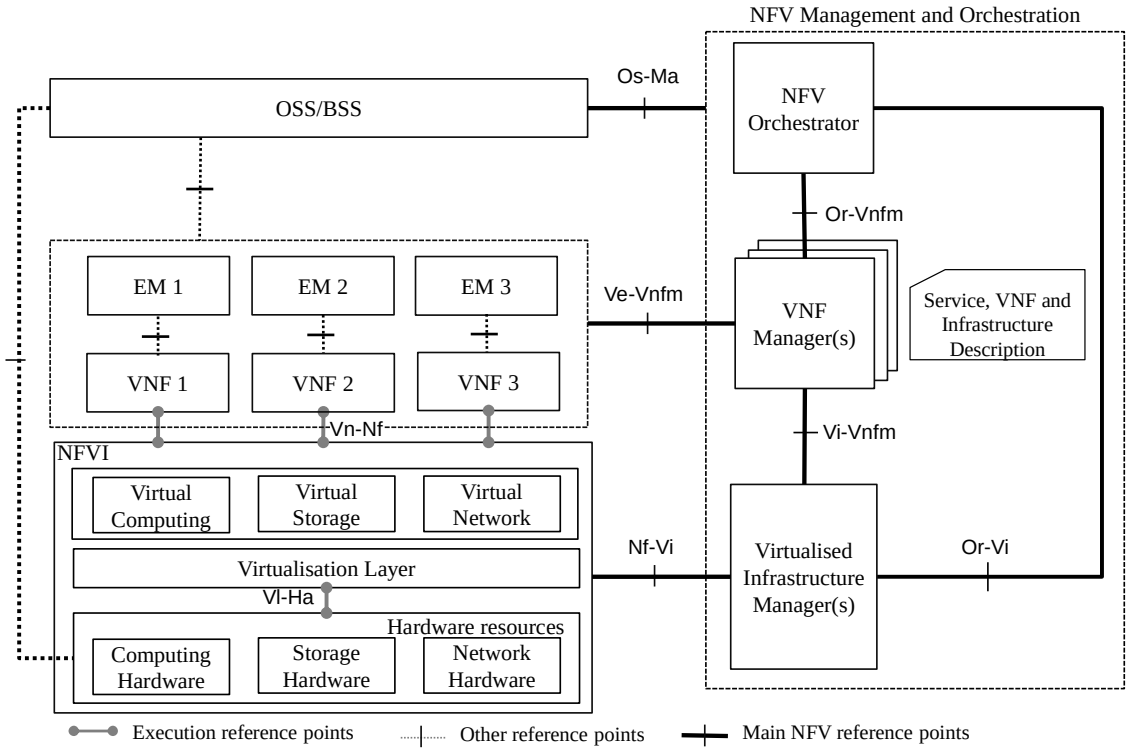
2.1.1 Benefits of the NFV architecture

Network Functions Virtualisation (NFV) has been proposed as an alternative to traditional service provisioning in the telecommunications industry (GUERZONI et al., 2012). Traditional service provisioning has been based on the deployment of physical proprietary devices, which have specific chaining requirements that must be reflected in the network topology (MIJUMBI et al., 2016). In addition, the lack of skilled professionals and the cost of offering the required space and energy have led to low service agility and high dependency on specialised hardware (HAN et al., 2015).

NFV decouples software from hardware, enabling software elements to advance independently from hardware. This brings flexibility to the network as it allows the network function software to be virtually deployed anywhere in the network topology on-demand. Therefore, NFV gives network operators finer granularity so they can provide resources according to the actual traffic (ETSI ISG NFV, 2013).

The NFV concept was coined in October 2012 when leading telecommunications service providers (TSPs) authored a white paper asking for industrial and academic attention towards NFV (GUERZONI et al., 2012). In November 2012, the European Telecom-

Figure 2.1: NFV architectural framework (ETSI ISG NFV, 2014a)



munications Standards Institute (ETSI) was chosen as the entity to host the Industry Specification Group for NFV (ETSI ISG NFV), which is responsible for developing NFV standards. ETSI has released multiple documents[1] specifying different aspects of the NFV architecture, which include an infrastructure overview, architectural framework, management and orchestration (MANO), security and trust, resilience and documents regarding the hypervisor and network domains of the infrastructure (MIJUMBI et al., 2016).

2.1.2 Architectural Framework

ETSI defines three main components to the NFV architecture (ETSI ISG NFV, 2013; ETSI ISG NFV, 2014a): (i) Network Functions Virtualisation Infrastructure (NFVI); (ii) Virtualised Network Functions (VNFs); and (iii) Management and Orchestration (MANO). These are further broken down into functional elements and connection points (also called reference points), comprising the NFV architectural framework, as shown in Figure 2.1. Implementation references of these blocks and their connections are publicly available.[1]

[1] http://www.etsi.org/standards-search#search=nfv

- The *Network Functions Virtualisation Infrastructure (NFVI)* represents all the hardware and software, which comprise the infrastructure where VNFs are deployed. The NFVI can span over different locations (*e.g.*, different servers or data centres), each represented as a different NFVI-PoP where a VNF can be deployed. The physical resources include computing power, storage and network, provided by commercial-off-the-shelf (COTS) equipment. The virtual computing, storage and network elements are abstractions allowed by the virtualisation layer (hypervisor). The virtualisation layer decouples the virtual elements from the underlying physical resources (ETSI ISG NFV, 2014a).
- A *Virtualised Network Function (VNF)* is the virtualisation of a network function (NF), which is a functional block of a network infrastructure that has well-defined behaviour and external interfaces. Examples of NFs are elements in a home network, such as dynamic host configuration protocol (DHCP) servers, firewalls, etc. The functional behaviour of an NF is usually independent of whether it is virtualised or not. Hence, the functional behaviour and external interfaces of a physical network function (PNF) and of a VNF are expected to be the same (ETSI ISG NFV, 2014a). It is fundamental that VNFs present near identical performance when compared to PNFs such that the migration from traditional network architectures to NFV does not impact on the overall service performance. However, current solutions have not reached this performance milestone yet (HAN et al., 2015).
- The *Management and Orchestration (MANO)* entity is responsible for the orchestration and lifecycle management of physical and virtualised resources that support the infrastructure virtualisation, and for the lifecycle management of VNFs. The NFV-MANO specification defines interfaces that allow the communication with the components that make up MANO as well as with other entities of the NFV architecture, *i.e.*, traditional network management systems such as operations support system (OSS) and business support systems (BSS). Moreover, it includes databases that describe how VNFs work, their resources, lifecycle properties and information regarding the required instantiation and configuration to allow groups of VNFs to work together (ETSI ISG NFV, 2014a; ETSI ISG NFV, 2014b).

The main NFV reference points (bold lines in Figure 2.1) are highly important to the NFV architecture, each having a model specification document providing detailed information on operations that must be supported, their inputs and outputs as well as their parameters.

One of the entities that is part of the NFV architectural framework is the *Element Management (EM)*. The EM is responsible for fault, configuration, accounting, performance and security (FCAPS) management of a VNF. EM configures, monitors and collects measurements from the VNF and exchanges information with the VNF Manager (VNFM), enabling MANO to make decisions and take actions to guarantee security, resilience and SLA requirements (ETSI ISG NFV, 2014b).

The *OSS/BSS* block represents traditional network management functions and business support functions that are not explicitly captured in the NFV architectural framework but are expected to have information exchanges with other components of this architecture. The interfaces between OSS/BSS and the other elements is yet to be fully defined.

Each VNF instance is assumed to have an associated *VNF Manager (VNFM)*. A VNFM can manage one or more VNFs. As pointed out by ETSI, most VNFM functions are assumed to be generic to any VNF. This entity is directly responsible for the following VNF operations: instantiation (including applying any configuration required for the instantiation); software upgrade; instance modification; instance scaling up/down; instance termination; management of the integrity of the VNF instance through its lifecycle and more. All the information regarding these operations is captured in a template called virtualised network function descriptor (VNFD) (ETSI ISG NFV, 2016b), which supports the provision of virtual resources for the instantiation of VNFs, which is done by the VIM.

The *Virtualised Infrastructure Manager* is responsible for managing the interaction of a VNF with computing, storage and network resources under its authority. The NFV orchestrator might use multiple VIMs as a VIM might be specialised in a certain type of NFVI resource, or if resources in an NFVI-PoP are not handled by a single VIM. This entity is responsible for orchestrating the allocation/upgrade/release of NFVI resources and managing the association of the virtualised resources to the physical resources.

The *Service, VNF and Infrastructure Description* is an aggregation of catalogues and repositories that provide key information to the NFV-MANO components (NFV Orchestrator, VNFM and VIM). We detail bellow the repositories and catalogues represented by this component.

- *VNF Descriptor*: guarantees the flexibility and portability of VNF instances on multi-vendor NFVI environments, *e.g.*, compute resources from different vendors, diverse virtual network technologies, etc. Hardware resources have to be properly abstracted, and the VNF requirements have to be specified using such abstractions. The VNFD repository might contain specifications of different versions of an NF.

An NF might be described by various VNFD, each containing details regarding different execution environments (*e.g.*, different hypervisors).

- *VNF Catalogue*: repository of all the available VNFs. This catalogue is composed of VNFD, software images, manifest files, etc. Both NFVO and VNFM can query the VNF Catalogue to retrieve a VNFD.
- *NFVI Resource*: repository that tracks all the available and reserved/allocated resources of the virtualised infrastructure. It plays an important role in supporting the NFVO operations.
- *NFV Instances*: repository that holds information of all VNF instances. Records in this repository are updated during the lifecycle of the respective VNFs, reflecting lifecycle management operations.

The *NFV Orchestrator* has two responsibilities: (i) the orchestration of NFVI resources through multiple VIMs; and (ii) the orchestration and lifecycle management of network services, which are detailed next.

2.1.3 NFV Orchestration

There are three main problems associated with NFV orchestration: (i) *selection*; (ii) *placement*; and (iii) *chaining*. The *selection* problem consists of logically selecting and connecting virtual network functions (VNFs) in an ordered manner to provide some service (MEDHAT et al., 2015). The solution to the selection problem is a virtualised network function forwarding graph (VNF-FG), which is an ordered set of VNFs and their interconnections represented as a graph, namely a *VNF forwarding graph* (VNF-FG) (ETSI ISG NFV, 2013). Most NFV orchestration approaches assume that the VNF-FG is provided as an input to their solutions (LUIZELLI et al., 2015; SUN et al., 2016)

The orchestrator component centralises all the VNF-related management operations, from the instantiation of the VNFM to the allocation of virtual resources for the instantiation of VNFs (in cooperation with the VNF's respective VNFM). The allocation of the VNFs into the physical network is known as the *placement* problem. It has drawn significant attention from researchers, as correctly provisioning resources and deciding where a VNF should be located in the infrastructure is critical to the realisation of NFV (XIA et al., 2015; BOUET et al., 2015; MOENS; TURCK, 2014).

According to ETSI, "a network service (NS) is a forwarding graph of network

functions (NFs) interconnected supported by the network infrastructure" (ETSI ISG NFV, 2013). NSs can be viewed as high-level NFs that represent end-to-end services.[2] Hence, NSs can be composed of not only NFs but also of other NF forwarding graphs. Nodes (NFs and end-points) in an NS are connected by virtual links, which can be unidirectional, bidirectional, multicast or broadcast. A simple example of an NS is a chain of network functions such as firewall, load balancer and a set of content delivery network (CDN) servers. Guaranteeing the correct interconnection of NFs is known as the *chaining* problem, which is often tackled together with the placement problem as placing VNFs many hops away could potentially compromise the performance and throughput of the NS. The correct configuration, deployment and management of NSs are the culminating point towards an effective orchestrator. The chaining problem has attracted attention from researchers, being tackled either alone (JIAO et al., 2017; THAI; LIN; LAI, 2016; ALAMEDDINE; QU; ASSI, 2017) or along with the placement problem (CHI; HUANG; LEI, 2015; ZHANG et al., 2016; KONG et al., 2017).

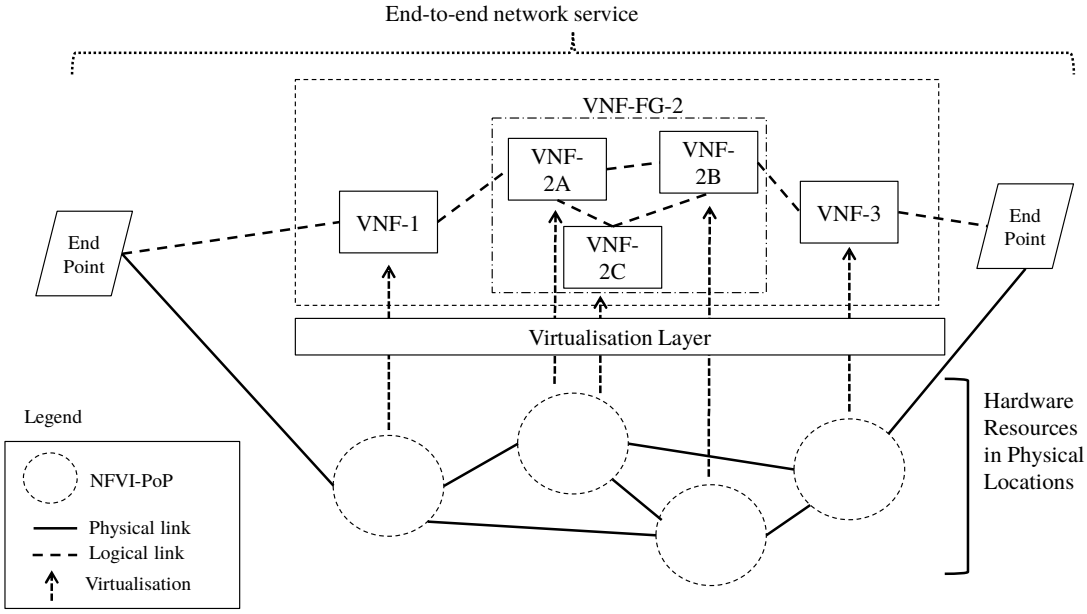
ETSI published a document (ETSI ISG NFV, 2016a) specifying the description of NSs in the form of metadata, allowing the specification of NFs that compose an NS and their connections. Moreover, it enables the description of more specific information such as minimal quality of service (QoS) requirement of links that connect NFs. Figure 2.2 shows an end-to-end NS and the different layers involved. It includes a nested VNF forwarding graph (VNF-FG). A VNF-FG is a forwarding graph composed entirely of virtualised NFs. The nested VNF-FG (VNF-FG-2) could be composed of virtualised resources managed by a different VIM or entirely managed by an orchestrator from another vendor. Nevertheless, due to the mutually accepted standardised interface they work together seamlessly (ETSI ISG NFV, 2013).

2.2 Autonomous Agents and Multi-agent Systems

In this section, we present the theoretical background of autonomous agents and multi-agent systems required to understand our proposed solution, which is based on the agent technology. In Section 2.2.1, we present key concepts and definitions. In Section 2.2.2, we detail the components of the BDI architecture, which is used to structure agents in our work. Finally, Section 2.2.3 describes negotiation in multi-agent systems,

[2] In the context of software-defined networking (SDN), an ordered set of service functions that compose an end-to-end service is called a service function chain (SFC) (HALPERN; PIGNATARO, 2015).

Figure 2.2: Forwarding graph and its supporting infrastructure (ETSI ISG NFV, 2013)



given that our solution involves negotiation among agents in order to allocate tasks to be accomplished.

2.2.1 Key Concepts and Definitions

An *agent* is an autonomous computer system, *i.e.*, it is capable of taking independent actions in order to satisfy its design objectives, rather than being told explicitly what to do at any given time. A collection of agents that interact with each other comprise a *multi-agent system*. Interactions usually take place through the exchange of messages over some media, *i.e.*, a computer network. Similarly to people, agents must cooperate, coordinate and negotiate in order to successfully interact. An agent is defined as follows.

Definition 1 (Agent) *An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its delegated objectives (WOOLDRIDGE, 2009).*

Although this definition is broad, according to Wooldridge (2009) there is much debate and controversy on the definition of the term *agent*. An agent may be provided with some form of intelligence, being in this case referred to as *intelligent agent*, which is a computational entity with *autonomous*, *reactive*, *proactive* and *social* characteristics. An autonomous agent functions without external intervention. A reactive agent can perceive

environmental changes and interact with its surroundings, through sensors and actuators, respectively, while a proactive agent is able to take the initiative, *i.e.*, take actions, towards satisfying its design objectives. Furthermore, since a single agent might not be enough to attack complex problems, an intelligent agent must have social ability to interact with other agents to achieve its goals.

An agent interacts with its environment through actuators. In most domains, an agent will not have complete control over its environment, at best it will have partial control. Sensors are responsible for perceiving the environment. They allow an agent to have a perspective of the environment it is embedded. In most cases, agents do not know their surroundings entirely as sensors provide only a partial view. The knowledge agents have about the environment are called *beliefs*. These do not represent the environment as is, but rather how the agent believes it is. This is because sensors might not precisely measure the environment or they can simply malfunction.

2.2.2 BDI Architecture

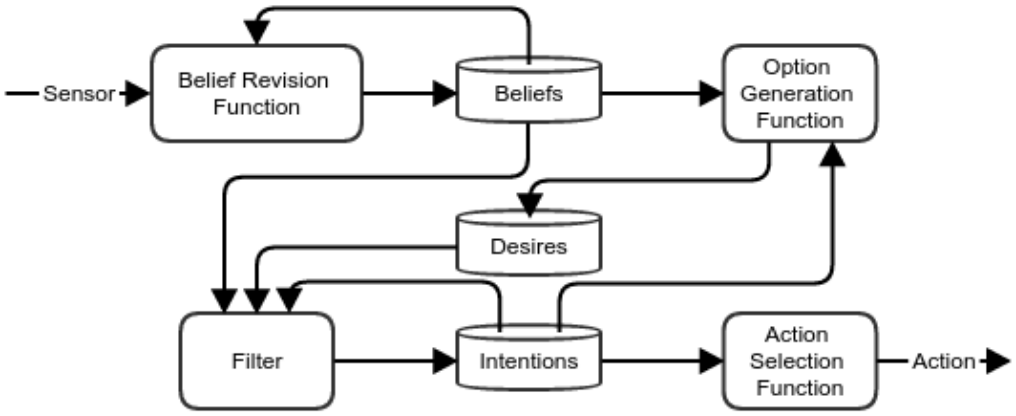
Belief-desire-intention (BDI) (RAO; GEORGEFF et al., 1995) is an architecture that includes a reasoning cycle, which provides agents with rational behaviour. Implementing Bratman (1987)'s model of practical reasoning, this architecture allows agents to deal with different scenarios in which flexible and intelligent behaviour is needed. In this section, we introduce the main components that comprise this architecture and briefly explain its limitations.

The BDI architecture has three main components. The first is *beliefs*, which represent the information that an agent has about itself and its environment. A set of beliefs, or belief base, can be implemented as a database, a set of logical expressions or other data structure. Beliefs are not static, they can be added, removed or changed over time.

The second component corresponds to objectives to be achieved by an agent, named *desires* or *goals*, representing the motivational state of an agent. They can be explicitly added to an agent by the designer or generated by the agent through interactions with its surroundings or as result of agent actions.

The deliberative state of an agent is called *intentions*, the third main BDI component. An intention is a goal that an agent is committed to achieve by the execution of a sequence of actions (or *plan*), in a structured manner intending to achieve one or more goals. That is, an agent is committed to achieve only what it believes that is feasible.

Figure 2.3: Overview of the BDI architecture (WOOLDRIDGE; JENNINGS, 1995).



For instance, an agent might have the belief *ipAddressMalicious(ip)*, which identifies an IP address that it considers ill-intentioned. This belief could result in the generation of *goal(ipBlock(ip))* to block the traffic that originates from this address. This desire can be fulfilled by configuring a firewall application to drop any packet from the IP address believed to be malicious.

These BDI components are connected through a reasoning cycle that is illustrated in Figure 2.3 (WOOLDRIDGE; JENNINGS, 1995). A BDI agent, through sensors, perceives events from its environment and updates its belief base using a *belief revision function*. Such beliefs, together with agent intentions, are used to update agent goals. This is performed by the *option generation function*. The selection of goals to be achieved, *i.e.*, those that will become intentions, is made by a *filtering function*. Finally, an *action selection function* chooses an appropriate sequence of actions, or plan, to achieve intentions.

The BDI architecture is a theoretical model associated with issues to be addressed when it is instantiated. Besides implementation issues, the BDI reasoning cycle includes many gaps to be fulfilled during the development of concrete BDI agents. An example of such a gap is the *action selection function* (often referred to as plan selection), which is responsible for selecting a suitable plan to achieve a goal. Often agents have multiple plans that are appropriate for achieving a goal; however, selecting one of them might be a challenging task due to unknown plan outcomes. Different approaches have tackled this problem from various perspectives. For example, Singh *et al.* (SINGH et al., 2010) focused on learning when a plan might fail, Baitiche *et al.* (BAITICHE; BOUZENADA; SAÏDOUNI, 2017) focused on calculating the probability of plans to succeed, and Faccin and Nunes (FACCIN; NUNES, 2015a) select the best plan in a given context considering agent preferences over secondary goals.

Another limitation of the original BDI architecture is the absence of abstractions

to promote modular design and software reuse, which is important when considering the development of real world applications. The *capability* concept was introduced by Busetta et al. (1999) as a way to build modular structures, promoting reuse of agent components. Capabilities include a subset of beliefs, goals and plans, which combined form an agent building block. Capabilities and other extensions of the BDI architecture make it suitable for modelling complex agents that together can address complex problems.

2.2.3 Negotiation in Multi-agent Systems

One of the key characteristics of an agent is its ability to communicate and become social. Genesereth and Ketchpel (1994) suggest that an entity is a software agent if and only if it communicates correctly in an agent communication language (FININ et al., 1994). Similarly to communication in other areas of computer science, the communication among agents has received significant attention (WOOLDRIDGE, 2009). In this section, we detail the interaction among agents and how mechanism designs, such as auction, are used to coordinate multi-agent systems.

Agent-to-Agent Interaction. Agent communication differs from traditional system-to-system communication. In multi-agent systems, one cannot guarantee that some request from an agent will result in some response or action by the agent receiving such request. Agents are autonomous software components that have their own will. As such, one cannot assume that agents can send orders or somehow manipulate the internal affairs of other agents (WOOLDRIDGE, 2009).

Different agent communication languages (ACL) have been proposed to support and standardise the communication among agents. ACLs are languages designed to support collaboration, negotiation and information exchange. They run in a logical layer on top of transport protocols such as TCP/IP and XML-RPC. These deal with communication issues on the transport level, while ACLs address communication issues on the intentional and social level (CHAIB-DRAA; DIGNUM, 2002).

Much research has been done to support agent communication (FININ et al., 1994; LABROU; FININ, 1997). One of the well-known message-based protocols for agent communication is the knowledge query and manipulation language (KQML) (FININ et al., 1994). This ACL defines a broad range of performatives, *i.e.*, communication primitives. These allow agents to, for example, send unicast and broadcast messages asking other agents to perform a certain task or recommend agents who can. This language

does not specify the message content format, it creates a container in which messages are embedded in.

Similarly to KQML, the FIPA Agent Communication Language (ACL)[3] specifies standards associated with message exchange. This specification is implemented in many of the existing agent and BDI platforms (WOOLDRIDGE, 2009). The main difference between this language and KQML resides in its comprehensive formal semantics based on Cohen and Levesque's theory of speech acts as rational action (COHEN; LEVESQUE, 1988). This semantics was named SL and provides BDI-style primitives, allowing the representation of beliefs, uncertain beliefs, desires and actions (WOOLDRIDGE, 2009; CHAIB-DRAA; DIGNUM, 2002).

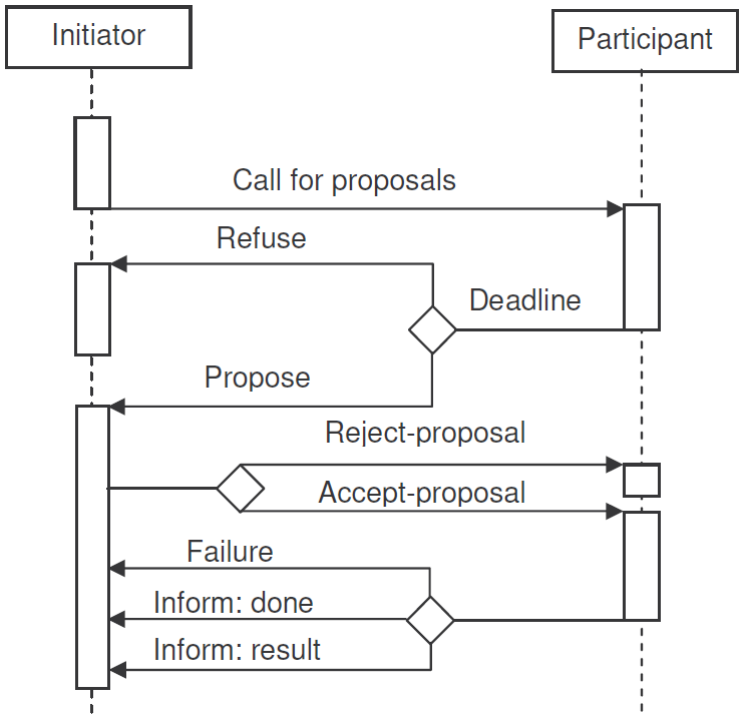
Effective communication allows agents to work together. The *cooperation* and *coordination* of intelligent software agents are made possible by ACLs. Furthermore, intelligent software agents act autonomously, that is, they decide about what to do at run-time rather than having their courses of actions defined at design time. Differently from traditional distributed and concurrent systems, in which all decisions are made at design time, agents have to *cooperate* with one another and *coordinate* their activities at run-time in order to work together.

Cooperative distributed problem solving (CDPS) studies how individual problem solvers such as agents, with little or no knowledge about others can work together to solve problems beyond their capacities. They have to cooperate as no single agent has sufficient resources or information to solve a problem, but others might be capable of solving different parts of the problem (DURFEE; LESSER; CORKILL, 1989). CDPS addresses the division of a problem into smaller tasks, the process of synthesising the solution of subproblems into a problem solution and the coordination of agents to maximise the effectiveness of the solution, *e.g.*, reducing unnecessary message exchange and avoiding destructive interactions (DAVIS; SMITH, 1983).

The Contract Net (CNET) (SMITH, 1980; SMITH; DAVIS, 1981) is a protocol for achieving cooperation through task sharing. An agent makes a *call for proposals* and then manages the task through its lifecycle. It can (i) send a message to all agents if it does not know which agents can perform such task; or (ii) send a message only to the agents it knows that are capable of achieving what it needs; or (iii) send announcements to those agents it knows that are appropriate for such task. Agents who receive an announcement assess their abilities in order to find out if they can achieve what is being asked. If they

[3] http://www.fipa.org/repository/aclspecs.html

Figure 2.4: The FIPA Contract Net (LANGERMAN, 2005)



can, then a *bid* is proposed informing that it wants to perform the task, otherwise a *refuse* message is sent. The manager sends an *accept-proposal* message to each agent that wins the bidding and a *reject-proposal* to those that lose. The winning agents become *contractors* of the auctioned task. Finally, contractors send a *report* message to the manager of the task once it has been completed. This process is depicted in Figure 2.4.

Auctions and Mechanism Design. Reaching agreements in a society of self-interested agents is a problem related to the issue of cooperation introduced previously. The ability to reach agreements upon unforeseen scenarios is fundamental for a multi-agent system to succeed in reaching its objectives. A negotiation mechanism must be commonly known such that negotiations are governed by previously established rules.

Auctions are simple and well-known mechanism design algorithms used to reach agreements on the allocation of scarce resources, where "resource" is a general concept. The resource could be, for example, the right to explore some good or simply to know some information. Auctions allow to allocate resources efficiently as they are assigned to the agents that give the highest bid value for them. Auctions take place between the *auctioneer* agent and the *bidder* agents. The auctioneer's objective is to allocate the "resource" to one or more of the bidders (WOOLDRIDGE, 2009).

There are different auction formats, the most common is the *English auction*. In this auction, a single item is offered by an auctioneer agent and bidding agents place

bids specifying the value they are willing to pay for the auctioned good. Bids are open, *i.e.*, agents are aware of how much other agents are offering for what is being auctioned. Finally, the auction is finished once no agent is willing to raise its bid, then the resource is allocated to the agent that made the highest bid. English auctions are classified as first-price (winner pays the price it bid), open cry (non-sealed bids), ascending auctions (highest bid wins).

Dutch auctions, in turn, are first-price, open cry descending auctions. In this type of auction, the auctioneer starts by dictating the good's value and lowers the price by a small amount until some agent accepts to pay the current price. In this case, the bidder sends the current value, then the auctioneer allocates the good for the winning agent.

Another common type of auction is the *first-price sealed-bid auction*. In this format, only one bid is placed by the bidders. There are no subsequent rounds of bids, and the auctioned good is awarded to the agent that made the highest bid.

Combinatorial auctions, in turn, are used when multiple resources are offered in groups. Let $\mathcal{Z} = \{z_1, ..., z_n\}$ be a set of items to be auctioned. Agents can place bids for every subset Z such as $Z \subseteq \mathcal{Z}$. Let Ag be a set of agents, the outcome of a combinatorial auction is a list of sets $\{Z_1, ..., Z_n\}$, one for each agent $i \in Ag$, such that $Z_i \subseteq \mathcal{Z}$, and for all $i, j \in Ag$ such that $i \neq j$, we have $Z_i \cap Z_j = \varnothing$, *i.e.*, no resource is allocated to more than one agent (WOOLDRIDGE, 2009). Frequently, one would like to maximise the allocation of resources given some preference agents have over allocations. This combinatorial optimization problem is called the winner determination problem and it is NP-hard (LEHMANN; MÜLLER; SANDHOLM, 2006).

Finally, *reverse auctions* take place when an auctioneer wants to buy a resource rather than sell. For example, suppose an agent wants to fulfil a goal but does not know how to do that. An alternative is to start a reverse auction, where other agents will offer bids informing their price for achieving the auctioned goal. In this case, the lowest bid will win as the selling agent is interested in paying the lowest possible price. The winning bidder then receives the task to be performed.

Agents of the solution presented in the next chapter are embedded with auction ability. These agents negotiate through auctions in order to choose which services they are responsible for, and consequently to compose the VNF-FG.

2.3 Final Remarks

Network function virtualisation (NFV) decouples network functions from the underlying hardware by means of virtualisation, thus enabling the replacement of proprietary middleboxes by software components that can be dynamically deployed and configured on demand. The selection of the most appropriate virtual network functions (VNFs) to achieve a particular objective, and the decision on where to deploy VNFs and through which paths they will communicate, are the responsibilities of an NFV orchestrator.

Autonomous agents organised in a multi-agent system can be used to implement an NFV orchestrator. The BDI architecture is a robust architecture that enables the creation of intelligent agents that together can solve complex and dynamic problems. Furthermore, communication and negotiation allow agents to exchange information and organise the allocation of resources.

In this chapter, we provided the required background to understand this research. Next, we detail how the BDI architecture was used to create a decentralised NFV orchestrator.

3 NFV ORCHESTRATION BASED ON BDI REASONING

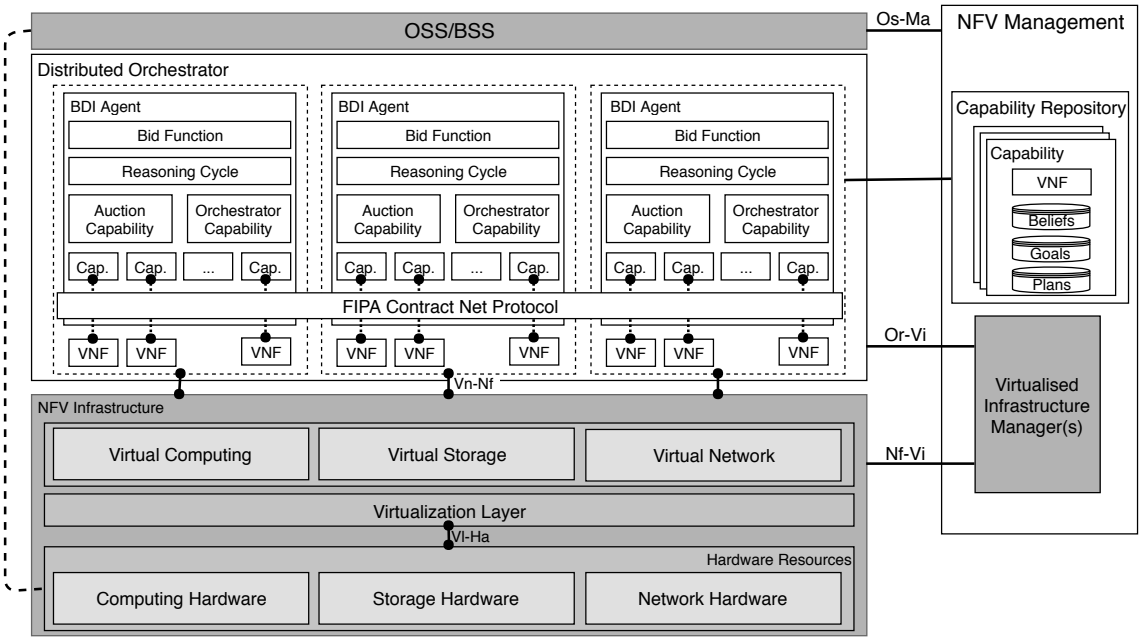
In this chapter, we begin by describing our extended NFV architecture. Next, we formalise the components of our orchestrator. Then, we detail how agents communicate and negotiate through auctions, outlining how our decentralised orchestrator attacks the selection, placement and chaining problems.

3.1 Extended NFV Architecture

To develop a decentralised orchestrator, we rely on the behaviour that emerges from the interaction of autonomous BDI agents. We use the BDI architecture because it allows the creation of flexible and robust decentralised systems, able to act reactively and proactively in order to solve dynamic and complex problems.

Our extended NFV architecture is summarised in Figure 3.1, showing our extension to the ETSI's NFV architecture (ETSI ISG NFV, 2014c). We replaced the original centralised orchestrator and VNF manager, with BDI agents. Components in grey correspond to the ETSI's framework, while those in white are our NFV orchestrator.

Figure 3.1: Extended NFV architecture



In our decentralised orchestrator, each agent controls an NFVI-PoP and is in charge of managing VNFs. Moreover, agents are capable of monitoring the resources of their NFVI-PoP and the network to make management decisions. Each agent is aware

of the connections of its NFVI-PoP as well as the hardware specifications through the Vn-Nf connection. In order to modularise an agent and allow VNFs to be dynamically instantiated, we structure agents in terms of capabilities. Capabilities are reusable components that contain the elements required to coordinate VNFs. Therefore, each VNF is associated with a capability. As result, an agent is a component that aggregates a set of capabilities.

Agents negotiate in order to choose which services they are responsible for. Negotiation is performed by means of *auctions*. In our solution, we have two built-in capabilities, *auction* and *orchestrator*, which are added to every agent, allowing them to, respectively, participate in auctions, deciding whether to activate other capabilities (*i.e.*, VNFs) and manage already instantiated VNFs. Part of the auction process consists of evaluating an agent's bid. This is performed by the *bid function*, which uses the capability repository to select a suitable capability (and, consequently, its VNF) to work on what is being auctioned.

3.2 BDI Agent Formalisation

Given the informal description of our extended NFV architecture above, we proceed to detail how this decentralised architecture attacks the selection, placement and chaining problems. Next, we provide formal definitions of the concepts described earlier.

Definition 2 (Capability) *A capability is a tuple $\langle v, B, G, P, \mathcal{B}, \mathcal{O}, \mathcal{F}, \mathcal{P} \rangle$, where v is a VNF, B is set beliefs, G is a set of goals, P is a set of plans, $\mathcal{B}$ is a belief revision function, $\mathcal{O}$ is an option generation function, $\mathcal{F}$ is a filtering function and $\mathcal{P}$ is a plan selection function.*

Definition 3 (Bid Function) $\mathcal{U} : A \times \wp(B) \rightarrow \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0}$ *— the bid function is a function that takes into account a capability (which is associated with a VNF) and the current set of beliefs (which contains the agent's knowledge about its physical resources and the network) and produces three real numbers, the bid components used in our auction protocol.*

Definition 4 (Agent) *An agent is a tuple $\langle G, I, T, \mathcal{U} \rangle$, where G is a set of current agent goals, I is a set of plans that correspond to agent intentions, T is a set capabilities, and $\mathcal{U}$ is a bid function.*

Definition 5 (Multiagent System) *A multiagent system is a tuple $\langle A, Prot \rangle$, where A is a set of agents, and $Prot$ is a message protocol that each $a \in A$ is able to understand.*

According to these definitions, it can be seen that capabilities specify parts of an agent. Each capability has a set of beliefs B, which collectively represent the agent belief base, *i.e.*, an agent's belief base is $\bigcup_{t \in T} B_t$. Goals of a capability consist of declarations of the goals that a capability can achieve (they represent the capability interface (NUNES, 2014)), and plans consist of a sequence of actions to achieve goals. As said above, capabilities command VNFs, which is achieved by four functions, which we detail next.

- $\mathcal{B} : \wp(E) \times \wp(B) \rightarrow \wp(B)$ — the belief revision function takes into account events E perceived by an agent, *e.g.*, messages received from other agents, and the current set of beliefs, and produces an updated set of beliefs.
- $\mathcal{O} : \wp(B) \times \wp(G) \rightarrow \wp(G)$ — the option generation function takes into account the current set of beliefs and goals, and produces an updated set of goals, *i.e.*, goals may be generated or dropped.
- $\mathcal{F} : \wp(G) \rightarrow \wp(G)$ — the filtering function takes into account a set of goals and produces a subset of goals, those that the agent will be committed to achieve.
- $\mathcal{P} : \wp(B) \times \wp(P) \rightarrow P$ — the plan selection function takes into account the current set of beliefs, *i.e.*, current context, and a set of plans that are candidates to achieve a goal, and selects one, among the candidates, to be executed.

Each capability is thus a part of an agent, and an agent has a set of capabilities. In addition, agents at runtime have a set of goals, which drives their behaviour. These are added as result of the functions described above or execution of plans. An agent may be committed to achieve not all of its goals, but a subset. Goals that an agent is committed to achieve are referred to as intentions, and the agent must have a plan to achieve them. Therefore, agent intentions are a set of plans, which are those that the agent is executing at the moment.

Example 1 *A firewall in this system is represented as a capability and associated to a VNF with the specific network function implementation, which performs changes to the traffic and consequently to the network, i.e. the environment in which the agent is in. This capability might believe a specific IP address or an entire IP block is malicious, which could be generated after recurrently blocking port scanning requests. The act of blocking port scanning requests is a plan that achieves a goal added to this capability a priori.*

Finally, the set of agents, with their capabilities that are responsible for coordinating all VNFs, comprise a multiagent system. This system is our decentralised NFV orchestrator. Given that all agents must communicate, message protocols are specified in this multiagent system, and all agents must be able to understand them.

3.3 BDI Agent Interaction

Given that our proposal consists of a decentralised orchestrator, in which independent and autonomous agents must collectively attack the selection, chaining and placement problems, we must provide means for agents to communicate and negotiate. Much research has been done to support agent communication (FININ et al., 1994; LABROU; FININ, 1997). Therefore, we adopt ACL, which specifies standards associated with message exchange. This specification is implemented in many agent and BDI platforms.

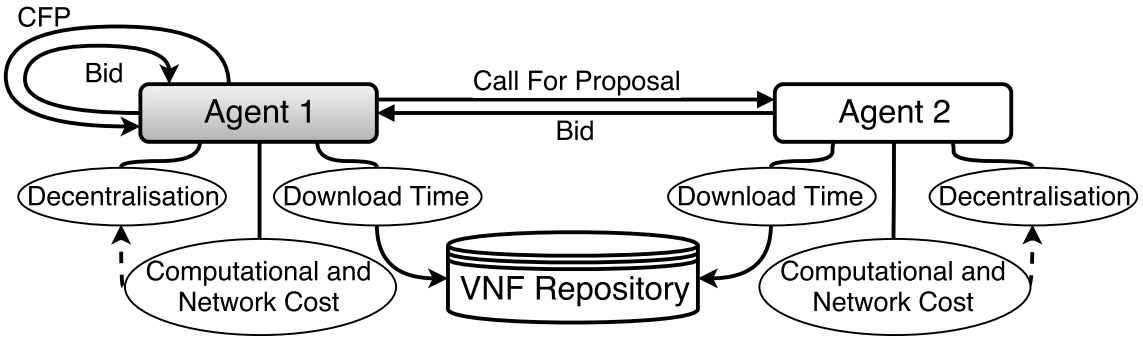
In this section, we detail how auctions are used in our system to attack the selection, placement and chaining problems, and explain how the auctioneer evaluates bids.

3.3.1 Auction Process: the NFV-A Protocol

Auction algorithms are simple and well-known mechanisms used to reach agreements on the allocation of scarce resources, where "resource" is a general concept. In the orchestrator problems, we use a first-price, sealed-bid, reverse auction to negotiate the placement of VNFs and to make decisions regarding the routes of network forwarding graphs. In our reverse auction, agents secretly respond to the auctioneer agent, i.e. agents do not know each other's bids, with their cost to accomplish the auctioned goal. Given that we use a reverse auction, agents look for the lowest price to accomplish a certain task. Therefore, the lowest bidder wins the auction.

We use the FIPA ACL specification for supporting agent communication. For the auction process, we adopt the FIPA Contract Net interaction protocol (introduced in Chapter 2) to build the NFV-A protocol. In this protocol, an initiator agent sends a call for proposal (CFP), and each participant agent either replies with a proposal— *i.e.*, a bid —or refuses to participate. Based on the replies, the initiator agent selects a bid (the lowest price wins the auction), and informs all participants of the result (accept proposal or reject proposal). The participant who had its proposal accepted, after performing what

Figure 3.2: Overview of the auction process.



was requested, informs the result to the initiator.

In our model, when an agent is unable to accomplish a particular service (or achieve a particular goal), it initiates an auction by sending CFPs to candidate agents, *i.e.*, auction participants. Candidate agents are those that meet the requirements of the VNF to be allocated. They must evaluate whether they have the required capabilities to accomplish the service; if not, whether the capability should be added to the agent; and the associated price. The price is the combination of deploying the VNF (if needed) and consuming computational and network resources. The agent that placed the lowest bid wins the auction and will be responsible for achieving the goal. When an agent has its proposal accepted, it is said that this agent *plays a role* in the multiagent system, and there is *commitment* between the initiator and participant agents.

An overview of this process is presented in Figure 3.2, in which Agent 1 starts an auction by sending CFPs to candidate agents, including itself. Although Agent 1 cannot achieve the goal considering its currently deployed VNFs, it might conclude, as result of the auction process, that it is the most suitable NFVI-PoP to deploy a new VNF to achieve this goal. Bids are composed of three components (shown in Figure 3.2), which are returned from the bid function $\mathcal{U}$. They provide the auctioneer with information to reason, considering its preferences, about the most adequate agent to delegate a goal. We formally define constrained goal, the NFV-A protocol and preferences as follows.

Definition 6 (Constrained Goal) *A constrained goal g_i^C is a tuple $\langle g_i, C \rangle$, where g_i is a goal to be achieved and C is a set of constraints that must be satisfied to achieve this goal. Each $c \in C$ is a tuple $\langle n, op, val \rangle$, where $n \in R_N \cup NM$ is a network resource or measurement, op is a comparison operator (e.g., $>$ or $\leq$) and val is a value.*

Example 2 *Consider the goal $isTrafficMalicious$ to be achieved, generated as a consequence of an unusual spike of 500 MB/s on the incoming traffic of an agent. Consequently, a constraint to achieve this goal is to have bandwidth higher than or equal to 500 MB/s.*

This is complemented by a design-time-specified constraint regarding packet loss, which should be lower than 20%. The constrained goal is, therefore, $\langle isTrafficMalicious, \{\langle B, \geq, 500\rangle, \langle P, <, 20\%\rangle\}\rangle$.

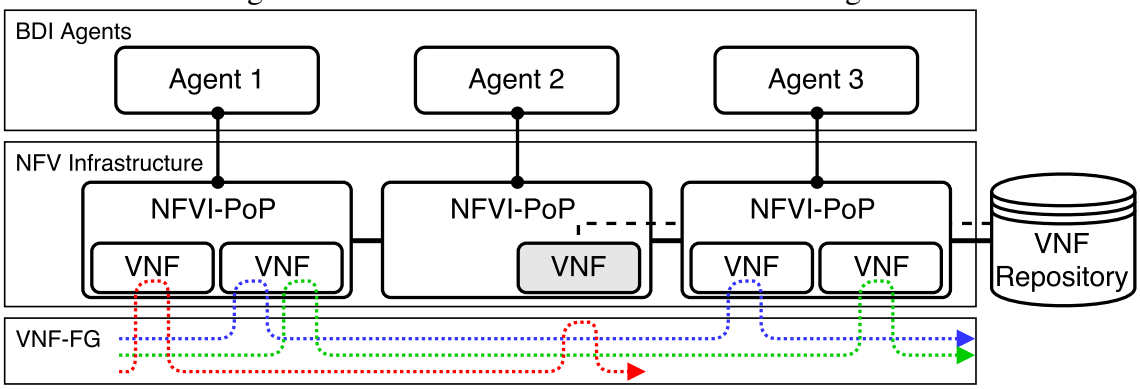
Definition 7 (NFV-A) *Let* $\mathcal{G} = \{g_1^C, ..., g_n^C\}$ *be a set of constrained goals to be auctioned, an atomic bid* β *is a tuple* $\langle g_i^C, \mathcal{T}, \mathcal{C}, \mathcal{D}\rangle$ *where* $g_i^C \in \mathcal{G}$*,* $\mathcal{T}$ *is the time to download a VNF to be deployed,* $\mathcal{C}$ *is the computational and network cost to achieve the goal, and* $\mathcal{D}$ *is a metric capturing the network decentralisation if* g_i^C *is achieved by the agent that placed the bid. Agents calculate* $\mathcal{T}$*,* $\mathcal{C}$ *and* $\mathcal{D}$ *using the bid function* $\mathcal{U}$*. The agent offering the best combination of* $\mathcal{T}$*,* $\mathcal{C}$*, and* $\mathcal{D}$ *considering the auctioneer's preferences wins the auction and is assigned to achieve* g_i^C*.*

Definition 8 (Preferences) *A preference* pf *is a real number* $\mathbb{R}$ *in* $[0, 1]$ *corresponding to the importance of a bid component to achieve a particular constrained goal, where 0 is the lowest importance and 1 the highest. Let preferences* $pf_{\mathcal{T}}$*,* $pf_{\mathcal{C}}$ *and* $pf_{\mathcal{D}}$ *represent the preferences of, respectively,* $\mathcal{T}$*,* $\mathcal{C}$ *and* $\mathcal{D}$ *for an auctioned goal. Moreover,* $pf_{\mathcal{T}} + pf_{\mathcal{C}} + pf_{\mathcal{D}} = 1$.

Because a goal can be achieved by different VNFs, the auction is a means of selecting which VNF will achieve a goal (selection problem). Furthermore, given that bids can be placed by agents representing different NFVI-PoPs using the same VNF, the auction is also a means of selecting the location of a VNF (placement problem). Finally, our solution assumes that bidders assess the path between them and the auctioneer when placing a bid (chaining problem).

We illustrate this agent-based solution in Figure 3.3. In this example, we have two VNF-FGs (blue and green dotted lines) that traverse multiple VNFs. In addition, this figure illustrates the creation of a new VNF-FG as the result of a finalised auction. The red line (bottom most dotted line) represents the network flow that will go through the VNF being downloaded from the VNF repository into the NFVI-PoP of agent 2, which won the auction and is now instantiating the new VNF (grey background). This new VNF is being downloaded through the physical link that connects the NFVI-PoP of agent 2 to the NFVI-PoP controlled by agent three, which is connected to the VNF repository.

Figure 3.3: An NFV architecture based on BDI agents.



3.3.2 Bid Evaluation

To choose a winner, auctioneers receive from each bidder the three bid components introduced above. The values of these components are used to calculate a utility value considering the auctioneer's preferences $pf_{\mathcal{T}}$, $pf_{\mathcal{C}}$ and $pf_{\mathcal{D}}$ over the auctioned goal. Currently, preferences are inputted at design time by a network operator for all constrained goals in the system.

Example 3 *Assume that an auctioneer has preferences 0.5, 0, and 0.5 associated with $\mathcal{T}$, $\mathcal{C}$ and $\mathcal{D}$, respectively, to achieve the $isTrafficMalicious$ goal. This means that the amount of the network infrastructure usage is irrelevant (as long as it meets the computational and network constraints), and the time to start analysing suspicious traffic and the decentralisation of VNFs are equally important to achieve this goal.*

Preferences are used as weights for each bid component, when they are combined in a weighted sum. Given that each bid component is in a different unit, we normalise them to a value in $[0, 1]$ considering the minimum and maximum values of received bids, for each component—note that the auctioneer receives bids only from agents that meet all hard constraints. Normalised values are denoted by $\hat{\beta}_{\mathcal{T}}$, $\hat{\beta}_{\mathcal{C}}$, and $\hat{\beta}_{\mathcal{D}}$. As result, the agent that placed the bid associated with the highest utility is the winner, that is, the lowest price considering the auctioneer's preferences, as shown in the equation below.

$$\underset{\beta \in Bid}{\operatorname{argmin}} \quad pf_{\mathcal{T}} \times \hat{\beta}_{\mathcal{T}} + pf_{\mathcal{C}} \times \hat{\beta}_{\mathcal{C}} + pf_{\mathcal{D}} \times (1 - \hat{\beta}_{\mathcal{D}}) \tag{3.1}$$

In this equation, we use the complement of $\hat{\beta}_{\mathcal{D}}$ because, as opposed to other components, the higher its value, the better.

3.4 Final Remarks

In this chapter, we introduced our extension to the NFV architecture proposed by ETSI (ETSI ISG NFV, 2013) that uses BDI agents to promote orchestration through negotiation using auctions. This system allows network operators to input their preferences despite three distinct characteristics: (i) time to achieve the goal; (ii) allocation of computational and network resources; and (iii) decentralisation. Finally, we detailed how an auctioneer evaluates auctions in our NFV-A protocol.

In the next chapter, we complete our decentralised architecture by introducing our bidding heuristic that together with the NFV-A protocol, provides agents with the capacity of attacking the selection, placement and chaining problems.

4 BIDDING HEURISTIC

We detailed above how our decentralised orchestrator works. In this chapter, we specify how agents calculate their bids through our bidding heuristic. Before detailing our bidding heuristic, we first formalise, in the next section, key concepts that are used in later sections. Next, we describe the algorithm used by bidder agents to place bids.

4.1 Notation and Definitions

In a computer network, nodes and links have a set of their available *resources*, $r_i \in R$. Each r_i can be either a computational resource in R_C or a network resource in R_N. Therefore, $R = R_C \cup R_N$ and $R_C \cap R_N = \emptyset$. Each type of computational resource and network resource are represented by R_C and R_N, respectively. In this work, we assume that there are four types of resources, from which three are computational resources and one is a network resource. The amount of these resources are represented using the notation introduced in Table 4.1 (first four rows), which also indicates to which set they belong. Moreover, there are measurements that can be collected from the network. The last rows of Table 4.1 correspond to the network measurements, $nm \in NM$, associated either with a network link or a path.

In the context of NFV, a node corresponds to a point-of-presence that has NFV-enabled infrastructure (NFVI-PoP), *i.e.*, hardware infrastructure that can host VNFs. For simplicity, we assume that their hardware resources are homogeneous in specification (*e.g.*, processor frequency and memory latency) but not in size (*e.g.*, amount of CPU and memory). Each NFVI-PoP p is characterised by a set of functions that give its specification and current state, detailed below, where NGB_p is the set of NFVI-PoP that are neighbours of p.

Table 4.1: Notation for resources and network measurements.

Symbol	Set	Description
U	R_C	Number of CPU cores.
M	R_C	Amount of RAM memory in GB.
D	R_C	Amount of disk memory in GB.
B	R_N	Bandwidth of a link in MB/s.
L	NM	Latency of a link in ms.
P	NM	Percentage of packet loss of a link.
H	NM	Number of hops in a path.

- $total_p : R_C \to \mathbb{R}_{>0}$: gives the total amount of a particular computational resource of p.
- $free_p : R_C \to \mathbb{R}_{\geq 0}$: gives the current available amount of a particular computational resource of p.
- $value_p : NPoP \times R_N \cup NM \nrightarrow \mathbb{R}_{>0}$: gives the value of a particular network resource or measurement of a link between p and a given NFVI-PoP $p' \in NPoP$, neighbour of p, that is, $\operatorname{dom} value_p = NGB_p \times R_N$.

NFVI-PoPs provide different amounts of resources, which are allocated and consumed by VNFs when instantiated. While NFVI-PoPs have an specification of *available* resources, VNFs are associated with *required* resources. The requirements of a VNF v are given by the following function.

- $req_v : R \to \mathbb{R}_{>0}$: gives the amount of a particular resource required by v.

We use p, p', p'', ... to denote NFVI-PoPs and v, v', v'', ... to denote VNFs. Using this introduced notation we proceed to the description of our NFV auction protocol. Next, we use the introduced formalisation to explain how agents calculate their bids.

4.2 Bidding Heuristic

When an agent receives a CFP, it first evaluates if it has a plan (executed by a capability) that can achieve the auctioned goal. If multiple plans can achieve the goal, the agent uses a plan selection function, *e.g.*, Faccin and Nunes (2015b). A simple approach is to prioritise plans that demand less time to execute the VNF, as implemented in our evaluation. Then, the bidder performs the following tasks: (i) evaluation if the NFVI-PoP meets computational requirements of the VNF that will execute the plan; and (ii) calculation of each bid component. Network hard constraints—in the VNF requirements and specified in the goal constraints—are evaluated while calculating the computational and network cost. If any hard constraint is unsatisfied, the agent informs the auctioneer that it refuses to participate in the auction. Otherwise, it sends its bid. Currently, we assume that agents belong to the same provider. Therefore, there is no need for incentivization for them to participate in auctions because they always evaluate CFPs and send a bid if constraints are satisfied. If there are other candidate plans available in case of failure in meeting VNF requirements and goal constraints, these other plans are evaluated before sending a refusal to the auctioneer. We describe the tasks of calculating a bid as follows.

4.2.1 Evaluation of Computational Requirements

Computational requirements are related to the instantiation of a VNF. In order to instantiate a VNF v in an NFVI-PoP p controlled by a bidder agent, p must have available computational resources that meet v's requirements. Formally, v's requirements are satisfied when $sat_{req}(p, v)$ holds, as shown below.

$$sat_{req}(p, v) := \forall r_C \in R_C, req_v(r_C) \leq free_p(r_C) \tag{4.1}$$

For instance, suppose an NFVI-PoP p has the following resources available: (i) 8 CPU cores: $free_p(U) = 8$; (ii) 16 GB of RAM: $free_p(M) = 16$; and (iii) 500 GB of disk: $free_p(D) = 500$. Consider the requirements of VNF v to be instantiated: (i) 2 CPU cores: $req_v(U) = 2$; (ii) 4 GB of RAM: $req_v(M) = 4$; and (iii) 50 GB of disk: $req_v(D) = 50$. Since all required resources to instantiate VNF v are available in NFVI-PoP p, then $sat_{req}(p, v)$ holds.

4.2.2 Bid Components

As introduced earlier, each bid has three components $\mathcal{T}$, $\mathcal{C}$, and $\mathcal{D}$, which correspond to the VNF download time, the computational and network cost, and network decentralisation, respectively. Their evaluation represent the implementation of the bid function $\mathcal{U}$ defined in Chapter 3. We next detail how each component is calculated.

VNF Download Time. In our solution, we currently assume that a VNF associated with an agent cannot be responsible for achieving multiple goals in parallel. Therefore, if a VNF is executing, another VNF must be deployed. However, after achieving a delegated goal, the VNF becomes idle and, consequently, becomes eligible to achieve another goal. VNFs are made available to agents through the VNF repository, a server that provides the service of making VNFs available for download, which is known to all agents.

The VNF download time is: (i) 0, if the VNF is already deployed and does not need to be downloaded; or (ii) the estimated time to download, from the VNF repository, the VNF that can execute the plan selected to achieve the auctioned goal. The estimated time considers the path with the highest available throughput to download the VNF in the shortest amount of time and the size of the VNF to be downloaded.

Computational and Network Cost. The computational and network cost is our

main bidding component. It assesses the amount of computational and network resources employed to instantiate a VNF and achieve a goal as well as evaluates if the instantiation of a VNF in a particular NFVI-PoP would leave resources unusable, thus preventing the instantiation of other VNFs. Moreover, while calculating this bidding component we: (i) verify whether goal constraints are met; and (ii) choose the path between the auctioneer and bidder agent to be used, if the bidder wins the auction (chaining problem).

The main idea is to find the path with the lowest cost between the auctioneer and the bidder. In order to do so, we represent the network as a graph, in which nodes are connected by links with weights capturing the amount of resources being consumed due to the use of this link. The identification of the path with the lowest cost is done using a customised version of the widely known Dijkstra's algorithm, which finds the shortest path in a graph in polynomial time. The key issue is thus to specify meaningful weights to the network links.

In Algorithm 1, we show the Dijkstra's algorithm, customised to calculate accumulated network measurements to verify goal constraints and the link weights. Our customisations correspond to the highlighted lines in the algorithm. It receives as input a graph representing the network, a source (bidder) and a destination (auctioneer) nodes, a constrained goal and the VNF to be deployed.

We first detail how we evaluate whether goal constraints are met. In the variables initialised in lines 4–6, we store accumulated latency, packet loss and hops in paths. Therefore, when evaluating the use of a particular link in a path, we update these variables. This is done in lines 21–23 in Algorithm 1. Then, in line 24, we check if goal constraints, associated with network resources (bandwidth) and measurements (latency, packet loss and hops), are satisfied. This is done using the $sat_{cst}(var, value, C)$ function, shown below.

$$sat_{cst}(var, value, C) := \forall c \in C | var = c[n], c[op](value, c[val]) \tag{4.2}$$

where var is the network resource or measurement to be evaluated, $value$ is its current value and C is the set of goal constraints. This given value is then checked against the value informed in goal constraints using the specified comparison operator. If any constraint is unsatisfied, the link is discarded.

We now focus on the link weight calculation. In order to evaluate consumed resources, we consider both the link and its source node and assess the utility of computational resources that remain left by going throught a node (NFVI-PoP) of the network.

Algorithm 1 $lowestCostPath(G, s, d, g_i^C, vnf)$

Require: Graph G
Require: Source node s
Require: Destination node d
Require: Constrained goal g_i^C
Require: VNF to be deployed vnf

for all $v \in V[G]$ **do**
  $dist[v] \leftarrow +\infty$
  $prev[v] \leftarrow 0$
  $latency[v] \leftarrow 0$
  $pktLoss[v] \leftarrow 0$
  $hops[v] \leftarrow 0$
end for
$dist[s] \leftarrow 0$
$Q \leftarrow V[G]$
while Q is not an empty set **do**
  $u \leftarrow \text{Extract}_{\text{Min}}(Q)$
  if u is d **then**
    $S \leftarrow$ empty set
    while $prev[u]$ is not undefined **do**
      $S \leftarrow S \cup \{(prev[u], u)\}$
      $u \leftarrow prev[u]$
    end while
    return S
  end if
  for all edge (u, v) outgoing from u **do**
    $latency[v] \leftarrow value_u(v, L) + latency[u]$
    $pktLoss[v] \leftarrow pktLoss[u] + ((1 - pktLoss[u]) \times value_u(v, P))$
    $hops[v] \leftarrow 1 + hops[u]$
    if $sat_{cst}(L, latency[v], C) \wedge sat_{cst}(P, pktLoss[v], C) \wedge sat_{cst}(H, hops[v], C) \wedge sat_{cst}(B, value_u(v, B), C)$ **then**
      $rscEval_u \leftarrow rscEval_{hop}(u)$
      if $u = s$ **then**
        $rscEval_u \leftarrow rscEval(u, vnf)$
      end if
      $weight \leftarrow value_u(v, B) \times (rscEval_u + rscEval_{hop}(v))$
      if $dist[u] + weight < dist[v]$ **then**
        $dist[v] \leftarrow dist[u] + weight$
        $prev[v] \leftarrow u$
      end if
    end if
  end for
end while
return undefined

Table 4.2: Definition of mathematical functions and logic predicates.

Function/Predicate	**Expression**	**Description**
$freeRscUtil(p)$	$\frac{\lvert R_C \rvert}{\sum_{r_C \in R_C} \left(\frac{free_p(r_C)}{total_p(r_C)} \right)^{-1}}$	Harmonic mean of the ratio of available computational resources of an NFVI-PoP p.
$consRscUtil(p, v)$	$\frac{\lvert R_C \rvert}{\sum_{r_C \in R_C} \left(\frac{req_v(r_C)}{total_p(r_C)} \right)^{-1}}$	Harmonic mean of the ratio of the computational resources required by VNF v from an NFVI-PoP p.
$full(p)$	$\forall v \exists r_C \in R_C,$ $req_v(r_C) > free_p(r_C)$	$full(p)$ is true when there is no VNF that can be deployed in the NFVI-PoP p, that is, there is at least one VNF requirement unsatisfied, for all VNFs.
$canDeploy(p, v)$	$\exists v' \forall r_C \in R_C,$ $req_{v'}(r_C) \leq$ $free_p(r_C) - req_v(r_C)$	$canDeploy(p, v)$ is true when there is a VNF that can be deployed in an NFVI-PoP p after the deployment of the VNF v in p.

Typically, to instantiate a VNF in the future we need to use part of all computational resources. Consequently, if there is only a subset of resources available, the instantiation of new VNFs may be problematic. Our heuristic thus aims to incentivise the balanced use of resources. This balance in the use of resources is captured by calculating the harmonic mean of the ratio of available computational resources of an NFVI-PoP. This is calculated by the functions $freeRscUtil(p)$ and $consRscUtil(p, v)$, shown in Table 4.2, which calculate the utility of an NFVI-PoP's free resources—current resources and resources consumed by a VNF instantiation, respectively.

These introduced functions are used to evaluate a link weight in the following way. We first assess the cost of using the source and target nodes of the link. If the node is a hop, no computational resource is used, only network resources (links). Hence, if VNFs can be instantiated in the node due to available resources, the cost is higher because links surrounding this node should preferably remain less used. The cost is then given by $freeRscUtil(p)$. If no VNF can be instantiated (*i.e.*, the node is full), the cost is 0, because the link can be used without compromising the instantiation of VNFs in the future in that node. This is formalised in the equation below, which uses the $full(p)$ logic predicate introduced in Table 4.2.

$$rscEval_{hop}(p) = \begin{cases} 0 & \text{, if } full(p) \\ freeRscUtil(p) & \text{, otherwise} \end{cases} \tag{4.3}$$

The evaluation of the source node (bidder) is different, because it requires consuming resources for instantiating the VNF. The idea is also to consider the instantiation of VNFs in the future. If, after instantiating the VNF, no other VNF can be instantiated in that node, the consumed resources are not only those consumed by the VNF but all resources ($freeRscUtil(p)$), as those remaining become useless. In the opposite case, the cost is lower, corresponding only to the consumed resources. This is also the case if, although no new VNFs can be deployed, there is no alternative VNF that can be instantiated in that node allowing the instantiation of other new VNFs, meaning that only one VNF can be deployed in the node anyway. This is formalised as follows, using the $canDeploy(p, v)$ predicate, also defined in Table 4.2.

$$rscEval(p, v) = \begin{cases} consRscUtil(p, v) & \text{, if } canDeploy(p, v) \vee \\ & \neg\exists v' \, canDeploy(p, v') \\ freeRscUtil(p) & \text{, otherwise} \end{cases} \tag{4.4}$$

Finally, the link weight is its bandwidth (the higher the bandwidth, the higher the cost) multiplied by the sum of the cost evaluation of the nodes connected to the link, as shown in line 29 of Algorithm 1. Lines 25–28 check if the link source is the path source to treat it in a distinguished way. This completes the explanation regarding this bidding component and we then proceed to the last component.

Decentralisation. The decentralisation component corresponds to how much physically distant the auctioneer and the bidder are. Given that our input information does not capture geographic information about NFVI-PoPs, we use latency to estimate decentralisation, assuming that geographically distant NFVI-PoPs tend to have higher latency than NFVI-PoPs that are geographically closer. This component is thus the accumulated latency of the path that links the auctioneer and the bidder, obtained from the algorithm presented above.

4.3 Final Remarks

In our system, when an agent participates in a negotiation, it evaluates its resources to find out the price of achieving another agent's goal. There are hard constraints that have to be met, such as having enough CPU cores or memory to instantiate a new VNF. Dijkstra's algorithm was adapted to calculate the cost of allocating a VNF in an NFVI-PoP

and using the bandwidth of available links to connect it with another VNF. This process was implemented in a novel testbed, which was used to evaluate the NFV-A protocol and the bidding heuristic, which is described next.

5 TESTBED AND EVALUATION

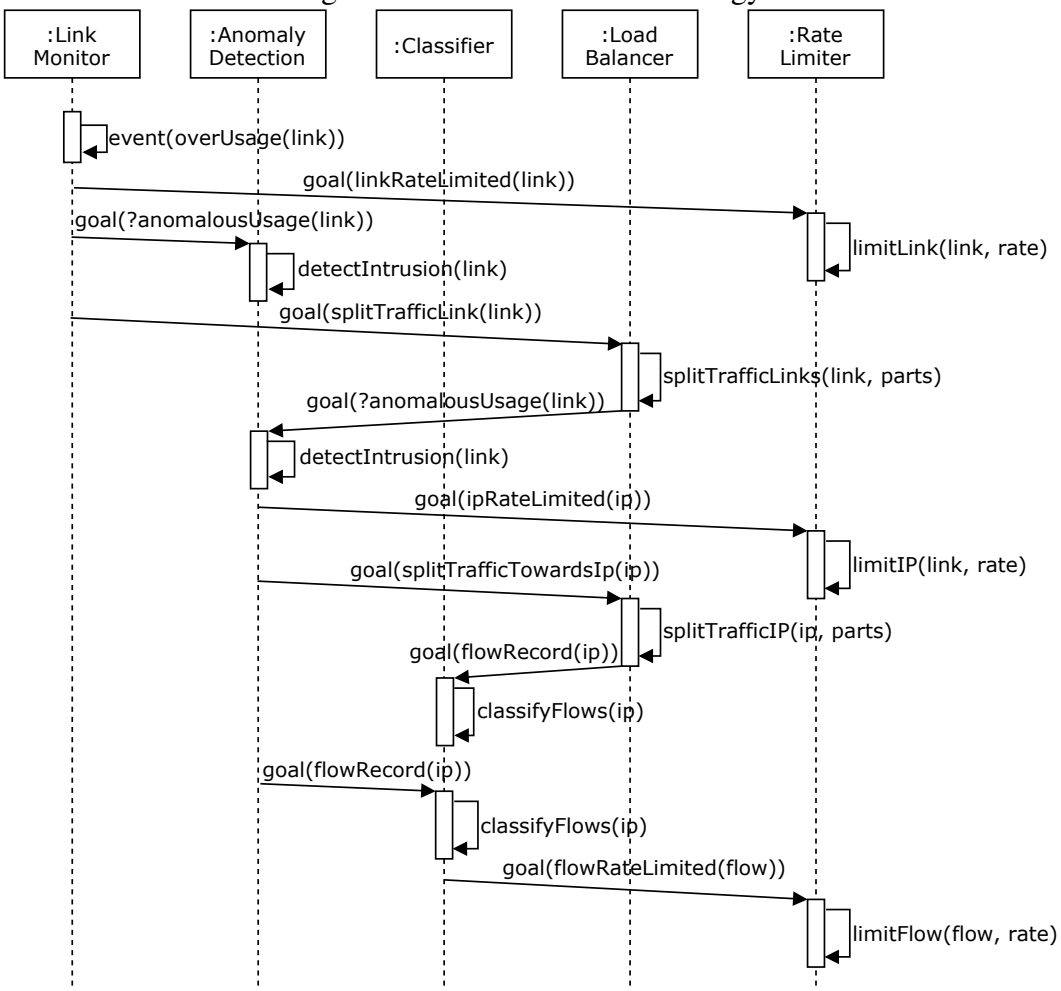
Now that we have presented our NFV-A protocol and the bidding heuristic that together promote the selection, placement and chaining of VNFs, we proceed to describe our implementation and evaluation. In this chapter, we begin detailing a DDoS resilience strategy used to evaluate our auction-based approach for NFV orchestration. Next, we describe a prototype implementation, an experimental evaluation and a scalability analysis of our system.

5.1 Scenario: DDoS Resilience Strategy

Network resilience can be defined as the ability that a network has to recover from an incident (STERBENZ et al., 2010). Different types of incidents can affect the operation level of a network, such as malicious attacks, equipment breakdown, and poor administration. We evaluate our model on a DDoS attack scenario, because it is a common issue, with a broad variety of solutions. In this section, we detail the network functions used in our scenario and how they are combined into a network resiliency strategy.

Initially, an analyst designs agents, capabilities and their VNFs, and indicates the initial state of the network, *i.e.*, which services should be provided and how. The VNFs associated with capabilities are stored in a VNF repository. We took inspiration from the capabilities built by Nunes *et al.* (NUNES; SCHARDONG; SCHAEFFER-FILHO, 2017), where they follow the resilience strategy created by Schaeffer-Filho *et al.* (SCHAEFFER-FILHO et al., 2012). We designed five capabilities that represent commonly used network functionality, and observed their collective behaviour emerging into a meaningful resilience strategy (NUNES; SCHARDONG; SCHAEFFER-FILHO, 2017): (i) *Link Monitor*: responsible for detecting any link overused in the network; (ii) *Rate Limiter*: performs traffic throttling to a specific link, to a specific IP address, and to specific flows (where both source and destination are known); (iii) *Anomaly Detector*: detects the target of a DDoS attack; (iv) *Classifier*: analyses and identifies malicious flows, among all traffic to a specific host; and (v) *Load Balancer*: enables agents to split traffic into two flows. As result of the interaction between agents, the emerged behaviour that can be observed in our multiagent system is shown in Figure 5.1. Differently from Nunes *et al.* (NUNES; SCHARDONG; SCHAEFFER-FILHO, 2017), in which a *Classifier* requires a *Flow Exporter* to perform its activities, our *Classifier* is embedded with the capacity of recording

Figure 5.1: DDoS resilience strategy.



flows. Furthermore, we add the *Load Balancer* capability to increase the chance that large flows can be dealt with. Given this brief overview of the capabilities used in our evaluation, we next detail them along with the *Auction* and *Orchestrator* capabilities.

Rate Limiter. Preventing a DDoS attack requires limiting network traffic in different granularities and under specific circumstances. This is achieved by the *Rate Limiter* capability, where three different plans limit the network traffic at different levels. The plans of the *Rate Limiter* capability are detailed in Table 5.1. Plans are named and characterised by: (i) the *goal* it can achieve; (ii) the required *preconditions* for the plan to be executed; and (iii) *actions* it performs. Belief actions, *belief(b)*, indicate that a belief *b* is being added (*belief(b)* or *belief(¬b)*) or removed (*belief(not b)*) from the belief base. Primitive actions, `action(param)`, are implemented by the capability's VNF, which can be seen as agent actuators.

The actions taken by plans change the belief base, adding preconditions required by other plans. For instance, plan LL achieves the goal of having *linkRateLimited(link)*

Table 5.1: Plans of the *Rate Limiter* capability.

Plan: LimitLink (LL)	**Plan: UnlimitLink (UL)**
Goal: $linkRateLimited(link)$ **Precondition**: - **Actions**: $\texttt{limitLink}(link, RATE)$ $belief(linkRateLimited(link))$	**Goal**: $\neg linkRateLimited(link)$ **Precondition**: $linkRateLimited(link)$ **Actions**: $\texttt{unlimitLink}(link)$ $belief(\neg linkRateLimited(link))$
Plan: LimitIP (LIP)	**Plan: UnlimitIP (UIP)**
Goal: $ipRateLimited(ip)$ **Precondition**: - **Actions**: $\texttt{limitIP}(ip, RATE)$ $belief(ipRateLimited(ip))$ $\forall link.(linkRateLimited(link) \wedge$ $\texttt{linkIP}(link, ip) \rightarrow$ $goal(\neg linkRateLimited(link)))$	**Goal**: $\neg ipRateLimited(ip)$ **Precondition**: $ipRateLimited(ip)$ **Actions**: $\texttt{unlimitIP}(ip)$ $belief(\neg ipRateLimited(ip))$
Plan: LimitFlow (LF)	
Goal: $flowRateLimited(flow)$ **Precondition**: - **Actions**: $\texttt{blockFlow}(flow)$ $belief(flowRateLimited(flow))$ $\forall ip.(ipRateLimited(flow) \wedge$ $\texttt{ipFlow}(ip, flow) \rightarrow$ $goal(\neg ipRateLimited(ip)))$	

in the belief base by triggering the operation $\texttt{limitLink}(link, RATE)$, where $RATE$ is fixed and defined at design time, then adding *linkRateLimited(link)* to the belief base. Consequently, the UL plan, which removes the limit of a given link, can now be selected by the plan selection function if the goal ¬*linkRateLimited(link)* is added, as all its preconditions are met. This happens when the plan LIP is executed.

Link Monitor. Network links connect and transport traffic between hosts. Under normal circumstances, their bandwidth is partially used, remaining below a threshold. Our *Link Monitor* capability has two goals: (i) monitor a link to detect anomalous usage; and (ii) guarantee that the bandwidth of the monitored link remains below a threshold.

We show the belief revision and goal generation functions of the *Link Monitor* capability in Table 5.2. When the *Link Monitor* detects a link being overused, the event *event(overUsage(link))* is triggered, adding the beliefs *overUsage(link)* and *not anomalousUsage(link)* to the beliefs base in the belief revision step. An overused link is due to either a peak in legitimate traffic or an ongoing attack. Two goals are generated because the belief *overUsage(link)* is present: (i) *goal(linkRateLimited(link))* is responsible

Table 5.2: Belief revision and goal generation functions of the *Link Monitor* capability.

Belief Revision
$event(overUsage(link)) \rightarrow belief(overUsage(link)) \wedge$ $belief(not\ anomalousUsage(link))$
$event(\neg overUsage(link)) \rightarrow belief(not\ overUsage(link))$
Goal Generation
$overUsage(link) \wedge not\ anomalousUsage(link) \rightarrow$ $goal(?anomalousUsage(link))$

for restricting the traffic in the overused link; and (ii) *goal(?anomalousUsage(link))* to determine if it is a malicious attack or not. Those goals are generated together in order to guarantee that the network remains functional while overcoming a challenge. Note that *goal(b)* adds the goal of having the predicate *b* as part of the agent belief base (*b* may also be $\neg b$, while *goal(?b)* adds the goal of having *b* or $\neg b$ as part of the agent belief base, *i.e.,* the agent must find out whether *b* or $\neg b$ is true.

Note that goals *goal(linkRateLimited(link))* and *goal(?anomalousUsage(link))* are generated together. However, considering that the ultimate objective is to guarantee that the network remains operational when confronted with a challenge, it is important to protect the network before finding out if an anomalous behaviour is due to a malicious attack. This prioritisation between the two goals is performed in the goal deliberation step. Both the goals *goal(linkRateLimited(link))* and *goal(?anomalousUsage(link))* have no plans that achieve them within the *Link Monitor* capability. The first can be achieved by the *Rate Limiter* capability, while the second by the *Anomaly Detector* capability, which is detailed next.

Anomaly Detector. To achieve the goal of discovering if the traffic is anomalous or not, *i.e. goal(?anomalousUsage(link))*, the *Anomaly Detector* uses the plan described in Table 5.3. The ALS plan analyses the traffic that flows in a link using `detectIntrusion(link)`, which is implemented and performed by its VNF. If there is traffic destined to an IP address that is an outlier, it is considered anomalous, and consequently the link usage is considered anomalous, *i.e. belief(anomalousUsage(link))*. Contrarily, if there are no outliers, then the link is considered to be normal, *i.e. belief(¬anomalousUsage(link))*. Once this belief is set, it is shared with the *Link Monitor* capability, whether it is part of the same agent or not. In the latter case, a message is sent, and the agent updates its belief base.

Two goals are generated for each outlier: (i) *goal(ipRateLimited(ip))* is responsible for restricting the traffic destined to an IP address; and (ii) *goal(flowRecord(ip))* to further

Table 5.3: Plan of the *Anomaly Detector* capability.

Plan: AnalyseLinkStatistics (ALS)
Goal: $?anomalousUsage(link)$
Precondition: -
Actions:
`detectIntrusion(link)`
$\forall ip.(outlier(ip) \rightarrow belief(anomalous(ip)) \wedge$
$\quad goal(ipRateLimited(ip) \wedge goal(flowRecord(ip)))$
$\exists ip.(anomalous(ip) \rightarrow belief(anomalousUsage(link)))$
$\nexists ip.(anomalous(ip) \rightarrow belief(\neg anomalousUsage(link)))$

Table 5.4: Plan of the *Classifier* capability.

Plan: AnalyseIPFlows (AIPF)
Goal: $flowRecord(ip)$
Precondition: -
Actions:
`classifyFlows(ip)`
$\forall flow.(malicious(flow) \rightarrow belief(threat(flow)))$
$\exists flow.(threat(flow) \wedge dstIP(flow) = ip \rightarrow$
$\quad goal(flowRateLimited(flow)))$

analyse the outlier to find the source IP address(es) of the malicious host(s). Note that there are no plans in this capability to achieve these goals. They are achieved by the *Rate Limiter* and *Classifier* capabilities, respectively. Next, we detail the *Classifier* capability.

Classifier. Flows that are destined to an *ip* are recorded and classified by the AIPF through the `classifyFlows(ip)` actuator, which identifies all the flows that are threats. Each malicious flow triggers a goal *flowRateLimited(flow)*, which can be achieved by the *Rate Limiter* capability.

Load Balancer. Since `detectIntrusion(link)` and `classifyFlows(ip)` are CPU intensive actions, the VNFs that perform them have limitations despite the amount of traffic that can be analysed in a single CPU core. In our system, the amount of traffic a VNF can process given the resources required for its instantiation is informed by an operator at design time. In scenarios with heavy traffic and short-resourced NFVI-PoPs or network links, often no *Anomaly Detector* and *Classifier* can be used because no NFVI-PoP can host their VNFs. In such cases, heavy traffic can be divided into lighter flows, which might allow VNFs to be instantiated. Thus, the *Load Balancer* capability implements two plans to divide traffic in such scenarios, which are shown in Table 5.5. Both plans divide the traffic into lighter flows. The number of divisions is inputted into the simulation *a priori*.

The *Load Balancer*'s plans should only be used in the scenarios described above,

Table 5.5: Plan of the *Load Balancer* capability.

Plan: LoadBalanceLink (LBL)
Goal: $splitTrafficLink(link)$
Precondition: $?anomalousUsage(link)$
Actions:
`splitTrafficLink(link, PARTS)`
$\forall virtualLink.(goal(?anomalousUsage(virtualLink)))$
Plan: LoadBalanceIP (LBIP)
Goal: $splitTrafficTowardsIp(ip)$
Precondition: $flowRecord(ip)$
Actions:
`splitTrafficIP(ip, PARTS)`
$\forall flow.(goal(flowRecord(ip)))$

as it is not interesting to divide traffic when a single *Anomaly Detector* or *Classifier* is enough. To achieve that, agents only add the goals *splitTrafficLink(link)* or *splitTrafficTowardsIp(ip)* if the goals *?anomalousUsage(link)* or *flowRecord(ip)* were not achieved, *i.e.* if the auction to instantiate an *Anomaly Detector* or a *Classifier* has failed.

Orchestrator and Auction. These capabilities are added to all agents as part of our decentralised orchestrator. The *Orchestrator* capability has the plan *InstantiateVNF*, which is used to instantiate new VNFs in an agent when new capabilities are activated. It downloads the VNF from the VNF Repository, starts it and creates the virtual path between the NFVI-PoP controlled by the agent and the NFVI-PoP controlled by the auctioneer. The *Auction* capability has two plans (i) *GoalRequest* plan, which can be used to request other agents to achieve a goal; and (ii) *GoalResponse* plan, which handles messages with goal requests. The *GoalRequest* and *GoalResponse* plans follow the NFV-A protocol described in Chapter 3. Auctions are triggered when the *GoalRequest* plan is selected. Agents send replies either accepting or refusing to participate in an auction. If an agent accepts, the three bid components are calculated in the *GoalResponse* plan using the bidding heuristic presented in Chapter 4. In our system, multiple auctions do not exist simultaneously. If an agent wants to trigger an auction, it waits for an ongoing negotiation to end. Moreover, it waits an additional random time, which helps prevent agents from starting auctions at the same time. Next, we describe how we implemented the VNFs used in our agents and the testbed that connects them to their VNFs.

5.2 Prototype Implementation

To evaluate the proposed framework and bidding heuristic, we developed an NFV testbed that integrates a BDI agent platform, namely `BDI4JADE` (NUNES; LUCENA; LUCK, 2011), with `containernet` (PEUSTER; KARL; ROSSEM, 2016), a fork of `Mininet` (LANTZ; HELLER; MCKEOWN, 2010) that implements `Mininet` hosts as Docker containers (MERKEL, 2014). We selected `containernet` because it provides operational system-level virtualisation and allows to fine tune the maximum resource usage of hosts (*e.g.*, memory, network and CPU). Differently from most BDI platforms, `BDI4JADE` is a multi-agent platform that implements agents in pure Java, which facilitates the integration with other applications. In our system, each VNF deployed to the virtualised network is represented by a host in `Mininet` (*i.e.*, a Docker container). Furthermore, the agent-to-VNF communication is implemented using `RESTful` (RICHARDSON; RUBY, 2008) calls. We provide VNFs with a light Python script that sends and receives `RESTful` calls, allowing agents to easily obtain VNF-specific information. Our prototype integrating `BDI4JADE` and `containernet` is publicly available.[1]

At the beginning of the simulation, a configuration file specifying the network topology (resource specification of NFVI-PoPs as well as their links and VNF repository location) is read by the `BDI4JADE` platform, which launches the BDI agents and a Python application responsible for starting `Mininet`. They exchange information through `RESTful` calls and coordinate network configuration and agent-to-PoP assignment. As result, agents are in control of NFVI-PoPs and pre-configured VNFs, which are those present in the initial state of the network, are deployed to specified agents according to the topology configuration file.

The VNFs in our system were implemented as follows: (i) *Link Monitor* uses `ifstat` to monitor incoming and outgoing traffic; (ii) *Rate Limiter* communicates with the `OpenVSwitch` (PFAFF et al., 2009) controller v. 2.5.0 to apply and remove queue-based rate limiting rules to network interfaces and specific source/destination IPs; (iii) *Anomaly Detector* and (iv) *Classifier* use `Snort` (ROESCH et al., 1999) v. 2.9.2.2 to detect suspicious traffic; and (v) *Load Balancer* communicates with `OpenVSwitch` and the `POX` controller to set load balancing rules. The `OpenVSwitch` switches are connected to a `POX` controller (GUDE et al., 2008).

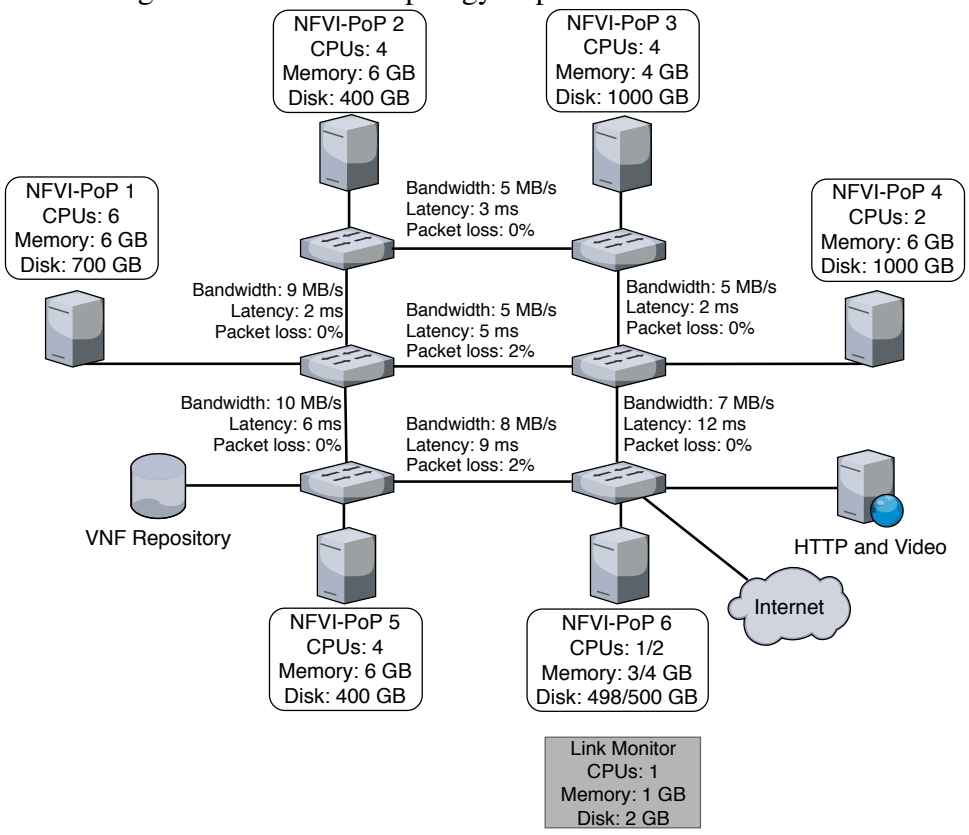
[1] https://sourceforge.net/projects/nfvauction/

5.3 Simulation Settings

To evaluate our solution, we created a server that answers HTTP requests through Python's `SimpleHTTPServer` module[2] version 2.7.14 and provides video streamming through VLC (MÜLLER; TIMMERER, 2011) version 2.2.2. Figure 5.2 illustrates our network topology, which is composed of six NFVI-PoPs. Also, the VNF repository is connected to the fifth NFVI-PoP. The HTTP and video streaming provider server and NFVI-PoP 6 are directly connected to the same switch, and a *Link Monitor* VNF is installed on NFVI-PoP 6, monitoring the link that connects the switch to the server.

The background traffic of our simulation consisted of 50 hosts requesting to watch the video streaming in a time frame, then they stop and make HTTP requests. As in Silva et al. (2016), for each host requesting video streaming traffic, there are other five hosts requesting HTTP traffic. We used `scapy` (SCAPY, 2007) to generate realistic background as well as anomalous traffic. The attack consisted of 8 malicious hosts performing a SYN flood attack, in which packets targeted the two ports of the service server (8000 for HTTP and 8080 for video stream).

Figure 5.2: Network topology implemented in our testbed.



[2]Simple HTTP request handler: https://docs.python.org/2/library/simplehttpserver.html

Table 5.6: VNF requirements and supported workloads.

VNF	CPU	Memory	Disk	Bandwidth
Link Monitor	1 core	1 GB	2 GB	-
Rate Limiter	1 core	1 GB	2 GB	-
Anomaly Detector	1 core	2 GB	10 GB	2 MB/s
Classifier	2 core	4 GB	10 GB	1 MB/s
Load Balancer	1 core	1 GB	2 GB	-

Table 5.7: Preference values and latency constraints of goals.

Goal	$pf_{\mathcal{T}}$	$pf_{\mathcal{C}}$	$pf_{\mathcal{D}}$	Latency Constraint
linkRateLimited(link)	0.8	0.1	0.1	100 ms
?anomalousUsage(link)	0.5	0.5	0.0	-
flowRecord(ip)	0.0	0.2	0.8	-
ipRateLimited(ip)	0.8	0.1	0.1	100 ms
splitTrafficLink(link)	0.3	0.7	0.0	-
splitTrafficTowardsIp(ip)	0.3	0.7	0.0	-
flowRateLimited(flow)	0.8	0.1	0.1	100 ms

The requirements of the VNFs used in our simulation are listed in Table 5.6. All VNFs have a download size of 500 MB, which is the amount of data transferred from the VNF repository to instantiate them.

Preferences in our system are tailored to goals. In our simulation, we used maximum packet loss of 8% for all goals, and maximum latency constraint of 100 ms for the goals *linkRateLimited(link)*, *ipRateLimited(ip)* and *flowRateLimited(flow)*. The preferences and latency constraint associated with all goals are listed in Table 5.7.

5.4 Evaluation Results

In this section, we describe our observations after executing the DDoS resilience strategy implemented in our testbed. First, the emergent solution to the selection, placement and chaining problems devised by agents to mitigate the DDoS attack is presented. Next, we detail the major auctions of our simulation to explain the decisions collectively made by agents. Finally, we provide a scalability analysis of our system.

5.4.1 Emergent Behaviour

By simulating the scenario described above, we obtained the configuration shown in Figure 5.3. The squares in grey show where VNFs have been allocated (selection and placement) and dotted lines represent the chaining of VNFs throughout the adopted mitigation strategy. They are numbered according to the time order that auctions occurred.

Figure 5.3: Emergent selection and resource allocation.

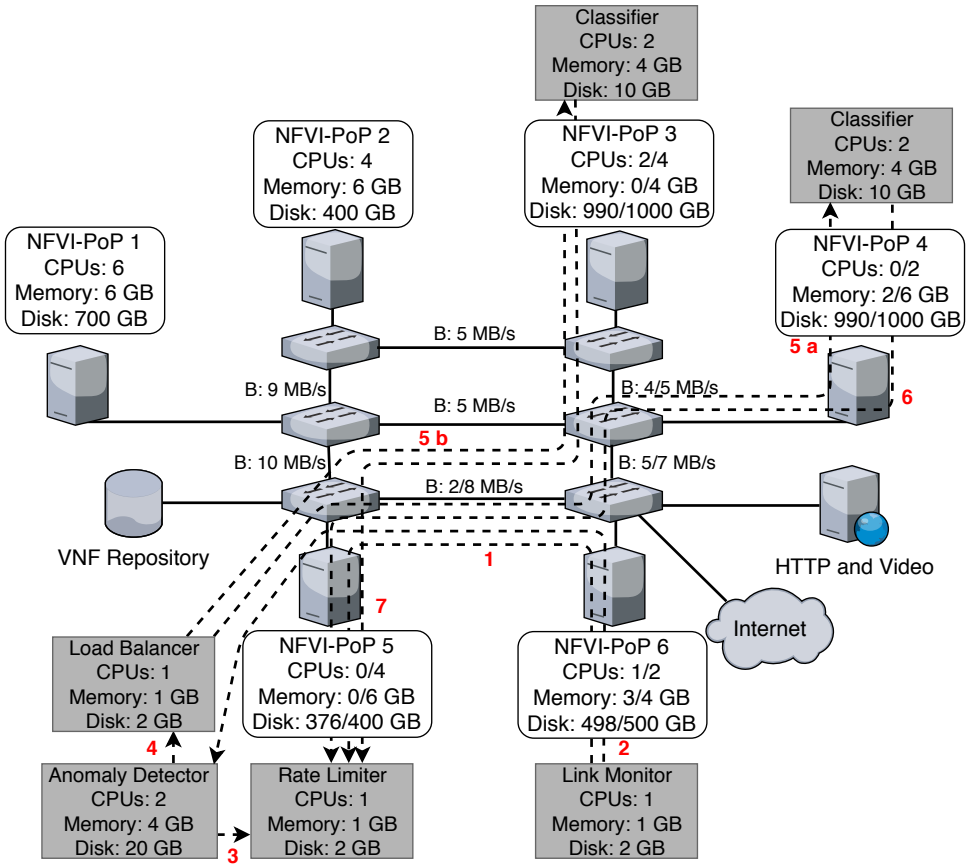
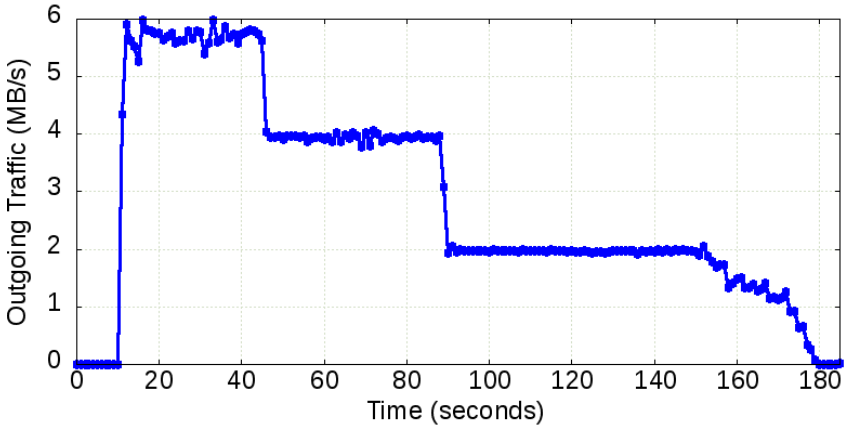


Figure 5.4 shows the amount of traffic from the switch connected to NFVI-PoP 6 to the HTTP and video server throughout the simulation. After a few seconds of simulation, the attack started and the outgoing traffic to the server reached 6 MB/s. Immediately, the *Link Monitor* detected the peak, triggering the auction to achieve the *linkRateLimited(link)* goal and agents selected the LL plan from the *Rate Limiter* capability to achieve this goal. Next, agents executed our bidding heuristic to place bids. Agent 5 won the auction and deployed the VNF to achieve the goal *linkRateLimited(link)*, which reduced the traffic by 50%, dropping to 4 MB/s at 45 seconds of the simulation.

After the *Link Monitor* have auctioned the *linkRateLimited(link)* goal, it started a new auction to have the *?anomalousUsage(link)* goal accomplished. Agents selected the *Anomaly Detector* capability and its VNF, then calculated and submitted their bids.

Figure 5.4: Traffic from the switch towards the HTTP and video server.



Note that the *Anomaly Detector* VNF requires 1 CPU core, 2 GB of memory and 10 GB of disk to work with 2 MB/s of traffic, as shown in Table 5.6. However, this auction requires a VNF to handle 4 MB/s. In this case, agents scaled up the requirements, so that 2 CPU cores, 4 GB of memory and 20 GB of disk were needed to instantiate this VNF, allowing it to handle 4 MB/s of traffic. Agent controlling NFVI-PoP 5 won this auction and immediately downloaded the VNF from the VNF repository and executed the plan to achieve the auctioned goal, reducing the capacity of the link connecting NFVI-PoPs 5 and 6 from 8 MB/s to 4 MB/s. It found the target IP of the DDoS attack and performed two auctions sequentially, *i.e.* starting the second only after the first is finished. First was the auction to fulfil goal *ipRateLimited(ip)*, which Agent 5 also won, because it had the VNF of the *Rate Limiter* capability already instantiated. The *Rate Limiter* capability executed the LIP plan, which reduced the traffic by 50% (from 4 MB/s to 2 MB/s) at 90 seconds of the simulation. After reducing the malicious traffic, the link limit previously established is removed, allowing legitimate traffic while the attack is mitigated. Next, a *flowRecord(IP)* goal was auctioned to deal with 2 MB/s of traffic, requiring a *Classifier* with capacity to process 2 MB/s of bandwidth. Following the requirements listed in Table 5.6, a *Classifier* could be scaled up to use 4 CPU cores, 8 GB of memory and 20 GB of disk to handle 2 MB/s of traffic. However, no NFVI-PoP had enough resources to host this scaled VNF.

Since it was not possible to achieve *Classifier*'s plans due to the bandwidth constraint of this goal, agents refused to participate in this auction as no agent was able to instantiate the *Classifier*'s VNF, *i.e.* $sat_{req}(p, v)$ did not hold. Next, this agent triggered and won the auction to have goal *splitTrafficTowardsIp(ip)* achieved, resulting in the instantiation of a *Load Balancer*. The traffic was separated into two flows, consequently triggering two auctions for *flowRecord(IP)*, each with 1 MB/s, which led to the allocation of two *Classifiers* in the network (Agents 3 and 4). They recorded and found the source

address of the attackers, triggering *flowRateLimited(flow)* for each malicious host discovered. The already instantiated *Rate Limiter* won all these auctions, which took place between 150 and 180 seconds of the simulation, reducing the traffic to operational levels. Once the *Rate Limiter* started limiting specific malicious flows, then legitimate traffic to the IPs being attacked was allowed without restrictions. Next, we detail the auctions that occurred in this simulation.

5.4.2 Auctions

In our system, decision making occurs through auctions, and the preferences inputted at design time play an important role in choosing the winning bidder. Table 5.8 summarises the main auctions that happened in our simulation. It shows the values of the bid components ($\mathcal{T}$, $\mathcal{C}$ and $\mathcal{D}$) sent by agents, along with the final bid utility evaluated by the auctioneer using Equation 3.1. We exclude an auction that had no bids, which is detailed below, and the eight auctions to achieve goal *flowRateLimited(flow)* because Agent 5 had a *Rate Limiter* instantiated to fulfil goal *linkRateLimited(link)*, consequently winning the auctions for goals *ipRateLimited(ip)* and *flowRateLimited(flow)*.

The first auction of the simulation was started by the *Link Monitor* to achieve goal *linkRateLimited(link)*, which is associated with the following preferences: $pf_{\mathcal{T}} = 0.8$, $pf_{\mathcal{C}} = 0.1$, and $pf_{\mathcal{D}} = 0.1$. These preferences mean that a network operator decided, at design time, that quickly limiting the traffic on a potentially anomalous link is more important than saving computational and network resources as well as decentralising this job. Agent 5 won this auction because it sent the lowest bid. Note that for this agent, the bid component $\mathcal{T}$ is 0 as its NFVI-PoP and the VNF repository share the same switch. In our simulation, we only accounted for the bandwidth between switches for simplicity. As the preference of the bid component $\mathcal{T}$ associated with this goal is high ($pf_{\mathcal{T}} = 0.8$), the bid utility of Agent 5 was consequently the lowest. Moreover, observe that the bid components $\mathcal{C}$ and $\mathcal{D}$ of Agent 6 are 0. This is because no bandwidth of the network infrastructure would be used if it won the auction.

Next, the goal *?anomalousUsage(link)* was auctioned by the *Link Monitor*. The objective of this goal is to find whether the traffic on a given link is anomalous or not. Differently from the previous auction, saving computational and network resources as well as downloading and doing the job quickly have the same preference ($pf_{\mathcal{T}} = pf_{\mathcal{C}} = 0.5$), while decentralising has no importance ($pf_{\mathcal{D}} = 0.0$). Agent 6 refused to participate

Table 5.8: Costs of achieving the auctioned goals and bid utility.

linkRateLimited(link)					*?anomalousUsage(link)*				
Agent	$\mathcal{T}$	$\mathcal{C}$	$\mathcal{D}$	**Bid**	**Agent**	$\mathcal{T}$	$\mathcal{C}$	$\mathcal{D}$	**Bid**
1	50.00	16.88	25.84	0.50	1	50.00	17.22	26.43	0.50
2	55.56	26.91	39.63	0.54	2	55.56	27.48	41.94	0.78
3	100.00	16.87	33.12	0.88	3	100.00	21.84	34.92	0.87
4	100.00	4.88	18.01	0.87	4	100.00	11.84	17.48	0.63
5	0.00	5.65	13.22	0.09	5	0.00	6.55	12.64	0.00
6	62.50	0.00	0.00	0.60	6	-	-	-	-
ipRateLimited(ip)					*splitTrafficTowardsIp(ip)*				
Agent	$\mathcal{T}$	$\mathcal{C}$	$\mathcal{D}$	**Bid**	**Agent**	$\mathcal{T}$	$\mathcal{C}$	$\mathcal{D}$	**Bid**
1	50.00	2.80	8.44	0.49	1	50.00	2.80	9.07	0.24
2	55.56	21.84	12.34	0.62	2	55.56	21.84	18.06	0.87
3	100.00	20.72	43.10	0.89	3	100.00	20.72	49.21	0.96
4	100.00	8.73	31.03	0.87	4	100.00	8.73	33.38	0.58
5	0.00	0.00	0.00	0.10	5	0.00	0.00	0.00	0.00
6	62.5	1.09	12.34	0.58	6	62.5	1.09	14.52	0.22
First *flowRecord(ip)*					Second *flowRecord(ip)*				
Agent	$\mathcal{T}$	$\mathcal{C}$	$\mathcal{D}$	**Bid**	**Agent**	$\mathcal{T}$	$\mathcal{C}$	$\mathcal{D}$	**Bid**
1	50.00	0.40	11.04	0.80	1	50.00	0.40	13.65	0.80
2	55.56	19.62	13.06	0.92	2	55.56	19.62	19.14	0.85
3	100.00	24.61	52.69	0.20	3	-	-	-	-
4	100.00	14.61	39.15	0.38	4	100.00	12.23	42.57	0.15
5	-	-	-	-	5	-	-	-	-
6	-	-	-	-	6	-	-	-	-

in this auction, because its NFVI-PoP did not have enough resources to host an *Anomaly Detector* with 2 CPU cores, 4 GB of memory and 20 GB of disk.

After finding the IP address targeted by the attackers, the *Anomaly Detector* triggered to have the goal *ipRateLimited(ip)* fulfilled. Agent 5 had a *Rate Limiter* idle at that time, which was used to calculate the bid components. Since no VNF needed to be downloaded, and no network link would be required (*Anomaly Detector* VNF was hosted at the same NFVI-PoP), it sent the bid components $\mathcal{T} = \mathcal{C} = \mathcal{D} = 0.00$. Agent 5 won the auction, and the malicious traffic was reduced by 50%, reaching 2 MB/s.

The fourth negotiation in our simulation was to have goal *flowRecord(ip)* satisfied, triggered by the *Anomaly Detector*. Since the amount of traffic to be analysed was 2 MB/s, a *Classifier* with 4 CPUs, 8 GB of memory and 20 GB of disk is needed. However, agents refused to send bids as no NFVI-PoP had 8 GB of memory. Then, the *Anomaly Detector* added the goal *splitTrafficTowardsIp(ip)* to have a *Load Balancer* dividing the traffic into

two flows of 1 MB/s. Agent 5 also won this auction, completely exhausting its CPU cores and memory, and therefore preventing it from participating in future negotiations. Note that the bid component $\mathcal{C}$ is equal to the previous auction for all agents. This is because two reasons: (i) the VNFs selected for this and the previous auction have the same instantiation requirements; and (ii) both auctions were triggered by the same auctioneer. Therefore, the bidding heuristic was executed by participants using the same parameters in these auctions.

Next, the *Load Balancer* started two *flowRecord(ip)* auctions, one for each flow of 1 MB/s. Since in our system multiple auctions cannot coexist, the second *flowRecord(ip)* auction happened after the first was finished. Agent 1 bid in both auctions, sending the lowest value for the $\mathcal{C}$ component in the simulation, which is a consequence of its proximity to Agent 5. Nonetheless, it lost the auctions because the preference for decentralisation is high and the preference for saving computational and network resources is low ($pf_{\mathcal{D}}$ = 0.8 and $pf_{\mathcal{C}}$ = 0.2).

The high preference for decentralisation awarded Agent 3 with the first auction and Agent 4 with the second. Agent 3 refused to participate in the second auction because it did not have enough resources to host another *Classifier*. During the bidding heuristic execution of Agents 3 and 4, they evaluated that predicate $canDeploy(p, v) \vee \neg\exists v' canDeploy(p, v')$ from Equation 4.4 was false. This means that no other VNF could be instantiated after the allocation of the *Classifier* VNF. Therefore, the result of Equation 4.4 was 1, which is the harmonic mean of the ratio of available computational resources of their NFVI-PoP (function $freeRscUtil(p)$). Although allocating this VNF incurred in high computational and network cost, given the high preference of this goal for decentralisation, the bid utility evaluated by the auctioneer selected Agents 3 and 4 as the winners. The importance of the preferences inputted by a specialist is prominent in these auctions. Although the $\mathcal{C}$ component from Agents 2 and 3 was manyfold greater than Agent 1, considering the preferences, their bid utility was almost the same.

Finally, eight auctions were triggered to have goal *flowRateLimited(flow)* achieved, one for each malicious host found by one of the two *Classifiers* deployed in the network. As shown in Table 5.7, the preference for the time component is high ($pf_{\mathcal{T}} = 0.8$). Consequently, Agent 5, which had a *Rate Limiter* already deployed, won these auctions. Next, we analyse how our system behaves in larger scenarios.

5.4.3 Scalability Analysis

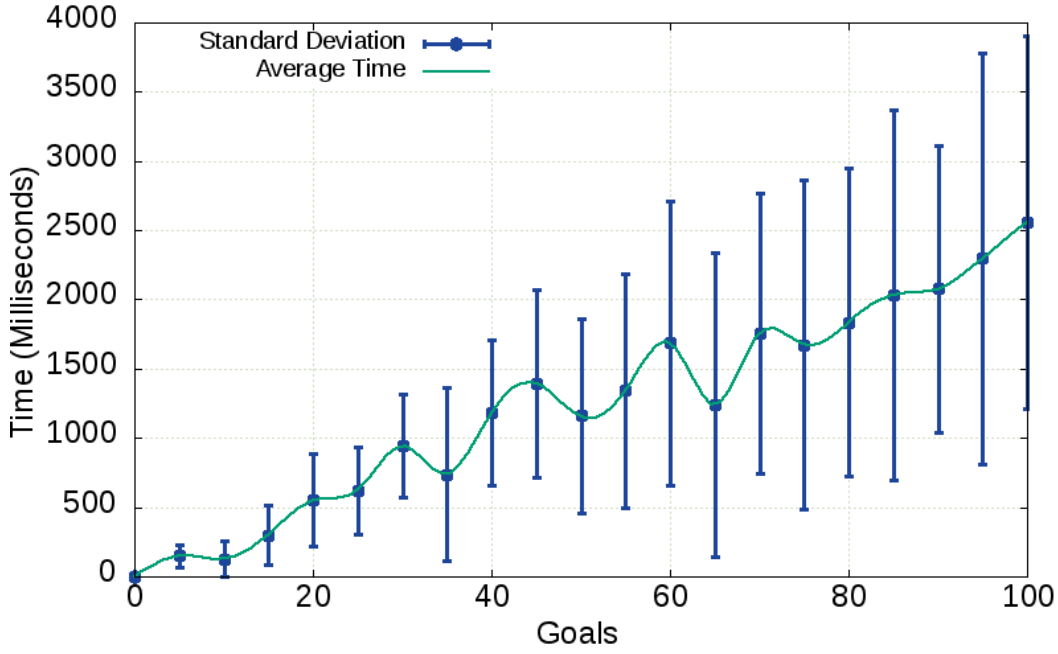
After detailing our implementation and experiment, we provide evidence despite the scalability of our decentralised orchestrator. We evaluated the scalability of our approach through two tests, whose details and results are given next.

In the first test, we evaluate the performance of the solution considering scenarios in which agents may have a large number of goals unrelated to the resilience strategy (*i.e.*, agents that perform multiple tasks). To do this, we used the scenario described in Section 5.3 and generated an increasing number of synthetic goals in the agent playing the *Rate Limiter* capability, which simulate the impact of real plans by taking a random amount of time to complete (between 0 and 100 milliseconds). We used 21 different configurations, where the number of goals started in 0, and 5 more were added in each new configuration so that the twenty-first had 100 synthetic goals. They were added to the agent along with the goal *linkRateLimited(link)*. The time measurement started right after the goals were added and ended when the throttling strategy was applied. Each configuration was executed 20 times.

Figure 5.5 shows how long the *Rate Limiter* took, on average, to apply the throttling strategy to the anomalous link, considering an increasing number of unrelated goals. The X-axis shows the increasing amount of synthetic goals, and the Y-axis represents time. Each blue vertical line shows the standard deviation of a configuration, while the circle point indicates the average measurement. A green line was added to connect all the average values.

In the second test, we evaluate two aspects of our approach with an increasing number of agents: (i) how much memory agents consume; and (ii) how many messages agents exchange in the NFV-A protocol. To comprehend how these aspects behave, we applied the DDoS resilience strategy described in Section 5.1 in a scenario where the resources of NFVI-PoPs and links bandwidth are unlimited. We used a ring topology so that we could easily automate the simulation to add more NFVI-PoPs to the loop. As we measure agent-related characteristics, and not their devised allocation of VNFs into NFVI-PoPs, changes in the network topology do not affect this test. An attacker is added to trigger the resilience strategy. This scenario was executed with 16 different configurations, where we only changed the number of agents. The first configuration starts with 6 agents, the same amount of the simulation presented in Section 5.3. Next, each configuration consisted of 6 more agents than the previous. Therefore, the sixteenth

Figure 5.5: Time taken by the *Rate Limiter* to apply the throttling strategy.

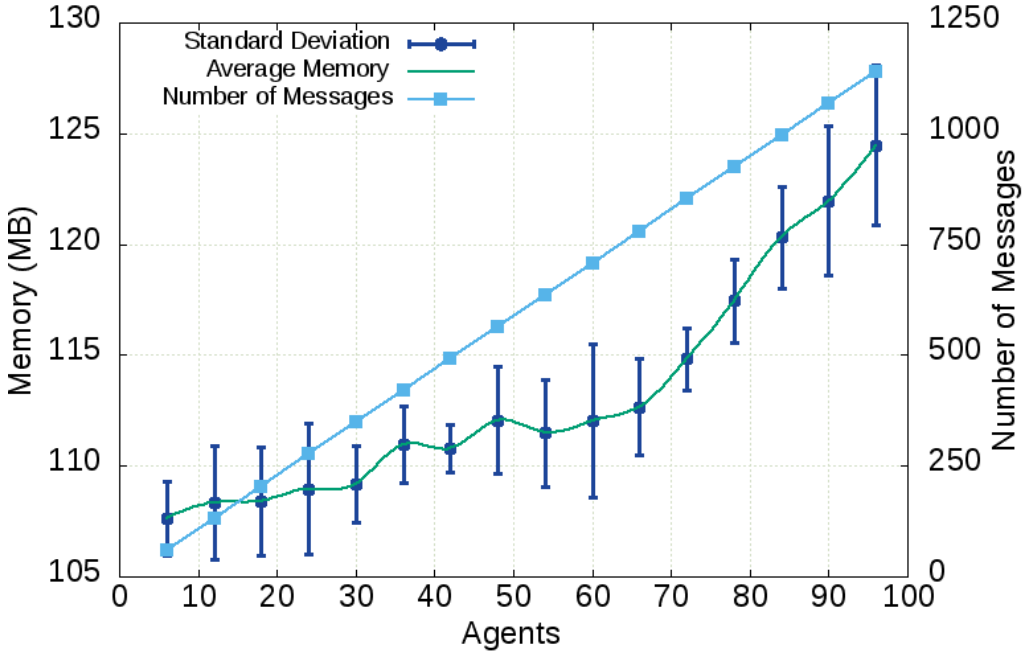


configuration consisted of 96 agents. Each configuration was simulated 20 times, where we collected the messages exchanged by the NFV-A protocol and memory consumption.

Figure 5.6 shows the average memory consumption and the number of messages exchanged in the scenario described above, using configurations with an increasing number of agents. The X-axis presents the number of agents, while the left-side Y-axis indicates the memory consumption and the right-side Y axis the number of messages. The blue line with square points shows that the number of messages exchanged in the NFV-A protocol grew linearly with the number of agents. The dark blue lines with a circle point represent the standard deviation of memory consumption of a given configuration. The circle point of these lines shows the average value for each configuration and are connected by a green line. Memory consumption only accounts for `BDI4JADE` agents, as we do not consider the memory used by `Mininet`.

The linear growth in both messages exchanged, memory usage and goal achievement time indicate that our system can scale to large networks. Note that we did not use goal filtering and prioritisation, which in a real scenario would be adopted to improve scalability, reducing the number of plans running at the same time.

Figure 5.6: Memory consumption and number of messages exchanged.



5.5 Final Remarks

In this chapter, we presented our testbed and evaluation of our system through a DDoS resilience strategy, where agents negotiated the allocation of VNFs dynamically, considering the preferences inputted by a network operator *a priori*. Then, we evaluated the emergent behaviour and auctions that occurred during the simulation. Next, we analysed the scalability of our solution by simulating agents with a large number of goals as well as the memory consumed by agents and the number of messages exchanged in our protocol. Finally, in next chapter, we analyse research efforts that also propose the creation of NFV orchestrator.

6 RELATED WORK

Recent research efforts have indicated that NFV orchestrators can be automated, relieving network operators from having to attack the placement and chaining problems (MIJUMBI et al., 2016). However, some key issues remain open, such as embedding the orchestrator with the knowledge to select VNFs to be deployed and removed (selection problem). In this chapter, we describe existing work that proposed NFV orchestrators.

6.1 Existing Solutions to the Automation of the Orchestration of VNFs

NFV has changed the way network functions are deployed and managed, posing new challenges to network operators. Management systems will have to cope with short-lived networks driven by per service demands, guaranteeing that all required functions are instantiated in an orderly and on-demand basis. This section describes previous work related to the automation of the orchestration of VNFs.

Giotis *et al.* (GIOTIS; KRYFTIS; MAGLARIS, 2015) proposed a policy-based approach, creating a centralised orchestrator based on Ponder2 (TWIDLE et al., 2008). They represent VNFs as an extension of Managed Objects (MO), allowing VNFs to interact with management policies and provide the desired NFV Service. The orchestrator is a collection of Event-Condition-Action (ECA) policies, authorisation and role assignment of resources. Differently from our work, however, ECA policies in (GIOTIS; KRYFTIS; MAGLARIS, 2015) have to be manually specified by a human operator, whereas in our work the behaviour of VNFs emerges through an automated reasoning process.

Focusing on security, Yasrebi *et al.* (YASREBI et al., 2015) proposed the orchestration of a Deep Packet Inspector (DPI) to detect and prevent DDoS attacks in SDN. Their approach is tested on the SAVI testbed (KANG; BANNAZADEH; LEON-GARCIA, 2013), which uses a centralised module aware of every network route. It uses this information to attach an Intrusion Detection System (IDS) where needed. Their system is capable of detecting, blocking and also redirecting the traffic to a honeypot for further investigation. Through the mobility provided by SDN, they are capable of implementing layers of security for different attacks, enabling the system to block attacks based on their severity in different parts of the network. In a different manner, we propose a decentralised approach where the interaction between autonomous components orchestrate the network behaviour.

With the objective of increasing energy efficiency, Donadio *et al.* (DONADIO et al., 2014) proposed a PCE-based (FARREL; VASSEUR; ASH, 2006) orchestrator, which is composed of a traffic analyser that controls computational resources and monitors log messages to perform orchestration in real-time. Their orchestrator can be executed in multiple physical hosts simultaneously, where each orchestrator manages its resources. Moreover, there is a centralised alternative in which one orchestrator is responsible for orchestrating resources in different network domains. Their work mainly focuses on minimising energy consumption of the IT infrastructure through the provision of orchestration algorithms that are oriented towards energy efficiency. Although Donadio *et al.* (DONADIO et al., 2014) proposed to use multiple orchestrators, they are still centralised components that control part of the virtualised network, while we propose that all virtualised components work together towards a decentralised NFV orchestrator.

Kuroki *et al.* (KUROKI; FUKUSHIMA; HAYASHI, 2015) focused on reducing the time of service of lifecycle management. They implemented an orchestrator that take as an input traditional service description (VNFs, virtual links, etc.) and an optimisation policy for each network service request. They use a set of algorithms that provide optimal results based on what is being optimised. Each network service request is submitted to an allocation algorithm based on its optimisation policy. By using algorithms that provide optimal resource allocation and open source network controllers to handle the low-level operations (create, remove and update VNFs), they were able to execute these operations in, on average, less than a second. Differently, instead of receiving as input the service requests, we attack the selection problem by embedding agents with knowledge that enables them to decide which VNFs are needed and build the VNF-FG dynamically.

Yoshida *et al.* (YOSHIDA et al., 2014) introduced an orchestrator that uses a multi-objective resource scheduling algorithm. They created a genetic algorithm that finds pareto-optimal solutions to the placement problem considering conflicting objectives. Similarly, preferences are given to their system *a priori* and used to determine the most suitable solutions. Although their system deals with requests dynamically, it only solves the placement problem (statically) while we also deal with selection and chaining.

6.2 Discussion

The NFV architecture separates network functions from hardware. This architecture decouples hardware from software, reducing network costs and ensuring that SLAs

Table 6.1: Comparison of related work.

Related Work	Decentralised	Selection	Placement	Chaining
Giotis *et al.*	x	x	✓	✓
Yasrebi *et al.*	x	x	✓	✓
Donadio *et al.*	✓	x	✓	✓
Kuroki *et al.*	x	x	✓	✓
Yoshida *et al.*	x	x	✓	x

are satisfied.

This overview introduced previous work whose aim was to provide the orchestration of VNFs. We summarise the related work discussed in the previous section in Table 6.1. From the literature review conducted, no work addresses the selection, placement and chaining problems altogether. All the work presented assume that the solution to the selection problem is given as an input to their orchestrators. So, there is still opportunities to create new approaches that, despite attacking the placement and chaining problems, can also relieve network operators from selecting VNFs to be added to the network in the face challenges.

Based on the analysis of existing work, we concluded that there are still opportunities to explore techniques to create intelligent NFV orchestrators. Differently from the related work discussed in this section, we proposed a decentralised NFV orchestrator in which the reasoning is embedded into BDI agents whose interactions and emerging behaviour attack the orchestrator problems.

6.3 Final Remarks

In this chapter, we presented other research efforts that also introduced NFV orchestrator, pointing out potential limitations and how our work address them. Next, we introduced a comparison between them, showing that there are opportunities for new approaches as there are open issues in this research field. Finally, in next chapter we conclude this dissertation detailing our contributions and opportunities for future work.

7 CONCLUSION AND FUTURE WORK

The NFV architecture provides an alternative to the cumbersome deployment, configuration and management of middleboxes, relieving network operators from vendor-specific technologies. This architecture leads to the challenges of selection, placement and chaining of VNFs, requiring solutions that attack these problems, ideally, at runtime and without human interference. Effectively solving these problems has the potential to improve the usage of resources, reducing CAPEX and OPEX.

Although different approaches have been proposed to attack the orchestration problem, previous work relies either on manual or on centralised approaches. Instead, in this dissertation, we proposed a distributed and decentralised NFV orchestrator based on BDI reasoning. We introduced an auction-based mechanism, in which agents bid on the allocation of VNFs. Cognitive agents are embedded with preferences from a domain specialist, allowing them to decide what action should be taken based on perceptions from the environment. While placing a bid, with our bidding heuristic, agents select a VNF to achieve a given goal and identify the path with the lowest cost to chain VNFs. The auction winner determines where to place the VNF. The emergent behaviour achieved by our approach is arguably more robust, since it can adapt to different situations and topologies, or in case of a component failure.

We evaluated our theoretical model through simulation in a testbed that integrates `BDI4JADE` and `Mininet`, using Docker containers to represent five distinct VNFs that work together to mitigate a DDoS attack. In our simulation, we explored the automatic orchestration of VNFs on a DDoS attack scenario and demonstrated that the traditional monolithic orchestrator can be implemented as distributed autonomous components.

7.1 Contributions

The main contributions of this dissertation are:

- **A decentralised NFV architecture.** In Chapter 3 we introduced our extension to the NFV architecture, replacing the centralised orchestrator by BDI agents. Each agent controls an NFVI-PoP, being able to perceive and make changes in the network through its VNFs. The proposed decentralised NFV architecture was published elsewhere (SCHARDONG; NUNES; SCHAEFFER-FILHO, 2017).

- **A bidding heuristic that addresses the selection, placement and chaining problems.** In Chapter 4, the NFV-A protocol, which specifies how agents start and respond to auctions, was detailed. The main component of this auction protocol is the bidding heuristic, which specifies how agents calculate their bids, considering the time to download a VNF from the repository, the cost of using a path to communicate with the auctioneer and a decentralisation metric. These are balanced through preferences, which allow network operators to fine-tune auctions based on their desires.
- **Implementation of the bidding heuristic.** In Chapter 5, we introduced a DDoS case study to evaluate the proposed decentralised orchestrator and NFV-A protocol. We implemented five VNFs and detailed their possible interactions. The simulations were conducted in our testbed, which is detailed next.
- **An integrated testbed.** In Chapter 5, we also detailed the integration between the agent platform BDI4JADE and the Mininet emulator to create a testbed that integrates BDI agents with our auction protocol to orchestrate VNFs. One can easily implement new VNFs and network topologies to test the emergent behaviour of our decentralised orchestrator. The reverse-auction heuristic, its implementation and the testbed were published elsewhere (SCHARDONG; NUNES; SCHAEFFER-FILHO, 2018).

7.2 Future Work

The contributions of this dissertation provide an initial step towards the consolidation of decentralised solutions to the orchestration problems. However, our work has limitations that can be explored by future work. These limitations are listed below.

- **Explore the emerging behaviour of agents.** In Chapter 5, we implemented five distinctive VNFs, each with a specific action that could be activated by the agent. However, in a DDoS attack, a network operator might have access to multiple middleboxes to classify or find anomalies in network traffic, each with its characteristics despite resource consumption, price and effectiveness in reaching its goal. Therefore, simulations with multiple VNFs that achieve the same objective, prompt agents with the task of selecting a VNF among many, allowing diverse and unforeseen solutions to emerge.

- **Evaluate the bidding heuristic in different network topologies and scenarios.** In Chapter 5, we tested our DDoS resiliency strategy in a small network topology composed of six NFVI-PoPs, six switches, one VNF repository and one application server. This topology does not reflect the complexity and size of the network of large corporations or university campuses. Likewise, scenarios involving, for instance, content delivery and telecom carriers pose different challenges. Therefore, larger network topologies and new scenarios should be used to analyse the emergent solution devised by agents through the bidding heuristic.
- **Address scenarios involving multiple providers.** Currently, we assume that agents belong to the same service provider. However, this assumption is unrealistic in other scenarios. For instance, in a VNF as a service (VNFaaS) market (HAWILO et al., 2014), agents from different providers have to trust each other to collaborate and receive rewards to perform some job from a different provider. Therefore, adding incentivization and reputation models would be a valuable contribution to our work.
- **Manage the lifecycle of VNFs.** The lifecycle management of VNFs consists of ensuring that VNFs provide their designed functionalities, allowing the instantiation, update, query, termination and also the scaling of their underlying resources. In our work, VNFs are scaled, instantiated and queried by their respective agents. However, agents can not terminate or update their VNFs, which limits the applicability of our solution in real life applications, because the resources of NFVI-PoPs are not released, thus fragmenting the network resources.

In summary, we introduce an extension to the NFV architecture, in which autonomous agents replace the centralised orchestrator. Agents use our NFV-A protocol and the bidding heuristic to address the selection, placement and chaining problems. We also implemented a testbed that integrates agents of BDI4JADE with a virtual network created in the Mininet emulator. Furthermore, we evaluated the emergent allocation of VNFs to mitigate a DDoS attack. With this work, we aim to popularise both multi-agent and NFV technologies, thus promoting the adoption of these technologies in the industry.

REFERENCES

ALAMEDDINE, H. A.; QU, L.; ASSI, C. Scheduling service function chains for ultra-low latency network services. In: IEEE. **2017 13th International Conference on Network and Service Management (CNSM)**. [S.l.], 2017. p. 1–9.

BAITICHE, H.; BOUZENADA, M.; SAÏDOUNI, D. E. Towards a generic predictive-based plan selection approach for bdi agents. **Procedia Computer Science**, Elsevier, v. 113, p. 41–48, 2017.

BOUET, M. et al. Cost-based placement of vdpi functions in nfv infrastructures. **International Journal of Network Management**, Wiley Online Library, v. 25, n. 6, p. 490–506, 2015.

BRATMAN, M. **Intention, plans, and practical reason**. [S.l.]: Cambridge, Mass., Harvard University Press, 1987.

BUSETTA, P. et al. Structuring bdi agents in functional clusters. In: SPRINGER. **International Workshop on Agent Theories, Architectures, and Languages**. [S.l.], 1999. p. 277–289.

CHAIB-DRAA, B.; DIGNUM, F. Trends in agent communication language. **Computational intelligence**, Wiley Online Library, v. 18, n. 2, p. 89–101, 2002.

CHI, P.-W.; HUANG, Y.-C.; LEI, C.-L. Efficient nfv deployment in data center networks. In: IEEE. **Communications (ICC), 2015 IEEE International Conference on**. [S.l.], 2015. p. 5290–5295.

CLAYMAN, S. et al. The dynamic placement of virtual network functions. In: IEEE. **Network Operations and Management Symposium (NOMS), 2014 IEEE**. [S.l.], 2014. p. 1–9.

COHEN, P. R.; LEVESQUE, H. J. **Rational interaction as the basis for communication**. [S.l.], 1988. 1–40 p.

DAVIS, R.; SMITH, R. G. Negotiation as a metaphor for distributed problem solving. **Artificial intelligence**, Elsevier, v. 20, n. 1, p. 63–109, 1983.

DONADIO, P. et al. A pce-based architecture for the management of virtualized infrastructures. In: IEEE. **Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on**. [S.l.], 2014. p. 223–228.

DURFEE, E. H.; LESSER, V. R.; CORKILL, D. D. Trends in cooperative distributed problem solving. **IEEE Transactions on knowledge and data Engineering**, IEEE, v. 1, n. 1, p. 63–83, 1989.

ETSI ISG NFV. **Network Functions Virtualisation (NFV): Architectural Framework**. 2013. Available from Internet: <http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01_60/gs_NFV002v010201p.pdf>.

ETSI ISG NFV. **ETSI GS NFV 003 V1.2.1: Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV**. 2014. Available from Internet: <http://www.etsi.org/deliver/etsi_gs/NFV/001_099/003/01.02.01_60/gs_NFV003v010201p.pdf>.

ETSI ISG NFV. **Network Functions Virtualisation (NFV); Management and Orchestration**. 2014. Available from Internet: <https://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf>.

ETSI ISG NFV. **Network Functions Virtualisation (NFV); Management and Orchestration**. 2014. Available from Internet: <https://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf>.

ETSI ISG NFV. **Network Functions Virtualisation (NFV); Management and Orchestration; Network Service Templates Specification**. 2016. Available from Internet: <http://www.etsi.org/deliver/etsi_gs/NFV-IFA/001_099/014/02.01.01_60/gs_NFV-IFA014v020101p.pdf>.

ETSI ISG NFV. **Network Functions Virtualisation (NFV); Management and Orchestration; VNF Packaging Specification**. 2016. Available from Internet: <http://www.etsi.org/deliver/etsi_gs/NFV-IFA/001_099/011/02.01.01_60/gs_NFV-IFA011v020101p.pdf>.

FACCIN, J.; NUNES, I. Bdi-agent plan selection based on prediction of plan outcomes. In: **2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)**. [S.l.: s.n.], 2015. v. 2, p. 166–173.

FACCIN, J.; NUNES, I. Bdi-agent plan selection based on prediction of plan outcomes. In: **Proceedings of the 2015 IEEE / WIC / ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT) - Volume 01**. Washington, DC, USA: IEEE Computer Society, 2015. (WI-IAT '15), p. 166–173. ISBN 978-1-4673-9618-9. Available from Internet: <http://dx.doi.org/10.1109/WI-IAT.2015.58>.

FARREL, A.; VASSEUR, J.-P.; ASH, J. **A path computation element (PCE)-based architecture**. 2006. <view-source:https://tools.ietf.org/html/rfc4655>. Accessed: 2017-09-28.

FININ, T. et al. Kqml as an agent communication language. In: ACM. **Proceedings of the third international conference on Information and knowledge management**. [S.l.], 1994. p. 456–463.

GENESERETH, M. R.; KETCHPEL, S. P. Software agents. **Commun. ACM**, v. 37, n. 7, p. 48–53, 1994.

GIOTIS, K.; KRYFTIS, Y.; MAGLARIS, V. Policy-based orchestration of nfv services in software-defined networks. In: IEEE. **Network Softwarization (NetSoft), 2015 1st IEEE Conference on**. [S.l.], 2015. p. 1–5.

GUDE, N. et al. Nox: towards an operating system for networks. **ACM SIGCOMM Computer Communication Review**, ACM, v. 38, n. 3, p. 105–110, 2008.

GUERZONI, R. et al. Network functions virtualisation: an introduction, benefits, enablers, challenges and call for action, introductory white paper. In: **SDN and OpenFlow World Congress**. [S.l.: s.n.], 2012. v. 1, p. 5–7.

HALPERN, J.; PIGNATARO, C. **Service function chaining (sfc) architecture**. [S.l.], 2015. Available from Internet: <https://www.rfc-editor.org/rfc/pdfrfc/rfc7665.txt.pdf>.

HAN, B. et al. Network function virtualization: Challenges and opportunities for innovations. **IEEE Communications Magazine**, IEEE, v. 53, n. 2, p. 90–97, 2015.

HAWILO, H. et al. Nfv: state of the art, challenges, and implementation in next generation mobile networks (vepc). **IEEE Network**, IEEE, v. 28, n. 6, p. 18–26, 2014.

HERRERA, J. G.; BOTERO, J. F. Resource allocation in nfv: A comprehensive survey. **IEEE Transactions on Network and Service Management**, IEEE, v. 13, n. 3, p. 518–532, 2016.

HUIN, N.; JAUMARD, B.; GIROIRE, F. Optimization of network service chain provisioning. In: IEEE. **Communications (ICC), 2017 IEEE International Conference on**. [S.l.], 2017. p. 1–7.

JIAO, S. et al. Joint virtual network function selection and traffic steering in telecom networks. In: IEEE. **GLOBECOM 2017-2017 IEEE Global Communications Conference**. [S.l.], 2017. p. 1–7.

KANG, J.-M.; BANNAZADEH, H.; LEON-GARCIA, A. Savi testbed: Control and management of converged virtual ict resources. In: IEEE. **Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on**. [S.l.], 2013. p. 664–667.

KONG, J. et al. Guaranteed-availability network function virtualization with network protection and vnf replication. In: IEEE. **GLOBECOM 2017-2017 IEEE Global Communications Conference**. [S.l.], 2017. p. 1–6.

KUROKI, K.; FUKUSHIMA, M.; HAYASHI, M. Framework of network service orchestrator for responsive service lifecycle management. In: IEEE. **2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)**. [S.l.], 2015. p. 960–965.

LABROU, Y.; FININ, T. Semantics for an agent communication language. In: SPRINGER. **International Workshop on Agent Theories, Architectures, and Languages**. [S.l.], 1997. p. 209–214.

LANGERMAN, J. J. **Agent-based models for the creation and management of airline schedules.** Thesis (PhD) — University of Johannesburg, 2005.

LANTZ, B.; HELLER, B.; MCKEOWN, N. A network in a laptop: rapid prototyping for software-defined networks. In: ACM. **Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks**. [S.l.], 2010. p. 19–25.

LEHMANN, D.; MÜLLER, R.; SANDHOLM, T. The winner determination problem. **Combinatorial auctions**, p. 297–318, 2006.

LUIZELLI, M. C. et al. Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions. In: IEEE. **2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)**. [S.l.], 2015. p. 98–106.

LUIZELLI, M. C. et al. The actual cost of software switching for nfv chaining. In: IEEE. **Integrated Network and Service Management (IM), 2017 IFIP/IEEE Symposium on**. [S.l.], 2017. p. 335–343.

MARTINS, J. et al. Clickos and the art of network function virtualization. In: USENIX ASSOCIATION. **Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation**. [S.l.], 2014. p. 459–473.

MEDHAT, A. M. et al. Near optimal service function path instantiation in a multi-datacenter environment. In: IEEE. **Network and Service Management (CNSM), 2015 11th International Conference on**. [S.l.], 2015. p. 336–341.

MERKEL, D. Docker: lightweight linux containers for consistent development and deployment. **Linux Journal**, Belltown Media, v. 2014, n. 239, p. 2, 2014.

MIJUMBI, R. et al. Network function virtualization: State-of-the-art and research challenges. **IEEE Communications Surveys & Tutorials**, IEEE, v. 18, n. 1, p. 236–262, 2016.

MOENS, H.; TURCK, F. D. Vnf-p: A model for efficient placement of virtualized network functions. In: IEEE. **10th International Conference on Network and Service Management (CNSM) and Workshop**. [S.l.], 2014. p. 418–423.

MÜLLER, C.; TIMMERER, C. A vlc media player plugin enabling dynamic adaptive streaming over http. In: ACM. **Proceedings of the 19th ACM international conference on Multimedia**. [S.l.], 2011. p. 723–726.

NUNES, I. Improving the design and modularity of bdi agents with capability relationships. In: SPRINGER. **International Workshop on Engineering Multi-Agent Systems**. [S.l.], 2014. p. 58–80.

NUNES, I.; LUCENA, C.; LUCK, M. Bdi4jade: a bdi layer on top of jade. In: **Proc. of the Workshop on Programming Multiagent Systems**. [S.l.: s.n.], 2011. p. 88–103.

NUNES, I.; SCHARDONG, F.; SCHAEFFER-FILHO, A. Bdi2dos: An application using collaborating bdi agents to combat ddos attacks. **J. Netw. Comput. Appl.**, Academic Press Ltd., London, UK, UK, v. 84, n. C, p. 14–24, abr. 2017. ISSN 1084-8045. Available from Internet: <https://doi.org/10.1016/j.jnca.2017.01.035>.

PEUSTER, M.; KARL, H.; ROSSEM, S. V. Medicine: Rapid prototyping of production-ready network services in multi-pop environments. In: IEEE. **Network Function Virtualization and Software Defined Networks (NFV-SDN), IEEE Conference on**. [S.l.], 2016. p. 148–153.

PFAFF, B. et al. Extending networking into the virtualization layer. In: **Hotnets**. [S.l.: s.n.], 2009. p. 1–6.

RAO, A. S.; GEORGEFF, M. P. et al. Bdi agents: From theory to practice. In: **ICMAS**. [S.l.: s.n.], 1995. v. 95, p. 312–319.

RICHARDSON, L.; RUBY, S. **RESTful web services**. [S.l.]: " O'Reilly Media, Inc.", 2008.

ROESCH, M. et al. Snort: Lightweight intrusion detection for networks. In: **LISA**. [S.l.: s.n.], 1999. v. 99, n. 1, p. 229–238.

SCAPY. 2007. <http://www.secdev.org/projects/scapy/>. Accessed: 2017-09-29.

SCHAEFFER-FILHO, A. E. et al. A framework for the design and evaluation of network resilience management. In: **NOMS**. [S.l.: s.n.], 2012. v. 2012, p. 401–408.

SCHARDONG, F.; NUNES, I.; SCHAEFFER-FILHO, A. A distributed nfv orchestrator based on bdi reasoning. In: IEEE. **Integrated Network Management (IM), 2017 IFIP/IEEE International Symposium on**. [S.l.], 2017. p. 107–115.

SCHARDONG, F.; NUNES, I.; SCHAEFFER-FILHO, A. Providing cognitive components with a bidding heuristic for emergent nfv orchestration. In: IEEE. **Network Operations and Management Symposium (NOMS), 2018 IEEE/IFIP**. [S.l.], 2018. p. 1–9.

SILVA, A. S. da et al. Atlantic: A framework for anomaly traffic detection, classification, and mitigation in sdn. In: IEEE. **NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium**. [S.l.], 2016. p. 27–35.

SINGH, D. et al. Learning context conditions for bdi plan selection. In: INTERNATIONAL FOUNDATION FOR AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS. **Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1**. [S.l.], 2010. p. 325–332.

SMITH, R. G. The contract net protocol: High-level communication and control in a distributed problem solver. **IEEE Transactions on computers**, IEEE, n. 12, p. 1104–1113, 1980.

SMITH, R. G.; DAVIS, R. Frameworks for cooperation in distributed problem solving. **IEEE Transactions on systems, man, and cybernetics**, IEEE, v. 11, n. 1, p. 61–70, 1981.

STERBENZ, J. P. et al. Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines. **Computer Networks**, Elsevier, v. 54, n. 8, p. 1245–1265, 2010.

SUN, Q. et al. Forecast-assisted nfv service chain deployment based on affiliation-aware vnf placement. In: IEEE. **Global Communications Conference (GLOBECOM), 2016 IEEE**. [S.l.], 2016. p. 1–6.

THAI, M.-T.; LIN, Y.-D.; LAI, Y.-C. A joint network and server load balancing algorithm for chaining virtualized network functions. In: IEEE. **Communications (ICC), 2016 IEEE International Conference on**. [S.l.], 2016. p. 1–6.

TWIDLE, K. et al. Ponder2-a policy environment for autonomous pervasive systems. In: IEEE. **Policies for Distributed Systems and Networks, 2008. POLICY 2008. IEEE Workshop on**. [S.l.], 2008. p. 245–246.

VERBRUGGE, S. et al. Methodology and input availability parameters for calculating opex and capex costs for realistic network scenarios. **Journal of Optical Networking**, Optical Society of America, v. 5, n. 6, p. 509–520, 2006.

WOOLDRIDGE, M. **An introduction to multiagent systems**. [S.l.]: John Wiley & Sons, 2009.

WOOLDRIDGE, M.; JENNINGS, N. R. Intelligent agents: Theory and practice. **The knowledge engineering review**, Cambridge Univ Press, v. 10, n. 02, p. 115–152, 1995.

XIA, M. et al. Network function placement for nfv chaining in packet/optical datacenters. **Journal of Lightwave Technology**, IEEE, v. 33, n. 8, p. 1565–1570, 2015.

YASREBI, P. et al. Security function virtualization in software defined infrastructure. In: IEEE. **Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on**. [S.l.], 2015. p. 778–781.

YOSHIDA, M. et al. Morsa: A multi-objective resource scheduling algorithm for nfv infrastructure. In: IEEE. **Network Operations and Management Symposium (APNOMS), 2014 16th Asia-Pacific**. [S.l.], 2014. p. 1–6.

ZHANG, S. Q. et al. Joint nfv placement and routing for multicast service on sdn. In: IEEE. **Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP**. [S.l.], 2016. p. 333–341.

APPENDIX A — RESUMO ESTENDIDO

Com o passar dos anos, a complexidade e o tamanho das redes aumentaram drasticamente, assim como a necessidade de um gerenciamento mais flexível. Os equipamentos físicos (também conhecidos como *middleboxes*) são produtos altamente especializados e proprietários que requerem encadeamento específico, instalação física e suas funcionalidades não podem ser facilmente alteradas. Além disso, a crescente demanda por redes mais diversificadas e de curta duração para lidar com altas taxas de dados, como infraestrutura-como-serviço e rede-como-serviço, exige que as operadoras de redes implantem rapidamente e operem equipamentos de rede complexos, levando a altas despesas de capital e gastos operacionais.

Para resolver esses problemas, a virtualização de funções de rede (NFV) foi proposta como uma forma de dissociar serviços de rede de dispositivos físicos através da virtualização. Há um amplo conjunto de serviços tradicionalmente executados pelas *middleboxes* — como firewalls, sistemas de detecção de intrusões, sistemas de prevenção de intrusões, modelagem de tráfego, conversão de endereços de rede, aceleradores de tráfego, caches e proxies — que podem ser virtualizados em funções de rede virtual (VNFs) baratas e fáceis de implementar. Diferentemente das *middleboxes*, múltiplas funções de rede virtualizadas podem compartilhar uma única máquina física, permitindo o uso de poder computacional, que de outra forma seria perdido, em *middleboxes* proprietárias. A evolução das redes em funções e serviços baseados em software são passos concretos para redes do futuro.

No entanto, as VNFs precisam ser gerenciados e compostas de maneira significativa, de modo que as funcionalidades desejadas sejam alcançadas. Esse processo é chamado de orquestração NFV. A orquestração NFV pode ser decomposta em três problemas principais: (i) *seleção* automática de VNFs; (ii) VNF *colocação* na rede virtualizada; e (iii) *encadeamento* de VNFs. Normalmente, os humanos projetam um grafo de encaminhamento de rede de VNFs, isto é, decidem como as VNFs são encadeadas. No entanto, à medida que a complexidade da rede aumenta e os requisitos de acordo de nível de serviço sobre as redes sob demanda se tornam mais rigorosos, garantir a orquestração de nós virtuais em tempo real torna-se vital para operadoras e provedores de serviços. No entanto, os esquemas de orquestração da NFV propostos até o momento não exploram os benefícios de componentes autônomos, portanto dependem de humanos para impor os SLAs, o que geralmente é impraticável.

Soluções centralizadas para os problemas de seleção, colocação e encadeamento foram propostas. No entanto, eles têm desvantagens naturais, como por exemplo, intolerância a falhas, pois um único componente controla a alocação de recursos na rede e pode exigir a interrupção completa devido a alterações de hardware ou de configurações. Em uma solução descentralizada, se um componente do sistema falhar, os outros percebem e assumem suas funções, mantendo assim a disponibilidade geral. Uma das consequências dessa característica é que as diferentes peças que compõem um sistema descentralizado podem ser atualizadas/alteradas sem afetar o sistema. Além disso, os sistemas descentralizados podem facilmente ser redimensionados para acomodar mudanças no tamanho do que eles estão tentando resolver. Com base nessas questões, a questão de pesquisa que norteia essa dissertação é: *Como abordar os problemas de seleção, colocação e encadeamento usando uma abordagem descentralizada?*

Buscando responder a esta questão, estendemos a arquitetura NFV substituindo seu orquestrador centralizado por componentes cognitivos estruturados com a arquitetura belief-desire-intention (BDI). A arquitetura BDI inclui um ciclo de raciocínio que fornece aos agentes um comportamento racional, permitindo que lidem com diferentes cenários nos quais o comportamento flexível e inteligente é necessário. Para desenvolver um orquestrador descentralizado, contamos com o comportamento que emerge da interação de agentes BDI autônomos. Utilizamos a arquitetura BDI, pois permite a criação de sistemas descentralizados flexíveis e robustos, capazes de atuar de forma reativa e proativa, a fim de resolver problemas dinâmicos e complexos.

Em nosso orquestrador descentralizado, cada agente controla um servidor capaz de receber VNFs (NFVI-PoP) e é responsável pelo gerenciamento de VNFs. Para isso, os agentes são capazes de monitorar os recursos do seu NFVI-PoP e da rede para tomar decisões de gerenciamento. Cada agente está ciente das conexões de seu NFVI-PoP, bem como das especificações de hardware.

Os agentes negociam para escolher os serviços (VNFs) pelos quais são responsáveis. A negociação é realizada por meio de leilões. Em nossa solução, os agentes gerenciam suas VNFs e coletivamente decidem qual agente instanciará uma nova VNF. Como um objetivo pode ser alcançado por diferentes VNFs, o leilão é um meio de selecionar qual VNF atingirá uma meta (problema de seleção). Além disso, dado que os lances podem ser colocados por agentes representando diferentes NFVI-PoPs usando o mesmo VNF, o leilão também é um meio de selecionar a localização de um VNF (problema de colocação). Finalmente, nossa solução assume que os agentes participantes avaliam o

caminho entre eles e o leiloeiro ao fazer um lance (problema de encadeamento).

Parte do processo de leilão consiste em calcular o lance de um agente. Isso é realizado pela função de lance, que seleciona uma VNF para trabalhar no que está sendo leiloado e resulta em três valores que são enviados para o agente leiloeiro escolher o vencedor. O leiloeiro utiliza as preferências a respeito dos três componentes, que são informadas pelo operador da rede em tempo de design, para determinar qual é o lance de menor custo que mais satisfaz as suas preferências.

Quando um agente recebe uma mensagem indicando o início de um leilão, ele realiza as seguintes tarefas: (i) avalia se o NFVI-PoP atende aos requisitos computacionais da VNF; e (ii) calcula cada componente da proposta. Durante estes passos, o agente avalia as restrições de rede impostas pelo leiloeiro. Se qualquer restrição não for satisfeita, o agente informa ao leiloeiro que ele se recusa a participar do leilão. Caso contrário, envia seu lance.

Para instanciar uma VNF em um NFVI-PoP, o NFVI-PoP deve ter recursos computacionais disponíveis que atendam aos requisitos da VNF. Por exemplo, suponha que um NFVI-PoP tenha os seguintes recursos disponíveis: (i) 8 núcleos de CPU; (ii) 16 GB de RAM; e (iii) 500 GB de disco. Considere os requisitos de uma VNF a ser instanciada: (i) 2 núcleos de CPU; (ii) 4 GB de RAM; e (iii) 50 GB de disco. Como todos os recursos necessários para instanciar a VNF estão disponíveis no NFVI-PoP, então ele participará do leilão.

Cada lance tem três componentes: (i) tempo de download da VNF; (ii) ao custo computacional e de rede; e (iii) descentralização.

O tempo de download da VNF é: (i) 0, se o VNF já estiver implantado e não precisar ser baixado; ou (ii) o tempo estimado para baixar, do repositório VNF, a VNF que pode executar o plano selecionado para atingir o objetivo leiloado. O tempo estimado considera o caminho com a maior largura de banda disponível para baixar a VNF no menor tempo possível.

O custo computacional e de rede é o principal componente do leilão. Ele avalia a quantidade de recursos computacionais e de rede empregados para instanciar uma VNF, bem como avalia se a instanciação de uma VNF em um determinado NFVI-PoP deixaria os recursos inutilizáveis, evitando assim a instanciação de outras VNFs. Além disso, ao calcular esse componente de licitação, o algoritmo: (i) verifica se as restrições de objetivos foram atendidas; e (ii) escolhe o caminho entre o leiloeiro e o agente licitante a ser utilizado, caso o licitante vença o leilão (problema de encadeamento). A ideia princi-

pal é encontrar o caminho com o menor custo entre o leiloeiro e o licitante. Para fazer isso, representamos a rede como um grafo, no qual os nós são conectados por links com pesos, capturando a quantidade de recursos que estão sendo consumidos devido ao uso desse link. A identificação do caminho com o menor custo é feita usando uma versão customizada do amplamente conhecido algoritmo de Dijkstra, que encontra o caminho mais curto em um grafo em tempo polinomial.

O componente de descentralização corresponde a quão fisicamente distante o leiloeiro e o licitante estão. Dado que nossas informações de entrada não capturam informações geográficas sobre NFVI-PoPs, usamos a latência para estimar a descentralização, assumindo que os NFVI-PoPs geograficamente distantes tendem a ter maior latência do que os NFVI-PoPs que estão geograficamente mais próximos. Este componente é, portanto, a latência acumulada do caminho que liga o leiloeiro e o licitante, obtido a partir do algoritmo de Dijkstra modificado.

Para testar empiricamente a solução, projetamos cinco VNFs que representam funcionalidades de rede comumente usadas e observamos seu comportamento coletivo emergindo em uma estratégia de resiliência significativa para atenuar um ataque de negação de serviço distribuido (DDoS), são eles: (i) Link Monitor: responsável por detectar qualquer link usado em excesso na rede; (ii) Rate Limiter: executa a limitação de tráfego para um link específico, para um endereço IP específico e para fluxos específicos (onde a origem e o destino são conhecidos); (iii) Anomaly Detector: detecta o alvo de um ataque de DDoS; (iv) Classifier: analisa e identifica fluxos maliciosos, entre todo o tráfego para um host específico; e (v) Load Balancer: permite que os agentes dividam o tráfego em dois fluxos.

Para avaliar o sistema proposto e a heurística de cálculo de lance do leilão, desenvolvemos um testbed que integra uma plataforma de agente BDI, chamada BDI4JADE, com o Containernet, que é uma extensão do Mininet onde hosts Mininet são implementados como containers Docker. A comunicação agente-para-VNF é implementada usando chamadas RESTful. Fornecemos as VNFs um script Python que envia e recebe chamadas RESTful, permitindo que os agentes obtenham facilmente informações específicas das VNFs.

No início da simulação, um arquivo de configuração especificando a topologia da rede (especificação de recursos de NFVI-PoPs assim como seus links e localização do repositório VNF) é lido pela plataforma BDI4JADE, que lança os agentes BDI e uma aplicação Python responsável por iniciar Mininet. Eles trocam informações por meio

de chamadas RESTful e coordenam a configuração da rede e a atribuição do agente ao NFVI-PoP. Como resultado, os agentes estão no controle de NFVI-PoPs e VNFs pré-configuradas, que são aqueles presentes no estado inicial da rede, implementados em agentes especificados de acordo com o arquivo de configuração de topologia.

Para avaliar nossa solução, criamos um servidor que responde a solicitações HTTP por meio do módulo SimpleHTTPServer do Python versão 2.7.14 e fornece um serviço transmissão de vídeo por meio do VLC versão 2.2.2. Nossa topologia de rede é composta por seis NFVI-PoPs. O repositório de VNFs está conectado ao quinto NFVI-PoP. O servidor HTTP e transmissão de vídeo, bem como o NFVI-PoP 6, são conectados diretamente ao mesmo switch. Um Link Monitor é instalado no NFVI-PoP 6, monitorando o link que conecta o switch ao servidor de HTTP e transmissão de vídeo. O tráfego da simulação consistiu em 50 hosts que solicitavam assistir a transmissão de vídeo em um intervalo de tempo, depois pararam e fizeram solicitações HTTP. Para cada host que solicita tráfego de streaming de vídeo, existem outros cinco hosts que solicitam tráfego HTTP.

Após alguns segundos de simulação, o ataque começou e o tráfego de saída para o servidor atingiu 6 MB/s. Imediatamente, o Link Monitor detectou o pico, acionando o leilão para atingir o objetivo linkRateLimited(link), que desencadeou uma série de leilões que mitigaram o ataque DDoS com sucesso.

Para avaliar a escalabilidade da solução multi-agente proposta, dois testes foram executados. No primeiro teste, avaliamos o tempo para executar atingir o objetivo considerando cenários nos quais os agentes podem ter um grande número de objetivos não relacionadas a estratégia de resiliência (como agentes que executam várias tarefas). No segundo teste, avaliamos dois aspectos de nossa abordagem com um número crescente de agentes: (i) quanta memória RAM os agentes consomem; e (ii) quantas mensagens os agentes trocam no protocolo de leilão. O crescimento linear em ambas as mensagens trocadas, uso de memória e tempo de realização dos objetivos indicam que nosso sistema pode escalar para grandes redes.

As principais contribuições desta dissertação são:

- **Uma arquitetura NFV descentralizada.** Introduzimos nossa extensão para a arquitetura NFV, substituindo o orquestrador centralizado por agentes autônomos. Cada agente controla um NFVI-PoP e suas conexões, sendo capaz de perceber e fazer mudanças na rede através de VNFs.
- **Uma heurística de lances que trata dos problemas de seleção, colocação e encadeamento.** O protocolo de leilão, que especifica como os agentes iniciam e re-

spondem aos leilões, foi detalhado. O principal componente deste protocolo de leilão é a heurística de lances, que especifica como os agentes calculam seus lances, considerando o tempo para baixar uma VNF do repositório, o custo de usar um caminho para se comunicar com o leiloeiro e uma métrica de descentralização. Eles são equilibrados por meio de preferências, que permitem que os operadores de rede ajustem os leilões com base em seus desejos.

- **Implementação da heurística de licitação.** Introduzimos um estudo de caso de DDoS para avaliar o orquestrador descentralizado proposto e o protocolo de leilão. Implementamos cinco VNFs e detalhamos suas possíveis interações. As simulações foram realizadas em nosso testbed, que é detalhado a seguir.
- **Um testbed integrado.** A integração entre a plataforma do agente BDI4JADE e o emulador Mininet para criar um testbed que integra agentes BDI com nosso protocolo de leilão para orquestrar VNFs. Pode-se facilmente implementar novos VNFs e topologias de rede para testar o comportamento emergente de nosso orquestrador descentralizado.

Em resumo, introduzimos uma extensão à arquitetura NFV, na qual agentes autônomos substituem o orquestrador centralizado. Os agentes usam nosso protocolo de leilão e a heurística de lances para resolver os problemas de seleção, colocação e encadeamento. Também implementamos um testbed que integra agentes do BDI4JADE com uma rede virtual criada no emulador Mininet. Além disso, avaliamos a alocação emergente de VNFs para mitigar um ataque DDoS. Com este trabalho, pretendemos popularizar as tecnologias multi-agente e NFV, promovendo assim a adoção dessas tecnologias na indústria.