



**HAL**  
open science

# Multidomain virtual network embedding under security-oriented requirements applied to 5G network slices

François Boutigny

► **To cite this version:**

François Boutigny. Multidomain virtual network embedding under security-oriented requirements applied to 5G network slices. Networking and Internet Architecture [cs.NI]. Institut Polytechnique de Paris, 2019. English. NNT : 2019IPPAS002 . tel-02469417

**HAL Id: tel-02469417**

**<https://theses.hal.science/tel-02469417v1>**

Submitted on 6 Feb 2020

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



INSTITUT  
POLYTECHNIQUE  
DE PARIS

NNT : 2019IPPAS002

Thèse de doctorat

TELECOM  
SudParis



IP PARIS

# Multidomain Virtual Network Embedding under Security-oriented Requirements applied to 5G Network Slices

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

École doctorale n°626 Institut Polytechnique de Paris (IPP)  
Spécialité de doctorat : Informatique, données, IA

Thèse présentée et soutenue à Évry-Courcouronnes, le 8 novembre 2019, par

**FRANÇOIS BOUTIGNY**

Composition du Jury :

|   |                    |
|---|--------------------|
| Guénaël Renault<br>Professeur, École Polytechnique                      | Président          |
| Yves Roudier<br>Professeur, Université Nice Sophia Antipolis            | Rapporteur         |
| Michaël Hauspie<br>Maître de conférences HDR, Université de Lille       | Rapporteur         |
| Christine Morin<br>Directrice de recherche, Inria                       | Examinatrice       |
| Isabelle Chrisment<br>Professeur, Télécom Nancy, Université de Lorraine | Examinatrice       |
| Nuno Neves<br>Professeur, Universidade de Lisboa                        | Examineur          |
| Samuel Dubus<br>Ingénieur de Recherche, Nokia Bell Labs                 | Examineur          |
| Hervé Debar<br>Professeur, Télécom SudParis                             | Directeur de thèse |
| Gregory Blanc<br>Maître de conférences, Télécom SudParis                | Invité             |
| Antoine Lavignotte<br>Maître de conférences, Télécom SudParis           | Invité             |

---

---

# Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction</b>   | <b>1</b>  |
| <b>2</b> | <b>State of the Art</b>                                     | <b>3</b>  |
| 2.1      | The Virtual Network Embedding (VNE) Problem . . . . .       | 4         |
| 2.2      | Extensions to the VNE Problem . . . . .                     | 5         |
| 2.2.1    | Rethinking the Objective Function . . . . .                 | 8         |
| 2.2.2    | Rethinking the Substrate . . . . .                          | 11        |
| 2.2.3    | Rethinking the Requests . . . . .                           | 13        |
| 2.2.4    | Leveraging Software-Defined Network (SDN) . . . . .         | 17        |
| 2.2.5    | Leveraging Network Function Virtualization (NFV) . . . . .  | 18        |
| 2.3      | Security Aspects for the VNE Problem . . . . .              | 20        |
| 2.3.1    | Security for the Tenants . . . . .                          | 20        |
| 2.3.2    | Security for the Infrastructure Providers (InPs) . . . . .  | 23        |
| 2.4      | Methods for Modeling the VNE Problem . . . . .              | 25        |
| 2.4.1    | The Integer Linear Programming (ILP) Model . . . . .        | 28        |
| 2.4.2    | The Boolean Satisfiability (SAT) Model . . . . .            | 31        |
| 2.4.3    | The Satisfiability Modulo Theories (SMT) Modeling . . . . . | 33        |
| 2.5      | Conclusion . . . . .  | 34        |
| <b>3</b> | <b>Security Constraint Model</b>                            | <b>37</b> |
| 3.1      | The Attribute Model . . . . .                               | 38        |
| 3.1.1    | Attribute Classification . . . . .                          | 38        |
| 3.1.2    | Exclusion Attributes: Semantics and Typology . . . . .      | 40        |
| 3.1.3    | Tolerant Attributes: Semantics and Typology . . . . .       | 43        |
| 3.1.4    | Tolerant Attributes: Structure . . . . .                    | 44        |
| 3.1.5    | Node and Link Attributes: Semantics and Typology . . . . .  | 47        |
| 3.2      | Application to the Single Domain Scenario . . . . .         | 48        |
| 3.2.1    | VNE Description . . . . .                                   | 48        |
| 3.2.2    | SMT Formulation . . . . .                                   | 51        |
| 3.3      | Model Refinement . . . . .                                  | 56        |

|          |  |            |
|----------|--|------------|
| 3.3.1    | The Theory of Non-Interpreted Function . . . . .                             | 57         |
| 3.3.2    | The Theory of Arrays . . . . .   | 60         |
| 3.3.3    | Conditional Requirements . . . . .   | 62         |
| 3.3.4    | Attribute-specific Exclusion and Collocation . . . . .                       | 70         |
| 3.4      | Conclusion . . . . .   | 75         |
| <b>4</b> | <b>Algorithm Resolution</b>  | <b>77</b>  |
| 4.1      | From the Single Domain to the Multi-Domain Formulation . . . . .             | 78         |
| 4.1.1    | SMT Formulation for the Intra-Domain Level . . . . .                         | 78         |
| 4.1.2    | SMT Formulation for the Inter-Domain Level . . . . .                         | 83         |
| 4.2      | Multi-level Resolution Algorithm . . . . .                                   | 84         |
| 4.2.1    | Architecture . . . . .   | 85         |
| 4.2.2    | Solution Enumeration . . . . .   | 85         |
| 4.2.3    | Inter-level Coordination . . . . .   | 89         |
| 4.2.4    | Solution Selection . . . . .   | 90         |
| 4.3      | Conclusion . . . . .   | 91         |
| <b>5</b> | <b>Prototyping and Evaluation</b>  | <b>93</b>  |
| 5.1      | Prototype Description and Use Case . . . . .                                 | 93         |
| 5.1.1    | Prototype Description . . . . .  | 94         |
| 5.1.2    | Use Case . . . . .   | 97         |
| 5.2      | Implementation Challenges for Mutable Attributes . . . . .                   | 99         |
| 5.3      | Results and Interpretation . . . . .   | 103        |
| 5.4      | Instantiation of the State-of-the-Art Security-oriented Attributes . . . . . | 107        |
| 5.5      | Conclusion . . . . .   | 114        |
| <b>6</b> | <b>Conclusion and Future Tracks</b>  | <b>117</b> |
| 6.1      | Conclusion . . . . .   | 117        |
| 6.2      | Cumulative Link Attributes . . . . .   | 118        |
| 6.3      | Time-dependent Requirement . . . . .   | 118        |
|          | <b>Bibliography</b>  | <b>121</b> |
|          | <b>A Chapter 5 Appendices</b>  | <b>129</b> |
|          | <b>Acronyms</b>  | <b>135</b> |
|          | <b>Table of all Notations</b>  | <b>137</b> |
|          | <b>List of Figures</b>   | <b>147</b> |
|          | <b>List of Tables</b>  | <b>149</b> |

# CHAPTER 1

---

## Introduction

From 1G to 4G, the technology mainly focused on increasing the bandwidth capacity of phones, to carry voice, message and Internet data. 5G has new goals: providing low latency, bandwidth, and increased support for diverse connected devices.

Network operators indeed anticipate an ever-growing number of more diverse connected devices, be them vehicles on roads, traffic lights in cities, robots in factories, or healthcare devices in human bodies. However, the companies producing those devices and operating them know little about network operators' business. For instance, they are not expected to own nor manage mobile network equipment.

Fortunately, the MVNO business model shows that this business knowledge is not mandatory. The MVNOs already provide services similar to other network operators, while owning no or a few equipment. They rent them instead, after negotiating specific agreements on specific resources. 5G proposes to generalize the MVNO business model so that the aforementioned companies can operate their devices on top of network operators' infrastructures, as if they had their own.

In the end, they get a dedicated, customized, virtual, end-to-end network on top of a shared infrastructure, which is called a "slice". Companies renting slices are called "tenants". And the shared infrastructure does not only rely on network operators, but more broadly on a subset of all possible Infrastructure Providers (InPs).

Each InP will then have to support multi-tenancy with a huge variety of devices (IoT, V2V, Industry 4.0) which will *autonomously* communicate, along with human-oriented devices.

In this thesis, we focus on slice customization. It is crucial: as tenants know little about network operators' business, network operators know little about tenants' business. Tenants have their own requirements to make their services operational, or regulation-compliant. For instance, robots need low latency, while a remote radiography needs high bandwidth.

We are already used to define such QoS-oriented requirements, and to pay for it, in the cloud model. But 5G invites us to go beyond, by considering more general requirements, like security requirements. For instance, to comply with a certain regulation in a certain country, tenants may require that their slices must rent resources in certain other countries, from certain InPs, with a certain level of certification.

There is an obstacle, though. As of today, MVNOs get their infrastructures through manually configured resources. With 5G and thousands of tenants, such configuration will be unpractical. We then need to automate slice deployment. In this thesis, we present how we solve this problem.

We borrow from the literature the concept of VNE, which refers to the allocation of resources from the physical network to fulfill requirements of a virtual network request. However, Fischer et al. (2013) show that the VNE problem is *NP*-hard. Besides, we identify two aspects of the slice embedding problem that previous works in VNE only covered partially: the multi-InP aspect, and the security requirement aspect. The algorithm we propose in this thesis successfully covers both.

Throughout this thesis, the term domain refers to the infrastructure owned by a certain InP. A given InP may own different domains, which are not directly interconnected together. When we use the term multi-domain, we mean that there are both multiple InPs and multiple domains operated by different InPs.

This thesis is organized as follows. Chapter 2 presents the state of the art works addressing the same problem, or a related problem, to the one of this thesis. Chapter 3 presents our attribute model for enabling security-oriented constraints. Chapter 4 presents the algorithm leveraging our model to solve the problem. Chapter 5 presents how we implement and evaluate our algorithm. Chapter 6 concludes this thesis and provides possible future tracks.

## CHAPTER 2

---

### State of the Art

While applying for a job position at Nokia Bell Labs, one of my interviewers recommended me to read the transcription of a talk given by Richard Hamming, “You and Your Research”. In this talk, Richard Hamming gave many advice about how to do significant research, based on his experience, as well as the ones of many other well-known scientists.

As I am writing this thesis, I recall one of his ground ideas, which is that science is cumulative. It is true that I took many inspiration from the works I will describe in this chapter. Like an investigator, although heavily assisted by modern research tools, I could gather information about the core problem of this thesis, which is the Virtual Network Embedding (VNE) problem. I call it so, because it is its recent name; but it covers so many fields that I could find the same problem with other names, such as network testbed mapping problem, or resource allocation problem. I will then, only to the best of my knowledge, presents the origins of this problem in Section 2.1.

By the light of this core description, we will then explore different variants of the VNE problem in Section 2.2. It will not be exhaustive; the works presented there have been selected because they each refine the VNE problem in their own way, and we will classify them according to some key aspects. This will allow us to introduce what we learned and how we differ from these works.

One aspect is of particular interest for this thesis, the security aspect, which is investigated on its own in Section 2.3. Although we use the singular, again, the works focusing on security for the VNE problem each addresses the security aspect in its own way.

Yet, VNE works do share some similarities in how they solve the VNE problem. This resolution method is analyzed in Section 2.4, where we position our



distinct resolution method, and explain why it is relevant and more appropriate to address the security aspect.

## 2.1 The VNE Problem

The VNE problem is a relatively old problem in computer science. To the best of my knowledge, its oldest mention is by Ricci et al. (2003). It is then referred to as the *network testbed mapping problem*. The VNE problem can also be found in other papers as the *resource allocation* problem, like Haider et al. (2009), but the description remains basically the same.

As the name suggests, the *network testbed mapping problem* has emerged during the creation of a network experiments platform at the University of Utah. The idea of such a platform is to simplify the realization of experiments by connecting a large quantity of switches together, and then to select the switches and cables that must be part of a given experiment. All possible topologies can be imagined and tested without having to physically modify the wiring. This differentiates the network used during an experiment (referred to as a *virtual network*), from the *physical network*. Another interesting property of this platform is that several virtual networks can, under certain conditions, be deployed in parallel.

To instantiate a virtual network, experimenters must send a *virtual network request*, in which they describe their needs. This description is given as a graph, where nodes are differentiated by their type. A (virtual) node whose type is “switch” (resp. “workstation”) must be assigned to a (physical) node whose type is “switch” (resp. “workstation”). The edges of the graph represent *virtual cables*, which allow the different virtual nodes to communicate. For instance, the virtual cable that connects two virtual nodes “a” and “b” physically corresponds to a path in the physical network that connects the physical node corresponding to “a” to the physical node corresponding to “b”. When requesting such a virtual cable, the experimenter must provide a quantity representing the bandwidth of the virtual cable. At the physical level, this same quantity represents the maximum bandwidth that the experimenter can consume. In practice, this also means that it is as much bandwidth that can not be reserved for any other virtual cable.

It is precisely during the deployment of a set of virtual networks that the *network testbed mapping problem* appears. On the one hand, for a given virtual topology, several physical topologies can match, or none at all. On the other hand, as already mentioned, each virtual network consumes bandwidth: the virtual networks are thus competing to acquire it.

In the end, not all deployments are equal, and some may even be undesirable. Typically, some deployments overload the physical network, and so a virtual network may impact the behavior of another virtual network. Since the goal is to

have independent experiences, such deployments are undesirable. The question then is to find the deployments that are desirable. This question is not simple. It turns out that the underlying problem is mathematically *NP*-hard (see Definition 2.3), as Ricci et al. (2003) and Fischer et al. (2013) remind us.

Besides, the initial state for the *network testbed mapping problem* may not be a case where no virtual network is deployed, and no physical resource is reserved. Instead, we may have a set of virtual networks already reserving physical resources. Finding deployments for newcoming virtual network requests then depends on the remaining, non-reserved resources. Consequently, the choice of resources for past virtual network requests matters, adding more complexity to the problem.

## 2.2 Extensions to the VNE Problem

Since then, the VNE problem has spread in many other fields of computer science. Today, it should be noted that although Ricci et al. (2003) uses the term “virtual network”, it rather means “logical network”, since every deployment merely changes the cabling in an abstract way between the different physical workstations. This precision is necessary, because state of the art works have refined the definition of virtual networks, by following the advances of virtualization technologies. As an example, Ballani et al. (2011), Papagianni et al. (2013), and Rabbani et al. (2013) define a virtual network in the cloud computing field, where virtual nodes are VMs instantiated in physical servers. In other words, several virtual nodes can be instantiated within the same physical node, which requires the introduction of another consumption data, namely the computing capacity. Similar to the bandwidth, this capacity is consumed by each VM.

Consequently, it is now very common in the VNE to consider that a physical node can host more than one virtual node (which can be a VM, but also a virtual router).

This section is intended to be an overview and an introduction to the different variants of the VNE problem, applied to different fields. To do so, we decide to present how different works, shown in Table 2.1, adapt the VNE problem to their needs. We also add one of our previous publications, as Boutigny et al. (2018), in Table 2.1.

In Table 2.1, we distinguish the problem the authors focus on (either the general VNE problem or a variant), its formulation (or resolution method), and the objective of their work. We warn the reader that two rows share the same reference. It is the case of Mehraghdam et al. (2014) and Liu et al. (2014). This is because in each of those papers, the authors propose three VNE problems, with each a different objective.

Overall, in the first column of Table 2.1, the different investigated problem variants are:

- **VNE**. The general VNE problem.
  - **Dynamic VNE**. A VNE handling requests dynamically. See Subsection 2.2.3.
  - **Service Function Chain Embedding (SFCE)**. One of the VNE problems in the Network Function Virtualization (NFV) field. See Subsection 2.2.5.
  - **Intra-Domain VNE**. One of the VNE problems in multi-domain context. See Subsection 2.2.2.
  - **Inter-Domain VNE**. One of the VNE problems in multi-domain context. See Subsection 2.2.2.
  - **Virtual Network Embedding Reconfiguration (VNER)**. A VNE variant supporting virtual network migration. See Subsection 2.2.3.
  - **Survivable Virtual Network Embedding (SVNE)**. A VNE variant supporting fault tolerance. See Subsection 2.2.3.
    - **Intra-Domain VNER**. One of the VNE problems in multi-domain context, handling virtual network migration. See Subsection 2.2.2 and Subsection 2.2.3.
    - **Inter-Domain VNER**. One of the VNE problems in multi-domain context, handling virtual network migration. See Subsection 2.2.2 and Subsection 2.2.3.
  - **Virtual Data Center Embedding (VDCE)**. One of the VNE problems in the cloud field. See Subsection 2.2.2.
    - **Dynamic VDCE**. One of the VNE problems in the cloud field, handling requests dynamically. See Subsection 2.2.2 and Subsection 2.2.3.

The second column of Table 2.1 is about the problem formulation. We identify two formulations. When a paper proposes both formulations, it is indicated as “ILP, heuristic”.

- **Integer Linear Programming (ILP)**. An exact resolution method. See Section 2.4.
- **Heuristic**. A resolution method relying on empirical evidence or apriori knowledge of the problem. The actual resolution algorithm depends on the authors.

Table 2.1 is a global presentation of all the works reviewed for the state-of-the-art in the VNE field. In the remainder of this section, we analyze those works using the following dimensions.

- **Optimization Objective**. In Subsection 2.2.1, we show that the optimization objective for the VNE problem is not unique, and that different concrete use cases lead to different optimization objectives, which can be conflictual.

- **Substrate Model.** Subsection 2.2.2, we show that the VNE problem changes drastically when considering that the *substrate* (which is the dedicated term to refer to the infrastructure in the VNE field, see Fischer et al. (2013)) is divided into multiple domains.
- **Request Model.** In Subsection 2.2.3, we present how multiple requests can be handled, and different request models.
- **Software-Defined Network (SDN) and NFV Impact.** In Subsection 2.2.4 and Subsection 2.2.5, we show how two new emerging paradigms of network virtualization, namely SDN and NFV, may redefine the VNE problem.

Table 2.1: Classification of VNE Research Papers

| Reference                | Problem          | Formulation    | Objective                                |
|--------------------------|------------------|----------------|--|
| Rost and Schmid (2018)   | VNE              | ILP            | Max acceptance                           |
| Boutigny et al. (2018)   | Intra-domain VNE | Heuristic      | Enumeration                              |
|                          | Inter-domain VNE | Heuristic      | Enumeration                              |
| Li et al. (2017)         | VNE              | Heuristic      | Min VN cost                              |
| Alaluna et al. (2017)    | VNE              | ILP            | Min VN cost                              |
| Mano et al. (2016)       | Inter-domain VNE | Heuristic      | Min VN cost                              |
| Fischer et al. (2016)    | VNE              | Heuristic      | Min VN cost                              |
| Bays et al. (2016)       | VNE              | ILP            | Min flow rules                           |
| Wang et al. (2015)       | VNE              | ILP            | Min VN cost                              |
| Nonde et al. (2015)      | VNE              | ILP, heuristic | Min power consumption                    |
| Dietrich et al. (2015)   | Inter-domain VNE | ILP, heuristic | Min VN cost                              |
| Bellavista et al. (2015) | VNE              | ILP, heuristic | Min infrastructure load                  |
| Zhang et al. (2014)      | Dyn. VDCE        | ILP, heuristic | Min power consumption and migration cost |
| Mehraghdam et al. (2014) | SFCE             | ILP, heuristic | Max residual bandwidth                   |
|                          | SFCE             | ILP, heuristic | Min # of hosts in use                    |
|                          | SFCE             | ILP, heuristic | Min latency                              |
| Liu et al. (2014)        | VNE              | ILP            | Max acceptance ratio                     |
|                          | VNE              | ILP            | Max long-term revenue                    |
|                          | VNE              | ILP            | Max long-term revenue to cost ratio      |
| Chau and Wang (2014)     | VNE              | ILP            | Min VN cost                              |
| Bays et al. (2014)       | VNE              | ILP, heuristic | Min VN cost                              |
| Riggio et al. (2013)     | VNE              | Heuristic      | Min virtual edge stress                  |

| Reference                 | Problem           | Formulation    | Objective                          |
|---------------------------|-------------------|----------------|------------------------------------|
| Rahman and Boutaba (2013) | SVNE              | ILP, heuristic | Min overconsumption                |
| Rabbani et al. (2013)     | VDCE              | Heuristic      | Max # of accepted requests         |
| Papagianni et al. (2013)  | VNE               | ILP            | Min VN cost                        |
| Di et al. (2013)          | Inter-domain VNE  | ILP, heuristic | Min VN cost                        |
| Houidi et al. (2011)      | Intra-domain VNE  | ILP, heuristic | Min VN cost                        |
|                           | Inter-domain VNE  | ILP, heuristic | Min VN cost                        |
| Fajjari et al. (2011)     | Dyn. VNE          | Heuristic      | Min congestion and reconfiguration |
| Ballani et al. (2011)     | VDCE              | Heuristic      | Unknown                            |
| Arora et al. (2011)       | Intra-domain VNER | Heuristic      | Min migration cost                 |
|                           | Inter-domain VNER | Heuristic      | Min migration cost                 |
| Chowdhury et al. (2010)   | Inter-domain VNE  | Heuristic      | Unknown                            |
| Bienkowski et al. (2010)  | VNER              | Heuristic      | Min migration cost                 |
| Yu et al. (2008)          | VNE               | Heuristic      | Max long-term revenue              |

## 2.2.1 Rethinking the Objective Function

In the *network testbed mapping problem*, the objective is to maximize the chances of accepting future virtual network requests. We can understand this objective, as it is in the context of an experimental platform.

For their part, all the works shown in Table 2.1 address a business use case, be it in a datacenter, multiple datacenters, or an Internet Service Provider (ISP) network. In all those cases, the authors oppose the tenant (the one requesting the virtual network), to the Infrastructure Provider (InP) (providing the physical resources). In some cases, another participant is added, the Virtual Network Provider (VNP), negotiating with InPs to rent their resources on behalf of the tenants. In all cases, the tenant is indeed renting the virtual network. A full presentation of the relations between the VNP, the InP and the tenant is given by Fischer et al. (2013).

Table 2.2 presents the same works as in Table 2.1 with a focus on their objective functions, as many different objective functions exist. For instance, Liu et al. (2014) propose to maximize the revenue of the InP. For their part, Alaluna et al. (2017) propose to minimize the cost of the virtual network to the tenant. Besides, in the field of cloud computing, Nonde et al. (2015) propose to minimize the power consumption of the whole data center, down to the optical devices. The reason why Nonde et al. (2015) minimize the power consumption is because they report it to be the greatest operational expenditure for data centers.

Table 2.2: Classification of Objectives of VNE Research Papers

| Objective Category | Objective           | References  |
|--------------------|---------------------|---|
| Tenant oriented    | Min VN cost         | Li et al. (2017) and Alaluna et al. (2017)<br>Fischer et al. (2016) and Mano et al. (2016)<br>Wang et al. (2015) and Dietrich et al. (2015)<br>Chau and Wang (2014) and Bays et al. (2014)<br>Papagianni et al. (2013) and Di et al. (2013)<br>Houidi et al. (2011) |
| InP oriented       | Min bottleneck load | Bays et al. (2016)<br>Bellavista et al. (2015)<br>Mehraghdam et al. (2014)<br>Riggio et al. (2013) and Rahman and Boutaba (2013)<br>Fajjari et al. (2011) and Arora et al. (2011)<br>Bienkowski et al. (2010)   |
|                    | Min expenditure     | Nonde et al. (2015) and Zhang et al. (2014)   |
|                    | Max revenue         | Liu et al. (2014) and Yu et al. (2008)  |
| Neutral            | Acceptance related  | Rost and Schmid (2018) and Rabbani et al. (2013)  |
| Unknown            | Unknown             | Ballani et al. (2011) and Chowdhury et al. (2010)   |

Minimizing the cost of the virtual network is the most widespread objective, although the actual definition of the cost depends on the paper. It is a tenant-oriented objective, in the sense that InPs are likely to prefer to charge more. This variety of cost definitions comes from the fact that the cost must be understood as what the tenant is charged for (it is then referred to also as the “revenue”), and this is up to the business strategy of the InP, which may vary. InPs get a revenue from renting their resources, but must also pay for their expenditures, like energy consumption, the buildings they rent, and their employees. To make profit, they need multiple tenants, or tenants with big enough requests. Meanwhile, as they rent the embedded virtual networks, tenants are sensitive to the rental price. Besides, virtualization naturally changes the perception of the virtual networks: they only become a set of files (VMs), software (VM hypervisor) and configuration (routing rules), consuming the desired amount of memory and CPU provided by whatever physical resources. This change in the perception of what a network represents may make it easier for tenants to consider to change their InPs.

Meanwhile, we consider that the objective functions from works like Mehraghdam et al. (2014), Nonde et al. (2015), and Yu et al. (2008) belong to the same InP-oriented objective category. Overall, those objectives are to minimize the load of the bottlenecks in the infrastructure as identified by the respective paper, or to minimize InP expenditure (mainly power consumption), or to maximize the InP revenue. The definition of the bottlenecks may vary. For Bays et al. (2016), the bottleneck is the capacity of programmable switches in the SDN context (see

Subsection 2.2.4), as their capacity is very limited. In the cloud context, Bellavista et al. (2015) proposes to minimize the traffic load in the whole infrastructure. Interestingly, Mehraghdam et al. (2014) propose, among other objectives, to minimize the number of hosts in use, which in turn may increase the load on the neighboring links. From their experiments, Fajjari et al. (2011) conclude that the main bottleneck in the link capacity.

The remaining works could not be classified as more tenant-oriented or more InP-oriented. Some focus on the acceptance of the requests, which may be beneficial for both the tenants and the InPs. As such, we classify them as neutral. This does not mean that they are better though. For instance, maximizing the number of accepted requests does not mean we maximize the revenue, unless the cost and the revenue of each request is equal.

For Ballani et al. (2011) and Chowdhury et al. (2010), we were not able to retrieve information about an objective function.

From a security viewpoint, Alaluna et al. (2017) and Bays et al. (2014) consider the security as a service, which is charged to the tenant. But other kinds of objective function may be considered. For instance, we can leverage the National Vulnerability Database<sup>1</sup>, to retrieve the known vulnerabilities of each product setup in a given infrastructure. The products are retrieved from the MITRE Common Platform Enumeration (CPE) database<sup>2</sup>, and the vulnerabilities are retrieved from the MITRE Common Vulnerabilities and Exposures (CVE) database. Each vulnerability is furthermore associated with a number, computed from the MITRE Common Vulnerability Scoring System (CVSS). This number represents the severity of the vulnerability. It is very different from a security level, in the sense that the CVSS is a scoring system which can be tuned by an organization to better adapt to its environment. This is possible by tuning the environmental score metrics (in CVSS version 3). We could agglomerate the environmental score metrics of all tenants, and minimize the severity of their virtual resources.

To the best of our knowledge, no work has proposed it yet. This approach has different downsides. First, it only deal with known vulnerabilities, and when a new vulnerability is published, the embedding may have to be updated. Second, the CPE database does not enumerate every known product, although every product should be described by the CPE naming system, but *at least every known vulnerable* product, even though the CPE can be contributed by organizations external to MITRE<sup>2</sup>; to the best of our knowledge, there is no list of current non-vulnerable products<sup>3</sup>.

---

<sup>1</sup>See <https://nvd.nist.gov/general>.

<sup>2</sup>See <https://nvd.nist.gov/Products/CPE>.

<sup>3</sup>As an illustration, the reader can verify that on 6/10/2019 (time of writing), there is no entry for the Google Chrome 76 stable version (see [https://en.wikipedia.org/wiki/Google\\_Chrome\\_version\\_history](https://en.wikipedia.org/wiki/Google_Chrome_version_history)), but one for Google Chrome 76 *beta* version. The Google Chrome

As a consequence, some CPE data can be misleading. For instance, vulnerability CVE-2013-6662<sup>4</sup> impacts *every* version of Google Chrome, as it is indicated by the CPE reference `cpe:2.3:a:google:chrome:*:*:*:*:*:*:*`; however, the CPE database does not define any entry for Google Chrome 76, or Google Chrome 77. In other words, although they are known to be vulnerable, there is no entry for them.

As such, when a non-referenced product is declared, we must ensure that we have enough and correct data to identify it, if it happens to be vulnerable in the long term.

To conclude this investigation around the objective functions in use, we want to point out the work from Ludwig (2016), which proposes to price the virtual networks on their *specificity*. Such definition is more market-related, and considers that the price of a virtual network may vary over time. The reason Ludwig (2016) proposes to do so is because a very specific virtual network topology, once embedded, may cause the reject of future virtual networks. The physical resources the embedded virtual network is using become then more valuable, because they become key to fulfill the demand. Ludwig (2016) proposes to reverebrate the fact they become more valuable by increasing their price. Changing the physical resources that are consumed by a virtual network is also a solution, called migration. This aspect is however addressed in Subsection 2.2.3.

## 2.2.2 Rethinking the Substrate

From a general viewpoint, the VNE problem can be described as the matching of an offer and a demand. The offer is brought by the InP, and the demand is brought by the tenant. If the offer was unlimited, the VNE problem would not be very complex. It is actually the meaning of being a *NP*-hard problem (see Definition 2.3). The VNE problem is a *NP*-hard problem in general. But some particular instances, like, a particular request, or a particular substrate, may lead to a simpler problem. At least, it may lead to heuristic that are really efficient for those particular instances.

Table 2.3 enumerates the same works as from Table 2.1 classifying them according to their substrate model. Our first discriminator is how many the model can handle, either one (single InP) or many (multiple InPs). For the single InP case, our second discriminator is if the substrate topology is structured or not.

For example, because data centers have structured topologies (usually in trees), particular heuristics can be applied, as Bari et al. (2013) report in their survey about virtual network embedding for cloud computing. These heuristics are not

---

76 version was released on 30/07/2019, and the 77 version was released on 24/09/2019.

<sup>4</sup>See <https://nvd.nist.gov/vuln/detail/CVE-2013-6662>



Table 2.3: Classification of Substrates of VNE Research Papers

| Substrate Category | Substrate Property      | References  |
|--------------------|-------------------------|---|
| Single InP         | Non-structured Topology | Rost and Schmid (2018) and Li et al. (2017)<br>Fischer et al. (2016)<br>Bays et al. (2016)<br>Wang et al. (2015) and Nonde et al. (2015)<br>Chau and Wang (2014)<br>Bays et al. (2014)<br>Liu et al. (2014) and Mehraghdam et al. (2014)<br>Papagianni et al. (2013) and Riggio et al. (2013)<br>Rahman and Boutaba (2013)<br>Arora et al. (2011) and Fajjari et al. (2011)<br>Bienkowski et al. (2010)<br>Yu et al. (2008) |
|                    | Structured Topology     | Bellavista et al. (2015) and Zhang et al. (2014)<br>Rabbani et al. (2013) and Ballani et al. (2011)   |
| Multiple InPs      | Fully disclosed         | Alaluna et al. (2017)   |
|                    | Partially disclosed     | Mano et al. (2016) and Dietrich et al. (2015)<br>Di et al. (2013) and Houidi et al. (2011)<br>Chowdhury et al. (2010)   |

applicable when we reason with a distribution of virtual networks between several data centers, or even when we consider the infrastructure of a telecom operator, which has much less structured topologies. More specifically, Ballani et al. (2011), Zhang et al. (2014), and Rabbani et al. (2013) address the Virtual Data Center Embedding (VDCE) problem. The VDCE problem considers that both the substrate and the requests are tree-like graphs, the substrate representing a datacenter, and the requests representing the virtual datacenter requests.

Even the offered resources can be different. For instance, we can distinguish the following substrate networks: 1) for Bays et al. (2014), the substrate network exposes only networking resources (i.e. switches, routers). Tenants request networking resources, and are provided with virtual switches/routers; 2) for Mano et al. (2016), Dietrich et al. (2015), Papagianni et al. (2013), Alaluna et al. (2017), and Liu et al. (2014), the substrate network exposes networking and computing resources (i.e. virtualization servers). Tenants request only computing resources, and are provided with VMs. Both cases are similar from the tenant’s viewpoint though, in the sense that they request a single type of resource, either computing or networking. Indeed, tenants are also provided with communication pathways interconnecting those resources.

Another way to rethink the substrate is to divide it into multiple subparts (referred to as *domains*), each owned by a different InP. This corresponds to the multiple InP case in Table 2.3. The idea then is that the InPs act as if there was

Table 2.4: Classification of Request Handling Methods of VNE Research Papers

| Request Handling Method | References  |
|-------------------------|---|
| Static                  | Rost and Schmid (2018)<br>Mano et al. (2016)<br>Wang et al. (2015) and Bellavista et al. (2015)<br>Chau and Wang (2014) and Mehraghdam et al. (2014)<br>Rahman and Boutaba (2013) and Di et al. (2013)<br>Chowdhury et al. (2010)<br>Yu et al. (2008)   |
| Online                  | Li et al. (2017) and Alaluna et al. (2017)<br>Bays et al. (2016)<br>Nonde et al. (2015) and Dietrich et al. (2015)<br>Bays et al. (2014) and Liu et al. (2014)<br>Papagianni et al. (2013), Riggio et al. (2013), and Rabbani et al. (2013)<br>Houidi et al. (2011) and Ballani et al. (2011) |
| Dynamic                 | Fajjari et al. (2011), Zhang et al. (2014), and Fischer et al. (2016)   |
| Reconfiguration         | Arora et al. (2011) and Bienkowski et al. (2010)  |

a single infrastructure. This does not mean that the requests will be always deployed over every InP, nor that they will always be deployed within a single InP.

Alaluna et al. (2017) apply such paradigm in the cloud context, with full information disclosure. Full information disclosure means that the VNE algorithm knows all the information from every domain, like nodes, edges, node capacity, edge capacity. It corresponds to the *fully disclosed* row in Table 2.3.

Full information disclosure, however, cannot be applied to every context, because topological information may be valuable, and sensitive, for the InPs. In that case, it is better to provide limited information disclosure. Limited information disclosure being related to InP protection, it will be treated within Section 2.3. It corresponds to the *partially disclosed* row in Table 2.3.

### 2.2.3 Rethinking the Requests

In their survey, Fischer et al. (2013) use the request processing as a criterion to classify VNE algorithms. Indeed, this aspect is already mentioned in the *network testbed mapping problem*. Fischer et al. (2013) identify two cases, namely, the static mode and the dynamic mode. We reuse this classification in Table 2.4, and we add two other classes, “Online” and “Reconfiguration”.

The static mode is the simplest mode of both. It assumes that the virtual network requests are known in advance. Then, the algorithm embeds them, while meeting the goal of the objective function. It corresponds to the “Static” row in Table 2.4.

The dynamic mode is more complicated. The first difficulty is that it implies to treat the requests in an online manner, as Houidi et al. (2011), Li et al. (2017), Alaluna et al. (2017), and Bays et al. (2014) do. Each request has a lifetime and a time of arrival. When the lifetime of a request is expired, the corresponding virtual network is shutdown, and the resources it occupied are released. At each instant, a new request may be submitted to the system, so the algorithm must be able to prepare enough resources to accommodate any new request. It must also memorize the residual capacities of the substrate network.

The second difficulty is that to increase the chances of a request to be accepted, the online treatment is not sufficient. Some VNE algorithms rely indeed on the concept of migration. This second aspect is key to be part of the dynamic category of the Fischer et al. (2013) classification. In Table 2.1, Fajjari et al. (2011), Arora et al. (2011), Bienkowski et al. (2010), Zhang et al. (2014), and Fischer et al. (2016) are the only works addressing both difficulties. For this reason, we distinguish in Table 2.4 works that treat requests in an online manner, without considering their reconfiguration, in class “Online”. Works that treat requests in an online manner *and* consider their reconfiguration are in class “Dynamic”.

This is such an intense research opportunity that simulation platforms (like ALEVIN from Fischer et al. (2011)) have been developed specifically to test dynamic VNE algorithms. It is because Fischer et al. (2016) is implemented within this platform that we consider it as dynamic.

To be more specific, Arora et al. (2011) and Bienkowski et al. (2010) address the Virtual Network Embedding Reconfiguration (VNER) problem, which seeks how to migrate virtual networks to make room for another given virtual network. They are listed in class “Reconfiguration” in Table 2.4. This class is distinct from the others in the sense that the VNER problem reconfigures already embedded requests without handling new ones. The other works, Fajjari et al. (2011) and Zhang et al. (2014), are true dynamic VNE algorithms as per the definition from Fischer et al. (2013), in the sense that the same algorithm which is responsible for the embedding task does also the re-embedding when needed. The slight difference with the VNER algorithms is that those latter are executed when the VNE algorithm counterpart fails to embed the newcoming request.

The use case of migration is as follows: the system receives a request and replies that it can not process it, because it does not find a way to deploy a virtual network corresponding to the request. In other words, there are not enough physical resources available. To solve this problem, there are two ways: either increase the number of physical resources (by buying more hardware), or increase the number of available resources. In the latter case, the idea is that an optimized deployment of  $n$  virtual networks does not necessarily allow to have an optimized deployment of  $n + 1$  virtual networks. To move from the  $n$  case to the  $n + 1$  case, it may be necessary to migrate the virtual elements, that is to

say to change the physical resources that are associated with the nodes and the virtual links.

Such an operation is not harmless though. For example, there are two migration paradigms: live migration and cold migration. Cold migration is the simplest, and involves taking a virtual network, disable it, re-deploy it, and then reactivate it. In other words, the virtual network, possibly in use, becomes unusable during the migration. In addition, data in memory (in nodes) or in transit (in links) can be lost. The cold migration thus has a great advantage for solving the VNE problem, but a great disadvantage for the user of the virtual network.

On the other hand, live migration makes the problem drastically complex. The basic idea is to effect the migration in a continuous way. There are multiple scenarios, which are surveyed by Leelipushpam and Sharmila (2013). We only give an insight of one of them. In the case of VMs, the memory pages are transmitted from the source machine to the destination machine. In the case of virtual cables, there is a path for old traffic during migration, and a path for new traffic. As, obviously, the transfer of the memory pages, the destination machine, and the path of the new traffic are added to the virtual network, an overconsumption of resources occurs. This overconsumption must therefore be processed upstream in order to decide whether or not live migration is even possible.

The migration of virtual networks intervenes also in another case of use, that of the creation of backups. These backups must allow the virtual network to continue to function, even in the event of a failure or unavailability of the physical resources on which it is deployed. This problem is investigated by Rahman and Boutaba (2013), and is called the Survivable Virtual Network Embedding (SVNE) problem. It is indeed a fault-tolerance oriented problem, as the algorithm tries to circumvent any failure at the substrate level by reserving resources for potential virtual network migrations.

Another approach is proposed by Alaluna et al. (2017). The idea is to allow tenants of the virtual network to ask if they want backups for this or that part of their virtual network. This service is obviously expensive, since the system must identify and prepare additional physical resources.

Another fundamental aspect has been questioned by Yu et al. (2008). This aspect can be summarized as follows: to what extent a virtual network should be considered and modeled like a physical network?

In the original description of the VNE problem, virtual cables are considered as an extension of the physical cables, which transmit packets without changing their chronological order up to their bandwidth capacity. To keep the same properties, the original description of the VNE problem then define virtual cables as physical paths, where a fraction of the bandwidth of each of its physical cable component is allocated for the given virtual cable. For Yu et al. (2008), we can also consider that a virtual cable correspond to a set of physical paths. The

bandwidth requirement of the virtual cable is then *distributed* over the different physical paths, deployed in parallel. Such operation is referred to as *path splitting* and increases the solution space for the VNE problem, as the bandwidth capacity is often the bottleneck (see Fajjari et al. (2011)), and as, for a given physical cable, fewer bandwidth may be required. Works like Li et al. (2017) also support path splitting. Concretely, path splitting means that different packets from the same traffic can take different paths. Consequently, this may not preserve their chronological order.

For their part, Dietrich et al. (2015) formulate an entirely new definition of virtual network requests. They use a matrix definition rather than a graph definition, where the cells of the matrix contain the bandwidth demands (the requests in computing capacity of the nodes are treated separately). Again, this definition increases the number of candidates to the VNE problem, since the same matrix can be used to generate several different topologies, hence several different graphs.

All in all, the virtual network request is not set in stone. We see that requests can be treated dynamically, or statically. Defining virtual network requests as dynamic lead to cope with the migration problem, in order to make room for other requests. The substrate can also be dynamic, in the sense that some physical resources may fail to work at some point, leading to migrate their guests onto other resources. What a virtual network request exactly represents is up to the virtualization technology. With enough care, we can support path splitting, that is, considering that a single virtual edge is represented by a bunch of physical paths. Other such transformations are covered in the next subsections.

A last point remains to be discussed. As we see, even the representation of virtual network requests as a graph can be questioned. If we stick to the graph representation though, it is important to consider that the request is still abstract. No work gives an exact definition of what the links in the graphs represent, for instance. And we can have various interpretations.

For instance, some work, like Yu et al. (2008), Alaluna et al. (2017), Houidi et al. (2011), and Chowdhury et al. (2010), assume that the virtual network request is an undirected graph. Others, such as Rost and Schmid (2018) and Bays et al. (2016), consider it to be a directed graph. Considering an undirected graph is meaningful, in the sense that we assume that virtual edges are like cables connecting two virtual nodes. The very idea is that all cables are bidirectional. However, this does not mean that the terminals at each end of the cable will use it in the same proportions: the tenant may want to discriminate an upload and a download bandwidth, or even the InPs may want to discriminate some of their physical resources. Directed graphs become helpful in such context, by enabling the tenant to create two directed edges between the same nodes, each with a different bandwidth.

## 2.2.4 Leveraging SDN

### Definition 2.1: n-1 Mapping, n-n Mapping

A n-1 mapping (or one-to-many mapping) is a relation from a set  $X$  to a set  $Y$  such that every element in  $Y$  is associated with at least one element of  $X$ . Mathematically, it actually corresponds to a surjective function. The notation “n-1 mapping” is inherited from computer science, and especially for database models. Being surjective, a n-1 mapping can associate multiple elements of  $X$  to the same element in  $Y$ . Besides, each element of  $X$  is associated to a unique element of  $Y$ .

The n-n mapping (or many-to-many mapping) is another important relation in computer science. It is a relation from a set  $X$  to a set  $Y$  such that each element of  $X$  can have more than one image in  $Y$ . Mathematically, it is a multi-valued function. Given an element of  $X$ , it is associated to multiple elements of  $Y$ , by definition. Besides, multiple elements of  $X$  can be associated to the same element  $Y$ .

For completeness, we mention that a 1-1 mapping (or one-to-one mapping) is a relation from a set  $X$  to a set  $Y$  such that two distinct elements of  $X$  have distinct images in  $Y$ . Mathematically, it actually corresponds to an injective function.

The latest advances in virtualization that are the SDN and NFV paradigms have also led to the formulation of specific VNE problems. SDN is commonly described as decoupling the control plane from the data plane. This means that switches and routers do no longer do anything by themselves, unlike legacy network elements. Instead, the management of these protocols is the responsibility of a new network component, referred to as the *controller*. The previously designated switches and routers then become *programmable switches*, and are all connected to the controller. Programmable switches have only one default behavior: send the received traffic to the controller. In response, the controller tells them what action to take on this traffic (for instance, to get it out of another port). Better, the controller can also tell them a rule to follow.

All these tasks revolutionize the world of transmission protocols, because the controller has a global view of the network it controls, which was not the case before. In addition, the controller can be physically distributed: that is, there are several physical controllers, but act logically as if there was only one controller. This should ensure scalability of this solution.

Returning to the description of SDN as a decoupling, what must be seen here is that the control plane corresponds to the traffic between the programmable switches and the controller. This traffic is carried by various protocols still un-

der study, the best known of which is OpenFlow. The data plane corresponds to the traffic exchanged by the programmable switches between them. SDN's interest in virtualization is to pave halfway. Thus, in the same way that we speak of hypervisors for virtual machine managers, we speak of *network hypervisors* to designate virtual network managers. And SDN is at the heart of these hypervisors.

There are different solutions and different architectures of network hypervisors; but overall, the principle is the same: we use the SDN controller as a proxy server, able to transform one set of routing rules into another set. This exposes a user Alice and a user Bob a virtual switch A and a virtual switch B when it is the physical switch C. When Alice's traffic arrives on C, it follows the routing rules of A, and when Bob's traffic arrives on C, it follows B's routing rules. This operation is a *n-1 mapping* (cf. Definition 2.1), which is the same kind of mappings used for virtual machines, which allows multiple VMs to coexist on the same host. However, network hypervisors enable also *n-n mappings* (cf. Definition 2.1), which are much more complex. With such transformations, the virtual switch A physically corresponds to a set of interconnected physical switches, for example C and D, and the network hypervisor must then distribute the routing rules between these switches in an intelligent manner.

The SDN technology thus makes it possible to imagine virtual networks where certain nodes would themselves be programmable switches. These programmable switches are actually virtual, and are managed by the system through the aforementioned n-1 and n-n transformations. Of the two, the n-n transformation is the one that poses the most difficulty for a VNE formulation. Indeed, the choice of all the programmable switches to be used to represent a given virtual switch is not trivial, because it must also be able to route the traffic between the various chosen programmable switches, and this can only pass through the selection of paths, themselves consumers of bandwidth. To the best of our knowledge, no work in VNE enables such transformations yet. Riggio et al. (2013) and Bays et al. (2016) leverage SDN in their works though. Note that in Bays et al. (2016), the objective function is to reduce the number of flow rules on the programmable switches. Without diving into the details, this is because of a current hardware limit. The configuration of a virtual network with SDN is made with flow rules, which are regular expression describing how an input flow on a programmable switch is transformed, and where it is outputted. However the memory which is specialized into saving those flow rules is very limited in capacity.

### 2.2.5 Leveraging NFV

NFV technology is based on the concept of *network function*. In abstract, a network function is an operation that occurs within a network and affects its behav-

ior. This definition is clearer if we compare it with the purpose of this technology. Indeed, NFV is originally a principle promulgated by operators, who spend time and energy to introduce devices into their networks. This is particularly true in security, where to protect the network, firewalls or intrusion detection systems are installed. The problem is that adding such devices to a network complicates its maintenance, especially since these devices are proprietary: to configure or update the network function they run, it is often necessary to have specific knowledge.

What NFV proposes is to use off-the-shelf hardware instead of dedicated hardware. The network function which is provided by the legacy hardware is provided by the means of a software program instead. This software program is referred to as a Virtual Network Function (VNF).

NFV's interest in virtualization, however, is to provide an opportunity for a tenant to have its own network function, while the configuration and management of the scaling of this network function is performed by the provider of this function.

In addition, the work in NFV also relates to the Service Function Chains (SFCs). This chain translates the fact that the network functions must execute in a certain order. For example, traffic is usually passed through a firewall, then through an IDS, in order to handle only the most complex attacks. In the case of VNF, we end up with software that can be installed anywhere in the network: the traffic must therefore follow a particular route for all these functions to run in order, as issued by the SFC.

The NFV technology thus makes it possible to imagine virtual networks where the holder specifies network functions to be applied along this or that virtual cable. The path used to represent the virtual cable must then follow the SFC as described. Several visions are opposed to the location of the VNFs of the SFC, however. Bellavista et al. (2015) considers that VNFs are instantiated at specific places by the InP: thus, when the specific holder has a network function, everything happens as if he was subscribing the virtual cable to a network function, and thus causing the path to be modified so that it reaches said VNF. Mehraghdam et al. (2014), however, offers another vision, where the InP seeks to optimize the placement of VNFs. In this case, in a way, the information given by the holder of the virtual network will be used to place the VNF that he himself asks. Optimization aims to minimize the number or the load of instantiated VNFs, as well as the over-consumption of bandwidth caused by overly complex paths within the infrastructure.

To be more precise, as Mehraghdam et al. (2014) report it themselves, the problem they address is a bit different from the VNE problem, and can be more accurately described as the Service Function Chain Embedding (SFCE) problem. The SFCs are the equivalent of the virtual network requests of the VNE problem.



Their topology is more structured and simple though, as they are chains. The main difference though is that the virtual edge between two VNFs of one SFC may be reused between the same VNFs of another SFC.

## 2.3 Security Aspects for the VNE Problem

VNE is a long-studied problem. As explained by Fischer et al. (2013), even if we are told where to map virtual nodes, finding the right link allocations corresponds to the unsplittable flow problem, which Kolliopoulos and Stein (1997) showed to be *NP*-hard. For this reason, the slice embedding problem is also *NP*-hard.

This section presents works that address security aspects of the VNE problem. We identify mainly two aspects. Works in Subsection 2.3.1 enable the tenants to require security within their virtual network requests. Works in Subsection 2.3.2 enable the InP to protect themselves.

### 2.3.1 Security for the Tenants

We identify two primitives to provide security requirements into the VNE problem. The first primitive uses a level to define a security requirement. When a tenant requires a certain level for a given virtual resource, only substrate resources with a greater level can be candidates to embed this virtual resource. In other words, there is a direct placement relation between the virtual resource and the substrate resource. This primitive is helpful, because it can directly be expressed in ILP, the problem formulation which is used by most VNE works, as shown in Table 2.1. Overall, this primitive is used to define security levels based on an absolute scale, which is then defined by the authors.

The second primitive defines a security requirement on an exclusion or a collocation principle on the virtual resources themselves. Contrary to the first primitive, this means that there is a direct placement relation between two or more virtual resources.

The reviewed papers provide a composition of the different primitives.

Liu et al. (2014) allow tenants to require a security level for nodes and links, then following the first primitive. The InP must provide a resource with higher security level.

Bays et al. (2014) focus on privacy issues for a tenant in an ISP infrastructure context. The tenants can enforce their security through two ways. First way, by demanding a *cryptographic support* for any virtual router. This translates into a binary attribute, following the first primitive. Second way, by enumerating the other requests with which they do not want to share any resource, following

the second primitive. In addition, tenants can request a virtual router to be in a specified *location*, following the first primitive.

Alaluna et al. (2017) focus on tenant protection in the cloud context. They propose four security attributes: cloud provider *trustworthiness*, node *backups*, node *security level*, and link *security level*. Node and link security levels follow the first primitive, as well as cloud trustworthiness level. In essence, the node and link backups are duplicate of other virtual resources, as if the tenants require the same node or link twice. What Alaluna et al. (2017) add is that a level determines where to place the duplicate relatively to its originator. The duplicate may then be in the same cloud as the originator, or in another cloud. For this reason, the node and link backups follow the second primitive.

Wang et al. (2015) allow tenants to require a security plan on three categories: virtual network wise, virtual node wise, and virtual link wise. Each plan further propose three levels. The high (resp. medium, resp. low) level of the virtual network wise security plan allows the virtual network to be collocated with no (resp. trusted, resp. any) other virtual network, in the same datacenter. The high (resp. medium, resp. low) level of the virtual node wise security plan allows the virtual node to be collocated with any virtual node of no (resp. trusted, resp. any) other virtual network, in the same host. The high (resp. medium, resp. low) level of the virtual link wise security plan means that there is point-to-point encryption (resp. end-to-end encryption, resp. no encryption). In essence, the virtual network wise security plan and the virtual node wise security plan are based on the second primitive, and the virtual link wise security plan is based on the first primitive.

Fischer et al. (2016) propose to implement security requirements on the ALEVIN simulator (see Fischer et al. (2011)). They classify security requirements into three types: node requirements, such as encryption support; link requirements, such as data encryption; and topological requirements, such as separating the virtual network request into two administrative domains, one public, and one private, this latter protected by a firewall. Their node and link requirements are then based on the first primitive, while the topological requirements are based on the second primitive.

In this thesis, and like Fischer et al. (2016), we consider that the notion of security or trustworthiness level is too abstract, as it may cover different security properties in a way which do not correspond to the various tenant needs. Infrastructure Providers (InPs) and tenants may not have the same vision. For this reason, we want to enable tenants to express their security needs with appropriate attributes. To do so, we want to provide an attribute model general enough to cover every security use case. In particular, we show that every reviewed security-oriented attribute is compatible with our model, and that we can model other relevant security-oriented attributes.

Our work is distinct to the aforementioned works as follows.

Liu et al. (2014) provide a single security level for the tenant. The definition of such a security level is however only relevant when all tenants have the same security policy, and the same security priorities. In a multi-tenant context as with the 5G network slices, we cannot expect such a situation. Tenants may use slice for a large variety of services: health care, vehicles, smart cities, governmental services, and industrial automation.

Bays et al. (2014) investigate the VNE in the ISP infrastructure context, which is slightly different from slice embedding in the sense that the tenants are requesting (virtual) routers only, and virtual routers cannot be collocated on the same substrate router.

Besides, Bays et al. (2014) propose a location requirement such that the tenant can require a single location for each virtual resource. We consider it to be too limited, as it can only be applied for a geographical scale, be it continents, countries, or cities, but not a mix of different scales: this model makes them exclusive to each other. For instance, if the location of every substrate resource is identified at a country level, we have no way to support a city-level requirement; and to support a continent-level requirement, the only solution would be to enable the tenant to require a *list* of countries. We consider that enabling tenants to require a list of authorized locations is a plus, and that requiring a list of values can also be generalized to other requirements.

Another limit of Bays et al. (2014) is that they leverage the second primitive at the request level, while we consider three levels of isolation (exclude tenant, exclude request, exclude resource), for the nodes as well as the edges of the tenant requests.

For Alaluna et al. (2017), the cloud provider trustworthiness is absolute, and does not vary for different tenants, while we think it should be up to the opinion of the tenant. In that latter case, it would mean that tenants can control in which cloud provider their virtual resources are hosted. Besides, the security levels are abstract, and the semantics of each level may vary among tenants, among InPs, and also between tenants and InPs. In the remainder of this thesis, we will show how to support the backup requirement they propose. We note also that, although they consider multiple cloud providers, they consider that they have full knowledge over their resources, while we follow a limited information disclosure principle.

Wang et al. (2015) distinguish two steps. The first step is to apply their three security plans (per virtual network, per virtual node, per virtual edge), and is referred to as the admission process. The second step is the execution of the VNE algorithm itself. The idea is that the admission process serves as a guide to the VNE algorithm, because the admission process produces an auxiliary graph where each substrate node is associated with a set of candidate virtual nodes.

Those candidates are obtained by computing the exclusion constraints according to the security plan of the current virtual network request.

These latter words are important, because we can imagine that a past request refuse to be collocated with any other request and that the current one accept it; with the algorithm proposed by Wang et al. (2015), only the plan of the current request is considered, leading to a break in the security plan of the past request. We consider this flaw to be due to a lack of a single requirement model, from which the algorithm of the admission process should be derived.

Last but not least, the attribute model proposed by Fischer et al. (2016), although general, is implementation dependent. To support the second primitive in their topological requirements for instance, they add another embedding algorithm to their system, the cross-domain link embedding. As we focus on a security-oriented VNE, we think it is better to have at least a sound mathematical formulation, which can be verified and audited.

### 2.3.2 Security for the InPs

This section presents works that aim to protect the InPs.

Liu et al. (2014) consider not only tenant protection (as shown in previous section), but also InP protection. This latter is modelled with a level, like the tenant protection, but it is a requirement *from* the InP. This means that tenants must declare the security level of their virtual resources, and that this level should be higher than the security level of the substrate resource of the InP. Regarding the 5G slice embedding, we may not be able to make the same criticism than in the previous section hold, that is, such a security level is too abstract, because we may expect that in a multi-InP scenario, the different InPs share some common priorities, as they are in a common business field.

Liu et al. (2014) is, to the best of our knowledge, the only work requiring information from the tenants. The other works providing security to the InPs rather focus on different aspects of limited information disclosure.

Limited information disclosure is part of the multi-provider VNE problem, as illustrated by Mano et al. (2016) and Dietrich et al. (2015), where InPs do not disclose their complete topologies, but only some nodes. In this variant of the VNE problem, the substrate is split into domains, each domain being owned by an InP. We can divide works supporting limited information disclosure into three categories.

The first category contains the work from Chowdhury et al. (2010). In this work, the authors provide a *distributed* algorithm that enable the domains to cooperate, and the tenants to sent their requests to any of those domains. In other words, there are multiple entry points for the VNE algorithm.

The second and the third category only use a single entry point for the tenant. It is in that case that we may have a VNP. The second and the third category distinguishes the inter-domain level and the intra-domain level, but do differ in how the two levels are related.

The second category follows a *top-down* approach. This basically means that the virtual network request is processed by an inter-domain level algorithm. This algorithm aims at selecting the domains that will embed the corresponding virtual network. Dietrich et al. (2015) and Houidi et al. (2011) provide each an inter-domain level algorithm. This algorithm leverages the only information that the InPs are not reluctant to share, like how the domains are interconnected, and what general types of resources they support. Then, it splits the virtual network request into subparts, that can each be considered as virtual network requests themselves to an intra-domain level algorithm. At this stage, we come back to a VNE problem where the substrate is the respective domain. Any suitable VNE algorithm can be used at this point, and Houidi et al. (2011) provide their own for instance.

For its part, the third category follows a *bottom-up* approach. This basically means that the virtual network request is directly sent to *each* participating InP. Those InPs then run an intra-domain level algorithm whose purpose is to generate candidates. A candidate is actually a subpart of the virtual network request that the InP accepts and is able to embed. Then, an inter-domain level algorithm runs to select the candidates. The set of candidates that are selected has the property of covering the whole original virtual network request. Such an algorithm is proposed by Mano et al. (2016) and Di et al. (2013).

Note that the information disclosure is limited, not null, in all approaches. This follows the natural assumption that InPs know how they are interconnected with their neighbors. For Houidi et al. (2011), the inter-domain level algorithm knows the full pricing function of each InP for each given virtual resource. For Dietrich et al. (2015), the VNP knows the InP's border nodes (i.e. nodes interconnecting InPs with each other), as well as the types and prices of virtual nodes each InP supports. For Mano et al. (2016) and Di et al. (2013), the inter-domain level algorithm knows the InP's border nodes and the bandwidth of the inter-domain links. Besides, for Di et al. (2013), the actual price of each candidate is also fully known.

From all these works, two in particular focus on restricting access to prices, that is, Dietrich et al. (2015) and Mano et al. (2016). The reason is that price information is also sensitive.

Our work is distinct to the aforementioned works as follows.

For Dietrich et al. (2015), the capacity attributes (for nodes and for edges) are deeply inset into the matrix-oriented model, making it difficult to generalize to other attributes. Besides, the matrix-oriented model is not compatible with

support for multiple attributes at once. Some topologies generated by the matrix model of the virtual network request may also be seen as undesirable by the tenant. It may be even uneasy for an InP to disclose node types when considering multiple attributes. In such a scenario, telling the external parties the features which are not available, or the combinations of features which are available, may be considered as very specific and then too sensitive.

For Mano et al. (2016), the partial embeddings are limited to connected sub-graphs of the request. In other words, InPs cannot be used as a connection provider. It follows that the InP topology should be full-mesh, which does not correspond to the slice embedding scenario: we cannot expect every InP to be connected to all others.

Besides, Mano et al. (2016) do not make explicit the inter-domain nor the intra-domain algorithms, even if they rely on modified versions of other works of Yu et al. (2008) and Chowdhury et al. (2010). Last but not least, they limit InP embedding propositions to be connected subparts, whereas we support more general pieces.

## 2.4 Methods for Modeling the VNE Problem

When humans face a problem, they have a certain understanding of it. Sometimes, they overcome such problem by themselves. Other times, the problem is big enough that it requires automation. Encoding a problem refers to this very step of agreeing on the description of the problem. Such description comes with definitions of the concepts of the problem, which are relevant to solve it. From this description, we can formulate a procedure that will solve the problem. But it is clear by how the description is given that it is still only an agreement. The problem may be totally described in another way, and solved by another procedure. Therefore, the encoding is not unique.

In this section, we present formal methods for describing a problem. One method in particular is heavily used by the state of the art, Integer Linear Programming (ILP). However, ILP is only a particular example of Constraint Satisfaction Problems (CSPs). In this thesis, we propose to describe the VNE problem with another formal method, to suit our purpose of enabling security-oriented requirements.

From a mathematical point of view, the VNE problem is essentially a constraint satisfaction problem (CSP). CSPs appear in several fields, such as software and hardware verification, type inference, static program analysis, test-case generation, scheduling, planning, and graph problems, as reported by De Moura and Bjørner (2011).

**Definition 2.2: CSP**

A CSP is a triple  $(X, D, C)$  where:

1.  $X$  is a set of variables, denoted  $\{X_1, \dots, X_n\}$ ,
2.  $D$  is a set of domains, denoted  $\{D_1, \dots, D_n\}$ ,
3.  $C$  is a set of constraints.

Each domain  $D_i$  is a set of values,  $\{v_{i_1}, \dots, v_{i_k}\}$ . Besides, each domain  $D_i$  corresponds to one variable  $X_i$ .

For their part, constraints are Boolean functions defined over every combinations of values from every domains. When the constraint is true for a combination of values, it is said to be satisfied. Otherwise, when the constraint is false, it is said to be violated.

Solving a CSP means that we must exhibit a solution. A CSP *solution* is a complete, consistent assignment. An *assignment* is a function that associates some or all variable  $X_i$  to one value  $v_i$  in  $D_i$ . A *consistent* assignment is an assignment that does not violate any constraint. A *complete* assignment is an assignment that associates a value to every variable.

Russell et al. (2010) provide a formal definition of CSPs, which we give in Definition 2.2. This definition is rather general: it says nothing about the formalism of the constraints, nor that of the domains<sup>5</sup>. This freedom makes CSP formalism rich, thus its applicability to different fields. It must be borne in mind that the most decisive step to mathematically solve a problem is still in the way of formalizing it. In the CSP terminology, we are talking about the encoding of the problem. In practice, it is a matter of translating a concrete problem possibly expressed in natural language into a mathematical problem. This translation, again, is not unique.

Note that Definition 2.2 is rather general. Domains may be finite or not, and nothing is said about how constraints are defined. In Examples 2.1 and 2.2, we show two simple examples of how the same constraint can be defined. Example 2.1 sees the constraint as a collection of allowed combinations, while Example 2.2 exhibits the (simple) equivalent relation.

Of course, when considering a non-finite domain, enumerating all allowed combinations is impossible. In such cases, representing a constraint as a relation may not be only a concise representation, but the only one. Consider a simple constraint like *being even* in Example 2.3. Here, we define the constraint using the modulo notation. This modulo notation is a common shorthand used in modular

<sup>5</sup>Domain here should be understood in its mathematical definition, as a set of values.

arithmetic, which is part of the standard definitions of the naturals, or  $D_1$  in the example.

**Example 2.1: Constraint as an enumeration**

Let  $D = \{D_1, D_2\}$ ,  $D_1 = \{a, b\}$  and  $D_2 = \{c, d\}$ . Table 2.5 defines a constraint  $C_1$  which is violated for the combinations  $(a, c)$  and  $(a, d)$ .

Table 2.5: Truth Table of  $C_1$

| Value in $D_1$ | Value in $D_2$ | $C_1$   |
|----------------|----------------|---------|
| $a$            | $c$            | $\perp$ |
| $a$            | $d$            | $\perp$ |
| $b$            | $c$            | $\top$  |
| $b$            | $d$            | $\top$  |

**Example 2.2: Constraint as a relation**

The same constraint  $C_1$  as presented in Example 2.1 can be expressed as a Boolean relation, as follows:

$$C_1 : D_1 \times D_2 \rightarrow \{\top, \perp\}$$

$$(x_1, x_2) \mapsto x_1 \neq a$$

**Example 2.3: Being even**

Let  $D = \{D_1\}$  and  $D_1 = \mathbb{N}$ . Let  $C_1$  be a constraint such that  $X_1$  must be even.

$$C_1 : D_1 \rightarrow \{\top, \perp\}$$

$$x_1 \mapsto (x_1 \equiv 0[2])$$

All the interest of the CSPs would be to have an automatic resolution tool, efficient (in terms of duration), capable of solving any CSP. For the moment, this is not the case. On the other hand, categories of CSPs have been identified, each rich enough to solve many problems, and each using its own method of resolution. In this thesis, we will only mention three of these categories: ILP problems, in Subsection 2.4.1, Boolean Satisfiability (SAT) problems, in Subsection 2.4.2, and Satisfiability Modulo Theories (SMT) problems, in Subsection 2.4.1. We have named these categories according to the resolution tools that are dedicated to them.



### 2.4.1 The ILP Model

ILP problems are problems whose variables are integers, and the goal is to optimize a vector of these variables under a number of constraints. The constraints themselves take the form of inequalities. It turns out that the ILP is used in the majority, if not all, of the work listed around the VNE problem.

ILP problems are part of linear optimization problems, where the variables are real. In practice, ILP problems are harder than linear optimization problems simply because of passing real variables to integer variables. In the same way, there is a sub-category to the problems in ILP, linear problems with binary variables. Binary linear problems are harder than ILP problems, simply because of the passage of integer variables to binary variables. VNE is such a binary linear problem. Indeed, the variables of the VNE problem in its classical formulation are Boolean variables which indicate if such virtual element is associated with such physical element. The other problem data, be it the physical resource capabilities or the capacity requirements within the queries, are rather to be seen as constants. They are known at the time of resolution of the problem.

The reader can refer to Table 2.1 to identify the works proposing an ILP formulation to solve the VNE problem. These works are Rost and Schmid (2018), Alaluna et al. (2017), Bays et al. (2016), Wang et al. (2015), Liu et al. (2014), Chau and Wang (2014), Riggio et al. (2013), and Papagianni et al. (2013). In some cases, the authors propose also a heuristic resolution. These works are Dietrich et al. (2015), Nonde et al. (2015), Bellavista et al. (2015), Zhang et al. (2014), Mehraghdam et al. (2014), Bays et al. (2014), Rahman and Boutaba (2013), Di et al. (2013), and Houidi et al. (2011).

We give in Example 2.4 an ILP formulation, adapted from Rost and Schmid (2018). Before analyzing it, we want to emphasize that this ILP formulation is not general, but sufficient to understand most other ILP formulations for the VNE problem. Depending on the features the researchers want to add, the following equations may be slightly different. The formulation given by Rost and Schmid (2018), for instance, focuses on only one virtual network request, described as a directed graph  $(N^R, L^R)$ , and on only one InP, described as a directed graph  $(N^S, L^S)$ . Both the virtual network request and the substrate are weighted. Substrate resources (nodes and links) have a limited capacity, modelled as an integer, which can be accessed through the *offer* function. Virtual resources (nodes and links) have a capacity request, modelled as an integer, which can be accessed through the *dem* (standing for “demand”) function. Besides, for Rost and Schmid (2018), every substrate node can host a virtual node.

The objective chosen by Rost and Schmid (2018) is to embed the virtual network request, and corresponds to Equation (2.1). It is modelled with a decision variable, *isEmbedded*, which is an integer in  $\{0, 1\}$ . If it equals 0 (resp. 1), the

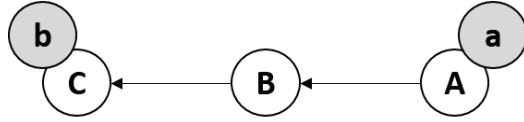


Figure 2.1: Flow Conservation Rule Illustration

Table 2.6: Flow Conservation Analysis for Figure 2.1

| $j \in N^S$ | $\sum_{k \in Egress(j)} l_{i,k}$ | $\sum_{k \in Ingress(j)} l_{j,k}$ | $n_{a,j} - n_{b,j}$ |
|-------------|----------------------------------|-----------------------------------|---------------------|
| <b>A</b>    | 1                                | 0                                 | 1                   |
| <b>B</b>    | 1                                | 1                                 | 0                   |
| <b>C</b>    | 0                                | 1                                 | -1                  |

virtual network request is not embedded (resp. embedded). As the objective is to maximize this variable, the ILP solver will always try to embed the virtual network request.

The substrate resources allocated for the virtual resources are given through two sets of unknowns, namely  $n_{ij}$  and  $l_{ij}$ . In this thesis, by convention, we use  $i$  to represent a virtual resource, and  $j$  to represent a substrate resource.

The  $n_{ij}$  represent the virtual node mapping. It is defined for  $i$  in  $N^R$ , and  $j$  in  $N^S$ . Its value is in  $\{0, 1\}$ , like *isEmbedded*.

The  $l_{ij}$  represent the virtual link mapping. It is defined for  $i$  in  $L^R$ , and  $j$  in  $L^S$ . Its value is in  $\{0, 1\}$ , like *isEmbedded*.

Equation (2.2) states that, given a substrate resource, the sum of the virtual node mapping unknowns over all virtual resources must be equal to the embedding decision variable. To analyze this equation, let first suppose that *isEmbedded* equals 1. Then, we have a sum of unknowns in  $\{0, 1\}$  which must be equal to 1. This means that *at least* and *at most* one unknown should be equal to 1, the others being equal to 0, for the given substrate resource. The equation then actually states that the virtual node mapping is an n-1 mapping: all virtual resources should be embedded, and each in one substrate resource. When *isEmbedded* equals 0 however, all unknowns for all substrate resources must be 0, which is also consistent with the fact that no substrate resource is allocated when the virtual network is not embedded.

Equation (2.3) is a rather complex equation which can be better understood as a flow conservation rule. The idea is that the substrate links (exceptionally denoted  $k$  here) allocated for the virtual link  $i$  must chain together to form a continuous path. Besides, this path should go from the substrate node allocated for the head node of the virtual link, to the substrate node allocated for the tail node of the virtual link. More precisely, the  $n_{i_h,j} - n_{i_t,j}$  part can only take three values: 1 when  $j$  is the host of  $i_h$ , -1 when  $j$  is the host of  $i_t$ , and 0 when  $j$  hosts neither, or both. The other part relies on the definition of *Ingress* and *Egress*.

$Egress(j)$  (resp.  $Ingress(j)$ ) is the set of edges whose tail node is (resp. whose head node is)  $j$ . Then the  $\sum_{k \in Ingress(j)} l_{i,k} - \sum_{k \in Egress(j)} l_{j,k}$  part only counts the number of egressing links minus the ingressing links. To have a path, which is a chain of edges, the equation only states that the substrate node hosting the head node (resp. the tail node) of the virtual link should have one more egress (resp. ingress) link allocated for the virtual link, and that all other substrate nodes should have as many egress links as ingress links allocated for the virtual link.

To illustrate this equation, consider Figure 2.1, associated with Table 2.6. Figure 2.1 represents a virtual link  $(\mathbf{a}, \mathbf{b})$ , for which a substrate path has been selected, going through substrate nodes  $\mathbf{A}$ ,  $\mathbf{B}$  and  $\mathbf{C}$ . Table 2.6 shows the calculation of Equation (2.3) for each of those substrate nodes. Consequently, Equation (2.3) simply describes a n-n mapping for the virtual links.

Finally, Equation (2.4) and Equation (2.5) only mean that the sum of all the demands from the virtual resources allocated to a given substrate resource must be lower than the substrate resource capacity. The difference between these equations is that Equation (2.4) use node variables, while Equation (2.5) use edge variables.

#### Example 2.4: An ILP formulation

The following ILP formulation is adapted from Rost and Schmid (2018), with the notations used in this thesis, especially in Chapter 3. Please refer to the table of all notations starting on page 137.

$$\max isEmbedded \quad (2.1)$$

$$\sum_{j \in N^S} n_{ij} = isEmbedded, \quad \forall i \in N^R \quad (2.2)$$

$$\sum_{k \in Egress(j)} l_{i,k} - \sum_{k \in Ingress(j)} l_{j,k} = n_{i_h,j} - n_{i_t,j}, \quad \forall i = (i_h, i_t) \in L^R, j \in N^S \quad (2.3)$$

$$\sum_{i \in N^R} dem(i) \times n_{ij} \leq offer(j), \quad \forall j \in N^S \quad (2.4)$$

$$\sum_{i \in L^R} dem(i) \times l_{ij} \leq offer(j), \quad \forall j \in L^S \quad (2.5)$$

This precision on the definition of the VNE problem allows us to see it in another form, that is to say no longer formulated in ILP, but formulated in SMT.

Before explaining what SMT is, we will explain the category of problems in SAT. It is this category that has historically led to a resolution method for SMT problems.

### 2.4.2 The SAT Model

#### Definition 2.3: *NP*-hardness, *NP*-completeness

*NP*-hardness and *NP*-complete are two concepts from the theory of complexity in mathematics. The theory of complexity tries to evaluate and classify the complexity of different problems. The two main classes are *P* and *NP*. The *P* class contains all problems that can be *solved* in polynomial time. Being solved in polynomial time means that there is an algorithm such that the time to solve the problem is expressed as a polynomial function on the input of the algorithm. It is said that *P* problems are easy to solve.

The other class, *NP*, contains all problems that can be *verified* in polynomial time. Being verified in polynomial time means that there is an algorithm such that the time to verify that a given input is a solution of the problem is expressed as a polynomial function on the input of the algorithm. It is said that *NP* problems are easy to verify.

Both definitions are relative to the existence of such algorithms. A problem may change of class if a better algorithm is found.

The equivalence between the *NP* class and the *P* class is an important question, often summarized as asking if  $P = NP$  is true, false, or non-demonstrable. Actually, many security algorithms rely on the  $P \neq NP$  hypothesis, that is: there are problems which are easy to verify, but not easy to solve.

The *NP*-hard and the *NP*-complete classes are derived from the *NP* class as follows. The *NP*-complete class contains the hardest problems in *NP*. In other words, it is a subset of *NP*.

The *NP*-hard class, for its part, contains all problems that are at least as hard as the hardest *NP* problem. Consequently, it encompasses the *NP*-complete problems and other problems outside the *NP* class.

The problems in SAT are CSPs where the variables are Boolean, and where the constraints are expressed using the propositional logic. Propositional logic is based on a number of operations, which are disjunction, conjunction, and negation. It thus makes it possible, typically, to model logic gates, and hence electronic circuits.

The interest of problems in SAT is that they are based on *the* SAT problem, which is a very important problem in complexity theory. The SAT problem is a

decision problem, which, given a set of constraints in propositional logic, aims to find out if there is a solution that does not violate any constraint. This problem is the first to have been proved to be *NP*-complete (see Definition 2.3), thanks to Cook–Levin’s theorem. In particular, it has been shown that the SAT problem is an *NP* problem, and that *any NP* problem can be *reduced* to the SAT problem.

One main consequence of this theorem is that, unless  $P = NP$ , we do not have an algorithm that can be used to solve *every* SAT problem instance in polynomial time. There is no “one-size-fits-all”. As such, the state of the art algorithm to resolve SAT problems has an exponential complexity of  $O(2^n)$ .

However, this does not mean that every problem instance is solved with an exponential time. First, we may have efficient heuristics, which are dependent on the problem instance. Second, the exponential time may be met for very specific problem instances, which are not used in practice.

The Davis–Putnam–Logemann–Loveland (DPLL) algorithm, developed by Davis et al. (1961), is originally a backtracking brute-force algorithm. What is important in the DPLL algorithm is that it is also a state machine, with an initial state, a final state, a fail state, transition states, and a set of transition functions. Every state except the fail state are in the form of a model and a formula. The model is a partial assignment of each variable to a value. The formula is a proposition in the form of a conjunctive normal form, which is a standard form to represent a set of propositions. This form means that we rewrite the formula as a conjunction of disjunctions of literals, where a literal is either a variable, its negation, the truth value, or the false value. The model has a relation with the formula, in the sense that if we assign the variables in the formula like given in the model, then the formula is true. For its part, the fail state indicates that the formula given originally as an input cannot be satisfied.

From the initial state, the algorithm tries to infer the variable values (either true or false) from the logical propositions. To do so, it runs a series of “unit propagation” function. The unit propagation is called as such because it is able to propagate a variable value from a disjunction to all other disjunctions, and infer new variable values from those disjunctions. However, when the algorithm cannot determine a value, it will decide to assign a value to some variable. When there is a conflict, the algorithm backtracks to its latest decision. It then takes the opposite decision.

Each time the algorithm forces a variable to be assigned to a value, it adds it to the model. If the algorithm succeeds, the model is an example of an assignment that makes the input formula true. Otherwise, the algorithm is in the fail state, and the model can not be exploited.

Nowadays, the DPLL algorithm is extended to support backjumping, also called non-chronological backtracking. This latter means that instead of backtracking to the latest decision, the algorithm will try to analyze the decision that

leads to the conflict. The conflict resolution is based on the construction of an implication graph.

### 2.4.3 The SMT Modeling

The SMT method focuses on all CSPs such that variables are part of some theories, and constraints are expressed under the grammar of those theories. In essence, it is an extension of SAT. The resolution of such problems relies on the DPLL algorithm from a SAT solver, communicating with a theory checker for each theory available in the SMT solver. The whole algorithm is then called DPLL Modulo Theories (DPLL(T)), and was developed by Ganzinger et al. (2004). The variables are typed, and treated with the axioms given their respective theory. De Moura and Bjørner (2011) describe SMT as a way to express a CSP with a rich mathematical language, rather than only propositional logic.

The SMT method has been applied in various fields, and acquired momentum especially for security purpose regarding system verification, as illustrated by Katz et al. (2017) for verification of deep neural networks, by Delmas et al. (2017) in safety, and by Shuyuan Zhang et al. (2012) for firewalls.

Basically, it is possible to express ILP problems into SMT. However, regarding performances, it may be better to use an ILP solver when the CSP can be reduced to an ILP problem, because the simplex algorithm is in general more efficient than the DPLL(T) algorithm.

For this thesis, although we propose a general requirement model, our original use case is to enable tenants to enumerate the locations they allow or the vendors they allow for their virtual resources. However, we failed to model those requirements within the ILP formulation. This failure is not due to those two requirements in particular. The reason is more general, because those two requirements imply a set structure (the set of authorized locations, the set of authorized vendors), which is very different from the integer structure. Therefore, we also failed to model any requirement using a set structure within the ILP formulation.

The difference between the set structure and the integer structure is that this latter has a total order, while sets do not. In other words, we can always compare any pair of integers (and saying which is lower or equal), but we cannot compare every pair of sets. We can only compare sets based on which contains the other: “authorizing vendor X and Y” implies “authorizing vendor X”, but we can say nothing about a pair of sets like “authorizing vendor X” and “authorizing vendor Y”. Hence, in this thesis, we decide to seek a CSP formulation that would support the set structure, and we do not rely on the ILP formulation.

The description we give of the DPLL(T) algorithm is very high-level. Indeed, there are different variations which are investigated, in order to make it even more efficient. Besides, not all those variations support the same theories. Heiz-

mann et al. (2018) organized the 13th International Satisfiability Modulo Theories Competition (SMT-COMP 2018). The four first best scores were assigned to CVC4 of Barrett et al. (2011), z3 of Moura and Bjørner (2008) (although not competing), Yices of Dutertre (2014) and SMTInterpol of Christ et al. (2012). In this thesis, we use the z3 solver, as it is one of the best solvers, supports a wide range of theories, and supports a Python API.

Although powerful, SMT solvers are not omnipotent. They are very dependent on the CSP they are solving—and by this sentence, we mean that they are very dependent on the way the user models the problem. In other words, translating a real world problem into a CSP is non-trivial, and we may have different alternatives. We can always imagine to use other theories, use less constraints, or even use a totally different set of constraints.

Despite this degree of freedom in the expression of the problem, there is always one property that we must ensure: that the problem is decidable. This property is key to ensure that the SMT solver will always answer. Otherwise, it may not terminate, or terminate but returning a meaningless result (typically “unknown”). In such cases, we have different options: re-run the solver (as the resolution is not fully deterministic), hoping to get a meaningful result; analyzing what makes the problem non-decidable, and circumvent it; or get rid of the SMT solver and use a dedicated algorithm.

## 2.5 Conclusion

This chapter presented several state-of-the-art works around the VNE problem. As a whole, it illustrated that it is a long-studied problem, with several ramifications in computer science, and a close relation with virtualization technology. It is then no surprise that new virtualization technologies such as SDN and NFV are also covered.

Our focus nevertheless was security, which can be mainly divided into two aspects: security for the tenants, and security for the InPs. In this thesis, we understand, from the security for the tenants, that security cannot be described as a single requirement. The needs of the tenants, especially in a 5G context where they may come from very different environments (industrial, healthcare, vehicle, cities), with little knowledge about how a network is operated, may therefore be very vast. Besides, in the same way they may have little knowledge about how a network is operated, the InPs may learn a lot from the different environments the tenants are dealing with.

This first idea leads us to propose a different VNE formulation, which should be able to treat a variety of security-oriented requirements, along with QoS requirements. This VNE formulation will leverage SMT.

We also propose, inspired by the security for the InPs, to follow the limited information disclosure paradigm. It is a reasonable assumption, as first, we expect multiple InPs to participate into the 5G network slicing, while being reluctant to share such sensitive information as their resources, their topologies, and their pricing policy.





## CHAPTER 3

---

### Security Constraint Model

This chapter presents our attribute model for the security-aware Virtual Network Embedding (VNE) in the 5G context. The goal of this model is to formalize the attribute concept, which will enable tenants and Infrastructure Providers (InPs) to agree upon what is offered, what is demanded, and how to fulfill a demand.

There is a close connection between the concept of demand and offer as used in this thesis, and the concept of demand and offer in economics. The idea in this thesis, though, is that we identify the components of the demand of the tenant, as well as the components of the offer from the InPs, and formalize rules that will tell how to automatically satisfy the tenant. Especially, we want to formalize such rules even for security needs. For this reason, in this chapter, we describe attributes in general terms, and provide security-oriented examples of attributes, which are meaningful for our use case and serve for the evaluation of this work.

The reader may notice that our focus is the tenant, not the InPs. Nevertheless, InPs may have security requirements too, regarding the tenants and their virtual resources. We also enable such requirements, but they will be treated in Chapter 4. The ground reason is that InPs do not need a model to formalize their needs, nor to enforce them onto the tenants, as InPs, by definition, have full control over the physical resources, and can always decide, for whatever reason, to not provide them to a given tenant.

To tackle the security-aware VNE in the 5G context, our methodology is as follows. First, we provide an attribute model in Section 3.1. It is the core of our contributions. We then apply this model to a single domain scenario in Section 3.2. We choose this scenario as the VNE problem is easier to reason within this scenario than directly with the multi-domain scenario. Besides, the single domain formulation will serve as a reference to build the multi-domain formu-

lation in Chapter 4. Section 3.3 extends the attribute model to two new features, namely conditional requirement, and attribute-specific exclusion and collocation. The reason why we separate our model into these two sections is because the Satisfiability Modulo Theories (SMT) formulation from Section 3.1 is the one implemented and evaluated in Chapter 4, while the features proposed by Section 3.3 have not been evaluated.

## 3.1 The Attribute Model

This section presents our attribute model. For this purpose, Subsection 3.1.1 first provides a definition and a classification of the attributes. One key category of attributes is the active attributes, which is split into exclusion and tolerant attributes. For both attributes, we give their semantics and further classifications in respectively Subsection 3.1.2 and Subsection 3.1.3. The tolerant attributes themselves have the Subsection 3.1.4 dedicated to the description of their structure, as they are the attributes which we want to be adapted and instantiated to whatever tenant requirement. Subsection 3.1.5 addresses another key dimension of the active attributes, which is the difference between the node and the link attributes.

### 3.1.1 Attribute Classification

Requirements are expressed in natural language. They can be technical, but also legal, security-based, or compliance-related. We propose here a model to translate requirements into mathematical constraints. This requirement model is built around the concept of *attribute*.

Attributes describe resources, either demanded by the tenants or owned by the Infrastructure Providers (InPs). For instance, when a tenant expresses the requirement “my server must be in the US or in the EU”, it translates as the virtual resource *tenant’s server* having a *location* attribute, which value is *US or EU*. This value is the tenant’s server demand (in location). Likewise, when an InP declares that “my server is in the US”, it translates as the physical resource *InP’s server* having a *location* attribute, which value is *US*. This value is the InP’s server offer (in location).

Attributes enable resources to be offered by providers to satisfy a demand from a tenant. To tell whether the mapping of the virtual resource to the physical resource fulfills the tenant’s requirements, we use mathematical constraints. These mathematical constraints describe the relation between an offer, a demand, and an attribute. For our location attribute example, it is clear that offer and demand are related through the subset relation: the physical resource location must be within the set of authorized locations of the virtual resource.

Before introducing all the mathematical constraints, we now give a general overview of our notations for the attribute model. We denote as  $i$  a resource required by a tenant, and as  $j$  a resource provided by an InP. We denote as  $d_i$  all the demands for the resource  $i$ , and as  $o_j$  all the offers for the resource  $j$ . We denote as  $d_i[a]$  (resp.  $o_j[a]$ ) a term in  $d_i$  (resp.  $o_j$ ), corresponding to the attribute  $a$ . These terms have values in the set  $V(a)$ , which is the set of allowed values for the attribute  $a$ . We denote as  $x_{ij} \in \{\perp, \top\}$  an unknown which equals  $\top$  (meaning “true”) if resource  $i$  is allocated to resource  $j$ , and  $\perp$  (meaning “false”) otherwise. The notations used in this thesis are detailed in the Table of all Notations starting on page 137.

Although aiming to be general, our attribute model does not provide a unique structure for every attribute, because we identified some relations between them. These relations lead us to an attribute classification.

The main cause of the distinction between different attributes is what we call *attribute derivation*. This relation leads us to differentiate the active from the inactive attributes. A second cause of distinction is the time-dependence, which leads us to differentiate the mutable from the immutable attributes. The third cause of distinction is semantics, leading us to differentiate exclusion from tolerant attributes. All those distinctions can be combined together.

*Active vs. Inactive Attributes.* Attributes can be derived from other attributes. For instance, given an attribute  $a$ , we can build  $a^*$ , an attribute such that the tenant can enter several values from  $V(a)$  (empty set included). It is an ordered attribute such that  $V(a^*)$  is  $\text{Powerset}(V(a))$ , where  $\text{Powerset}(Set)$  stands for the powerset of set  $Set$ . If we consider an offer in  $a$ , its value is, by definition, in  $V(a)$ . At the tenant side, the demand is given with  $V(a^*)$ , whose value is, by definition, in  $\text{Powerset}(V(a))$ .

Comparing directly the offer and the demand is not possible. We need either to convert the demand into an element of  $V(a)$ , which is impossible, or to convert the offer into an element of  $\text{Powerset}(V(a))$ , which can be simply done by putting it into a singleton. This conversion actually describes an attribute derivation: we are actually defining the offer in  $a^*$ , so that this offer can be compared with the demand in the same attribute,  $a^*$ .

Yet, for the InP, it may be more convenient, more accurate, to define the offer in  $a$  in their substrate resources. For this reason, we enable both  $a$  and  $a^*$ . More specifically,  $a$  is referred to as an *inactive* attribute, while  $a^*$  is referred to as an *active* attribute. Active attributes enable to compare an offer and a demand. As such, they can be part of a VNE algorithm. Inactive attributes, for their part, do not enable such a comparison directly: their values, to be treated by a VNE algorithm, must be used within a derived and active attribute.

*Mutable vs. Immutable Attributes.* We further classify attributes regarding their time-dependency into *mutable* and *immutable* attributes. For this purpose,

we define a transition function  $Transition_a$  such that  $Transition_a(i_t, j_t) \in V(a)$ , where  $i_t$  and  $j_t$  are respectively a resource request  $i$  and a resource offer  $j$  at a given instant  $t$ . Then  $Transition_a(i_t, j_t)$  is the value of  $o_j[a]$  at next instant  $t + 1$ , or  $o_{j_{t+1}}[a]$ , and indicates how a substrate resource state changes over time.

$$Transition_a(i_t, j_t) = j_t[a] - i_t[a] \quad (3.1)$$

$$Transition_a(i_t, j_t) = Transition_a(j_t) \quad (3.2)$$

A typical immutable attribute for a substrate resource is its geographical location ( $Transition_a$  is then a constant function), whereas a typical mutable attribute is the link bandwidth or the link bit error rate ( $Transition_a$  is then a general function). More precisely, the link bandwidth is a *stateful* resource, whereas the link bit error rate is a *stateless* resource. Stateful attributes depend on the past substrate network state and on the tenant request. The expression of the  $Transition_a$  can vary, and is very dependent on the semantics of the attribute. For instance, we can represent the depletion of the offer  $j_t$  due to its consumption by  $i_t$  as a subtraction, as in Equation (3.1). On the contrary, stateless attributes do not depend on the tenant request, as shown in Equation (3.2). In that case,  $Transition_a$  can model a time-dependent physical process occurring on substrate resource  $j$ .

*Exclusion vs. Tolerant Attributes.* The original use case for this thesis was to enable two sets of requirements. One was to enable a tenant to enumerate a set of authorized locations, which will be referred to as the **!loc** attribute in the remainder of this thesis, as well as to enumerate a set of authorized vendors, which will be referred to as the **!ven** attribute. The other set of requirements encompassed exclusion relations between virtual resources of different tenants. Such exclusion relation were special, as they do not depend on the substrate resources, while the **!loc** and the **!ven** do. These latter are encompassed in the *tolerant* attribute category, opposed to the *exclusion* attribute category. These categories are more deeply described in the remainder of this thesis.

### 3.1.2 Exclusion Attributes: Semantics and Typology

*Exclusion* attributes enable tenants to forbid their virtual resources to share physical resources with other virtual resources, be them owned by the same tenant or by other tenants. In practice, when tenants have an exclusion requirement on one of their virtual resources (for instance, **a**), they enumerate the other virtual resources they want to exclude (**b** and **c** for instance). Exclusion is special, because it does not depend on any property of the substrate resources. Once we know that **a** excludes **b** and **c**, we can select any substrate resource for **b** (let it be

**B**), any substrate resource for **c** (let it be **C**, or **B** again); the only real constraint is that we must have another substrate resource, denoted **A**, distinct from **B** (and **C**), to host **a**. For this reason, we use a distinct model for exclusion attributes. Besides, exclusion by itself is a symmetric relation, as shown in Demonstration 3.1.

### Demonstration 3.1: Exclusion symmetry

Let  $Resources^V$  be the set of virtual resources, and  $Resources^P$  the set of physical resources. Let  $Exclude$  be the exclusion predicate, as a binary relation. It can be expressed as:

$$\begin{aligned} \forall (i, i') \in Resources^V \times Resources^V, i \neq i', \\ Exclude(i, i') = (\forall j \in Resources^P, x_{ij} \Rightarrow \neg x_{i',j}) \end{aligned}$$

By contraposition,  $x_{ij} \Rightarrow \neg x_{i',j}$  is equivalent to  $x_{i',j} \Rightarrow \neg x_{ij}$ . It follows that  $Exclude(i, i') \equiv Exclude(i', i)$ .  $\square$

We exhibit three exclusion attributes, namely **!exc-resources** (resource exclusion), **!exc-requests** (request exclusion), and **!exc-tenants** (tenant exclusion). They follow respectively Equation (3.3), Equation (3.4) and Equation (3.5). These three equations only differ in what the tenant expresses as the elements to be excluded, respectively resources, requests, and tenants. Each equation then derives exclusion relationships from those elements. An excluded tenant will lead its requests to be excluded, and an excluded request will lead its resources to be excluded. In other words, Equation (3.3) implies Equation (3.4), which implies Equation (3.5).

$$x_{ij} \Rightarrow \bigwedge_{i' \in d_i[!exc-resources]} \neg x_{i',j} \quad (3.3)$$

Equation (3.3) means that if  $i$  is allocated to  $j$ , then no excluded resource  $i'$  can be allocated to  $j$ , where  $d_i[!exc-resources]$  is the set of all resources  $i'$  which are excluded by the requester.

$$x_{ij} \Rightarrow \bigwedge_{\substack{R \in d_i[!exc-requests] \\ i' \in Resources^R}} \neg x_{i',j} \quad (3.4)$$

Equation (3.4) means that if  $i$  is allocated to  $j$ , then no virtual resource  $i'$  from any excluded request  $R$  can be allocated to  $j$ , where  $d_i[!exc-requests]$  is the set of all requests  $R$  which are excluded by the requester. The set  $Resources^R$

Table 3.1: Building Blocks of Some Tolerant Attributes

| Attribute   | $V(a)$                       | Ordering operator<br>(demand $\leq$ offer)                       | +                 | $m_a$         | $M_a$       |
|-------------|------------------------------|--|-------------------|---------------|-------------|
| <b>!mem</b> | $\mathbb{N} \cup \{\infty\}$ | Natural order  | Standard addition | 0             | $\infty$    |
| <b>!bw</b>  | $\mathbb{N} \cup \{\infty\}$ | Natural order  | Standard addition | 0             | $\infty$    |
| <b>!loc</b> | Powerset of locations        | $\supset$ (note the orientation: offer must be <i>in</i> demand) | $\cap$            | All locations | $\emptyset$ |
| <b>!ven</b> | Powerset of vendors          | $\supset$ (note the orientation: offer must be <i>in</i> demand) | $\cap$            | All vendors   | $\emptyset$ |
| <b>!ber</b> | Percentage                   | $\geq$ (note the orientation)                                    | min               | 100%          | 0%          |

represents the set of resources in a request  $R$ .

$$x_{ij} \Rightarrow \bigwedge_{\substack{\tau \in d_i[\text{!exc-tenants}] \\ R \in \text{Requests}_\tau \\ i' \in \text{Resources}^R}} \neg x_{i',j} \quad (3.5)$$

Equation (3.5) means that if  $i$  is allocated to  $j$ , then no virtual resource  $i'$  from any request  $R \in \text{Requests}_\tau$  from any excluded tenant  $\tau$  can be allocated to  $j$ , where  $d_i[\text{!exc-tenants}]$  is the set of all tenants  $\tau$  which are excluded by the requester. The set  $\text{Requests}_\tau$  represents the set of requests owned by a tenant  $\tau$ .

The reason why we define three distinct exclusion attributes is because the needs of the tenants may evolve over time. They may need new slices, new virtual resources. And of course, new tenants may appear. Our exclusion attributes capture those three layers. In practice, we apply these exclusion attributes for different cases.

Excluding resources from each other in a tenant's own request is useful if they carry data that are sensitive, see Equation (3.3). Limiting collocations in between the tenant's requests is useful when the tenant is a company, as one request may represent the company's intellectual property department slice, another the financial department slice, see Equation (3.4). Excluding tenants is useful when the other tenants are identified as rivals, see Equation (3.5). All those cases are driven by mistrust in slice isolation at the physical level.

To be more precise, a tenant may trust the resource isolation as enforced by the InPs for some resources, but not for some critical ones. In that case, this tenant may desire more control over the physical resources in which those critical resources are hosted, perhaps even to the point of allocating a dedicated physical resource for them.

### 3.1.3 Tolerant Attributes: Semantics and Typology

Aside from the exclusion attribute, we have *tolerant* attributes. We call them as such because they tolerate unspecified values for both InPs and tenants. We want the list of tolerant attributes to be extended freely by InPs. In this thesis, we provide a first list of tolerant attributes in Table 3.1. Table 3.1 also shows the building blocks of the corresponding mathematical structure. Besides, tolerant attributes are divided into *binding* and *non-binding* attributes.

*Binding* attributes result in constraints where the acceptance of a demand can impact the acceptance of another one. It is typically the case when we model an attribute where demands will consume an offer. Let `!mem` be the memory attribute, which is a binding attribute. Let  $d_1$  be a demand allocated to an offer  $o_j$ , that is, the variable  $x_{1,j}$  is true. This means that the demand  $d_1$  in `!mem` is lower than the offer  $o_j$  in `!mem` (that is,  $d_1[!mem] \leq o_j[!mem]$ ). Now, let  $d_2$  be another demand such that the demand  $d_2$  in `!mem` is lower than the offer  $o_j$  in `!mem`. The condition allowing to allocate  $d_2$  along  $d_1$  to  $o_j$  (that is, both variables  $x_{1,j}$  and  $x_{2,j}$  are true), is *stricter* than only having  $d_1[!mem] \leq o_j[!mem]$  and  $d_2[!mem] \leq o_j[!mem]$ , because the demand  $d_1$  in `!mem` already consumes the offer  $o_j$  in `!mem`. In other words, the demand  $d_2$  in `!mem` can only consume what remains of the offer  $o_j$  in `!mem`, that is,  $o_j[!mem] - d_1[!mem]$ . The aforementioned condition is then that the demand  $d_2$  in `!mem` should be lower than the remaining `!mem` of  $o_j$ , that is:  $d_2[!mem] \leq o_j[!mem] - d_1[!mem]$ , better rewritten as:  $d_1[!mem] + d_2[!mem] \leq o_j[!mem]$ .

$$\sum_{i \in Resources^V, x_{ij}=\top} d_i[a] \leq o_j[a] \quad (3.6)$$

Binding attributes follow Equation (3.6). This latter means that the sum of the demands  $d_i[a]$  of all the virtual resources  $i \in Resources^V$  that have been allocated to  $j$  must be lower than the offer  $o_j[a]$ . There is one such constraint per offer  $o_j$  and per attribute  $a$ .

$$x_{ij} \Rightarrow d_i[a] \leq o_j[a] \quad (3.7)$$

On the contrary, *non-binding* attributes result in constraints where each demand acceptance can be checked separately. It is typically the case of the bit error rate (BER) attribute. Let `!ber` be the BER attribute, which is a non-binding attribute. Let  $d_1$  be a demand allocated to an offer  $o_j$ , that is, the variable  $x_{1,j}$  is true. This means that the demand  $d_1$  in `!ber` is lower than the offer  $o_j$  in `!ber`, that is:  $d_1[!ber] \geq o_j[!ber]$  (note the orientation of  $\geq$ :  $d_i[a]$  here is the maximal accepted BER, as shown in Table 3.1). Now, let  $d_2$  be another demand such that the demand  $d_2$  in `!ber` is lower than the offer  $o_j$  in `!ber`. The condition allow-



ing to allocate  $d_2$  along  $d_1$  to  $o_j$  (that is, both variables  $x_{1j}$  and  $x_{2j}$  are true), is equivalent to have  $d_1[!ber] \geq o_j[!ber]$  and  $d_2[!ber] \geq o_j[!ber]$ . The aforementioned condition is then that any demand allocated to an offer  $o_j$  should be lower than the offer in  $!ber$ , better rewritten as:  $\min\{d_1[!mem], d_2[!mem]\} \geq o_j[!mem]$  (again, see Table 3.1 regarding the orientation of  $\geq$ ).

### 3.1.4 Tolerant Attributes: Structure

#### Demonstration 3.2

Let  $(W_a, \leq, +)$  be a commutative, partially ordered monoid. Let  $e$  be an *ex nihilo* element not in  $W_a$ . Let  $\leq'$  and  $+'$  be extensions of  $\leq$  and  $+$  such that:

$$\begin{aligned} \forall x \in W_a \cup \{e\}, x \leq' e \\ e +' x = e +' e = e \\ \forall x, y \in W_a, x \leq y \text{ is equivalent to } x \leq' y \\ x +' y = x + y \end{aligned}$$

Then, we have stability of  $+'$ :

$$\begin{aligned} \forall x, y \in W_a, \quad x +' y = x + y & \in W_a \subset W_a \cup \{e\} \\ \forall x \in W_a, \quad e +' x = e + e = e & \in W_a \cup \{e\} \end{aligned}$$

We have associativity of  $+'$  (demonstration is shortened thanks to commutativity assumption):

$$\begin{aligned} \forall x, y, z \in W_a, (x +' y) +' z &= (x + y) + z \\ &= x + (y + z) \\ &= x +' (y +' z) \\ \forall x, y \in W_a, (e +' x) +' y &= (x +' y) +' e = e \end{aligned}$$

We have reflexivity of  $\leq'$ :

$$\begin{aligned} \forall x \in W_a, x \leq' x &\Rightarrow x \leq x \\ e \leq' e \end{aligned}$$

We have antisymmetry of  $\leq'$ , as  $e$  is the greatest element by construction:

$$\begin{aligned}\forall x, y \in W_a, x \leq' y \wedge y \leq' x &\Rightarrow x \leq y \wedge y \leq x \Rightarrow x = y \\ \forall x \in W_a, x \leq' e \wedge e \leq' x &\Rightarrow e = x\end{aligned}$$

We have transitivity of  $\leq'$ . Let  $x, y, z \in W_a \cup \{e\}$  such that  $x \leq' y$  and  $y \leq' z$ .

- 1) if  $z = e$ , then  $x \leq' z$  is true
- 2) if  $z \neq e$ , then  $y \neq e$  hence  $x \neq e$  then  $x \leq z$  so  $x \leq' z$

And we have compatibility between  $\leq'$  and  $+'$ . Let  $x, y \in W_a \cup \{e\}$  such that  $x \leq' y$ .

We can show that:  $\forall t \in W_a \cup \{e\}, x +' t \leq' y +' t$ .

- 1) if  $y = e$ , we have  $\forall t \in W_a \cup \{e\}, x +' t \leq' e +' t$
- 2) if  $y \neq e$ , then  $x \neq e$ 
  - 2a) if  $t = e$ , we have  $x +' e \leq' y +' e$
  - 2b) if  $t \neq e$ , we have  $x + t \leq y + t$  hence  $x +' t \leq' y +' t$

Consequently,  $(W_a \cup \{e\}, \leq', +')$  is also a partially ordered monoid.  $\square$

More generally, we define an attribute  $a$  as a tuple  $(V(a), \leq, +, m_a, M_a, \times)$ . The building blocks of this structure for different tolerant attributes is shown in Table 3.1. Note that  $\times$  does not appear in this table. We actually introduce it later, and define it from the other building blocks.

We build this tuple from a commutative, partially ordered monoid denoted  $(W_a, \leq, +)$ . Monoids are structures such that the law is associative, and supports a neutral element. We denote  $m_a$  such a neutral element. We suppose that there exists in  $W_a$  an element which is both its greatest<sup>1</sup> and its absorbing element. This element is denoted  $M_a$ . Then  $V(a)$  is the positive cone of  $W_a$ , that is, the set of all values which are greater than  $m_a$ . It follows that  $m_a$  is both the least<sup>1</sup> and the neutral element in  $V(a)$ .

We show in Demonstration 3.2 that we can always extend a monoid to get the  $M_a$  element, and the **!mem** attribute in Table 3.1, is an example of a monoid  $(\mathbb{N}, \leq, +)$  we extend to get such  $M_a$  element, which is denoted as  $\infty$ .

<sup>1</sup> In a partially ordered structure, the greatest (resp. least) element (it is unique) is the element of the structure which is greater (resp. lower) than any other element of the same structure. They should not be confused with the definition of the maximum (resp. minimum), as those require a totally ordered structure.

**Definition 3.1: Ideal node and link**

An *ideal* node (or link) is a resource for which we do *not* apply any tolerant nor exclusion constraints. For a tolerant attribute  $a$ , this is done by using the value  $M_a$ . As exclusion attributes do not exhibit such a convenient value, we do not generate exclusion constraints from ideal nodes and links. In other words, they will accept whatever request they receive.

We also define an operation  $\times$  as a function which takes a boolean ( $x_{ij}$ ) and an attribute value ( $d_i[a] \in V(a)$ ) and returns another attribute value. It follows Equation (3.8).

$$x_{ij} \times d_i[a] = \begin{cases} d_i[a] & \text{if } x_{ij} = \top \text{ (} x_{ij} \text{ is true)} \\ m_a & \text{otherwise} \end{cases} \quad (3.8)$$

As a general note, we can also define the  $\sum$  symbol in Equation (3.6) with the law of the monoid, by applying recursively the associativity of this law, in the same way  $\sum$  is a recurrence over  $+$ . In this thesis though, all the binding attributes we present are derived from integers.

Among those building blocks,  $m_a$  and  $M_a$  happen to play a convenient role for both tenants and InPs.

The  $m_a$  value of tolerant attributes as well as the value  $\emptyset$  of the exclusion attributes can be used as a *default* value. In other words, when tenants fill their requests, they can skip some attributes, and focus only on those which are meaningful to them. The  $m_a$  value guarantees through Equation (3.6) and Equation (3.7) that *all* offers will be considered. For the exclusion constraint, the  $\emptyset$  value guarantees through Equation (3.3) that all offers *but* the ones we have excluded will be considered (since exclusion is a symmetric relation).

At the same time,  $m_a$  can be used by the InPs to indicate that the value for this attribute is unknown or that the attribute itself is unsupported. In that case, the  $m_a$  value guarantees through Equation (3.6) and Equation (3.7) that *only* demands which use  $m_a$  too will be considered. In other words, the equations follow the legitimate rule that demands with a meaningful value for the tenant (that is, different from  $m_a$ ) will not be allocated to offers with an unknown value or not supporting the attribute (that is, equal to  $m_a$ ).

For its part,  $M_a$  is a convenient way to implement a resource with an unlimited capacity (or a capacity which cannot be exhausted), while still complying with capacity rule (R6). Indeed, by definition, when  $o_j[a] = M_a$ , Equation (3.6) and Equation (3.7) are true. In other words, when  $o_j[a] = M_a$ , the offer can accept as many demands as desired. Note that unlike  $m_a$ , we do not have any equivalent of  $M_a$  for exclusion attribute.

However, disabling the exclusion attribute for some offers happens to be useful in some cases. These cases are actually described in Sections 4.1.1 and 4.1.2, when applying the attribute model to the multi-domain scenario. This leads us to the definition of ideal nodes and ideal links, given in Definition 3.1.

### 3.1.5 Node and Link Attributes: Semantics and Typology

#### Definition 3.2: Aggregation function

A *tolerant, link* attribute  $a$  is a tuple  $(V(a), \leq, +, m_a, M_a, \times, \cdot_a)$  where  $(V(a), \leq, +, m_a, M_a, \times)$  is an attribute, and  $\cdot_a$  is a binary relation called the aggregation function. This latter is used to define the offer  $o_j[a]$  for some link  $j$ , as follows. We denote as  $i_h$  and  $i_t$  the nodes at the tips of the link  $j$ .

$$\forall i = (i_h, i_t) \in L^S, o_j[a] := o_{i_h}[a] \cdot_a o_{i_t}[a]$$

The relation  $\cdot_a$  must have the following properties, with respect to the attribute  $(V(a), \leq, +, m_a, M_a, \times)$ .

- 1) neutral element:  $\forall v \in V(a), M_a \cdot_a v = v \cdot_a M_a = v$
- 2) absorbing element:  $\forall v \in V(a), m_a \cdot_a v = v \cdot_a m_a = m_a$
- 3) idempotence:  $\forall x \in V(a), x \cdot_a x = x$
- 4) commutative:  $\forall x, y \in V(a), x \cdot_a y = y \cdot_a x$
- 5) translation-invariance:  $\forall x, y, t \in V(a), x \leq y \Rightarrow x \cdot_a t \leq y \cdot_a t$

Note that the neutral (resp. absorbing) element of  $\cdot_a$  is the absorbing (resp. neutral) element of  $+$ .

Along the tolerant-exclusion distinction, we also distinguish *node* and *link* attributes. Node attributes are on tenant virtual nodes, and link attributes are on tenant virtual links. In other words, attributes are classified according to the tenant's viewpoint.

As shown in the Table of all Notations starting on page 137, we denote as  $Binding_N$  (resp.  $NonBinding_N$ ) the set of binding (resp. non-binding) node attributes, and as  $Binding_L$  (resp.  $NonBinding_L$ ) the set of binding (resp. non-binding) link attributes.

The semantics of some attributes may not involve a virtual node and a physical node, or a virtual link and a physical link, but a virtual link and physical nodes. In Table 3.1, we present such a *link* attribute (from the tenant's viewpoint), **!ven**, whose offer is related to the nodes at the tips<sup>2</sup> of the physical link.

<sup>2</sup>We use "tips" here instead of "ends", to not introduce confusion with "end nodes", already

For such attribute, in Equation (3.6) and Equation (3.7), we consider that  $o_j[a]$  is the result of some aggregation function denoted  $\cdot_a$ , as defined in Definition 3.2.

As an example, for the tolerant link attribute `!ven`, we define  $\cdot_{!ven}$  as the union ( $\cup$ ) of the node offers, for instance. The reader can easily verify that the union follows the properties given in Definition 3.2 as  $M_{!ven}$  is the empty set ( $\emptyset$ ),  $m_{!ven}$  is the set of all vendors, and the ordering operator is the containment ( $\supset$ ), as shown in Table 3.1.

All the attributes in Table 3.1 can be formulated in Satisfiability Modulo Theories (SMT). As usual, the integer attributes leverage integer theory, and the exclusion attribute leverage boolean theory. Our novelty is that the powerset attributes leverage bitvector theory. The bitvector theory is a convenient way to represent elements of a powerset  $Powerset(Set)$ , where  $Set$  is *finite*, because we can associate each bit to the presence/absence of a certain element of  $Set$ . Then for  $Set = \{a, b, c\}$  we can associate a sequence  $(a, b, c)$  that tells which bit corresponds to which element, and then  $\emptyset = 000$ ,  $S = 111$ ,  $\{a, c\} = 101$ . The inclusion operation,  $x \subset y$  (see Table 3.1), is then equivalent to the predicate  $(x = x \odot y)$  where  $\odot$  denotes the bitwise logical “and”.

## 3.2 Application to the Single Domain Scenario

This section applies the attribute model given in Section 3.1.1 to the single domain scenario. The single domain scenario corresponds to the original VNE problem, with a unique InP. Subsection 3.2.1 identifies the key rules of the VNE problem in such a scenario and compares it with the Integer Linear Programming (ILP) formulation we provided in Subsection 2.4.1. Subsection 3.2.2 provides the SMT formulation of the problem, while leveraging our attribute model. This formulation does not only serve as an illustration of how we can leverage our attribute model. It is also a key part of our methodology to solve the VNE problem in a multi-domain context. For this reason, it will be reused and adapted in Chapter 4.

### 3.2.1 VNE Description

Let us describe the VNE problem for a single domain slice embedding. The situation itself is represented in Figure 3.1. Let the substrate be a weighted graph modelling the infrastructure. Substrate nodes model compute resources, and substrate edges model network links. Their weights correspond to some capacity. In Figure 3.1, such weights are labeled as *cpu* (node weight) or *bw* (edge weight).

---

used in eligible substrate hosts rule (R3).

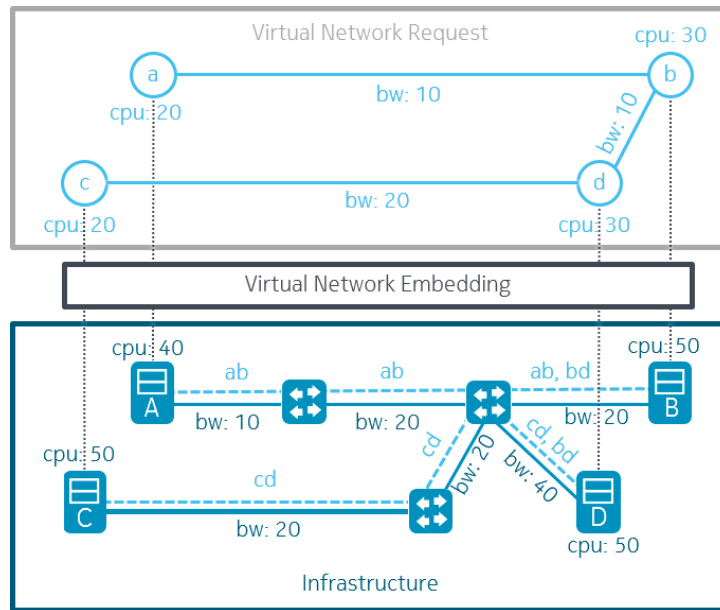


Figure 3.1: VNE illustration

A tenant willing to rent resources from this infrastructure must submit a virtual network request, which is also modelled as a weighted graph. The weights of virtual nodes and virtual links correspond to what the tenant demands. The goal of the problem is to embed this request into the substrate, under the following rules:

- R1 Node mapping rule.** Each virtual node is mapped to a *unique* substrate host.
- R2 Link mapping rule.** Each virtual link is mapped to a *unique* substrate path. A path is surely a set of contiguous links.
- R3 Eligible substrate hosts rule.** There are two kinds of substrate nodes: transit nodes (usually network devices) and end nodes (usually servers), also called substrate hosts in node mapping rule (R1). Only substrate hosts can host a virtual node.
- R4 Eligible paths rule.** Paths in link mapping rule (R2) must be drawn between substrate hosts (as per R3), which must be distinct.
- R5 Node-link mapping coordination rule.** The virtual nodes at the ends of each virtual link must be mapped to the hosts at the ends of the substrate path to which the virtual link is mapped.
- R6 Capacity rule.** The substrate node/links must accommodate all the demands of all the virtual node/links which are mapped to them.

In R1 and R2, we define as a *mapping* a relation between a tenant-desired virtual resource (a node or a link) and a physical resource (a *host* or a link). This mapping relation technically corresponds to different tasks. For instance, if virtual nodes represent virtual machines (VMs) and physical hosts represent servers, then the mapping represent the fact that the VM is installed in a server through a hypervisor. It is a n-1 mapping (cf. Definition 2.1), as illustrated by the association between the nodes  $\mathbf{a}$  and  $\mathbf{A}$  in Figure 3.1.

Other node mappings are possible. For instance, Bays et al. (2014) consider virtual nodes to be virtual routers, that must be instantiated in physical routers. In other words, in their work, every substrate node could host a virtual node. In this thesis, like in works from Alaluna et al. (2017) for instance, we consider that substrate nodes are classified into two categories: transit nodes, and end nodes; and that only end nodes can actually host virtual nodes. For this reason, we follow R3.

Meanwhile, the development of network hypervisors (see Blenk et al. (2016)) in the Software-Defined Network (SDN) context enables a single virtual SDN switch to be mapped to multiple physical SDN switches. As such, it is a n-n mapping (cf. Definition 2.1), which we decided to not address in this thesis. For this reason, we follow R1.

Link mapping is another n-n mapping, where the virtual link is mapped to a physical path, as illustrated by the association between the links  $ab$  and the links along the path between  $A$  and  $B$  in Figure 3.1. This raises the question of what happens when the path is empty, that is, when, technically, we are trying to emulate a link within a node. In this thesis, we consider that this is still a tedious task, and we will avoid it at the physical level. This is why, in the eligible paths rule (R4), we explicitly mention distinct hosts.

Besides, there is an important relation between the link and the node mapping, which is described in node-link mapping coordination rule (R5). R5 is actually very important for an exact mapping. With this rule, we consider that virtual links represent an authorized traffic from a source node to a destination node, and that it must flow through a continuous set of links from this source to that destination. It is typically this rule that becomes difficult to formulate when enabling a n-n node mapping. The reader can refer to Definition 2.1 for a reminder of the definition of n-n mapping and n-1 mapping.

The concept of demand (and offer) from R6 will be described in the next section. In Figure 3.1, the demands are formulated by the weights within the virtual network request, and the offers by the weights within the infrastructure graph. There are some relations indeed between them. For instance,  $ab$  and  $bd$  can be mapped on the link between the rightmost switch and the server  $B$  because the sum of their requested bandwidths ( $bw$ ) is lower than what the physical link provides.

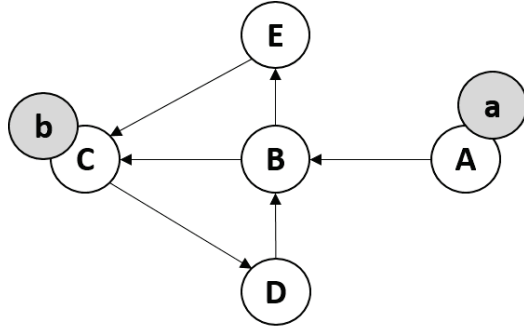


Figure 3.2: Flow Conservation Rule Limit

Note that the general ILP formulation we give in Example 2.4 follows some of these rules. Indeed, Equation (2.2) corresponds to R1, Equation (2.3) encompasses to R2 and R5, and Equation (2.4) and Equation (2.5) correspond to R6. The two missing rules are R3, because this formulation considers that every substrate node can host a virtual node, and R4, by definition of the flow conservation rule, from Equation (2.3).

### 3.2.2 SMT Formulation

We leverage our attribute-based requirement model to express the slice embedding problem as a Constraint Satisfaction Problem (CSP). In this section, we present these mathematical constraints for a limited case, with only one InP and only one domain, namely, the single domain scenario. We will build upon it in the next sections.

The CSP formulation serves as an interface between the human language, and the slice embedding algorithm. We consider this interface as an important feature for InPs, tenants, and system auditors, as it enables them to verify what the algorithm means, and how the attributes are applied. For this reason, we use an SMT formulation. SMT solvers take a CSP, and return one solution at a time. This solution is not necessarily optimal. To get an optimal solution, we must add an optimality constraint to the solver.

In this thesis, we do not focus on finding an optimal solution. Instead, we enumerate all of them, to verify that the constraints are applied as intended. The reason is that in VNE, the ILP formulations are including optimization features. For instance, paths are often expressed using a flow conservation rule, like Equation (2.3) in the general ILP formulation we give in Example 2.4. If we consider this constraint alone, it authorizes very convoluted paths, which can loop and go through the source and destination many times. A example of such a convoluted but legitimate path is given in Figure 3.2. The verification of Equation (2.3) applied on this example is given in Table 3.2.



Table 3.2: Flow Conservation Analysis for Figure 3.2

| $j \in N^S$ | $\sum_{k \in Egress(j)} l_{i,k}$ | $\sum_{k \in Ingress(j)} l_{j,k}$ | $n_{\mathbf{a},j} - n_{\mathbf{b},j}$ |
|-------------|----------------------------------|-----------------------------------|---------------------------------------|
| <b>A</b>    | 1                                | 0                                 | 1                                     |
| <b>B</b>    | 2                                | 2                                 | 0                                     |
| <b>C</b>    | 1                                | 2                                 | -1                                    |
| <b>D</b>    | 1                                | 1                                 | 0                                     |
| <b>E</b>    | 1                                | 1                                 | 0                                     |

The ground reason for this is that the ILP formulation we propose, taken from Rost and Schmid (2018) is wrong to some extent, but remains also a good example of why it is easy to be lost when encoding a problem as difficult as the VNE problem into the ILP paradigm. The problem raised by Figure 3.2 is that it is very difficult to give a meaning to such a convoluted path in a computer network. The first thing which visually appears in a set of two loops, **BCD** and **BCDEC**, which is also the very first thing we want to avoid for a traffic. The second thing is that it is a very suboptimal path: we traverse the node **B** multiple times, and consume capacity of some substrate links without any concrete purpose.

Nevertheless, the same equation appears in several VNE works, within ILP formulations. And each time, the use of this equation is legitimate; it is only wrong from an operational viewpoint in Rost and Schmid (2018), because the other works use the *price* as an objective, instead of the *isEmbedded* decision variable. In other words, as they seek to minimize the cost of the virtual network charged to the tenant, or to minimize the load of the infrastructure, solutions like in Figure 3.2 are never chosen.

In this thesis, as we leverage more theories than the integer one, it is important to be sure, before any optimization, that our problem is correctly formulated. It is for this reason that, although we take inspiration from many equations of the ILP formulation, we cannot follow them blindly, and especially, we cannot reuse the flow conservation rule. The semantical description of the single domain scenario is given in Section 3.2.1 as a set of rules to comply to. We now translate those rules into mathematical constraints, as follows.

The substrate, or physical infrastructure, or single domain, is denoted  $S$ . It is a directed, connected graph:  $S = (N^S, L^S)$ . Some nodes are hosts:  $H^S \subset N^S$ . Only them can host slice nodes. Besides, we have a set of (slice) requests, denoted *Requests*. Each request is owned by a tenant. Each request  $R$  is modeled as a directed graph  $R = (N^R, L^R)$ . We treat one request at a time, so we distinguish a request-to-embed, denoted *NewRequest*, from already embedded slices.

$$\bigvee_{j \in H^S} \left( n_{ij} \wedge \bigwedge_{j' \neq j} \neg n_{i,j'} \right) \quad \forall R \in Requests, i \in N^R \quad (C1)$$

$$\bigvee_{j \in P^S} \left( p_{ij} \wedge \bigwedge_{j' \neq j} \neg p_{i,j'} \right) \quad \forall R \in Requests, i \in L^R \quad (C2)$$

### Remark 3.1: Notations

As we consider a directed graph, we now want to emphasize the notation for edges. Given two nodes  $\mathbf{A}$  and  $\mathbf{B}$ , the edge whose head is  $\mathbf{A}$  (resp.  $\mathbf{B}$ ) and tail is  $\mathbf{B}$  (resp.  $\mathbf{A}$ ) is denoted  $(\mathbf{A}, \mathbf{B})$  (resp.  $(\mathbf{B}, \mathbf{A})$ ). Conversely, given an edge denoted  $e$ , its head is denoted  $e_h$  and its tail is denoted  $e_t$ . The head and the tail of the edge are called the *tips* of the edge.

$$(n_1, (n_1, n_2), n_2, \dots, n_l, (n_l, n_{l+1}), n_{l+1}) \quad (3.9)$$

Besides, we want to emphasize the notation for paths. A path is a sequence alternating nodes and edges, as depicted in Equation (3.9), where the  $n_i$  for  $i \in [1, l + 1]$  are the nodes along the path, and the  $(n_i, n_{i+1})$  for  $i \in [1, l]$  are the edges along the path. The number  $l$  represents the length of the path. The node  $n_1$  is called the node at the beginning of the path, or *head* to make a parallel with the edge notation, and the node  $n_{l+1}$  is called the node at the end of the path, or *tail*, to make a parallel with the edge notation.

Conversely, given a path  $p$ , the set of nodes along the path  $p$  is denoted  $Along_N(p)$ , the set of edges along the path  $p$  is denoted  $Along_L(p)$ , the head of the path is denoted  $p_h$ , the tail of the path is denoted  $p_t$ , and the sequence  $(p_h, p_t)$  is also referred to as the tips of the path, denoted  $Tips_p$ .

Those notations are also present in the Table of all Notations starting on page 137.

Equation (C1) (resp. Equation (C2)) implements R1 (resp. R2), expressed as a one-and-only-one predicate over the  $n_{ij}$  (resp.  $p_{ij}$ ). The  $n_{ij}$  and the  $p_{ij}$  correspond to the unknowns  $x_{ij}$  as used in Section 3.1. As such,  $n_{ij}$  (resp.  $p_{ij}$ ) are Boolean variables which are true (denoted  $\top$ ) when a virtual node  $i$  (resp. a virtual link  $i$ ) is mapped to a substrate host  $j$  from the set of hosts  $H^S$  (resp. a path  $j$  from the set of paths  $\in P^S$ ), and false (denoted  $\perp$ ) otherwise. Especially,  $j \in H^S$  complies with R3, and  $j \in P^S$  complies with R4. We will give further details about  $P^S$  at the end of this section, as not all paths are actually candidates.

$$p_{ij} = \left. \begin{aligned} & (n_{i_h, j_h} \wedge n_{i_t, j_t}) \wedge \bigwedge_{j' \in \text{Along}_L(j)} l_{i, j'} \wedge \bigwedge_{j' \in L^S \setminus \text{Along}_L(j)} \neg l_{i, j'} \\ & \forall R \in \text{Requests}, i \in L^R, j \in P^S, i = (i_h, i_t), \text{Tips}_j = (j_h, j_t) \end{aligned} \right\} \quad (\text{C3})$$

The  $p_{ij}$  variables are derived from  $n_{ij}$  and  $l_{ij}$  through Equation (C3). It defines the link-to-path variables as the conjunction of mapping link tips to path tips (cf. R5), then mapping the link to the links along the path, and then not mapping the link to the other links in the infrastructure. As a reminder, the definition of  $\text{Along}_L(j)$  and  $\text{Tips}_j$  is given in the Table of all Notations starting on page 137. It is also given in Remark 3.1. We will give further details about  $P^S$  at the end of this section, as not all paths are actually candidates.

$$n_{i, \text{memorized}_N(i)} = \top \quad \forall R \in \text{Requests} \setminus \{\text{NewRequest}\}, i \in N^R \quad (\text{C4})$$

$$p_{i, \text{memorized}_P(i)} = \top \quad \forall R \in \text{Requests} \setminus \{\text{NewRequest}\}, i \in L^R \quad (\text{C5})$$

Equation (C4) (resp. Equation (C5)) tells that each virtual node  $i$  (resp. each virtual link  $i$ ) of the already embedded requests should be mapped to the physical node  $\mu_N(i)$  (resp. physical path  $\mu_P(i)$ ), where  $\text{memorized}_N$  (resp.  $\text{memorized}_P$ ) is a function that memorizes the mappings for the currently embedded slices.

$$\sum_{\substack{i \in N^R \\ R \in \text{Requests}}} n_{ij} \times d_i[a] \leq o_j[a] \quad \forall a \in \text{Binding}_N, j \in H^S \quad (\text{C6})$$

$$n_{ij} \Rightarrow d_i[a] \leq o_j[a] \quad \forall a \in \text{NonBinding}_N, R \in \text{Requests}, i \in N^R, j \in H^S \quad (\text{C7})$$

$$\sum_{\substack{i \in L^R \\ R \in \text{Requests}}} l_{ij} \times d_i[a] \leq o_j[a] \quad \forall a \in \text{Binding}_L, j \in L^S \quad (\text{C8})$$

$$l_{ij} \Rightarrow d_i[a] \leq o_j[a] \quad \forall a \in \text{NonBinding}_L, R \in \text{Requests}, i \in L^R, j \in L^S \quad (\text{C9})$$

Equation (C6) (resp. Equation (C7), Equation (C8), Equation (C9)) guarantee requirements based on tolerant attributes  $\text{Binding}_N$  (resp.  $\text{NonBinding}_N$ ,  $\text{Binding}_L$ ,  $\text{NonBinding}_L$ ). They express that the offer  $o_j[a]$  of an InP's resource  $j$  in a given attribute  $a$  is limited, and that the demands  $d_i[a]$  of the tenant resources  $i$  allocated to  $j$  are bounded by the offer. They comply with R6. Equation (C6) and Equation (C8) are based on Equation (3.6), using  $n_{ij}$  and  $l_{ij}$  instead of  $x_{ij}$ . They describe binding attributes. For their part, equations Equation (C7) and Equation (C9) are based on Equation (3.7), using  $n_{ij}$  and  $l_{ij}$  instead of  $x_{ij}$ . They describe non-binding attributes.

**Definition 3.3: Loop-free path**

A loop-free path from a *source* to a *target* in a graph is a path that traverse each node of the graph *at most* once when  $source \neq target$ . When  $source = target$ , and if there exists a self-loop ( $source, source$ ), there is a unique loop-free path between *source* and *source*. It is the sequence  $(source, (source, source), source)$ , which traverses the node *source* twice and no other node.

$$n_{ij} \Rightarrow \bigwedge_{i' \in d_i[!exc-resources]} \neg n_{i',j} \quad \forall R \in Requests, i \in N^R, j \in H^S \quad (C10)$$

$$n_{ij} \Rightarrow \bigwedge_{\substack{R' \in d_i[!exc-requests] \\ i' \in N^{R'}}} \neg n_{i',j} \quad \forall R \in Requests, i \in N^R, j \in H^S \quad (C11)$$

$$n_{ij} \Rightarrow \bigwedge_{\substack{\tau \in d_i[!exc-requests] \\ R' \in Requests_\tau \\ i' \in N^{R'}}} \neg n_{i',j} \quad \forall R \in Requests, i \in N^R, j \in H^S \quad (C12)$$

Equation (C10), Equation (C11), and Equation (C12) express node exclusion constraints. They are based respectively on Equation (3.3), Equation (3.4) and Equation (3.5), using  $n_{ij}$  instead of  $x_{ij}$ . They also comply with R6.

$$l_{ij} \Rightarrow \bigwedge_{i' \in d_i[!exc-resources]} \neg l_{i',j} \quad \forall R \in Requests, i \in L^R, j \in L^S \quad (C13)$$

$$l_{ij} \Rightarrow \bigwedge_{\substack{R' \in d_i[!exc-requests] \\ i' \in L^{R'}}} \neg l_{i',j} \quad \forall R \in Requests, i \in L^R, j \in L^S \quad (C14)$$

$$l_{ij} \Rightarrow \bigwedge_{\substack{\tau \in d_i[!exc-requests] \\ R' \in Requests_\tau \\ i' \in L^{R'}}} \neg l_{i',j} \quad \forall R \in Requests, i \in L^R, j \in L^S \quad (C15)$$

Finally, Equation (C13), Equation (C14), and Equation (C15) express link exclusion constraints. Likewise, they are based respectively on Equation (3.3), Equation (3.4) and Equation (3.5), using  $l_{ij}$  instead of  $x_{ij}$ . They also comply with R6.

While describing Equation (C2) and Equation (C3), we mentioned that not all paths are actually candidates. For this reason,  $P^S$  does not represent the set of all paths, but a subset of them: the set of what we call “loop-free paths”. Loop-free paths enable us to enforce R4. They are defined by Definition 3.3. They are indeed a special case of simple paths. For this reason, we can easily design a loop-free path generating algorithm from a simple path generating algorithm.

### 3.3 Model Refinement

In the earliest stage of this work, we envisioned two features that would be helpful when declaring a virtual network request. The first feature can be referred to as *conditional requirement*, and should help tenants to formulate trade-offs, which minimize the chances of their requests to be rejected by the system. Basically, a conditional requirement is divided into different alternatives. Each alternative is itself a set of requirements upon different resources. The conditional requirement is then met if at least one alternative is met.

A use case of the conditional requirement feature could be as follows. The tenant primarily asks for 100 of capacity (arbitrary unit), for a virtual node  $\mathbf{a}$ . The tenant also requires a security level of 3 (arbitrary level) for the same virtual node  $\mathbf{a}$ . In the same time, the tenant knows that it is possible to get the same level of security by running security software by himself. Then, alternatively, the tenant asks for 200 of capacity for the virtual node  $\mathbf{a}$ , the extra capacity being used to run the software. The resulting conditional requirement is given in Equation (3.10).

$$(d_{\mathbf{a}}[!cap] = 100 \wedge d_{\mathbf{a}}[!sec] = 3) \vee (d_{\mathbf{a}}[!cap] = 200) \quad (3.10)$$

In Equation (3.10), each equality associates a demand ( $d_{\mathbf{a}}[!cap]$  for instance) to a value (100 for instance). Actually, Alaluna et al. (2017) propose a similar grammar to declare the virtual network requests. To be more precise, the tenant declares a request using their grammar, and each possible alternative is converted into its own virtual network request. In other word, Equation (3.10) would be split among two different requests, one stating that  $\mathbf{a}$  should have 100 capacity and a security level of 3; and the other stating that  $\mathbf{a}$  should have 200 capacity. What we are proposing, is that the system will choose between the two alternatives. This is very different from the model in Section 3.1, in the sense that the system will assign itself some values to the demands. The assignment is only limited to the alternatives we provide.

$$\mu_N(\mathbf{a}) \neq \mu_N(\mathbf{b}) \quad (3.11)$$

$$o_{\mu_N(\mathbf{a})}[!loc] \neq o_{\mu_N(\mathbf{b})}[!loc] \quad (3.12)$$

The second feature can be called *attribute-specific exclusion* (or collocation). Contrary to conditional requirement, exclusion and collocation relations do not describe the virtual resources themselves, but a relation upon them. For instance, it enables to state that two virtual nodes  $\mathbf{a}$  and  $\mathbf{b}$  should be in a different host, as done in Equation (3.11). As a reminder,  $\mu_N$  is the function that assigns a host to each virtual node.

What we propose with *attribute-specific* exclusion (or collocation) is to apply it also when comparing offer values. Equation (3.12) is an example of an attribute-specific exclusion, which states that  $\mathbf{a}$  and  $\mathbf{b}$  should be in a different location. It reads as: the host of  $\mathbf{a}$ , which is  $\mu_N(\mathbf{a})$ , should have a location, which is denoted  $o_{\mu_N(\mathbf{a})}[\!|1oc]$ , different from the location of the host of  $\mathbf{b}$ .

It is also possible to combine the different features. For instance, we can require  $\mathbf{a}$  and  $\mathbf{b}$  to be in the same authorized locations,  $\{X, Y, Z\}$ , and that they should also be in distinct locations among the authorized locations, as in Equation (3.13).

$$o_{\mu_N(\mathbf{a})}[\!|1oc] \neq o_{\mu_N(\mathbf{b})}[\!|1oc] \wedge d_{\mathbf{a}}[\!|1oc] = \{X, Y, Z\} \wedge d_{\mathbf{b}}[\!|1oc] = \{X, Y, Z\} \quad (3.13)$$

The model presented in Section 3.1 does not encompass those two features, but we can build from it to encompass them. To do so, we leverage the non-interpreted function theory and the theory of arrays. These theories are described in Subsection 3.3.1 and Subsection 3.3.2 respectively. We then leverage them to encompass our two features. Subsection 3.3.3 presents how we inset conditional requirements into our model, and Subsection 3.3.4 presents how we inset attribute-specific exclusion and collocation into our model, along with conditional requirements.

### 3.3.1 The Theory of Non-Interpreted Function

#### Definition 3.4: Total function

A *total function*  $f$  is an application from a set  $X$  (also called the *domain*<sup>a</sup>) to a set  $Y$  (also called the *codomain*) such that it associates every element  $x$  in  $X$  to one element  $y$  of  $Y$ . It is said that a total function is defined for every of its input.

We also define the *image* of  $f$  as the set of all values  $y$  which are associated to at least one  $x$  in  $X$ . The codomain of  $f$  may be a superset of, or equal to, the image of  $f$ .

A *n-ary function*  $f$  is a total function whose domain is  $X_1 \times \cdots \times X_n$ , where  $\times$  is the Cartesian product. The (positive) number  $n$  represents the number of arguments of the *n-ary function*  $f$ . It is also called the *arity* of the function  $f$ . A *n-ary function* associates then a value  $y$  to each tuple  $(x_1, \cdots, x_n)$  where  $x_i \in X_i$  and so forth.

<sup>a</sup>Mathematical domain, not to confuse with a network domain.

The theory of non-interpreted functions is one of the most important theories for SMT. The functions the theory deals with are not computer theory functions,

which can have side-effect (such as raising an exception, returning no result, or even never terminate if it encounters an infinite-loop), but total functions. The definition of total functions is given in Definition 3.4, along with the definition of  $n$ -ary functions. A particular case of  $n$ -ary functions, the nullary functions, is defined in Definition 3.5, which comes from Burris (2012).

**Definition 3.5: Nullary function**

A *constant* is a 0-ary (or nullary) function.

The definition of nullary functions is an extension of the definition of  $n$ -ary functions. The extension is as follows. Let  $S$  be a set. Let  $n$  be a positive number. For  $n > 0$ , we define as  $S^n$  the product  $S \times \dots \times S$  where  $S$  is repeated  $n$  times.  $S^n$  is then a particular case of  $X_1 \times \dots \times X_n$  where  $X_1 = \dots = X_n = S$ .

For  $n = 0$  we define  $S^0$  as being the set containing the empty set,  $\{\emptyset\}$ . Let  $f$  be a function from  $S^0$  to  $Y$ . The only element in the domain of  $f$  is then  $\emptyset$ . It is associated with the value  $f(\emptyset)$ .

In practice, we substitute  $f(\emptyset)$  and  $f$ . For instance, let  $a = 2$ . We say that  $a$  is actually a nullary function from  $\{\emptyset\}$  to  $\mathbb{N}$  such that  $a(\emptyset) = 2$ .

It follows from Definition 3.4 and Definition 3.5 that every symbol used by the SMT grammar can be considered as a  $n$ -ary function.

The theory of uninterpreted functions is called so because those  $n$ -ary functions are not interpreted. This is what we do in Definition 3.4, in the sense that we declare some symbol  $f$ ,  $x$ ,  $y$ , without stating explicitly what  $x$  is, what  $y$  is, and how to compute  $f(x)$  from that  $x$ .

To present things differently, we do not say, for instance, that  $f(x) = 2 \times x + 1$ , where  $\times$  and  $+$  are the usual arithmetic operations.

The theory of uninterpreted functions used alone in SMT only allows symbols (representing  $n$ -ary functions), equalities ( $=$ ,  $\neq$ ) between those symbols, and logical operators ( $\wedge$ ,  $\vee$ ). Example 3.1 gives different examples of constraints with non-interpreted functions. We give those examples to show that even though the grammar is limited to equalities and symbols, it is very flexible and powerful.

**Example 3.1**

Let consider  $f$  a unary function from  $\mathbb{N}$  to  $\mathbb{N}$ , and  $a$  and  $b$  two natural constants. The following constraint can be expressed in SMT:

$$(a = b) \wedge (f(a) \neq f(b))$$

It means that we want a function  $f$  such that for two equal inputs  $f$  return a different value. If such a constraint is provided to an SMT solver,

it will obviously return that the constraint cannot be satisfied, by definition of total functions.

Chapoutot (n.d.) gives a more complex example (also unsatisfiable):

$$(f(f(f(a))) = a) \wedge (f(f(f(f(f(a)))))) = a) \wedge (f(a) \neq a)$$

Which can be solved as follows:

$$\begin{aligned} (f(f(f(a))) = a) \wedge (f(f(f(f(f(a)))))) = a &\Rightarrow (f(f(a)) = a) \\ (f(f(f(a))) = a) \wedge (f(f(a)) = a) &\Rightarrow (f(a) = a) \\ (f(a) \neq a) \wedge (f(a) = a) &\Rightarrow \perp \end{aligned}$$

In an SMT solver such as z3, it is also possible to use uninterpreted domains (or sorts in z3 terms) instead of actual domains like  $\mathbb{N}$ .

Let  $S$  be a set. Let  $x$  and  $y$  be two constants of  $S$ . Let  $f$  be a unary function from  $S$  to  $S$ . We use the following constraint:

$$(f(f(x)) = x) \wedge (f(x) = y) \wedge (x \neq y)$$

This constraint is satisfiable. Consequently, we can retrieve a model from the solver. The code snippet below represents an example of an obtained model when the constraint is implemented with z3.

```
>>> from z3 import *
>>> S = DeclareSort("S")
>>> f = Function("f", S, S)
>>> x = Const("x", S)
>>> y = Const("y", S)
>>> part1 = (f(f(x)) == x)
>>> part2 = (f(x) == y)
>>> part3 = (x != y)
>>> constraint = And(part1, part2, part3)
>>> s = Solver()
>>> s.add(constraint)
>>> s.check()
sat
>>> s.model()
[y = S!val!0,
```



```
x = S!val!1,
f = [S!val!1 -> S!val!0, else -> S!val!1]]
```

The last line must be read as follows. A solution to the constraint is to interpret  $y$  as being a value  $v_0$  of  $S$ , and  $x$  as being a distinct value  $v_1$  of  $S$ , and  $f$  a function that maps every input to  $v_1$  except for  $v_1$  which is mapped to  $v_0$ . What should be noted here is that when we declare a set (like  $S$ ), it is supposed to be infinite, hence ensuring that  $v_0$  and  $v_1$  both exist.

To model a finite set, we use the `EnumSort` constructor instead, which will tell the SMT solver how many distinct elements there are inside. For instance, let  $S = \{v_0, v_1\}$ .

```
>>> from z3 import *
>>> S, (v0, v1) = EnumSort("S", ["v0", "v1"])
>>> f = Function("f", S, S)
>>> x = Const("x", S)
>>> y = Const("y", S)
>>> part1 = (f(f(x)) == x)
>>> part2 = (f(x) == y)
>>> part3 = (x != y)
>>> constraint = And(part1, part2, part3)
>>> s = Solver()
>>> s.add(constraint)
>>> s.check()
sat
>>> s.model()
[y = v1, x = v0, f = [v0 -> v1, else -> v0]]
```

The model is essentially the same as in the infinite set example.

### 3.3.2 The Theory of Arrays

Arrays are one of the main structures in computer science. Tables in C or lists in Python are kind of arrays. Arrays are a convenient way to store data. They can be thought as functions from an index set (for instance, the naturals) to the data set. It is said that an array *stores* a value  $v$  at an index  $i$ , and that by *reading* the array at its index  $i$ , the value  $v$  can be retrieved. In Python, the dictionary type (`dict`) acts as an array where indices can be tuples, for instance. In other words, in the mathematical description of the arrays, the choice of the index is free. In particular, in computer science, we never use the naturals as the index set, as we are limited by the memory available to store the array, we use rather an interval.

Mccarthy (1962) propose a theory of arrays which is still widely used. Essentially, this theory reuses the theory of non-interpreted functions, and add to

it two main functions, namely *Read* and *Write*, as well as the axioms given in Equation (3.14), where  $a$  in an array,  $v$  a value,  $i$  and  $j$  two indices. The function *Read* has two parameters (the array, the index), and the function *Write* has three parameters (the array, the index, the value). For Python lists, *Read* would correspond to `__getitem__` (whose shortcut notation is  $v = l[i]$ ) and *Write* would correspond to `__setitem__` (whose shortcut notation is  $l[i] = v$ ).

$$(i = j) \Rightarrow \text{Read}(\text{Write}(a, i, v), j) = v \quad (3.14)$$

$$(i \neq j) \Rightarrow \text{Read}(\text{Write}(a, i, v), j) = \text{Read}(a, j) \quad (3.15)$$

Moura and Bjorner (2009) propose an extension of the theory of arrays which is implemented in `z3`. In this theory, arrays can come with a pre-defined default value, which is the value stored in the array at any index where we do not write. Such pre-defined default value is defined by the function *default*, which is added to the original theory of arrays, among other functions. Example 3.2 describes how the same problem can be modelled with variables, non-interpreted functions and arrays, with their respective benefits.

### Example 3.2

Let us assume we have a phenomenon which is very rare and very short, but we want to track it over time, so we measure its presence or absence over a long period of time. We want to use our measures as an input of a SMT solver to verify that the phenomenon obeys to some sets of physics equations.

We have multiple ways to model the measures. This example shows different approaches.

*Integer Theory.* Suppose we have  $N$  measures. For instance, we can create  $N$  variables  $x_i$  for  $i$  in  $[1, N]$ , and add to the solver a constraint that tells the value of the variable:  $x_i = \perp$  or  $x_i = \top$ .

The integer theory is a convenient way to model the measures, until we cope with a physics equation that says, for instance, that two consecutive measures cannot be true together, for instance:  $x_i \wedge x_{i+1} = \perp$ . We then have to create  $N - 1$  constraints, one for each value of  $i$ . The number of variables and constraints that a SMT solver can treat is not infinite. We currently have  $2 \times N - 1$  constraints. If  $N$  is very huge, it may be necessary to use another model.

*Non-Interpreted Function Theory.* Another way to model the measures is to let  $f$  be a function that assigns each time to a value. In other words, we have a function  $f$  such that  $f(i) = x_i$ . We would then have to create one constraint that tells, for each  $i$ , the value of  $f$ :  $f(i) = \perp$ , or  $f(i) = \top$ , and

one constraint for the physics equation:  $f(i) \wedge f(i + 1) = \perp$ . We then only have  $N + 1$  constraints.

*Array Theory.* Again,  $N + 1$  constraints may still be too much. A more precise analysis of the data may then be required. For instance, we supposed at the beginning that the phenomenon is very rare, so many measures will be false in practice. To model this situation, we can use an array  $a$  instead. We then have to add the following constraints:  $Default(a) = \perp$ ,  $Write(a, i, \top)$  for the few indices  $i$  for which we measure the presence of the phenomenon, and the physics equation as  $Read(a, i) \wedge Read(a, i + 1) = \perp$ . This means we have  $n + 2$  constraints, with  $n$  much lower than  $N$ .

As a conclusion for this example, the reader may notice that we use three different models for the data, each leveraging a different theory. No model is better than another in absolute; it really depends on the concrete problem to solve. Nevertheless, it is a good illustration of how the theory of non-interpreted functions and the theory of arrays can become useful in practice.

### 3.3.3 Conditional Requirements

To leverage the non-interpreted function theory, we declare the following functions. Equation (3.16) defines a function that assigns a substrate host to each virtual node of each request. Equation (3.17) defines the association between a virtual link and a set of substrate links. Equation (3.18) defines the function that assigns a substrate path to each virtual link of each request.

$$\mu_N: \bigcup_{R \in Requests} N^R \rightarrow H^S \quad (3.16)$$

$$\mu_L: \bigcup_{R \in Requests} L^R \times L^S \rightarrow \{\perp, \top\} \quad (3.17)$$

$$\mu_P: \bigcup_{R \in Requests} L^R \rightarrow P^S \quad (3.18)$$

Besides, we define for each node and link attribute  $a$  two functions  $D_a$  and  $O_a$ . Those functions serve to extend the definition of  $d_i[a]$  and  $o_j[a]$  respectively. In the model from Section 3.1,  $d_i[a]$  and  $o_j[a]$  can be considered as *constant*. In this section, what we propose is that  $D_a(i)$  (and  $O_a(j)$  for convenience) are *variables* instead. The values that  $D_a(i)$  (on which we focus) can take are still given by the tenant. We will explain shortly after how the tenant manipulate these functions.

$$\forall a \in Active_N, D_a: \bigcup_{R \in Requests} N^R \rightarrow V(a) \quad (3.19)$$

$$\forall a \in \text{Active}_N, O_a: H^S \rightarrow V(a) \quad (3.20)$$

### Grammar 3.1: Grammar for Conditional Requirement

The following grammar states that a request from a tenant is an *expression* made of conjunctions ( $\wedge$  operator, represented as AND below due to some L<sup>A</sup>T<sub>E</sub>X package limitations) and disjunctions ( $\vee$  operator, represented as OR below due to some L<sup>A</sup>T<sub>E</sub>X package limitations). The core is made of predicates in the form ( $D_a(i) = \text{value}$ ), where  $a$  is the name of an attribute (whose format is !<string>),  $i$  is a virtual resource (either a node or a link), and  $\text{value}$  is a string representing a value in  $V(a)$ .

The grammar itself is written in the Backus–Naur form.

$$\begin{aligned} \langle \text{expression} \rangle &::= \langle \text{expression} \rangle \text{ OR } \langle \text{addend} \rangle \mid \langle \text{addend} \rangle \\ \langle \text{addend} \rangle &::= \langle \text{addend} \rangle \text{ AND } \langle \text{multiplicand} \rangle \mid \langle \text{multiplicand} \rangle \\ \langle \text{multiplicand} \rangle &::= ( \langle \text{expression} \rangle ) \mid ( \langle \text{predicate} \rangle ) \\ \langle \text{predicate} \rangle &::= D_{\{ \langle \text{attribute} \rangle \}} ( \langle \text{resource} \rangle ) = \langle \text{value} \rangle \\ \langle \text{resource} \rangle &::= \langle \text{node} \rangle \mid \langle \text{link} \rangle \\ \langle \text{node} \rangle &::= \langle \text{string} \rangle \\ \langle \text{link} \rangle &::= ( \langle \text{string} \rangle , \langle \text{string} \rangle ) \\ \langle \text{attribute} \rangle &::= ! \langle \text{string} \rangle \\ \langle \text{value} \rangle &::= \langle \text{string} \rangle \mid \{ \langle \text{set} \rangle \} \\ \langle \text{set} \rangle &::= \langle \text{string} \rangle \mid \langle \text{set} \rangle , \langle \text{string} \rangle \\ \langle \text{string} \rangle &::= \langle \text{character} \rangle \mid \langle \text{string} \rangle \langle \text{character} \rangle \\ \langle \text{character} \rangle &::= \langle \text{letter} \rangle \mid \langle \text{digit} \rangle \\ \langle \text{letter} \rangle &::= a \mid \dots \mid z \\ \langle \text{digit} \rangle &::= 0 \mid \dots \mid 9 \end{aligned}$$

$$\forall a \in \text{Active}_L, D_a: \bigcup_{R \in \text{Requests}} L^R \rightarrow V(a) \quad (3.21)$$

$$\forall a \in \text{Active}_L, O_a: L^S \rightarrow V(a) \quad (3.22)$$

We define two functions to hold the exclusion constraints, namely  $\text{Exclude}_N$  and  $\text{Exclude}_L$ . More specifically, they are implemented as arrays, enabling us to apply *Read* and *Write* predicates on them.

$$\text{Exclude}_N: \bigcup_{R \in \text{Requests}} N^R \times \bigcup_{R \in \text{Requests}} N^R \rightarrow \{\perp, \top\} \quad (3.23)$$

$$\text{Exclude}_L: \bigcup_{R \in \text{Requests}} L^R \times \bigcup_{R \in \text{Requests}} L^R \rightarrow \{\perp, \top\} \quad (3.24)$$

We define also two collocation functions  $\text{Collocate}_N$  and  $\text{Collocate}_L$ . More specifically, they are implemented as arrays, enabling us to apply *Read* and *Write* predicates on them.

$$\text{Collocate}_N: \bigcup_{R \in \text{Requests}} N^R \times \bigcup_{R \in \text{Requests}} N^R \rightarrow \{\perp, \top\} \quad (3.25)$$

$$\text{Collocate}_L: \bigcup_{R \in \text{Requests}} L^R \times \bigcup_{R \in \text{Requests}} L^R \rightarrow \{\perp, \top\} \quad (3.26)$$

Finally, we define two functions to identify the ideal nodes and links in the substrate.

$$\text{Ideal}_N: \bigcup_{R \in \text{Requests}} N^R \rightarrow \{\perp, \top\} \quad (3.27)$$

$$\text{Ideal}_L: \bigcup_{R \in \text{Requests}} L^R \rightarrow \{\perp, \top\} \quad (3.28)$$

To specify the virtual network requests, we leverage a grammar which is similar to Alaluna et al. (2017), although our grammar must enable tenants to define more attributes. The grammar itself is given in Grammar 3.1.

Then we rewrite all our equations from Section 3.1, namely from Equation (C1) to Equation (C15), by applying the substitutions given by Equation (3.29), Equation (3.30), Equation (3.32), and Equation (3.33).

Equation (3.29) reads as follows. Every variable  $n_{ij}$  should be replaced by the predicate  $(\mu_N(i) = j)$ . They are equivalent:  $n_{ij}$  is true when virtual node

$i$  is allocated to substrate node  $j$ ;  $(\mu_N(i) = j)$  is true when the substrate node allocated to  $i$ , namely  $\mu_N(i)$ , is  $j$ .

Equation (3.30) reads as follows. Every variable  $l_{ij}$  should be replaced by the function call  $\mu_L(i, j)$ . They are equivalent:  $l_{ij}$  and  $\mu_L(i, j)$  are true when virtual link  $i$  is allocated to substrate link  $j$ .

Equation (3.31) reads as follows. Every variable  $p_{ij}$  should be replaced by the predicate  $(\mu_P(i) = j)$ . They are equivalent:  $p_{ij}$  is true when virtual link  $i$  is allocated to substrate path  $j$ ;  $(\mu_P(i) = j)$  is true when the substrate path allocated to  $i$ , namely  $\mu_P(i)$ , is  $j$ .

Equation (3.32) reads as follows. Every variable  $d_i[a]$  (resp.  $o_j[a]$ ) should be replaced by the function call  $D_a(i)$  (resp.  $O_a(j)$ ). They are equivalent:  $d_i[a]$  (resp.  $o_j[a]$ ) gets the value of the attribute  $a$  from the demand vector  $d_i[]$  (resp. the offer vector  $o_j[]$ ) of the virtual resource  $i$  (resp. of the substrate resource  $j$ ), while  $D_a(i)$  (resp.  $O_a(j)$ ) gets the value of the attribute  $a$  for the virtual resource  $i$  (resp. the substrate resource  $j$ ).

As the (mathematical) domains that we manipulate for our functions are finite, we can use SMT quantifiers, like *ForAll*, and apply it for variables like  $i$  and  $j$ . This is very different from Section 3.1, where the only variables are the  $n_{ij}$ , the  $l_{ij}$  and the  $p_{ij}$ . Those variables are also the *unknowns* of Section 3.1. In the current section, the  $n_{ij}$ , the  $l_{ij}$  and the  $p_{ij}$  are still part of the unknowns, but they are not the only variables, as  $i$  and  $j$  are also variables. Besides, for a given virtual resource  $i$  and a given attribute  $a$ ,  $d_i[a]$  is also an unknown, whose possible values are given by the tenant with Grammar 3.1.

$$n_{ij} \rightarrow (\mu_N(i) = j) \quad (3.29)$$

$$l_{ij} \rightarrow \mu_L(i, j) \quad (3.30)$$

$$p_{ij} \rightarrow (\mu_P(i) = j) \quad (3.31)$$

$$d_i[a] \rightarrow D_a(i) \quad (3.32)$$

$$o_j[a] \rightarrow O_a(j) \quad (3.33)$$

As  $\mu_N$  and  $\mu_P$  are total functions (see Definition 3.4), they already implement R1 (resp. R2). For this reason, we do not need Equation (C1) nor Equation (C2) anymore.

Equation (C3d), Equation (C4d), Equation (C5d), Equation (C6d), Equation (C7d), Equation (C8d), Equation (C9d) are all directly derived from the aforementioned substitutions.

$$\left. \begin{aligned}
 (\mu_P(i) = j) &\equiv (\mu_N(i_h) = j_h) \\
 &\wedge (\mu_N(i_t) = j_t) \\
 &\wedge \bigwedge_{j' \in \text{Along}_L(j)} \mu_L(i, j') \\
 &\wedge \bigwedge_{j' \in L^S \setminus \text{Along}_L(j)} \neg \mu_L(i, j') \\
 &\forall R \in \text{Requests}, i \in L^R, j \in P^S, i = (i_h, i_t), \text{Tips}_j = (j_h, j_t)
 \end{aligned} \right\} \text{(C3d)}$$

$$(\mu_N(i) = \text{memorized}_N(i)) \quad \forall R \in \text{Requests} \setminus \{\text{NewRequest}\}, i \in N^R \quad \text{(C4d)}$$

$$(\mu_P(i) = \text{memorized}_P(i)) \quad \forall R \in \text{Requests} \setminus \{\text{NewRequest}\}, i \in L^R \quad \text{(C5d)}$$

$$\sum_{\substack{i \in N^R \\ R \in \text{Requests}}} (\mu_N(i) = j) \times D_a(i) \leq O_a(j) \quad \forall a \in \text{Binding}_N, j \in H^S \quad \text{(C6d)}$$

$$(\mu_N(i) = j) \Rightarrow D_a(i) \leq O_a(j) \quad \forall a \in \text{NonBinding}_N, R \in \text{Requests}, i \in N^R, j \in H^S \quad \text{(C7d)}$$

$$\sum_{\substack{i \in L^R \\ R \in \text{Requests}}} \mu_L(i, j) \times D_a(i) \leq O_a(j) \quad \forall a \in \text{Binding}_L, j \in L^S \quad \text{(C8d)}$$

$$\mu_L(i, j) \Rightarrow D_a(i) \leq O_a(j) \quad \forall a \in \text{NonBinding}_L, R \in \text{Requests}, i \in L^R, j \in L^S \quad \text{(C9d)}$$

For the exclusion attributes, we propose a revision to our constraints from Section 3.1. This revision allows us to be transparent to our definition of ideal resources (see Definition 3.1), so that the constraints of this section can be reused for the single domain case and the multi-domain case.

Our revision holds into two sets of constraints. Equation (C10d), Equation (C11d) and Equation (C12d) each convert the exclusion attribute value from the tenant (resp. `!exc-resources`, `!exc-requests`, and `!exc-tenants`) into a constraint over the exclusion function `ExcludeN`. Likewise, Equation (C13d), Equation (C14d), and Equation (C15d) convert tenant exclusion attribute values into a constraint over the exclusion function `ExcludeL`.

$$\left. \begin{array}{l} \text{Write}(\text{Exclude}_N, (i, i'), \top) \\ \forall R \in \text{Requests}, i \in N^R, j \in H^S, i' \in D_{! \text{exc-resources}}(i) \end{array} \right\} \quad (\text{C10d})$$

$$\left. \begin{array}{l} \text{Write}(\text{Exclude}_N, (i, i'), \top) \\ \forall R \in \text{Requests}, i \in N^R, j \in H^S, R' \in D_{! \text{exc-requests}}(i), i' \in N^{R'} \end{array} \right\} \quad (\text{C11d})$$

$$\left. \begin{array}{l} \text{Write}(\text{Exclude}_N, (i, i'), \top) \\ \forall R \in \text{Requests}, i \in N^R, j \in H^S, \tau \in D_{! \text{exc-tenants}}(i), R' \in \text{Requests}_\tau, i' \in N^{R'} \end{array} \right\} \quad (\text{C12d})$$

$$\left. \begin{array}{l} \text{Write}(\text{Exclude}_L, (i, i'), \top) \\ \forall R \in \text{Requests}, i \in L^R, j \in L^S, i' \in D_{! \text{exc-resources}}(i) \end{array} \right\} \quad (\text{C13d})$$

$$\left. \begin{array}{l} \text{Write}(\text{Exclude}_L, (i, i'), \top) \\ \forall R \in \text{Requests}, i \in L^R, j \in L^S, R' \in D_{! \text{exc-requests}}(i), i' \in L^{R'} \end{array} \right\} \quad (\text{C14d})$$

$$\left. \begin{array}{l} \text{Write}(\text{Exclude}_L, (i, i'), \top) \\ \forall R \in \text{Requests}, i \in L^R, j \in L^S, \tau \in D_{! \text{exc-tenants}}(i), R' \in \text{Requests}_\tau, i' \in L^{R'} \end{array} \right\} \quad (\text{C15d})$$

The second set of constraints for our revision of the exclusion requirement encompass Equation (C16d) and Equation (C17d). Equation (C16d) is a relation between the node exclusion function  $\text{Exclude}_N$ , the ideal node function  $\text{Ideal}_N$ , and the mapping function  $\mu_N$ . It reads as follows: given two virtual nodes  $i$  and  $i'$ , and given a substrate host  $j$ , such that  $j$  hosts  $i$  (written  $\mu_N(i) = j$ ), and  $j$  is not an ideal node (written  $\neg \text{Ideal}_N(j)$ ); if  $i$  excludes  $i'$  (written  $\text{Write}(\text{Exclude}_N, (i, i'), \top)$ ), then  $j$  cannot host also  $i'$ .

Likewise, Equation (C17d) is a relation between the link exclusion function  $\text{Exclude}_L$ , the ideal link function  $\text{Ideal}_L$ , and the mapping function  $\mu_P$ .

$$\left. \begin{array}{l} (\mu_N(i) = j) \wedge (\text{Read}(\text{Exclude}_N, (i, i')) = \top) \wedge \neg \text{Ideal}_N(j) \Rightarrow \mu_N(i') \neq j \\ \forall (i, i') \in \bigcup_{R \in \text{Requests}} N^R \times \bigcup_{R \in \text{Requests}} N^R, j \in H^S \end{array} \right\} \quad (\text{C16d})$$



$$\left. \begin{aligned} \mu_L(i, j) \wedge (Read(Exclude_L, (i, i')) = \top) \wedge \neg Ideal_L(j) \Rightarrow \neg \mu_L(i', j) \\ \forall (i, i') \in \bigcup_{R \in Requests} L^R \times \bigcup_{R \in Requests} L^R, j \in L^S \end{aligned} \right\} \quad (C17d)$$

As we use arrays for  $Exclude_N$  and  $Exclude_L$ , we can define that when the tenant does not explicitly set an exclusion relation between two virtual resources, then we implicitly consider no exclusion relation between those virtual resources. In other words, the tenant does not have to define, for every pair of virtual resources, that they exclude each other or not: unless the tenant states otherwise, we consider that no virtual resources exclude each other, as per Equation (C18d) and Equation (C19d). Besides, we also assume that both arrays define a symmetric relation, as per Equation (C20d) and Equation (C21d).

$$Default(Exclude_N) = \perp \quad (C18d)$$

$$Default(Exclude_L) = \perp \quad (C19d)$$

$$Read(Exclude_N, (i, i')) = Read(Exclude_N, (i', i)) \quad \forall (i, i') \in N^R \times N^R \quad (C20d)$$

$$Read(Exclude_L, (i, i')) = Read(Exclude_L, (i', i)) \quad \forall (i, i') \in L^R \times L^R \quad (C21d)$$

The same kind of equations can be used to model a collocation attribute, **!col-resources**. The collocation attribute enables tenants to enumerate, within their requests, the virtual resources that must be collocated. Two collocated virtual nodes hence have the same host, and two collocated virtual links hence have the same path. Equation (C22d) and Equation (C23d) are similar to Equation (C10d) and Equation (C13d) respectively. Note that we do not define a collocation attribute at the request or the tenant level, as we do not have a use case to guide us for their actual definition<sup>3</sup>.

$$(Write(Collocate_N, (i, i'), \top) \quad \forall R \in Requests, i \in L^R, j \in L^S, i' \in d_i[!col-resources]) \quad (C22d)$$

$$(Write(Collocate_L, (i, i'), \top) \quad \forall R \in Requests, i \in L^R, j \in L^S, i' \in d_i[!col-resources]) \quad (C23d)$$

---

<sup>3</sup>For instance, if we had a *per-request* collocation attribute, like we have the **!exc-requests** attribute, does that mean that we must collocate a virtual resource with *every* or *at least one* virtual resource of the requests given by the tenant? The problem of collocation is that it is transitive: if  $Collocate_N(i, i')$  is true and  $Collocate_N(i', i'')$  is true, then  $Collocate_N(i, i'')$  is also true; whereas for exclusion:  $Exclude_N(i, i')$  and  $Exclude_N(i', i'')$  does *not* imply that  $Exclude_N(i, i'')$ .

Then Equation (C24d) and Equation (C25d) are similar to Equation (C16d) and Equation (C17d) respectively. They both mean that given two virtual resources  $i$  and  $i'$ , and a substrate resource  $j$ , such that  $j$  hosts  $i$ , if  $i$  must be collocated with  $i'$  then  $j$  also hosts  $i'$ . However, contrary to Equation (C16d) and Equation (C17d), we do not consider the ideal functions ( $Ideal_N$  and  $Ideal_L$ ). This is because, until now, in our model, the ideal nodes and links (see Section 3.1) are only the domains themselves and their self-loops (at the inter-domain level), the outside node, and the outside self-loop (at the intra-domain level). The idea of ideal resources is that they are abstract containers for further physical resources. In other words, we need the  $Ideal_N$  and the  $Ideal_L$  functions for the exclusion attribute, because two exclusive virtual resources can be in the same ideal resource; what matters is that, within this ideal resource, they are hosted on distinct physical resources. The reasoning for collocation relation is actually simpler: two collocated virtual resources must be in the same physical resource, so they must be in the same ideal resource.

$$\left. \begin{aligned} &(\mu_N(i) = j) \wedge (Read(Collocate_N, (i, i')) = \top) \Rightarrow \mu_N(i') = j \\ &\forall (i, i') \in \bigcup_{R \in Requests} N^R \times \bigcup_{R \in Requests} N^R, j \in H^S \end{aligned} \right\} \quad (C24d)$$

$$\left. \begin{aligned} &\mu_L(i, j) \wedge (Read(Collocate_L, (i, i')) = \top) \Rightarrow \mu_L(i', j) \\ &\forall (i, i') \in \bigcup_{R \in Requests} L^R \times \bigcup_{R \in Requests} L^R, j \in L^S \end{aligned} \right\} \quad (C25d)$$

Besides, as we use arrays for  $Collocate_N$  and  $Collocate_L$ , we can define that when the tenant does not explicitly set a collocation relation between two virtual resources, then we implicitly consider no collocation relation between those virtual resources. In essence, it is the same idea than for Equation (C18d) and Equation (C19d). The corresponding equations for  $Collocate_N$  and  $Collocate_L$  are Equation (C26d) and Equation (C27d) respectively. Finally, we also assume that both arrays define a symmetric relation, as per Equation (C28d) and Equation (C29d). This is because collocation is a symmetric relation, like the exclusion relation. When a virtual resource is collocated with another virtual resource, then the other virtual resource is collocated with the first virtual resource.

$$\text{Default}(\text{Collocate}_N) = \perp \quad (\text{C26d})$$

$$\text{Default}(\text{Collocate}_L) = \perp \quad (\text{C27d})$$

$$\text{Read}(\text{Collocate}_N, (i, i')) = \text{Read}(\text{Collocate}_N, (i', i)) \quad \forall (i, i') \in N^R \times N^R \quad (\text{C28d})$$

$$\text{Read}(\text{Collocate}_L, (i, i')) = \text{Read}(\text{Collocate}_L, (i', i)) \quad \forall (i, i') \in L^R \times L^R \quad (\text{C29d})$$

### 3.3.4 Attribute-specific Exclusion and Collocation

We now present a model for the attribute-specific exclusion and collocation. It extends Subsection 3.3.3, and use also functions. For each node and link attribute  $a$ , we define the attribute-specific exclusion function,  $\text{Exclude}_a$ , and the attribute-specific collocation function,  $\text{Collocate}_a$ . Equation (3.34) (resp. Equation (3.35)) defines the attribute-specific exclusion function applied to nodes (resp. links), while Equation (3.36) (resp. Equation (3.37)) defines the attribute-specific collocation function applied to nodes (resp. links). More specifically, they are implemented as arrays, enabling us to apply  $\text{Read}$  and  $\text{Write}$  predicates on them.

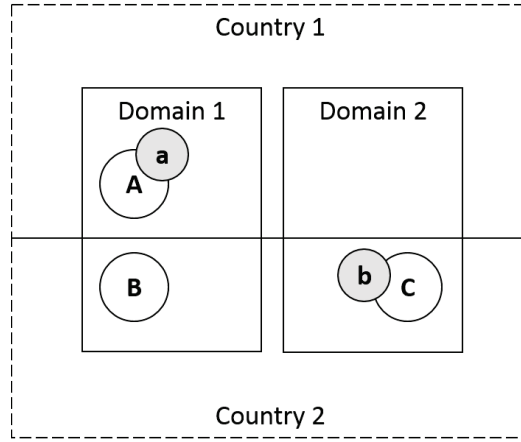
$$\forall a \in \text{Active}_N, \quad \text{Exclude}_a: \bigcup_{R \in \text{Requests}} N^R \times \bigcup_{R \in \text{Requests}} N^R \rightarrow \{\perp, \top\} \quad (3.34)$$

$$\forall a \in \text{Active}_L, \quad \text{Exclude}_a: \bigcup_{R \in \text{Requests}} L^R \times \bigcup_{R \in \text{Requests}} L^R \rightarrow \{\perp, \top\} \quad (3.35)$$

$$\forall a \in \text{Active}_N, \quad \text{Collocate}_a: \bigcup_{R \in \text{Requests}} N^R \times \bigcup_{R \in \text{Requests}} N^R \rightarrow \{\perp, \top\} \quad (3.36)$$

$$\forall a \in \text{Active}_L, \quad \text{Collocate}_a: \bigcup_{R \in \text{Requests}} L^R \times \bigcup_{R \in \text{Requests}} L^R \rightarrow \{\perp, \top\} \quad (3.37)$$

Figure 3.3 illustrates the broad idea behind attribute-specific exclusion. The tenant formulates a constraint, which is that the location of the substrate resource to which a virtual resource  $\mathbf{a}$  is assigned should not be the same than the location of the substrate resource to which another virtual resource,  $\mathbf{b}$ , is assigned. In the figure, this results in  $\mathbf{a}$  and  $\mathbf{b}$  being in two different countries, namely “Country 1” and “Country 2”.



$$O_{!loc}(\mu_N(\mathbf{a})) \neq O_{!loc}(\mu_N(\mathbf{b}))$$

Figure 3.3: Attribute-specific Exclusion Example for the **!loc** Attribute**Grammar 3.2: Grammar for Attribute-specific Exclusion and Collocation**

This grammar extends Grammar 3.1 with two new functions, which are referred to as *Collocate* and *Exclude*. We expose them as functions to the tenant.

```

⟨request⟩ ::= ⟨request⟩ OR ⟨addend⟩ | ⟨addend⟩
⟨addend⟩ ::= ⟨addend⟩ AND ⟨multiplicand⟩ | ⟨multiplicand⟩
⟨multiplicand⟩ ::= ( ⟨expression⟩ ) | ( ⟨predicate⟩ )
⟨predicate⟩ ::= D_{ { ⟨attribute⟩ } } ( ⟨resource⟩ ) = ⟨value⟩
  | ⟨function⟩_{ { ⟨attribute⟩ } } ( ⟨node⟩ , ⟨node⟩ )
  | ⟨function⟩_{ { ⟨attribute⟩ } } ( ⟨link⟩ , ⟨link⟩ )
⟨function⟩ ::= Exclude | Collocate
⟨resource⟩ ::= ⟨node⟩ | ⟨link⟩
⟨node⟩ ::= ⟨string⟩
⟨link⟩ ::= ( ⟨string⟩ , ⟨string⟩ )
⟨attribute⟩ ::= !⟨string⟩

```

```

⟨value⟩ ::= ⟨string⟩ | { ⟨set⟩ }
⟨set⟩ ::= ⟨string⟩ | ⟨set⟩ , ⟨string⟩
⟨string⟩ ::= ⟨character⟩ | ⟨string⟩ ⟨character⟩
⟨character⟩ ::= ⟨letter⟩ | ⟨digit⟩
⟨letter⟩ ::= a | ... | z
⟨digit⟩ ::= 0 | ... | 9
    
```

Instead of considering attribute-specific exclusion and attribute-specific collocation as their own attributes, we consider that the structure of the attribute  $a$  is extended with the aforementioned functions, namely  $Exclude_a$ , and  $Collocate_a$ . Then, we enable tenants to manipulate the attribute-specific exclusion function,  $Exclude_a$ , and the attribute-specific collocation function,  $Collocate_a$ , along the demand function, as described in Grammar 3.2.

We then define the node (resp. link) attribute-specific exclusion in Equation (C30d) (resp. in Equation (C31d)) as a relation between  $Exclude_a$ , the ideal resource function ( $Ideal_N$  and  $Ideal_L$  respectively), the mapping function ( $\mu_N$  and  $\mu_L$  respectively), and the offer function,  $O_a$ . The relation is as follows: given two virtual resources  $i$  and  $i'$ , and two substrate resources  $j$  and  $j'$ , such that  $j$  hosts  $i$ , and  $j'$  is not an ideal resource (note the prime symbol); if  $j$  and  $j'$  have the same offer in  $a$ , then  $j'$  cannot host  $i'$ .

The attribute-specific exclusion relation is more complex than for Equation (C16d) and Equation (C17d) as it involves *two* substrate resources which are compared against their offer in some attribute  $a$ . It is a remote relation: the host chosen for  $i$  impacts the choice of the host chosen for  $i'$ .

$$\left. \begin{aligned}
 & (\mu_N(i) = j) \wedge (O_a(j) = O_a(j')) \wedge (Read(Exclude_a, (i, i')) = \top) \wedge \neg Ideal_N(j') \\
 \Rightarrow & \mu_N(i') \neq j' \\
 & \forall a \in Active_N, (i, i') \in \bigcup_{R \in Requests} N^R \times \bigcup_{R \in Requests} N^R, (j, j') \in H^S \times H^S
 \end{aligned} \right\} \text{(C30d)}$$

$$\left. \begin{aligned}
 & \mu_L(i, j) \wedge (O_a(j) = O_a(j')) \wedge (Read(Exclude_a, (i, i')) = \top) \wedge \neg Ideal_L(j') \\
 \Rightarrow & \neg \mu_L(i', j') \\
 & \forall a \in Active_L, (i, i') \in \bigcup_{R \in Requests} L^R \times \bigcup_{R \in Requests} L^R, (j, j') \in L^S \times L^S
 \end{aligned} \right\} \text{(C31d)}$$

$$\left. \begin{aligned}
& (\mu_N(i) = j) \wedge (O_a(j) \neq O_a(j')) \wedge (\text{Read}(\text{Collocate}_a, (i, i')) = \top) \wedge \neg \text{Ideal}_N(j') \Rightarrow \\
& \mu_N(i') \neq j' \\
& \forall a \in \text{Active}_N, (i, i') \in \bigcup_{R \in \text{Requests}} N^R \times \bigcup_{R \in \text{Requests}} N^R, (j, j') \in H^S \times H^S
\end{aligned} \right\} \quad (\text{C32d})$$

$$\left. \begin{aligned}
& \mu_L(i, j) \wedge (O_a(j) \neq O_a(j')) \wedge (\text{Read}(\text{Collocate}_a, (i, i')) = \top) \wedge \neg \text{Ideal}_L(j') \Rightarrow \\
& \neg \mu_L(i', j') \\
& \forall a \in \text{Active}_L, (i, i') \in \bigcup_{R \in \text{Requests}} L^R \times \bigcup_{R \in \text{Requests}} L^R, (j, j') \in L^S \times L^S
\end{aligned} \right\} \quad (\text{C33d})$$

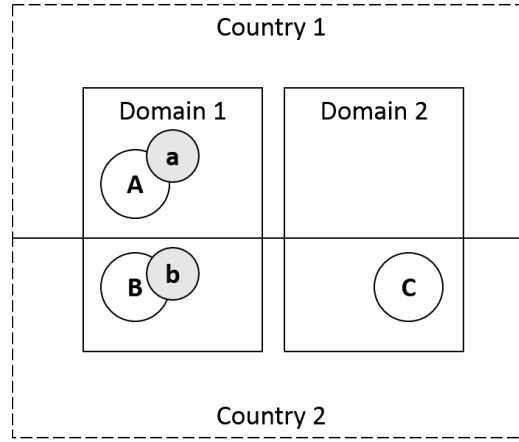
Besides, Equation (C32d) (resp. Equation (C33d)) defines the attribute-specific collocation as a relation between  $\text{Collocate}_a$ , the ideal resource function ( $\text{Ideal}_N$  and  $\text{Ideal}_L$  respectively), the mapping function ( $\mu_N$  and  $\mu_L$  respectively), and the offer function,  $O_a$ . The relation is as follows: given two virtual resources  $i$  and  $i'$ , and two substrate resources  $j$  and  $j'$ , such that  $j$  hosts  $i$ , and  $j'$  is not an ideal resource (note the prime symbol); if  $j$  and  $j'$  do not have the same offer in  $a$ , then  $j'$  cannot host  $i'$ .

At this point of the formulation of the model, a discussion is needed. The model works; yet, its application is limited, due to limited information disclosure design. In essence, attribute-specific exclusion (resp. collocation) is a remote relation between two virtual resources: the choice of a substrate resource for one virtual resource impacts the choice of the substrate resource for the other virtual resource, because we want the substrate resources to have different (resp. same) attribute values.

If we take again Figure 3.3, this means that assigning  $\mathbf{b}$  to  $\mathbf{c}$  forces the system to select either  $\mathbf{A}$  for  $\mathbf{a}$ , as  $\mathbf{A}$  is the only substrate resource in a different country than  $\mathbf{c}$ .

This remote relation naturally opposes to the limited information disclosure design. A trade-off appears about which information the InPs can disclose or not. Depending on the level of disclosure, some attribute-specific exclusion (or collocation) may be fully enabled, or not. We can imagine, like Dietrich et al. (2015), that the InPs expose more information to the system, like a typology of the different attribute values that they resources have.

In the case of limited information disclosure, attribute-specific exclusion (or collocation) are still enabled, but not *fully* enabled. It is still possible to apply them within the same domain. The situation is illustrated by Figure 3.4. In this



$$O_{!loc}(\mu_N(\mathbf{a})) \neq O_{!loc}(\mu_N(\mathbf{b}))$$

Figure 3.4: Attribute-specific Exclusion Example for the `!loc` Attribute, under Limited Information Disclosure

Table 3.3: Building Blocks of `!domain` Attribute

| Attribute            | $V(a)$                 | Ordering operator<br>(demand $\leq$ offer)                          | +      | $m_a$       | $M_a$       |
|----------------------|------------------------|---|--------|-------------|-------------|
| <code>!domain</code> | Powerset<br>of domains | $\supset$ (note the orientation:<br>offer must be <i>in</i> demand) | $\cap$ | All domains | $\emptyset$ |

case, “Domain 1” is the only able to guarantee that `a` and `b` are in different countries.

However, our model may also produce the result depicted in Figure 3.5, due to the definition of outside nodes (`Out1` and `Out2` in the figure), as ideal nodes. By definition, ideal nodes make Equation (C30d) true. Then, at the intra-domain level, “Domain 1” can assign `a` to `A` and `b` to `Out1`. And, *independently*, “Domain 2” can assign `a` to `Out2` and `b` to `D`. In the end, the tenant requirement is not fulfilled, as `a` and `b` are in the same country.

In other words, when tenants want an attribute-specific exclusion (or collocation) upon some virtual resources, it is necessary to add another requirement, explicitly telling that those virtual resources must be in the same domain. This other requirement is the last stone of our model extension. We refer to it as `!domain`, and it enables the tenant to enumerate the domains which are authorized to host the virtual resource. Its structure is given in Table 3.3. It is a tolerant, non-binding attribute.

The reason why `!domain` solves the problem despite being defined like other attributes is that, contrary to the other attributes, the domains in which a virtual

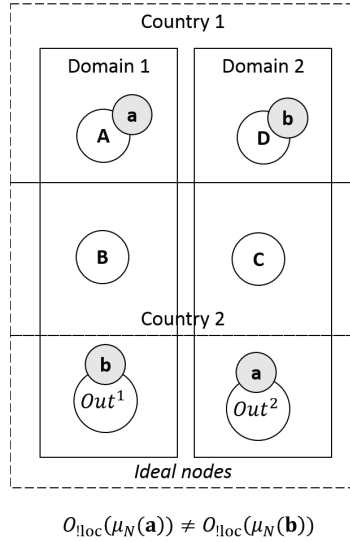


Figure 3.5: Undesired Result from Attribute-specific Exclusion Model, under Limited Information Disclosure

network is instantiated are supposed to be known by the tenant. It is indeed the only output of our algorithm. As such, it complies with limited information disclosure. Algorithm wise, this new attribute is directly applied at the inter-domain level.

All in all, when a tenant desires that  $\mathbf{a}$  and  $\mathbf{b}$  must be in different locations, we convert it in the constraint depicted in Equation (3.38). It reads as follows: the substrate resource hosting  $\mathbf{a}$  must be in a different location that the substrate resource hosting  $\mathbf{b}$ , provided that  $\mathbf{a}$  and  $\mathbf{b}$  must be in the same domain.

$$O_{!loc}(\mathbf{a}) \neq O_{!loc}(\mathbf{b}) \wedge O_{!domain}(\mathbf{a}) = O_{!domain}(\mathbf{b}) \quad (3.38)$$

### 3.4 Conclusion

In this chapter, we presented our attribute model for the security-aware VNE in the 5G context. This model aims to be general enough so that we can instantiate every kind of requirement. More specifically, we identify the tolerant attributes as the ones which can be instantiated, and describe their key building blocks. We illustrated how to leverage this model in the single domain scenario, where we could identify the key rules of a VNE. Two other features are proposed, which have not been evaluated. It is namely the conditional requirement and the attribute-specific exclusion and collocation. As such, they will not be part of Chapter 4, dealing with the implementation and the evaluation.





### Algorithm Resolution

To solve the slice embedding problem, where multiple Infrastructure Providers (InPs) are expected, several approaches exist. The tenant can submit the request to any InP, and the InPs collectively negotiate which part of the slice they embed, following a protocol like the one proposed by Chowdhury et al. (2010). It is a *distributed* approach.

A second approach is to have a broker in between the tenant and the InPs. This broker (or Virtual Network Provider (VNP)) sends the slice request to each InP participant. The InPs then enumerate their embedding propositions, which can fulfill fully or partially the original slice request. The role of the broker is then to gather the propositions that will fully fulfill the original slice request. This approach is a *bottom-up* approach. It is investigated by Mano et al. (2016).

A third approach is to have the same VNP broker, but instead, this broker uses information from the InPs to pre-define the distribution of the slice among those InPs. The broker then sends his or her own requests to the different InPs, the different requests being determined by how he or she wants to distribute the slice. This approach is a *top-down* approach. Dietrich et al. (2015) and Houidi et al. (2011) assume that to embed those requests, the InP can use any Virtual Network Embedding (VNE) method.

In this thesis, we perceive that the distributed approach would be difficult to achieve, because it exposes multiple entry points of the system to the tenant. Instead, we favor a single entry point approach. In this thesis, we decide to follow the top-down approach, as presented in Section 4.2, for which there are more works with which we can compare, such as Dietrich et al. (2015) and Di et al. (2013). Our main contribution then is that we minimize the disclosure of topological information, and that we formulate the partial slice embedding problem,

because the distribution of the original virtual network request among the different InPs does not result in a set of requests which can be directly used by any VNE algorithm.

## 4.1 From the Single Domain to the Multi-Domain Formulation

The Constraint Satisfaction Problem (CSP) as formulated in Section 3.2.2 for a single domain can be reused for a multi-domain scenario. An obvious way is to consider as the substrate the aggregation of all the resources from all the involved domains. Yet, for slices, the domains are the infrastructure networks and are owned by several Infrastructure Providers (InPs) which are likely reluctant to share their information, as they are commercially sensitive.

Instead, we need to make the InPs cooperate while keeping their information private. In this thesis, we consider two levels of cooperation, the inter-domain level and the intra-domain level. The intra-domain level is presented in Section 4.1.1, and the inter-domain level is presented in Section 4.1.2.

Broadly speaking, the inter-domain level embeds a *whole* slice in the *domains* without any knowledge about what they contain. The constraints hold on how the domains are interconnected.

On the contrary, the intra-domain level embeds a *part* of a slice in the *resources* of a domain with full knowledge about their attributes. By a part, we mean that all requests must be embedded when considering *all* the domains, but at the same time, some domains may not embed any request. Indeed, the resources allocated for a given request may belong to completely different domains.

Modeling a partial mapping is not easy though, because node mapping and link mapping are interrelated. In this thesis, to overcome the problem, we use a topological heuristic, which makes us able to reuse the equations from Section 3.2. The idea is to transform the substrate graph. We then seek an exact mapping in this transformed graph. The exact mapping then corresponds to a partial mapping on the original substrate graph.

### 4.1.1 Satisfiability Modulo Theories (SMT) Formulation for the Intra-Domain Level

This section presents how we adapt the VNE formulation from Section 3.2.2 to the intra-domain level, thanks to a topological heuristic.

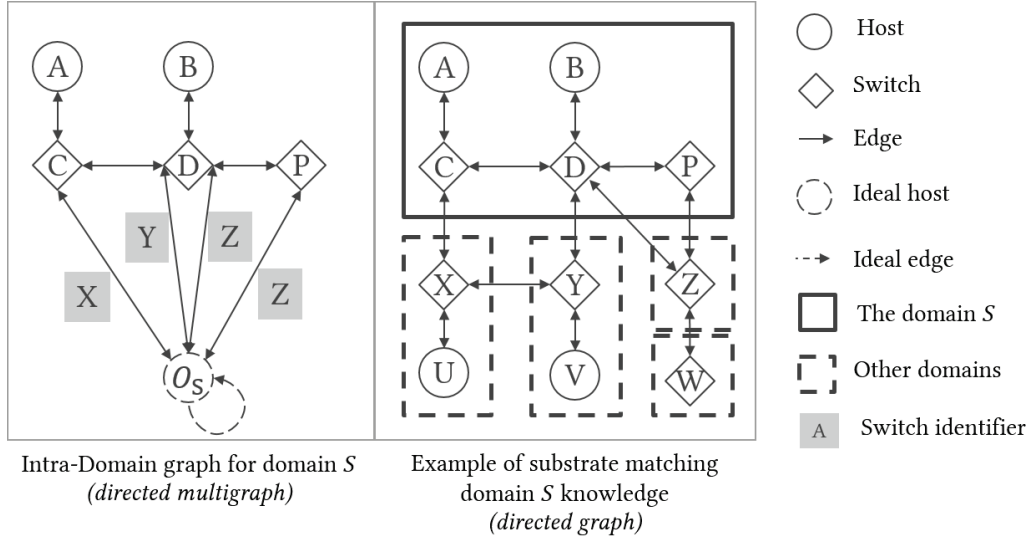


Figure 4.1: Example of Intra-Domain Graph

Before detailing more the heuristic, let us model what a domain can know. We use Figure 4.1 for this purpose. On the left, we present some intra-domain level graph with which we work for some domain  $S$ . It actually represents what we assume  $S$  can know, in the form of a directed, connected multigraph. On the right, we present a *possible* substrate corresponding to this knowledge. As in Section 3.2.2, it is a directed, connected graph.

We consider that any domain knows:

- i*) its own resources, as well as their attributes ( $A$ ,  $B$ ,  $C$  and  $D$ , as well as the links between them in Figure 4.1),
- ii*) the inter-domain links, as well as their attributes ( $(C, X)$  and  $(D, Y)$  for instance in Figure 4.1), and
- iii*) the switches to which it is connected in its neighboring domains, but *not* their attributes ( $X$ ,  $Y$  and  $Z$  in Figure 4.1).

However, the inner topology of other domains ( $(X, U)$  for instance), as well as if and how they could be interconnected ( $(X, Y)$  for instance), is not known.

The heuristic relies on a special *host* node, denoted  $Out^S$  where  $S$  is the domain being considered. This node models the outside of the domain, which encompasses all other domains, being its neighbors or not. When we map a virtual node to the outside, we say that we have a partial mapping (or that the node is outsourced). In other words, we consider that such virtual node will be hosted in another domain. The outside host is connected to every border nodes

( $C$ ,  $D$  and  $P$  for  $S$  in Figure 4.1). This follows a legitimate assumption, that there is no disjoint domain in our system.

Besides, we consider a self-loop  $(Out^S, Out^S)$ , which allows us to outsource a virtual link to the outside. The existence of  $(Out^S, Out^S)$  is required to enable and define the outsourcing of a virtual edge  $(x, y)$ , when both  $x$  and  $y$  must be outsourced too.

From this topology, we define  $\hat{P}^S$  the set of loop-free paths in  $P^S$  that start from a host inside the domain, go out through  $Out^S$  and then end in another host inside the domain. The candidate paths we consider for our embedding are then in  $P^S \setminus \hat{P}^S$ . By construction, there is a unique path in  $P^S \setminus \hat{P}^S$  that contains  $(Out^S, Out^S)$ . It is the path  $(Out^S, (Out^S, Out^S), Out^S)$ , as explained in Definition 3.3. As such, as we have a unique way to outsource a virtual node, we have also a unique way to outsource a virtual link. This unicity is important to guarantee when generating solutions, otherwise we would generate the same solution multiple times.

Another property of the host  $Out^S$  and the self-loop  $(Out^S, Out^S)$  is that they are ideal, which is defined in Definition 3.1. By doing so,  $Out^S$  and  $(Out^S, Out^S)$  limit the possible embeddings in the given intra-domain in terms of topology, rather than attribute values, as any partial embedding requires to go through an inter-domain link.

Inter-domain links are the most critical part of the intra-domain graph indeed, as each of them is known by two domains. For most attributes, the inter-domain link attribute values do not depend on the intra-domain being considered, as we assume that such values are shared between the two involved InPs. Consequently, a virtual link is allocated to an inter-domain link, only if both domains allocate this virtual link to the inter-domain link at their intra-domain level. Furthermore, if we had the full knowledge of the topology, this virtual link could also be allocated to the same inter-domain link, as we have the same attribute values.

However, the reader may be interested by the case of tolerant attributes which rely on aggregation functions, as defined in Definition 3.2, like the **!ven** attribute. In that case, the inter-domain link attribute values are *different*, due to the presence of  $Out^S$ . They are also different from the inter-domain link attribute value under the fully disclosed infrastructure. Thanks to the properties of aggregation function (cf. Definition 3.2), this difference does not impact the validity of the embedding, as we show in Demonstration 4.1. More precisely, we show that if an embedding is rejected with full knowledge of the attributes, then it is also rejected by our solution.

**Demonstration 4.1**

Let  $D$  and  $D'$  be two domains. Let  $S$  be the infrastructure with full knowledge about  $D$  and  $D'$ . Let  $(h, t) \in L^S$  be an inter-domain link between  $D$  and  $D'$ . Let  $a$  be a tolerant link attribute with an aggregation function  $\cdot_a$ .

Thanks to the aggregation function properties, as given in Def. 3.2, we can show that having both domains separately accepting the demand on abstract links  $(h, Out^D)$  and  $(Out^{D'}, t)$  respectively implies to accept the demand with full knowledge on the physical link  $(h, t)$ .

By definition of outside hosts  $Out^D$  and  $Out^{D'}$ , as well as per Def. 3.2, we have:

$$\begin{aligned} o_{(h, Out^D)}[a] &= o_h[a] \cdot_a M_a = o_h[a] \\ o_{(Out^{D'}, t)}[a] &= M_a \cdot_a o_t[a] = o_t[a] \\ o_{(h, t)}[a] &= o_h[a] \cdot_a o_t[a] \end{aligned}$$

Let  $R$  be a request and  $i \in L^R$  a link. Let assume that each domain  $D$  and  $D'$  separately accepts the demand  $d_i[a]$  for  $o_h[a]$  and  $o_t[a]$  respectively. In other words:

- 1)  $d_i[a] \leq o_h[a]$
- 2)  $d_i[a] \leq o_t[a]$

It follows:

$$\begin{aligned} \text{translation-invariance of 1): } & d_i[a] \cdot_a o_t[a] \leq o_h[a] \cdot_a o_t[a] \\ \text{translation-invariance of 2): } & d_i[a] \cdot_a d_i[a] \leq o_t[a] \cdot_a d_i[a] \\ \text{idempotence of } \cdot_a: & d_i[a] \cdot_a d_i[a] = d_i[a] \\ \text{commutativity of } \cdot_a: & d_i[a] \leq d_i[a] \cdot_a o_t[a] \\ \text{transitivity of } \leq: & d_i[a] \leq o_h[a] \cdot_a o_t[a] \end{aligned}$$

As we have  $d_i[a] \leq o_h[a] \cdot_a o_t[a]$ , we know that  $S$  will also accept the demand  $d_i[a]$  for the link  $(h, t)$ .  $\square$

We now present how this heuristic helps us to adapt the equations from Section 3.2.2. Every rule from Section 3.2.1 which is modified is explicitly cited. Those which are not cited are not modified. The modification of the equations is also highlighted.

Equation (C1) still holds because  $Out^S$  is a host. Equation (C2) and Equation (C3) only hold for  $P^S \setminus \hat{P}^S$  and not  $P^S$ . This is to reduce recursivity: otherwise, if we enable paths going out through  $Out^S$ , it means that the other domains

would have to also validate the link mapping. In other words, we alter eligible paths rule (R4) by using a subset of the available paths. See Equation (C2b) and Equation (C3b).

Equation (C4) and Equation (C5) still hold because  $Out^S$  is a host. Equation (C6) and Equation (C7) still hold because  $Out^S$  is an ideal node (cf. Def. 3.1). Equation (C8) and Equation (C9) still hold because  $(Out^S, Out^S)$  is an ideal link (cf. Def. 3.1). Equation (C10), Equation (C11) and Equation (C12) only hold for host  $j \neq Out^S$ , because  $Out^S$  is an ideal node (cf. Def. 3.1). In other words, we alter capacity rule (R6) to make an exception for exclusion attributes on  $Out^S$ , which is consistent with its abstract nature. See Equation (C10b), Equation (C11b), and Equation (C12b).

Equation (C13), Equation (C14), and Equation (C15) only hold for link  $j$  different from  $(Out^S, Out^S)$ , because  $(Out^S, Out^S)$  is an ideal link (cf. Def. 3.1). In other words, we alter R6 to make an exception for exclusion attributes on  $(Out^S, Out^S)$ , which is consistent with its abstract nature. See Equation (C13b), Equation (C14b), and Equation (C15b).

$$\bigvee_{j \in P^S \setminus \hat{P}^S} \left( p_{ij} \wedge \bigwedge_{j' \neq j} \neg p_{i,j'} \right) \quad \forall R \in Requests, i \in L^R \quad (C2b)$$

$$\left. \begin{aligned} p_{ij} = & (n_{i_h, j_h} \wedge n_{i_t, j_t}) \wedge \bigwedge_{j' \in Along_L(j)} l_{i,j'} \wedge \bigwedge_{j' \in L^S \setminus Along_L(j)} \neg l_{i,j'} \\ \forall R \in Requests, i \in L^R, & j \in P^S \setminus \hat{P}^S, i = (i_h, i_t), Tips_j = (j_h, j_t) \end{aligned} \right\} \quad (C3b)$$

$$n_{ij} \Rightarrow \bigwedge_{i' \in d_i[!exc-resources]} \neg n_{i',j} \quad \forall R \in Requests, i \in N^R, j \in \hat{H}^S \quad (C10b)$$

$$n_{ij} \Rightarrow \bigwedge_{\substack{R' \in d_i[!exc-requests] \\ i' \in N^{R'}}} \neg n_{i',j} \quad \forall R \in Requests, i \in N^R, j \in \hat{H}^S \quad (C11b)$$

$$n_{ij} \Rightarrow \bigwedge_{\substack{\tau \in d_i[!exc-requests] \\ R' \in Requests_\tau \\ i' \in N^{R'}}} \neg n_{i',j} \quad \forall R \in Requests, i \in N^R, j \in \hat{H}^S \quad (C12b)$$

$$l_{ij} \Rightarrow \bigwedge_{i' \in d_i[\text{!exc-resources}]} \neg l_{i',j} \quad \forall R \in \text{Requests}, i \in L^R, j \in \hat{L}^S \quad (\text{C13b})$$

$$l_{ij} \Rightarrow \bigwedge_{\substack{R' \in d_i[\text{!exc-requests}] \\ i' \in L^{R'}}} \neg l_{i',j} \quad \forall R \in \text{Requests}, i \in L^R, j \in \hat{L}^S \quad (\text{C14b})$$

$$l_{ij} \Rightarrow \bigwedge_{\substack{\tau \in d_i[\text{!exc-requests}] \\ R' \in \text{Requests}_\tau \\ i' \in L^{R'}}} \neg l_{i',j} \quad \forall R \in \text{Requests}, i \in L^R, j \in \hat{L}^S \quad (\text{C15b})$$

### 4.1.2 SMT Formulation for the Inter-Domain Level

At the inter-domain level, we only know the different domains and the inter-domain links. The inner domain topologies are unknown. Yet, we can still reuse the rules in Section 3.2.1, and use them to adapt the equations from Section 3.2.2, as follows.

We model the inter-domain level as a directed, connected, multigraph. Each node represents a domain, and each link represents an inter-domain link. Domains may be connected through more than one pair of border routers.

We leverage the multigraph to know which domains can be used for our slice, and which inter-domain links will be used. The inner details of the embedding are left to the domains.

In such graph, according to node mapping rule (R1) and eligible substrate hosts rule (R3), we need to consider that each domain is a host. Then, we must relax R4 to enable two virtual nodes to be embedded in the same domain. To do so, we will use loop-free paths (cf. Def. 3.3), and benefit from the fact that, by definition, when there is a self-loop joining a node to itself, there exists a unique loop-free path joining the same node to itself. Finally, we must relax R6, because we cannot treat neither exclusion nor tolerant attribute constraint at this level due to our lack of knowledge. This is also true for the self-loops on the domains.

From this update of the rules, it follows that we only need to consider domains as hosts, with a self-loop edge. As we do not consider exclusion nor tolerant attributes on domains nor on their self-loops, we can make them all ideal, as per Definition 3.1.

We now present how this heuristic helps us to adapt the equations from Section 3.2.2. Like in Section 4.1.1, the equations which are modified are highlighted. The equations which are not modified are not cited.

Equation (C1), Equation (C2), Equation (C3), Equation (C4) and Equation (C5) still hold because domains are hosts. Equation (C6) and Equation (C7) still hold



because domains are ideal nodes. Equation (C8) and Equation (C9) still hold because domain self-loops are ideal links. Equation (C10), Equation (C11) and Equation (C12) are removed because domains are ideal nodes. Equation (C13), Equation (C14), and Equation (C15) only hold for inter-domain links, as domain self-loops are ideal links. See Equation (C13c), Equation (C14c), and Equation (C15c).

$$l_{ij} \Rightarrow \bigwedge_{k \in d_i[\text{!exc-resources}]} \neg l_{kj} \quad \forall R \in \text{Requests}, i \in L^R, j \in L^S \setminus \text{SelfLoops}^S \quad (\text{C13c})$$

$$l_{ij} \Rightarrow \bigwedge_{\substack{R' \in d_i[\text{!exc-requests}] \\ k \in L^{R'}}} \neg l_{kj} \quad \forall R \in \text{Requests}, i \in L^R, j \in L^S \setminus \text{SelfLoops}^S \quad (\text{C14c})$$

$$l_{ij} \Rightarrow \bigwedge_{\substack{T \in d_i[\text{!exc-requests}] \\ R' \in \text{Requests}_\tau \\ k \in L^{R'}}} \neg l_{kj} \quad \forall R \in \text{Requests}, i \in L^R, j \in L^S \setminus \text{SelfLoops}^S \quad (\text{C15c})$$

## 4.2 Multi-level Resolution Algorithm

In this section, we present our algorithm resolution following a top-down approach, where the VNP determines the subparts of the original virtual network request that the different InP must embed. The InPs must then solve a partial slice embedding problem, as each subpart sent by the VNP is a set of virtual resources which must be connected to virtual resources instantiated by other InPs.

This algorithm resolution relies on SMT solvers. Our main motivation for leveraging those is that they provide some guarantees, such as mathematical consistency, trustworthiness (the SMT solver in use is *z3*, which is well-known and open source, and relying on Barrett et al. (2017), which is a SMT standard), and auditability (as the InPs can fully verify the constraints that the solver is using).

Subsection 4.2.1 presents the architecture of the system, enabling the multi-domain VNE. We describe our algorithm as a multi-level algorithm, as we have to treat the inter-domain SMT formulation and the intra-domain SMT formulation given in Section 4.1. Our main goal is to enumerate the candidate embeddings, which is described in Section 4.2.2. This goal has been chosen instead of an optimization goal because we want first to verify that our SMT formulation is correct and generates correct solutions, which is evaluated in Chapter 5. Subsection 4.2.3 is dedicated to the description of how the inter-domain level and the intra-domain level cooperates in the system to enumerate the candidates. We also describe how a candidate can be selected by the tenant in Subsection 4.2.4. As such, we enable the tenant to apply whatever decision criterion upon the set

of candidates to choose the embedding candidate which will be instantiated to provide the virtual network itself.

### 4.2.1 Architecture

Our algorithm is presented in Figure 4.2. It follows a delegation system, where an *InterDomainSolver* serves as the unique interface between the tenant and the InPs. The *InterDomainSolver* views only the interconnections between the different domains. It is the only solver belonging to what is called the “inter-domain level”. Each *IntraDomainSolver* corresponds to a domain, and is the only entity being able to access the data of that domain. They belong to what is called the “intra-domain level”. The delegation system appears in how the two levels interact. Indeed, the *InterDomainSolver* acts as an initiator, which queries each *IntraDomainSolver*. It uses their replies to generate the solutions.

The system involves a tenant  $\tau$  submitting a new request *NewRequest* to the unique *InterDomainSolver* (denoted  $U$ ). This latter collaborates with several *IntraDomainSolvers* (denoted  $D$ ). From now, the inter-domain level must be distinguished from the intra-domain level. To do so, the letters  $U$  and  $D$  will be used in *superscript*. For instance,  $x_{ij}^D$  is an allocation variable  $x_{ij}$  from the intra-domain level problem (any  $n_{ij}$ ,  $p_{ij}$  or  $l_{ij}$  in Section 4.1.1), while  $x_{ij}^U$  is an allocation variable  $x_{ij}$  from the inter-domain level problem (any  $n_{ij}$ ,  $p_{ij}$  or  $l_{ij}$  in Section 4.1.2).

The generation algorithm, presented in Fig. 4.2 starts with an **initialization** step, where the tenant  $\tau$  sends the new request *NewRequest* (step 1) to the inter-domain solver  $U$ , which relays it to the different intra-domain solver instances  $D$  (step 3). When receiving *NewRequest*, each solver runs independently a *genCstr* (for “generate constraints”) function. The function *genCstr<sup>D</sup>* (resp. *genCstr<sup>U</sup>*) is the function which generates the intra-domain level (resp. inter-domain level) mathematical constraints as per the formulation in Section 4.1.1 (resp. Section 4.1.2).

Then at the **enumeration** step, all solvers participate in a *while* loop. The embeddings are outputted in the variable *results*. We will describe how the enumeration works shortly after. The algorithm ends with the **finalization** step, which sends the *results* back to the tenant  $\tau$ .

### 4.2.2 Solution Enumeration

Enumerating solutions with a SMT solver is a known algorithm: while the constraints are satisfied (through an *isSatisfied* function), the solver gets a model  $m$  (through the *getModel* function, see step 6), stores it in the results, and adds a

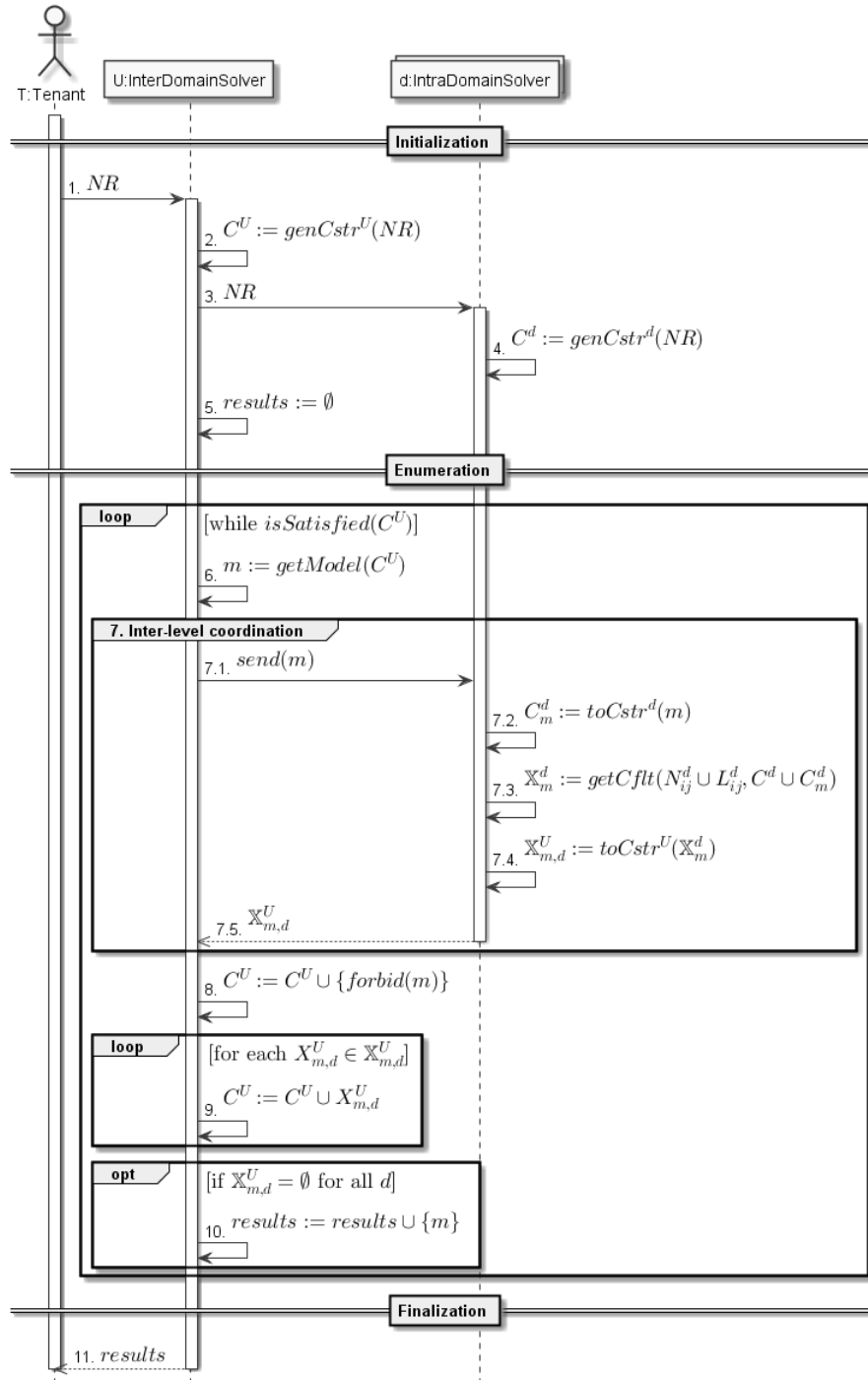


Figure 4.2: Embeddings Generation Sequence Diagram

new constraint to the problem which prevents itself from re-generating  $m$ . This new constraint is returned by the function *forbid* (step 8).

$$\bigvee_{x \in X} x \neq m(x) \quad (4.1)$$

$$\bigvee_{x \in N_{ij} \cup P_{ij}, m(x) = \top} \neg x \quad (4.2)$$

By definition, a model  $m$  is a function that assigns a value  $m(x)$  to each variable  $x \in X$  of the original problem. Hence forbidding a model  $m$  can be done with Equation (4.1). This expression is for the general case. If we adapt it to our slice embedding formulation, we can exhibit a shorter (in terms) forbidding constraint, represented in Equation (4.2).

The reason why we can do so is that knowing, for each virtual resource  $i \in N^R \cup L^R$  of a request  $R$ , the resource  $j \in H^D \cup \hat{P}^D$  (resp.  $j \in H^U \cup P^U$ ) to which it is mapped at the intra-domain level (resp. the inter-domain level), *fully determines* the values of all the variables  $N_{ij}$ ,  $P_{ij}$  and  $L_{ij}$  (for the respective level), as we show in Demonstration 4.2. Thus Equation (4.2) is the chosen expression for *forbid*.

In our algorithm (see Figure 4.2), the **enumeration** stage features three steps, numbered 7, 9 and 10. They are all related to the **inter-level coordination**. At step 7.1, it sends the inter-domain level model  $m$  to the solver  $D$ . At step 7.2, this model is translated into intra-domain level constraints with *toCstr<sup>D</sup>* (for “translate to constraints”). At step 7.3, it generates the set of minimal unsatisfiable subsets. We will explain its exact meaning later, but the reader can interpret it globally as the set of conflicting variables. At step 7.4, those minimal unsatisfiable subsets are translated back into inter-domain level constraints with *toCstr<sup>U</sup>*, which are sent to  $U$  at step 7.5.

#### Demonstration 4.2: Solution Determination by True-valued Variables

This demonstration aims to show that we can fully determine an embedding solution by only knowing an appropriate set of  $n_{ij}$  variables and  $p_{ij}$  variables.

For the sake of simplicity, this demonstration ignores the difference between the inter-domain level and the intra-domain level. We rely on the embedding formulation in Section 3.2.2. Considering inter-domain level equations or intra-domain level equations does not alter the reasoning.

Let suppose we have a solution. It follows:

1) Equation (C1) is true:

$$\forall i \in N^R, \bigvee_{j \in H^S} \left( n_{ij} \wedge \bigwedge_{j' \neq j} \neg n_{i,j'} \right) = \top$$

2)  $\vee$ -split:

$$\forall i \in N^R, \exists j \in H^S, n_{ij} \wedge \bigwedge_{j' \neq j} \neg n_{i,j'} = \top$$

3)  $\wedge$ -split:

$$\forall i \in N^R, \exists j \in H^S, (n_{ij} = \top) \wedge (\forall j' \neq j, n_{i,j'} = \perp)$$

4)  $j$  is indeed unique:

$$\forall i \in N^R, \exists! j \in H^S, (n_{ij} = \top) \wedge (\forall j' \neq j, n_{i,j'} = \perp)$$

As such, knowing a resource  $j$  such that  $n_{ij}$  is true means that it is *the*  $j$  such that  $n_{ij}$  is true and that for all  $j' \neq j$ , the  $n_{i,j'}$  are false.

We apply the same reasoning to Equation (C2). In other words, knowing all the  $n_{ij}$  which are true and all the  $p_{ij}$  which are true is equivalent to know the values of all the  $n_{ij}$  and  $p_{ij}$ .

Besides, let  $i = (i_h, i_t) \in L^R$ . Let  $j \in P^S$  such that  $p_{ij} = \top$ . Let  $Tips_j = (j_h, j_t)$ . It follows:

1) Equation (C3) is true:

$$p_{ij} = (n_{i_h, j_h} \wedge n_{i_t, j_t}) \wedge \bigwedge_{j' \in Along_L(j)} l_{i,j'} \wedge \bigwedge_{j' \in L^S \setminus Along_L(j)} \neg l_{i,j'} = \top$$

2)  $\wedge$ -split:

$$\begin{aligned} & (n_{i_h, j_h} = \top) \wedge (n_{i_t, j_t} = \top) \wedge \\ & (\forall j' \in Along_L(j), l_{i,j'} = \top) \wedge \\ & (\forall j' \in L^S \setminus Along_L(j), l_{i,j'} = \perp) \end{aligned}$$

In other words, for a given  $i$ , knowing a  $j$  such that  $p_{ij}$  is true implies that we know the values of all the  $l_{i,j'}$  for  $j' \neq j$ . It follows that knowing all the  $p_{ij}$  which are true implies to know the values of all  $l_{ij}$ .

Knowing all the  $n_{ij}$  which are true and all the  $p_{ij}$  which are true is then sufficient to describe the solution.  $\square$

At step 9, the algorithm uses the minimal unsatisfiable subsets as new forbidding constraints, similarly to the function *forbid* at step 8. At step 10, the algorithm determines if the model  $m$  is a solution. It is the case if every domain

$D$  returns an empty set of minimal unsatisfiable subsets. Naturally, if  $m$  is a solution, no forbidding constraint is added at step 9, but as the *forbid* function is called in step 8, the solver never solves twice the same constraints. Consequently, the enumeration loop will always terminate.

### 4.2.3 Inter-level Coordination

Let us now talk about the **inter-level coordination**. Above all the algorithm uses *getCflt* (for “get conflicts”), a function relying on another known SMT algorithm, called MARCO and designed by Liffiton et al. (2016), which produces all minimal unsatisfiable subsets. Given a problem as a set of constraints (like  $\{a, \neg a, b, \neg b\}$ ), an unsatisfiable subset is a subset of the problem constraints which are unsatisfiable. For instance,  $\{a, \neg a, b\}$  is such an unsatisfiable subset, as well as  $\{a, \neg a\}$ . A *minimal* unsatisfiable subset is then an unsatisfiable subset which does not contain any other unsatisfiable subset. As  $\{a, \neg a, b\}$  contains  $\{a, \neg a\}$ , it is not a minimal unsatisfiable subset, but  $\{a, \neg a\}$  is. Minimal unsatisfiable subsets are not necessarily unique. For instance,  $\{b, \neg b\}$  is also a minimal unsatisfiable subset of the same set of constraints. The MARCO algorithm can be directly interconnected with z3 thanks to Bjørner (2016).

In our case, *getCflt* use a slight modification of the MARCO algorithm so that the minimal unsatisfiable subsets are only composed of our allocation variables. In other words,  $getCflt(X_{ij}, C)$  enumerates all variables in  $X_{ij}$  that make the problem  $C$  unsatisfiable. To do so, the algorithm first translates the inter-domain level mode  $m$  in terms of intra-domain level constraints  $C_m^D$  with  $toCstr^D$ . The problem given to *getCflt* is then  $C := C^D \cup C_m^D$  where  $C^D$  is the initial intra-domain level problem (from  $genCstr^D$ ). The result is denoted as  $\mathbb{X}_m^D$  and is a set of sets of intra-domain level allocation variables, which cannot be true together. We then translate back the intra-domain level allocation variables in  $\mathbb{X}_m^D$  into inter-domain level allocation variables thanks to  $toCstr^U$ . The result is denoted  $\mathbb{X}_{m,D}^U$ .

The translation from the inter-domain level to the intra-domain level realized by  $toCstr^D$  is given in Equation (4.3), Equation (4.4), and Equation (4.5). Equation (4.3) means that mapping a tenant node  $i$  to a domain  $D$  is translated to the constraint “do not map  $i$  to the outside” within the domain  $D$ . On the contrary, Equation (4.4) means that mapping a tenant node  $i$  to a domain different from  $D$  is translated to the constraint “map  $i$  to the outside” within the domain  $D$ . Equation (4.5) means that mapping a tenant link  $i$  on a inter-domain link from/to the domain  $D$  is kept as is within the domain  $D$ .

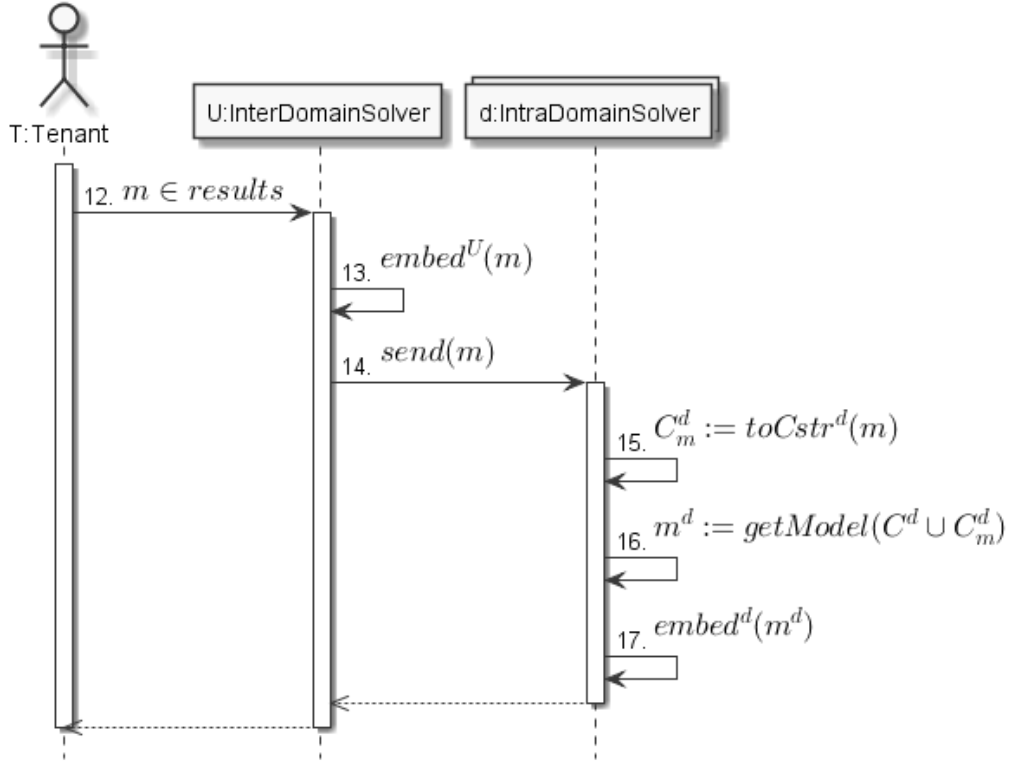


Figure 4.3: Embedding Selection Sequence Diagram

$$\forall i \in N^{NewRequest}, \quad \exists n_{ij}^U, j = D \Rightarrow \neg n_{i,Out^D}^D \quad (4.3)$$

$$\forall i \in N^{NewRequest}, \quad \exists n_{ij}^U, i \neq D \Rightarrow n_{i,Out^D}^D \quad (4.4)$$

$$\forall i \in L^{NewRequest}, \quad \exists l_{ij}^U, j \in InterdomainLinks_D^U \Rightarrow l_{ij}^D \quad (4.5)$$

When generating constraints with  $genCons^U$  (or  $genCons^D$ ), the solver of course need to know the set of our loop-free paths  $P^U$  (or  $P^D$ , see Table of all Notations starting on page 137). If we assume that the infrastructure topology does not change, then this set can be considered as constant, like Yu et al. (2008), and produced with a depth-first search algorithm.

#### 4.2.4 Solution Selection

Figure 4.3 presents what happens after the tenant chooses which embedding is the most suitable. Those steps are run after the generation algorithm. First, at step 12, the tenant chooses a model  $m$  in the *results* the tenant got from the generation algorithm. Then at step 13, the inter-domain solver runs the  $embed^U$

function, which will memorize, for the future executions, which mapping variables  $n_{ij}$  and  $p_{ij}$  must be set to true. It is actually how (C4) and (C5) are filled. Steps 14 and 15 are the same as 7.1 and 7.2. The reason why they are repeated is because we consider that  $D$  does not remember  $C_m^D$  for every  $m$ . Step 16 generates a new model,  $m^D$ , which represents the partial mapping of the newcoming request *NewRequest* at the intra-domain level. Model  $m^D$  is only known by  $D$ . It is not unique, but it is guaranteed to exist, because if  $m \in results$ , then neither  $D$  nor any other intra-domain solver sent any minimal unsatisfaction subset at step 7.5. At step 17, the intra-domain solver runs its counterpart of  $embed^U$ , namely  $embed^D$ , to fill (C4) and (C5) at the intra-domain level.

The tenant  $\tau$  now has a slice. Note however that the process by which the physical resources are technically reserved and configured is out of the scope of this thesis, therefore not presented in Figure 4.3.

## 4.3 Conclusion

In this chapter, we present how we solve the security-aware VNE in the 5G context. It is a top-down approach, distinguishing two levels: the inter-domain level, and the intra-domain level. The inter-domain level is managed by the VNP, while the intra-domain level is managed by the different InPs. Our methodology to address the problem was to leverage the single domain scenario, formulated with SMT in Section 3.2, and a topological heuristic. The algorithm itself leverages different techniques from the SMT field, notably how to enumerate a set of solutions, and how to identify the causes of rejection.





## CHAPTER 5

---

### Prototyping and Evaluation

This chapter presents how we implement our work and evaluate it. More precisely, our implementation leverages the Satisfiability Modulo Theories (SMT) oriented model from Chapter 3, and follows the top-down approach described in Chapter 4. Section 5.1 presents our prototype, as well as an illustration with a use case request. The implementation itself was not an easy task though, and nourished our attribute model. Section 5.2 summarizes the challenges that occurred during the implementation. We then consider the evaluation of the developed system itself. Section 5.3 presents the experiments we conducted to evaluate our work thoroughly. Then, in Section 5.4, we justify that our model is generic enough, by leveraging it to implement the security-oriented attributes proposed by the state of the art. Section 5.5 concludes this chapter.

#### 5.1 Prototype Description and Use Case

Developing a prototype for our work is a necessity for any correct evaluation and improvement. This prototype serves also other purposes. We presented it at an event organized by Nokia Bell Labs France entitled “Nokia 5G Campus Event”, in 2018, where we could gathered positive feedback. The audience encompassed Nokia employees, and employees of telecom companies. The prototype is also part of a demonstration for the sub-project TANDEM of the SENDATE European project (see Pointurier (2016)), and included in the security work package, WP4. SENDATE stands for Secure Networking for a Data Center Cloud in Europe, and TANDEM for Tailored Network for Data Centers in the Metro.

In Subsection 5.1.1, we give a technical description of our prototype. In Subsection 5.1.2, we present a tenant scenario, and how we leverage the prototype

to instantiate the tenant’s request. Although we use the term “technical”, we do not intend to present the prototype down to the programming aspects. Our main focus instead is to show how the prototype behaves, what are our design choices, and which technical challenges have been great enough to make us rethink the implementation.

### 5.1.1 Prototype Description

This section describes our prototype, implementing the model presented in Section 3.1 and the algorithm presented in Section 4.2. The purpose of this prototype is to solve the multi-provider security-aware virtual network embedding problem.

The core of the prototype is a web server, which comes with a GUI and a REST API. Both are used for demonstrations of the prototype, and then, share some ground features. They do differ however, because of the stakes of the demonstration. The GUI was mainly used within a standalone demonstration, with some processes directly run by the web server without control from the GUI. For its part, the REST API is mainly used in a demonstration where it is connected to an orchestrator. For integration purpose, the server has to cooperate with the orchestrator.

As a whole, the GUI encompass the following processes:

**Request creation** The tenant customizes a graph representing the virtual network. Once the graph is submitted, a virtual network request is created. The request is considered to be in the state “non-embedded”.

**Request embedding** The tenant requests the embedding of one (and optionally, multiple) virtual network requests. This steps actually runs our Virtual Network Embedding (VNE) algorithm. It also runs extra steps. The first one is to *memorize* the embedding, both at the inter-domain level and the intra-domain level. The second one is to display back the output of the algorithm *at the inter-domain level*.

**Candidates generation** The tenant can ask (for demonstration purpose), to enumerate all candidate embeddings for the selected virtual network request. A candidate embedding is an output of our VNE algorithm.

**Embedding selection** The tenant can select one candidate embedding from the candidates generation page. This embedding is then memorized and displayed (like in “Request embedding”).

**Request freeing** An embedded request can be freed, which means that our VNE algorithm must forget the association between the virtual resources

and the infrastructure resources it memorized. The request comes back to the “non-embedded” state.

**Request deletion** A non-embedded request can be deleted, which means that the prototype forgets the virtual network request as a whole.

**Infrastructure view** The Infrastructure Providers (InPs) can view the change in their respective infrastructure, as memorized by the prototype.

Then the REST API, enabling connection to an orchestrator, encompasses the following processes:

**Request creation** Like for the GUI, but using a JSON format.

**Request embedding** Like for the GUI, but using a JSON format.

**Infrastructure initialization** Infrastructure initialization is currently out-of-scope of our prototype, and done in an improper manner. Currently, the prototype expects JSON data describing the whole infrastructure of every InPs. Then, it distributes the data to different servers, which are part of the intra-domain level. The different servers are the one responsible for holding and protecting the data for a particular InPs. We do so because we do not implement a mechanism for the InPs to register themselves and plug an external intra-domain solver to our algorithm.

**Attribute mutability** When new hardware is acquired, new options may be offered to the tenants, such as new supported vendors, or new supported locations. Attribute mutability enables to update the attribute structure accordingly in a single place.

Note that we distinguish request creation (resp. deletion) from request embedding (resp. freeing). We do so because we want to empower tenants with full control over when their virtual network is instantiated or not. This feature is interesting when the tenant only needs (and accepts to pay for) the virtual network for a short period of time, and may need it again later.

Besides, we mention the connection to an orchestrator, because orchestrators are the entities dedicated to actually instantiate virtual networks. The output of our prototype actually corresponds to the placement of the virtual resources within the infrastructure. Naturally, our prototype and the so-called orchestrator must cooperate closely. Another example of such cooperation is that, besides the virtual network request, our prototype needs the infrastructure data. Such data are only known by the orchestrator of the particular infrastructure. When multiple InPs must be considered, our prototype seeks to communicate with the orchestrators of the respective domains.

Table 5.1: Overview of Python modules in use

| Python module name     | Purpose                     |
|------------------------|-----------------------------|
| <code>cffi</code>      | For ZODB dependencies       |
| <code>ZODB</code>      | Object-oriented database    |
| <code>ZEO</code>       | Run ZODB as a server        |
| <code>passlib</code>   | Password creation library   |
| <code>bottle</code>    | Web server                  |
| <code>z3-solver</code> | SMT solver                  |
| <code>networkx</code>  | Graph library               |
| <code>pydot</code>     | Graph visualization library |
| <code>requests</code>  | REST API calls              |

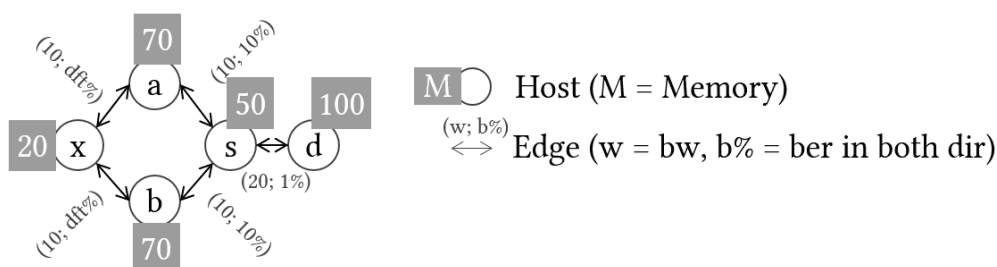
The prototype itself has been developed under Python 3.6 and has been run on an Ubuntu 16.04 OS. The Python package dependencies are enumerated in Table 5.1. For the purpose of the SENDATE demonstration, the prototype can also come in a Docker container. The Docker container format helps the prototype to be more portable.

The choice of Python for programming is only driven by my familiarity with this language. I have encountered it in my engineering school, and during my end-of-studies internship. Python proved to me to be flexible enough to start any kind of project, be it job-related or personal. For this prototype implementation, it was just a tool to start, until I could face some drastical limits. When I tried the implementation of the first proposed model, it was slow enough to make me think about using another language; but as SMT stepped into the picture and as I could rework my model around it, I continued to use Python.

Packages in Table 5.1 serve different purposes. `z3-solver` (by Moura and Bjørner (2008) and Bjørner and Moura (2018)), `networkx` (by Hagberg (2018)) and `ZODB` (by Fulton (2018)) are closely related to the implementation of our model. `z3-solver` provides the `z3` SMT solver, which is at the core of our algorithm. `networkx` provides graph manipulation and modelling tools. Particularly, we use their simple path generation algorithm, which is based on a breadth-first search. For its part, `ZODB` provides database tools for storing Python objects, especially graphs from `networkx` or the instances from our model.

Another widely used package used by our prototype is the `multiprocessing` standard Python package, as we have decided to implement the domain isolation by instantiating the intra-domain solvers in distinct processes, run in parallel. This design choice is driven by the decision to run our algorithm at first on a single (but powerful) physical machine.

The other packages are related to the demonstration purpose of our prototype. The `bottle` package of Hellkamp (2018) provides tools for making a web server, and is the core of our server architecture. I actually use it because I am



Other attributes:

- (x,b) excludes (x,a) and (a,x)
- (b,x) excludes (x,a) and (a,x)
- (a,s), (s,a), (b,s), (s,b), (s,d), (d,s) forbid vendor V3
- d forbids location Loc1
- b excludes a
- s excludes all other nodes in request
- d excludes all other nodes in request

Figure 5.1: Use case request

already familiar with it, and I could work with it without encountering major obstacle.

### 5.1.2 Use Case

Figure 5.1 shows an exemplary use case. Here, the tenant wants to outsource some IT resources. The request is composed from an access node, **x**, two remote workspaces for two users **a** and **b**, a server **s** and a database **d**. Apart from the access part, the tenant wants to explicitly avoid switch vendor *V3*. The database containing some sensitive or regulation-protected information cannot be stored in location *Loc1*. Using the exclusion attribute, the tenant explicitly avoids the collocation of some virtual nodes and virtual edges. This is to prevent **a** and **b** from spying on each other, and also from acquiring protected data from **d**.

Our prototype runs on a server. The tenant connects to it and describes the request on a form page. Figure 5.2 shows the specific web form to fulfill for the resource **d**.

In our use case, the tenant only modifies the fields for *Requested Memory* (!mem attribute), the *Authorized Locations* (!loc attribute) and *Forbidden Resources* (!exc-resources attribute). Others are left to their default value.

The tenant then saves the request and can now find it on a dashboard page, which encompasses all already submitted requests and their states (embedded, not embedded). This dashboard is displayed in Figure 5.3, along with a summary of the demands for each resource. The table on the right presents how each value given by the tenant is translated by the server. The fields *Requested Memory*, *Authorized Locations* and *Forbidden Resources* semantically describe what the tenant asked for.

## Prototyping and Evaluation

| Save                 | Cancel | Design your node  |
|----------------------|--------|---|
| Node name            |        | <input type="text" value="d"/>  |
| Requested Memory     |        | <input type="text" value="100"/>  |
| Authorized Locations |        | except <input type="checkbox"/> Loc1 <input type="checkbox"/> Loc2 <input checked="" type="checkbox"/> Loc34 <input checked="" type="checkbox"/> all others <input type="checkbox"/>  |
| Forbidden Tenants    |        | except <input type="checkbox"/> Competitor <input type="checkbox"/> MyCompany <input type="checkbox"/> Tenant <input type="checkbox"/> Tenant1 <input type="checkbox"/><br>Tenant2 <input type="checkbox"/> all others <input type="checkbox"/> |
| Forbidden Requests   |        | except <input type="checkbox"/><br>add another <input type="checkbox"/><br>all others <input type="checkbox"/>  |
| Forbidden Resources  |        | except <input type="checkbox"/><br>add another <input type="checkbox"/><br>all others <input checked="" type="checkbox"/>   |

Figure 5.2: Declaring resource **d** on prototype

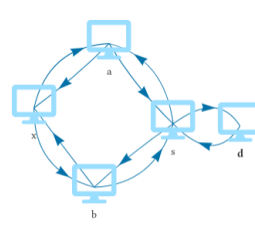
| Select                   | Date      | Request   | Hide                     | Node d   |
|--------------------------|-----------|---|--------------------------|--|
| <input type="checkbox"/> | 3/15/2019 |  | <input type="checkbox"/> | Requested Memory: 100<br>Authorized Locations: LOCATIONS EXCEPT Loc1<br>Forbidden Resources: ANY OTHER NODE<br>__model_mem: 100<br>__model_loc: 3<br>__model_x_ten: NONE<br>__model_x_req: NONE<br>__model_x_res: s-Request-Tenant, AND a-Request-Tenant, AND b-Request-Tenant, AND x-Request-Tenant |
| Invasive Embedding       |           | Discreet Embedding  |                          | Delete Free  |

Figure 5.3: Description of resource **d** on prototype

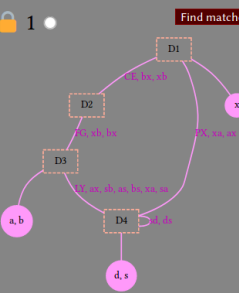
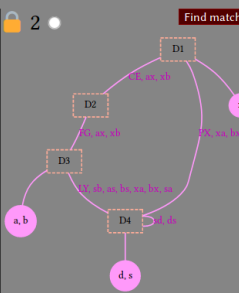
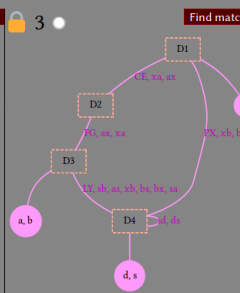
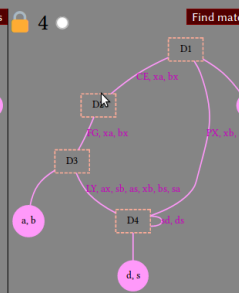
| 1   | 2   | 3  | 4   |
|---|---|--|---|
|  |  |  |  |
| Embed selected solution   |   |  | Forget it   |

Figure 5.4: Proposed embeddings

The tenant now executes our embedding generation algorithm (as a reminder, it is depicted in Figure 4.2 in Chapter 4). The tenant is then redirected to an interface similar to the one shown on Figure 5.4. This interface shows every found embedding, depicted as a graph. The embeddings are numbered by the order in which they were found. The bullet on top of each embedding enables the tenant to select the desired embedding. Once an embedding is selected, we run the embedding selection algorithm (as a reminder, it is depicted in Figure 4.3 from Chapter 4).

## 5.2 Implementation Challenges for Mutable Attributes

This section is about the difficulties encountered during prototype development. We actually encountered one major difficulty, to which this section has been dedicated. This difficulty is a good example of why it is necessary to go back and forth between the model and the implementation, as it is due to an aspect which has been overseen in the active attribute model. This aspect is the mutability of the attributes, as introduced in Section 3.1.

In this prototype, we decouple request declaration from request embedding. As such, tenants may declare requests and embed them later, or re-embed them after freeing resources, for instance. Consequently, we have to store requests in the database.

Yet, this raises a technical problem for the powerset-based attributes. For attributes like `!loc` or `!ven`, the set of available locations, or the set of available vendors, is expected to change over time, as the InPs may acquire more hardware, or even because new InPs can participate into the system. The problem is then that, although such attributes may change, the data we store, describing requirements, should not change. This does not mean that the stored data may not change over time; but that the requirement itself should be guaranteed to have the same meaning over time. In other words, providing new options, new vendors, new locations, should not affect the data using the old set of options, vendors or locations.

Guarantee that the requirement keeps the same meaning over time happened to be a challenge. We now illustrate it by describing how we tackle this problem.

We want to enable a certain attribute  $a$  to be a mutable set, growing over time. We suppose that  $V(a)$  is a powerset of a set  $Set_a$ :  $V(a) = Powerset(Set_a)$ . Let  $Set_a(t)$  denote such time-dependent set. At the initialization time ( $t_0$ ), we assume that  $Set_a(t_0) = \{X, Y\}$ . At another (later) instant,  $t_1$ , we assume that  $Set_a(t_1) = \{X, Y, Z\}$ . The situation is presented in two distinct tables, Table 5.2



Table 5.2: Compute before Store Strategy for Attribute  $a$  Such That  $V(a) = \text{Powerset}(Set_a)$ 

| a Positive Requirement |                |                        |
|------------------------|----------------|------------------------|
| Time                   | $t = t_0$      | $t = t_1$              |
| $Set_a$                | $\{X, Y\}$     | $\{X, Y, Z\}$          |
| Intent                 | $X$ only       | $X$ only               |
| Computation            | Store $\{X\}$  | Get $\{X\}$            |
| Semantics              | $\{X\}$        | $\{X\}$                |
| Interpretation         | $X$ only       | $X$ only               |
| b Negative Requirement |                |                        |
| Time                   | $t = t_0$      | $t = t_1$              |
| $Set_a$                | $\{X, Y\}$     | $\{X, Y, Z\}$          |
| Intent                 | All except $X$ | All except $X$         |
| Computation            | Store $\{Y\}$  | Get $\{Y\}$            |
| Semantics              | $\{Y\}$        | $\{Y\}$                |
| Interpretation         | All except $X$ | All except $X$ and $Z$ |

 Table 5.3: Compute on Demand Strategy for Attribute  $a$  Such That  $V(a) = \text{Powerset}(Set_a)$ 

| a Positive Requirement |                             |                           |
|------------------------|-----------------------------|---------------------------|
| Time                   | $t = t_0$                   | $t = t_1$                 |
| $Set_a$                | $\{X, Y\}$                  | $\{X, Y, Z\}$             |
| Intent                 | $X$ only                    | $X$ only                  |
| Computation            | Store $(\{X\}, pos, \perp)$ | Get $(\{X\}, pos, \perp)$ |
| Semantics              | $\{X\}$                     | $\{X\}$                   |
| Interpretation         | $X$ only                    | $X$ only                  |
| b Negative Requirement |                             |                           |
| Time                   | $t = t_0$                   | $t = t_1$                 |
| $Set_a$                | $\{X, Y\}$                  | $\{X, Y, Z\}$             |
| Intent                 | All except $X$              | All except $X$            |
| Computation            | Store $(\{X\}, neg, \top)$  | Get $(\{X\}, neg, \top)$  |
| Semantics              | $\{Y\}$                     | $\{Y, Z\}$                |
| Interpretation         | All except $X$              | All except $X$            |

and Table 5.3. Those tables actually present two strategies. The first strategy is the one that leads to a problem. It is called “Compute before Store” and presented in Table 5.2. The second strategy is how we solve the problem raised by the first strategy. It is called “Compute on Demand” and presented in Table 5.3.

For both strategies, we consider two requirements. The first requirement is a positive requirement, where the tenant enumerates all the authorized values. In Table 5.2a and Table 5.3a, this positive requirement is “ $X$  only”, meaning that only  $X$  is the authorized value. The second requirement is a negative requirement, where the tenant enumerates all the unauthorized values. In Table 5.2b and Table 5.3b, this negative requirement is “all except  $X$ ”, meaning that all values except  $X$  are authorized.

Besides, for both strategies, at time  $t_0$ , the tenant creates the request with the desired requirement (either positive or negative); the request is then stored in the database. At time  $t_1$ , the system runs the embedding algorithm; for this purpose, it retrieves the request of the tenant from the database.

Consider the first strategy. Table 5.2a is the case where everything works right. The *intent* of the tenant (expressed in human language: “ $X$  only”) is converted and *computed* as the set  $\{X\}$ . This set is stored as-is in the database. *Semantically*, with the first strategy, the semantics is in accordance with the computation. For this reason, the *interpretation* of the value  $\{X\}$  is that it represents the requirement “ $X$  only”. If we now move to time  $t_1$ , we are now retrieving the value from the database; the *computation* gives  $\{X\}$ . Again, *semantically*, with the first strategy, the semantics is in accordance with the computation, so we have the same *interpretation* as for time  $t_0$ . In other words, for a positive requirement, the interpretation is constant, and matches, at all times, the intent.

Consider now Table 5.2b with the negative requirement. The intent of the tenant is “all except  $X$ ”. The first strategy states that we “compute before store” so we convert the negative requirement into a positive requirement by using the set subtraction operation on the value set at time  $t_0$  ( $\{X, Y\}$ ); this leaves us with the value  $\{Y\}$ . *Semantically*,  $\{Y\}$  means that we accept only  $Y$ ; but at time  $t_0$ , it is also equivalent to accept every value but  $X$ . However, if we now move to time  $t_1$ , the value we retrieve from the database is  $\{Y\}$ . As the value set has changed (it is now  $\{X, Y, Z\}$ ), the value  $\{Y\}$  means that we accept only  $Y$ ; and at time  $t_1$ , it is also equivalent to accept every value but  $X$  and  $Z$ . The interpretation of the requirement at time  $t_1$  is then “all but  $X$  and  $Z$ ”, which is not the same interpretation than at time  $t_0$ . In other words, for a negative requirement, the interpretation is not constant, and does not match, at all times, the intent.

To correctly match the intent, our solution is to re-compute the value each time we retrieve it. The data we store have a different structure too. Instead of a bare set, it is a tuple of a set, a type and a Boolean. The set in first position is a subset of  $Set_a(t)$  at time  $t$  when the request is registered. The type indicates

Table 5.4: Data Structure for Mutable Attribute At Time  $t_i$  With  $Set_a(t_i) = \{X, Y, Z\}$ 

| Input Set  | Type       | All others | Interpretation      | Store at $t_i$           | Retrieve at $t_j > t_i$         |
|------------|------------|------------|---------------------|--------------------------|---------------------------------|
| $\{X, Y\}$ | <i>pos</i> | $\perp$    | $X$ and $Y$ only    | $(\{X, Y\}, pos, \perp)$ | $\{X, Y\}$                      |
| $\{X, Y\}$ | <i>pos</i> | $\top$     | All but $Z$         | $(\{Z\}, pos, \top)$     | $Set_a(t_j) \setminus \{Z\}$    |
| $\{X, Y\}$ | <i>neg</i> | $\perp$    | All but $X$ and $Y$ | $(\{X, Y\}, neg, \perp)$ | $Set_a(t_j) \setminus \{X, Y\}$ |
| $\{X, Y\}$ | <i>neg</i> | $\top$     | $Z$ only            | $(\{Z\}, neg, \top)$     | $\{Z\}$                         |

whether the requirement is positive (*pos*) or negative (*neg*). The Boolean indicates for its part if the tenant authorizes future values ( $\top$ ) or not ( $\perp$ ). Table 5.4 shows how the different possible values for this structure are treated. The first three columns represent the tenant input, that is, as if the tenant used the GUI presented in Figure 5.2. We use the same “input set” and show how the options “type” and “all others” affect it. The computation at time  $t_i$  column gives how we instantiate the data structure from the tenant input, and store it in the database. The computation at time  $t_j$  column shows how we retrieve and treat the value from the database. In Table 5.4, we assume that  $Set_a(t_i) = \{X, Y, Z\}$ .

When the “all others” option is set by the tenant, we use the complement to the current set,  $Set_a(t_i)$ , in our data structure, as shown in rows 2 and 3 of Table 5.4. It is for this reason that we store  $\{Z\}$ , as  $\{Z\} = Set_a(t_i) \setminus \{X, Y\}$ . Otherwise, we use the input set as is, as shown in rows 1 and 4.

When retrieving a value, we use the complement to the new current set,  $Set_a(t_j)$ , only when the type is *pos* (resp. *neg*) and the “all others” option is not used (resp. is used). Otherwise, we use the first element of the data structure as-is.

Consider now the second strategy with this new structure. In Table 5.3a, the intent “ $X$  only” is converted into  $(\{X\}, pos, \perp)$ . At time  $t_1$ , the value  $(\{X\}, pos, \perp)$  semantically corresponds to  $(X)$ , whose interpretation is “ $X$  only”. The interpretation matches the intent. In Table 5.3b, the intent “all except  $X$ ” is converted into  $(\{X\}, neg, \top)$ . It semantically corresponds to the subtraction of the value set by  $\{X\}$ . At time  $t_1$ , the value  $(\{X\}, neg, \top)$  semantically corresponds to  $(Y, Z)$ , whose interpretation is “all except  $X$ ”. The interpretation, again, matches the intent.

In our implementation, every attribute with a value set derived from a powerset supports this mechanism. In practice, it is required for the tenant-based exclusion attribute, as the set of all tenants registered in the system is expected to grow over time. Attributes such as **!ven** and **!loc** may benefit from such a mechanism when the infrastructure grows and the InPs acquire new hardware.

Note however that this structure has been designed to address a *growing* mutable value set. In other words, the value set at time  $t_{i+1}$  contains the value set

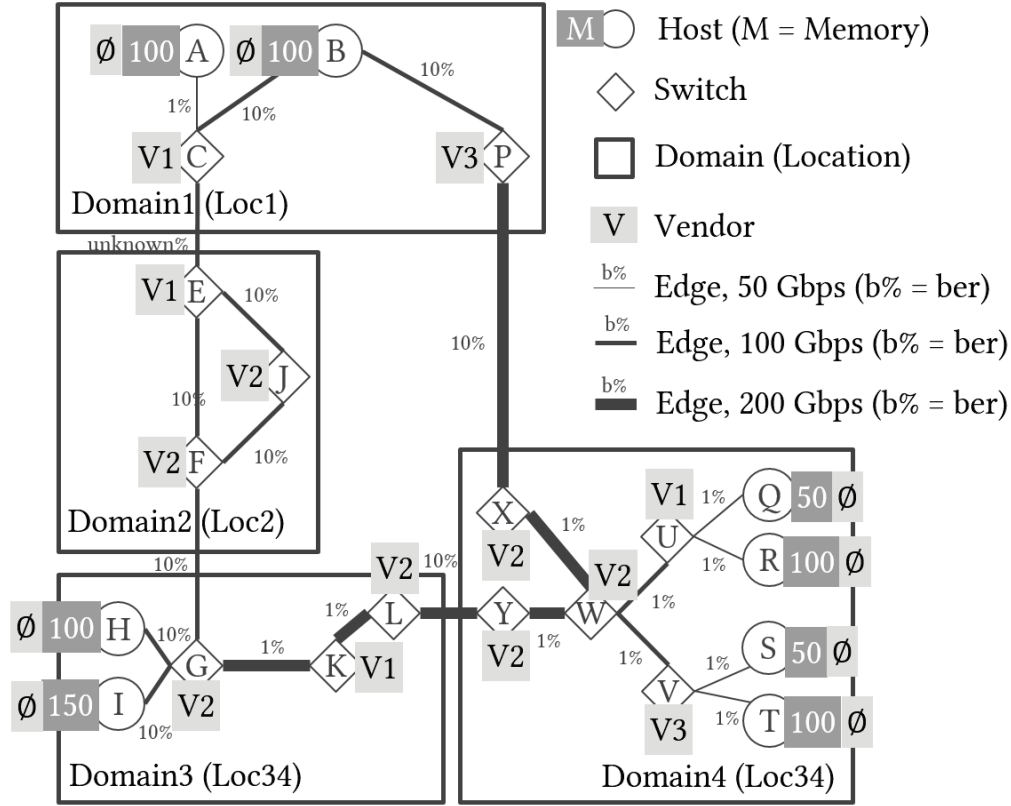


Figure 5.5: The use case substrate, with its sub-domains

at time  $t_i$ . We are unsure if it can be applied if the value set can shrink (lose of values), or if freely evolve (either acquire new values or lose some).

## 5.3 Results and Interpretation

In this section, we evaluate the correctness of our algorithm, and provide some measures about its time computation. We conduct the following experiments: *i)* we show that computation time increases faster with the number of embeddings, and that all embeddings are correctly generated; *ii)* we show how time computation evolves when no suitable embedding exists.

We implement our work under Ubuntu 16.04 on a i7-6700HQ @ 2.60GHz machine with 16 GB RAM and 300 GB of storage. Domains are modeled as separate processes. We do not enforce any delay between calling a domain server and getting its reply, even though we expect such delay to have an impact on the time performances of our algorithm when deployed in a real system. We leave such optimization for future works and here evaluate our algorithm as a proof of concept.

The substrate which is used for all our evaluations is described in Figure 5.5. This figure represents the substrate as a set of interconnected boxes (the domains), with their hosts, switches and links inside the boxes. The substrate is divided into four domains (*Domain1* to *Domain4*). Domains are interconnected following a ring topology. The *Domain2* is special is that there are no host inside. The attribute values for each host, switch and link are also represented. The node location is applied domain-wise, hence the location value is indicated next to the domain name, between brackets. Host memory and vendor are indicated next to the host name. For their part, edges come in three bandwidth flavors (50 Gbps, 100 Gbps, 200 Gbps). The bit error rate is represented as a percentage. Note that for the edge (C, E), the value is *unknown%*. This corresponds to the case when the Infrastructure Providers (InPs) are unable to provide attribute information. As described in Section 3.1, the actual value is set to  $m_{\text{!ber}}$ , which is equal to 100%.

Figure 5.6 evaluates our algorithm for different requests in time and in correctness. The requests come with three different topologies, in which we increase the number of nodes. Those topologies are chain, full-mesh, and star. They do not enforce any requirement: all attributes are set to their default value. Consequently, we generate the maximal number of embeddings for each request.

More specifically, Figure 5.6a shows the measured number of embeddings to be generated for each request. This measured number of embeddings is then compared to the theoretical number of embeddings in Figure 5.6b. For a chain topology, the theoretical number of embeddings is  $3 \times 5^n$  where  $n$  is the number of nodes (see Demonstration 5.1). For a star topology, the theoretical number of embeddings is  $3 \times 5^n$  where  $n$  is the number of nodes (see Demonstration A.1 in appendix). For a full-mesh topology, the theoretical number of embeddings for  $n \in (1, 2, 3, 4)$  is given in Demonstration A.2 in appendix, and is respectively 3, 27, 673, 41985.

### Demonstration 5.1

Let  $f(n)$  be the number of solutions for a chain of  $n$  nodes, with the substrate given in Figure 5.5. We show by induction that  $f(n) = 3 \times 5^n$ . We denote as  $N_i$  where  $i \in (1, \dots, n)$  the nodes in the graph. We denote as  $m_k^n$  a solution for a chain of  $n$  nodes where  $k$  is an index identifying the solution.

**Basis.** When  $n = 1$ , the request is a single node,  $N_1$ . Among the four domains in our substrate, only three contain hosts: *Domain1*, *Domain3*, and *Domain4*. Then we have 3 ways of embedding our request:  $m_1^0(N_1) = \text{Domain1}$ ,  $m_2^0(N_1) = \text{Domain3}$  and  $m_3^0(N_1) = \text{Domain4}$ . Then  $f(1) = 3$ .

**Induction step.** Let  $m_k^n$  be a solution for a chain of  $n$  nodes. Let  $m_q^{n+1}$  be a solution for a chain of  $n + 1$  nodes such that:

$$\begin{aligned} \forall i \in (1, \dots, n), & \quad m_q^{n+1}(N_i) = m_k^n(N_i) \\ \forall i \in (1, \dots, n-1), & \quad m_q^{n+1}((N_i, N_{i+1})) = m_k^n((N_i, N_{i+1})) \end{aligned}$$

Then we need to embed  $N_{n+1}$  and  $(N_n, N_{n+1})$ . Let  $d = m_q^{n+1}(N_n)$  be the domain hosting  $N_n$ .

If  $m_q^{n+1}(N_{n+1}) = d$ , then  $m_q^{n+1}((N_n, N_{n+1})) = (d, (d, d), d)$  by Definition 3.3. We then have described one (1) solution from  $m_k^n$ .

If  $m_q^{n+1}(N_{n+1}) \neq d$ , then we can choose among the two (2) other domains. Let  $d' = m_q^{n+1}(N_{n+1})$ ,  $d' \neq d$ . Let  $d''$  be the remaining domain,  $d'' \neq d'$ ,  $d'' \neq d$ . As the substrate topology is a ring, we have two (2) paths that join  $d$  to  $d'$ . Those paths are  $(d, (d, d'), d')$  and  $(d, (d, d''), d'', (d'', d'), d')$ . They can be chosen for the mapping of  $(N_n, N_{n+1})$ . We then have described four ( $4 = 2 \times 2$ ) other solutions from  $m_k^n$ .

Then  $f(n+1) = (1+4) \times f(n) = 5 \times f(n)$ .

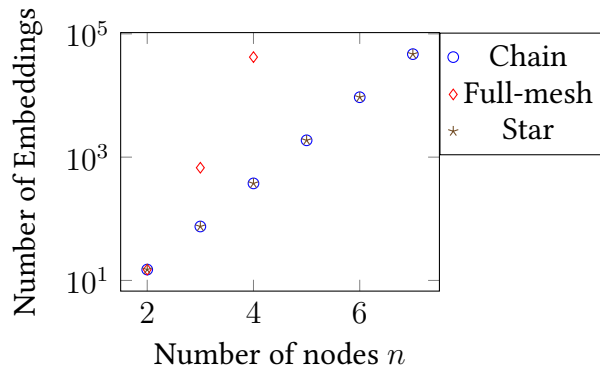
As  $f$  is such that  $f(n+1) = 5 \times f(n)$  and  $f(0) = 3$ , it is a geometric progression whose formula is  $f(n) = 3 \times 5^n$ .  $\square$

As shown in Figure 5.6b, there is no deviation from the expected number of embeddings for any tested request. For star and chain topologies, the experiment was conducted from  $n = 2$  to  $n = 7$  and repeated 50 times until  $n = 5$  and 5 times for  $n = 6$  and  $n = 7$  due to computation time. For full-mesh topology, the experiment was conducted from  $n = 2$  to  $n = 4$  and repeated 5 times due to computation time.

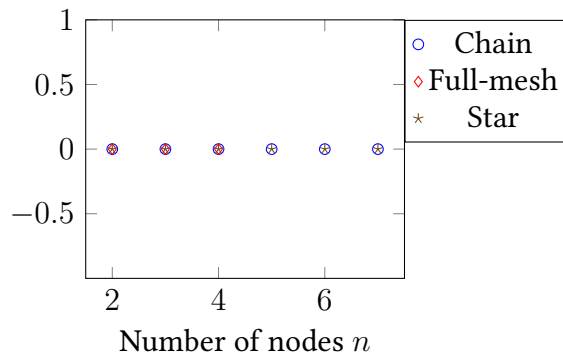
In Figure 5.6c, we show the computation time for each execution. As explained before, this computation time is not expected to represent real time deployment, with communication delays between the domains. We can notice that the computation time grows faster than the number of embeddings between  $n = 6$  and  $n = 7$ . These computation times are way shorter than with the implementation (not disclosed) of our previous algorithm, Boutigny et al. (2018), which took at least several minutes for the  $n = 3$  case.

Figure 5.7 shows computation time for different requests. The idea of this experiment is to evaluate how much time the algorithm needs to find out that there is no solution. For this experiment, we have a reference request (cf. Figure 5.8a) whose requirements are strong enough to not be met, thus there are 0 embeddings. We then strengthen the requirements of this reference request and measure the computation time. Each time, we reset the substrate. The different requests are derived from the reference request following Figure 5.8b.

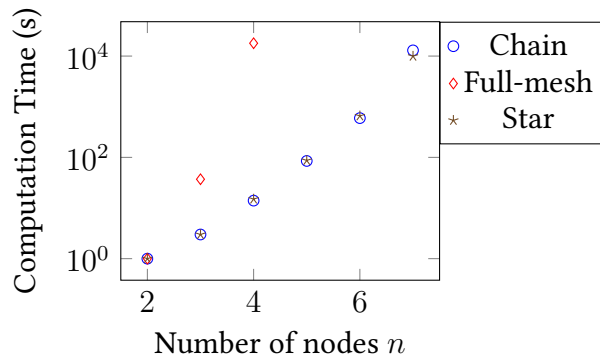
We interpret our results as follows. First, as expected, the time taken by the algorithm to ensure there is no solution is finite, and varies depending on the



a Number of Embeddings per Number of Nodes  $n$  for Chain, Full-mesh and Star Topology



b Difference between Theoretical and Measured Numbers of Embeddings per Number of Nodes  $n$  for Chain, Full-mesh and Star Topology



c Computation Time per Number of Nodes  $n$  for Chain, Full-mesh and Star Topology

Figure 5.6: Computation Time Trend for Chain, Full-mesh and Star Topology

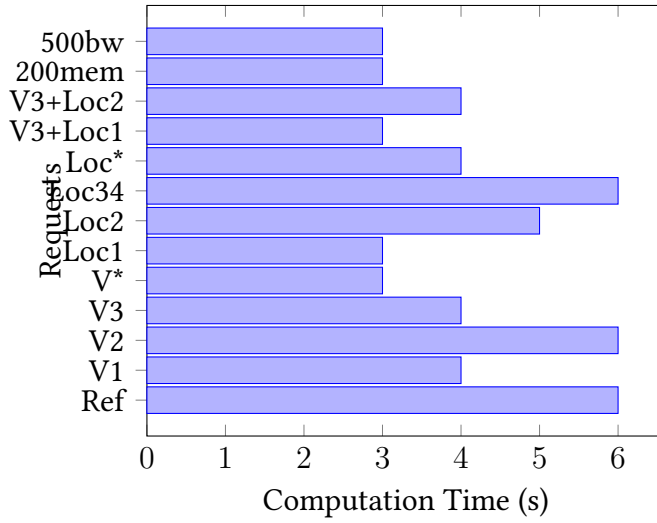


Figure 5.7: Computation Time Trend with 0-Solution Request

requirements. Second, strengthening or adding requirements to a request does not increase the computation time, but keeps it as-is or lowers it.

Third, we cannot conclude that there is a relation between computation time and the requirement values, for a given attribute. For instance, we expected the time to discover that there is no solution to be lower for  $V^*$  than for  $V1$ ,  $V2$  and  $V3$  respectively, but we did not expect the time to discover that there is no solution to be greater for  $Loc^*$  than for  $Loc1$ . In the same time, when combining requirements, such as in  $V3 + Loc2$  and  $V3 + Loc1$  cases, we get a lower time than when considering  $V3$ ,  $Loc2$  or  $Loc1$  independently.

## 5.4 Instantiation of the State-of-the-Art Security-oriented Attributes

In Table 2.1 of the state of the art, we presented different works in the VNE field. Among those works, Alaluna et al. (2017), Fischer et al. (2016), Wang et al. (2015), Liu et al. (2014), and Bays et al. (2014) propose several security requirements. In this section, we show how our model can handle them.

As a general remark, some proposed security requirements may indeed correspond to a combination of several attributes from our model. It is then, somehow, more atomic.

The security level (resp. the cloud trustworthiness level), denoted  $sec$  (resp.  $cloud$ ) in the work from Alaluna et al. (2017), is implemented using a non-binding tolerant attribute, denoted as  $!sec-demand$  (resp.  $!cloud$ ) in Table 5.5. The levels are modelled using integers, where 0 is the lowest level and  $N$  is the highest




 a The Reference Request (*Ref*)

| Request name     | Difference with <i>Ref</i> request  |
|------------------|---|
| <i>500bw</i>     | $d_{(s,d)}[!bw] = d_{(d,s)}[!bw] = 500$   |
| <i>200mem</i>    | $d_s[!mem] = 200$   |
| <i>V3 + Loc2</i> | $d_{(s,d)}[!ven] = d_{(d,s)}[!ven] = Vendors \setminus \{V3\}$<br>$\wedge d_s[!loc] = Locations \setminus \{Loc2\}$ |
| <i>V3 + Loc1</i> | $d_{(s,d)}[!ven] = d_{(d,s)}[!ven] = Vendors \setminus \{V3\}$<br>$\wedge d_s[!loc] = Locations \setminus \{Loc1\}$ |
| <i>Loc*</i>      | $d_s[!loc] = \emptyset$   |
| <i>Loc34</i>     | $d_s[!loc] = Locations \setminus \{Loc34\}$   |
| <i>Loc2</i>      | $d_s[!loc] = Locations \setminus \{Loc2\}$  |
| <i>Loc1</i>      | $d_s[!loc] = Locations \setminus \{Loc1\}$  |
| <i>V*</i>        | $d_{(s,d)}[!ven] = d_{(d,s)}[!ven] = \emptyset$   |
| <i>V3</i>        | $d_{(s,d)}[!ven] = d_{(d,s)}[!ven] = Vendors \setminus \{V3\}$  |
| <i>V2</i>        | $d_{(s,d)}[!ven] = d_{(d,s)}[!ven] = Vendors \setminus \{V2\}$  |
| <i>V1</i>        | $d_{(s,d)}[!ven] = d_{(d,s)}[!ven] = Vendors \setminus \{V1\}$  |

b The Other Requests

Figure 5.8: Tested Requests in Figure 5.7

#### 5.4. Instantiation of the State-of-the-Art Security-oriented Attributes

Table 5.5: Building Blocks of State of the Art Security-oriented Attributes

| Reference work        | Authors Attribute      | Our Attribute        | $V(a)$                      | Ordering operator<br>(demand $\leq$ offer)   | +              | $m_a$             | $M_a$       |
|-----------------------|------------------------|----------------------|-----------------------------|--|----------------|-------------------|-------------|
| Alaluna et al. (2017) | <i>sec</i>             | <b>!sec-demand</b>   | $[0, N] \subset \mathbb{N}$ | Natural order                                | max            | 0                 | $N$         |
| Liu et al. (2014)     | <i>dem<sup>V</sup></i> |                      |                             |  |                |                   |             |
| Alaluna et al. (2017) | <i>avail</i>           | <b>!domain</b>       | Powerset of domains         | $\supset$ (offer must be in demand)          | $\cap$         | All domains       | $\emptyset$ |
| Wang et al. (2015)    | virtual network plan   |                      |                             |  |                |                   |             |
| Alaluna et al. (2017) | <i>cloud</i>           | <b>!cloud</b>        | $[0, N] \subset \mathbb{N}$ | Natural order                                | max            | 0                 | $N$         |
| Liu et al. (2014)     | <i>dem<sup>S</sup></i> | <b>!sec-level</b>    | $[0, N] \subset \mathbb{N}$ | $\geq$<br>(demand must be higher than offer) | min            | $N$               | 0           |
| Bays et al. (2014)    | $K_{r,i}^V$            | <b>!crypto</b>       | Powerset of crypto suites   | $\supset$<br>(offer must be in demand)       | $\cap$         | All crypto suites | $\emptyset$ |
|                       | $S^V$                  | <b>!loc</b>          | Powerset of locations       | $\supset$<br>(offer must be in demand)       | $\cap$         | All locations     | $\emptyset$ |
| Bays et al. (2014)    | $X$                    | <b>!exc-requests</b> | Powerset of <i>Requests</i> |  | Not applicable |                   |             |
| Wang et al. (2015)    | virtual node plan      |                      |                             |  |                |                   |             |
| Bays et al. (2014)    | $K_{r,i}^V$            | <b>!has-crypto</b>   | $\{\perp, \top\}$           | Boolean order                                | $\vee$         | $\perp$           | $\top$      |
| Wang et al. (2015)    | virtual edge plan      |                      |                             |  |                |                   |             |
| Wang et al. (2015)    | virtual edge plan      | <b>!tunnel</b>       | $\{\perp, \top\}$           | Boolean order                                | $\vee$         | $\perp$           | $\top$      |
| Fischer et al. (2016) | data encryption        |                      |                             |  |                |                   |             |
| Fischer et al. (2016) | <i>TH</i>              | <b>!tpm</b>          | $\{\perp, \top\}$           | Boolean order                                | $\vee$         | $\perp$           | $\top$      |
|                       | Firewall               | <b>!is-firewall</b>  | $\{\perp, \top\}$           | Boolean order                                | $\vee$         | $\perp$           | $\top$      |

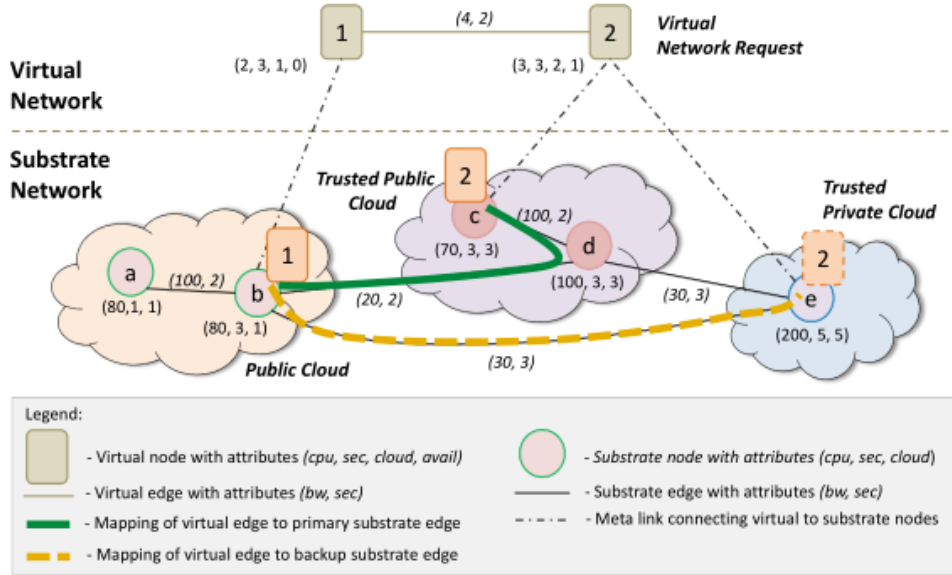


Figure 2: Example of the embedding of a virtual network request (top) onto a multi-cloud substrate network (bottom). The figure also illustrates the various constraints and the resulting mapping after the execution of our MILP formulation.

Figure 5.9: Example of Embedding from Alaluna et al.

level.  $N$  itself is an integer whose value is arbitrary chosen. The order is such that the offer must have a higher level than the demand. Each substrate resource has a security level, and also a cloud trustworthiness level. The cloud trustworthiness level is constant within the same cloud.

To model the backup requirement, denoted *avail* in Alaluna et al. (2017), we proceed step by step. First, we note that Alaluna et al. (2017) explicitly distinguish the primary or working resource from the secondary or replicated resource. In other words, the tenant can only require at most one backup. Alaluna et al. (2017) says that “[t]his [the backup requirement] causes the embedding to allocate an additional node and the necessary links to connect it to the other nodes”. This means that the actual virtual network request treated by their VNE algorithm is an augmented graph where some resources are duplicated, except that some are labelled as being *backup*. This is reproduced in Figure 5.9. Note that their *avail* attribute is a scale, where 0 means no backup, 1 means a backup in another cloud, and 2 means a backup in the same cloud.

To enable the same requirement in our model, we propose:

1. that tenants create the so-called augmented graph by themselves: if they want a backup for a node or an edge, they duplicate the appropriate node or edge

in the graph; absence of duplicate for a node or an edge corresponds to *avail* level 0;

2. leverage the **!domain** attribute (introduced in Subsection 3.3.4; its structure has been given in Table 3.3, and repeated in Table 5.5 for convenience), to force the backup node (or edge) to be in the same cloud (*avail* level 2), thanks to attribute-specific collocation, or a different cloud (*avail* level 1), thanks to attribute-specific exclusion.

Our proposition enables also:

1. to require more than one backup, by duplicating their nodes or edges as many times as needed;
2. to require that the different backups are all in different clouds;
3. to apply more advanced exclusion and collocation constraints if needed, for instance based on geographical location, or switch vendors.

Liu et al. (2014) define two security-oriented attributes:  $dem^V$  and  $dem^S$ .  $dem^V$  is tenant-oriented: it is a security level that the substrate resource should reach (the substrate resource can have a higher security level). The other attribute,  $dem^S$ , is InP-oriented: it is a security level that the virtual resource should reach (again, the virtual resource can have a higher security level). Modelling  $dem^V$  is straightforward, as it directly corresponds to the *sec* attribute from Alaluna et al. (2017). It is modelled as the **!sec-demand** in Table 5.5.

The  $dem^S$  attribute, for its part, is enabled by a similar structure, but keeping in mind its InP-oriented nature. It is instantiated as the **!sec-level** attribute. With **!sec-demand**, the demand represents the desired security level, while the offer represents the security level of the substrate resource. For **!sec-level**, the demand represents the security level of the virtual resource, while the offer represents the desired security level. As such, contrary to **!sec-demand**, we want the demand to be greater than the offer. For this reason, we do not use the Natural order, conventionally  $\leq$ , but its dual,  $\geq$ . It follows that when comparing multiple demands to the same offer, all demands are satisfied if the least of them is greater than the offer. Then, the addition in our structure is based on the *min* function, instead of the *max* function for **!sec-demand**. It also follows that the least element for the  $\geq$  order is  $N$ , and 0 is the greatest element.

The main consequence of our structure for **!sec-demand** is that it still is tenant-oriented, in the sense that there is no incentive for tenants to declare the security level of their virtual resources: by default, it is set to  $N$ . This is a direct consequence of how this model has been designed. We want the tenants to make requirements, and the InPs to fulfill those requirements. When we try to enable InPs to make requirements, as we do with **!sec-demand**, it can only

be a requirement to which the tenant *consents*. In other words, the **!sec-demand** attribute is only applied when tenants explicitly accepts to declare the security level of their resources.

The actual incentive is elsewhere. At the intra-domain level, InPs have full control over whether or not they accept the virtual network requests they receive. To make the **!sec-demand** attribute mandatory, InPs can then only treat virtual network requests declaring their security levels.

Bays et al. (2014) propose three attributes. In their work,  $K_{r,i}^V$  is a Boolean variable that tells if a virtual router  $r$  needs a substrate resource with a cryptographic support. This can be modelled with a Boolean structure as well, within our attribute **!has-crypto**. It is also possible to enable the tenant to specify which cryptographic suite is needed, through our attribute **!crypto**.

The second attribute Bays et al. (2014) propose is denoted  $X$  and enable tenants to exclude requests as a whole. In other words, they are enabled to declare, at the virtual network request level, which other virtual network requests should not share any substrate resource with them. This is a particular case of our **!exc-requests** indeed, where all virtual resources within the same request have their **!exc-requests** set to the same set of virtual network requests. Our attribute, **!exc-requests**, then supersedes the proposed attribute,  $X$ .

The last attribute Bays et al. (2014) propose is denoted  $S^V$  and enable tenants to force a location for their virtual resources. It is again a particular case of one of our attributes, **!loc**, where the virtual resources only authorize a single location. Our attribute enables also tenants to authorize multiple locations, in which their virtual resources can be instantiated.

Wang et al. (2015) allow tenants to require a security plan on three categories: virtual-network-wise, virtual-node-wise, and virtual-link-wise. Each plan further propose three levels. Each time, the lowest level implies no restriction whatsoever. We then focus on the modelling of the medium and the high levels of these plans.

The virtual-network-wise plan works as follows. The medium security level states that virtual networks from other tenants can be in the same datacenter than the virtual network of one tenant, only if those virtual networks are trusted by this tenant. In other words, virtual networks which are untrusted cannot be in the same datacenter. We then recognize in this sentence the application of an attribute-specific exclusion, based on the **!domain** attribute. In practice, with our model, this means that the tenant will apply Equation (5.1). This equation reads as follows. Given  $Untrusted$  the set of all requests untrusted by the tenant, a constraint  $O_{!domain}(i) \neq O_{!domain}(i')$  is added for each virtual resource of the tenant and each virtual resource of the aforementioned request. The constraint itself means that those two virtual resources cannot be in the same domain.

$$\forall i \in \text{Resources}^R, i' \in \text{Resources}^{R'}, R' \in \text{Untrusted}, O_{!domain}(i) \neq O_{!domain}(i') \quad (5.1)$$

The high security level of the virtual-network-wise plan states that no other virtual network can be in the same datacenter. It then corresponds to an infrastructure dedication. This is also encompassed by Equation (5.1), provided that *Untrusted* is the set of all the other virtual network requests.

Consequently, the virtual-network-wise plan is directly derived from our **!domain** attribute.

The virtual-node-wise plan is very similar. The medium level states that the virtual nodes of the tenant can only be collocated with virtual nodes from other trusted virtual networks. The high level states for its part that the virtual nodes of the tenant cannot be collocated with any other virtual node of any other virtual network. This directly corresponds to the definition of our **!exc-requests** attribute.

The last plan is the virtual-link-wise plan. The medium level states that the physical resources hosting the virtual nodes at the tips of the virtual edge must have some cryptographic support. The high level states for its part the same thing than the medium level, and adds that every physical node along the path chosen for the virtual edge should also have some cryptographic support. We here recognize two attributes: one node attribute, and one link attribute. The node attribute has already been mentioned with Bays et al. (2014): it is **!has-crypto**. The link attribute is similar. It is actually an aggregation link attribute (we remind that the aggregation function is given in Definition 3.2), whose aggregation function is the conjunction ( $\wedge$ ), as shown in Equation (5.2). This attribute is denoted **!tunnel**. It works as follows. Each substrate node indicates if it supports cryptographic tunneling. Then, thanks to the aggregation function, each substrate link is said to be supporting cryptographic tunneling if its both tips supports it. For the tenant, this means that each virtual edge can have two values for the **!tunnel** attribute. If the value is  $\perp$ , there is no end-to-end cryptographic support. If the value is  $\top$ , there is end-to-end cryptographic support.

$$\forall (i_h, i_t) \in L^S, o_{i_h}[!tunnel] \cdot !tunnel o_{i_t}[!tunnel] = o_{i_h}[!tunnel] \wedge o_{i_t}[!tunnel] \quad (5.2)$$

As a summary, the virtual-network-wise plan is enabled by our **!domain** attribute, the virtual-node-wise plan is enabled by our **!exc-requests** attribute, and the virtual-link-wise plan is enabled by a combination of two attributes, namely **!has-crypto** and **!tunnel**.

Finally, Fischer et al. (2016) propose three examples of security attributes. The first one is a node attribute, denoted  $TH$ , which stands for “trusted hardware”. The idea is that the tenant can demand that a substrate host supports some trusted hardware, or not. The semantics is very close to the `!has-crypto` attribute, hence they share a common structure in Table 5.5.

The second proposed attribute is about “data encryption” along a path. This example has already been treated with the virtual edge plan of Wang et al. (2015), thanks to the `!tunnel` attribute.

The last proposed attribute is described as topological. The idea is that, within the same virtual network request, the tenant identifies two (or more) areas<sup>1</sup>. One area should be protected by a firewall, and the area should not. Besides, traffic coming from the non-protected area to the protected area should go through the firewall.

To reproduce the example, we actually only need one attribute, `!is-firewall`, which enables the tenant to tell whether a virtual resource is a firewall or not. Likewise, the InPs declare which substrate hosts are dedicated to firewalling, or not. The two areas themselves are defined by the tenant, by manipulating the topology of the virtual network request. The tenant only adds one special node, `fw` for instance, such that its `!is-firewall` attribute is set to true.

But we can actually take a step further by taking inspiration from the Network Function Virtualization (NFV) paradigm and the Service Function Chain (SFC) concept. In their formulation of the firewall example, Fischer et al. (2016) implicitly make the assumption that the given virtual network is protected by a single firewall, although multiple firewalls are provided by the InPs. With NFV, we can imagine instead that the tenant explicitly tells instead which traffic should be protected by *a* firewall. As such, different firewalls can be leveraged to protect one area from the other. To do so, the tenant only needs to add one node with the `!is-firewall` set to true, in the middle of each virtual edge corresponding to a traffic that should be protected.

## 5.5 Conclusion

This chapter presented our implementation, and how we evaluate it. The implementation leverages the SMT oriented model from Chapter 3, and follows the top-down approach described in Chapter 4. The implementation of our work led to interesting challenges to implement the mutability of some attributes. For its part, the evaluation has been done around two axes. The first axis was to verify

---

<sup>1</sup>Fischer et al. (2016) use actually the term “domain”, but as we already widely use this term to refer to the domains of the InPs, we prefer to use “area” instead.

that the algorithm produces only correct solutions; and to measure the computation time in different cases. The second axis was to verify that we are indeed able to model every security-oriented attribute from the state of the art. In other words, that our model is atomic enough. We also showed that we are able to provide substantially interesting features when modelling such attributes.





### Conclusion and Future Tracks

#### 6.1 Conclusion

In this thesis, we present how to solve the slice embedding problem in a multi-domain context where domain owners are reluctant to expose their resource attributes or topology. Our solution distinguishes two levels, the intra-domain and the inter-domain, where the inter-domain is known by a trusted third party, namely the Virtual Network Provider (VNP), and the intra-domain is only known by the Infrastructure Provider (InP) owning it.

To solve this problem, we first express an attribute; formulate the Virtual Network Embedding (VNE) problem in a single domain scenario; use a topological heuristic to build the inter-domain and the intra-domain formulations; these formulations are covered by our algorithm, enabling the Infrastructure Providers (InPs) and the VNP to cooperate when generating the final embedding solution. The proposed system displays interesting performance, and prove to encompass the security-oriented requirements given by the state-of-the-art. Besides, our attribute model is general enough to encompass other security-oriented requirements.

This thesis is the conclusion of a three-year work; it has been a very interesting and stimulating personal experience. As I am concluding this work, I would like to open it to other possible future tracks, which I did not address. There are three main ideas I think of. First of all, although we solve the slice embedding problem, which is related to 5G, we do not have a proper model for attributes like latency. My suggestion of how to encompass such an attribute is given in Section 6.2. Another use case I had in mind while developing this thesis is the time-dependent requirement. In other words, it is the idea that the tenants may

have a requirement depending on the time, and they want our system to automatically take it into account. This is described in Section 6.3.

## 6.2 Cumulative Link Attributes

We propose to refer to such a new link attribute as a cumulative link attribute. The idea is that we leverage the addition of the tolerant structure, that is, the  $+$  in the tuple  $(V(a), \leq, +, m_a, M_a, \times)$ , as it is the operation we already use to compose multiple demands together with binding attributes (see Equation (3.6) as a reminder). As we already generate the set of loop-free paths in our Satisfiability Modulo Theories (SMT) formulation, we can, in the same time, give an attribute value for all cumulative link attributes upon these paths. This will lead also to define a new set of equations, very similar to Equation (C8), but using  $j$  in  $P$  instead of  $j$  in  $L^S$ .

Although adding the equations is simple, we will face the same problem as when we designed the attribute-specific exclusion and collocation in Section 3.3. Indeed, we must still respect the limited information disclosure, which is broken by paths going through multiple domains: naturally, each InP can deduce the latency upon links from neighboring InPs from the latency they compute within their own domain, and the information that the virtual link latency demand is fulfilled, or not.

To overcome this problem, we propose the same analysis as for Section 3.3, which is that latency requirement (or, likewise, any of those new cumulative link attribute requirement) should come with a domain collocation constraint, using the *domain* attribute.

## 6.3 Time-dependent Requirement

We already overcome time-dependence and mutability of attributes in Section 5.2. The time-dependence we now think of is of different nature, as it the tenant requirement itself which can vary over time. The use case for such behavior is simple indeed: a tenant may only need some virtual resources during work hours; during time, the virtual resources can be removed, or paused.

In other words, the virtual network request itself has different states. In our example, there are two states. One where a given link exists, because it is required for transferring some traffic between two virtual nodes. The other where the same link does not exist, because it is not required, and the tenant does not want to be charged for it. It may also be a defensive mechanism, to ensure that the virtual nodes really are inactive over night.

The fact that we have different states slightly changes our resolution method. What we suggest is that the embedding algorithm should also consider the substrate as a set of graphs, each graph describing the state of the substrate at a given period. The embedding algorithm then periodically checks and re-embeds, if necessary, the virtual networks, according to the new state of the virtual network request.

## Conclusion and Future Tracks

---

---

## Bibliography

- Alaluna, Max, Luís Ferrolho, José Rui Figueira, Nuno Neves, and Fernando Ramos (2017). “Secure virtual network embedding in a multi-cloud environment”. In: *Computing Research Repository* abs/1703.01313.
- Arora, Dushyant, Marcin Bienkowski, Anja Feldmann, Gregor Schaffrath, and Stefan Schmid (2011). “Online strategies for intra and inter provider service migration in virtual networks”. In: ACM Press, p. 1. ISBN: 978-1-4503-0975-2. DOI: 10.1145/2124436.2124449. URL: <http://dl.acm.org/citation.cfm?doid=2124436.2124449> (visited on 05/03/2018).
- Ballani, Hitesh, Paolo Costa, Thomas Karagiannis, and Ant Rowstron (2011). “Towards predictable datacenter networks”. In: *ACM SIGCOMM Computer Communication Review*. Vol. 41. ACM, pp. 242–253. URL: <http://dl.acm.org/citation.cfm?id=2018465> (visited on 04/08/2016).
- Bari, M. F., R. Boutaba, R. Esteves, L. Z. Granville, M. Podlesny, M. G. Rabbani, Q. Zhang, and M. F. Zhani (2013). “Data Center Network Virtualization: A Survey”. In: *IEEE Communications Surveys Tutorials* 15.2, pp. 909–928. ISSN: 1553-877X. DOI: 10.1109/SURV.2012.090512.00043.
- Barrett, Clark, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli (2011). “CVC4”. In: *Computer Aided Verification*. Ed. by Ganesh Gopalakrishnan and Shaz Qadeer. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 171–177. ISBN: 978-3-642-22110-1.
- Barrett, Clark, Pascal Fontaine, and Aaron Stump (2017). “The SMT-LIB Standard”. In: p. 104.
- Bays, L. R., L. P. Gasparý, R. Ahmed, and R. Boutaba (Apr. 2016). “Virtual network embedding in software-defined networks”. In: *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium. Istanbul, Turkey: IEEE, pp. 10–18. ISBN: 978-1-5090-0223-8. DOI: 10.1109/NOMS.2016.7502791. URL: <http://ieeexplore.ieee.org/document/7502791/> (visited on 07/31/2019).

- Bays, L. R., R. R. Oliveira, L. S. Buriol, M. P. Barcellos, and L. P. Gasparly (2014). “A heuristic-based algorithm for privacy-oriented virtual network embedding”. In: *2014 IEEE Network Operations and Management Symposium (NOMS)*. 2014 IEEE Network Operations and Management Symposium (NOMS), pp. 1–8. DOI: 10.1109/NOMS.2014.6838360.
- Bellavista, Paolo, Franco Callegati, Walter Cerroni, Chiara Contoli, Antonio Corradi, Luca Foschini, Alessandro Pernaflini, and Giuliano Santandrea (Dec. 2015). “Virtual network function embedding in real cloud environments”. In: *Computer Networks* 93, pp. 506–517. ISSN: 13891286. DOI: 10.1016/j.comnet.2015.09.034. URL: <http://linkinghub.elsevier.com/retrieve/pii/S1389128615003588> (visited on 10/18/2017).
- Bienkowski, Marcin, Anja Feldmann, Dan Jurca, Wolfgang Kellerer, Gregor Schaf-frath, Stefan Schmid, and Joerg Widmer (2010). “Competitive analysis for service migration in VNets”. In: ACM Press, p. 17. ISBN: 978-1-4503-0199-2. DOI: 10.1145/1851399.1851403. URL: <http://portal.acm.org/citation.cfm?doid=1851399.1851403> (visited on 05/04/2018).
- Bjørner, Nikolaj (2016). *Enumeration of Minimal Unsatisfiable Cores and Maximal Satisfying Subsets*. URL: <https://github.com/Z3Prover/z3/blob/master/examples/python/mus/marco.py> (visited on 10/01/2019).
- Bjørner, Nikolaj and Leonardo de Moura (2018). *The Z3 Theorem Prover, ver. 4.8.1*. URL: <https://github.com/Z3Prover/z3/tree/z3-4.8.1> (visited on 10/01/2019).
- Blenk, A., A. Basta, M. Reisslein, and W. Kellerer (2016). “Survey on Network Virtualization Hypervisors for Software Defined Networking”. In: *IEEE Communications Surveys Tutorials* 18.1, pp. 655–685. ISSN: 1553-877X. DOI: 10.1109/COMST.2015.2489183.
- Boutigny, Francois, Stephane Betge-Brezetz, Herve Debar, Gregory Blanc, Antoine Lavignotte, and Ion Popescu (Feb. 2018). “Multi-Provider Secure Virtual Network Embedding”. In: *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*. 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS). Paris: IEEE, pp. 1–5. ISBN: 978-1-5386-3662-6. DOI: 10.1109/NTMS.2018.8328706.
- Burris, Stanley (2012). *A Course in universal algebra: the millenium edition*. OCLC: 994848166. ISBN: 978-0-9880552-0-9. URL: <https://www.math.uwaterloo.ca/~snburris/htdocs/UALG/univ-algebra2012.pdf> (visited on 08/19/2019).
- Chapoutot, Alexandre (n.d.). “Résolution efficace des modèles logiques - IA303”. In: (), p. 48.
- Chau, P. and Y. Wang (Oct. 2014). “Security-Awareness in Network Virtualization: A Classified Overview”. In: IEEE, pp. 545–550. ISBN: 978-1-4799-6036-1 978-1-4799-6035-4. DOI: 10.1109/MASS.2014.59. URL: <http://ieeexplore.ieee.org/document/7035742/> (visited on 06/14/2017).

- Chowdhury, Mosharaf, Fady Samuel, and Raouf Boutaba (2010). “Polyvine: policy-based virtual network embedding across multiple domains”. In: *Proceedings of the Second ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures*. ACM, pp. 49–56.
- Christ, Jürgen, Jochen Hoenicke, and Alexander Nutz (2012). “SMTInterpol: An Interpolating SMT Solver”. In: *Model Checking Software*. Ed. by Alastair Donaldson and David Parker. Red. by David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, and Gerhard Weikum. Vol. 7385. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 248–254. ISBN: 978-3-642-31758-3 978-3-642-31759-0. DOI: 10.1007/978-3-642-31759-0\_19. URL: [http://link.springer.com/10.1007/978-3-642-31759-0\\_19](http://link.springer.com/10.1007/978-3-642-31759-0_19) (visited on 07/11/2019).
- Davis, Martin, George Logemann, and Donald Loveland (1961). *A machine program for theorem-proving*. In collab. with Institute of Fine Arts Library New York University. New York: Courant Institute of Mathematical Sciences, New York University. 44 pp. URL: <http://archive.org/details/machineprogramfo00davi> (visited on 10/06/2019).
- De Moura, Leonardo and Nikolaj Bjørner (Sept. 1, 2011). “Satisfiability modulo theories: introduction and applications”. In: *Communications of the ACM* 54.9, p. 69. ISSN: 00010782. DOI: 10.1145/1995376.1995394. URL: <http://dl.acm.org/citation.cfm?doid=1995376.1995394> (visited on 02/13/2019).
- Delmas, Kevin, Remi Delmas, and Claire Pagetti (June 2017). “SMT-based architecture modelling for safety assessment”. In: *2017 12th IEEE International Symposium on Industrial Embedded Systems (SIES)*. 2017 12th IEEE International Symposium on Industrial Embedded Systems (SIES). Toulouse, France: IEEE, pp. 1–8. ISBN: 978-1-5386-3166-9. DOI: 10.1109/SIES.2017.7993379. URL: <http://ieeexplore.ieee.org/document/7993379/> (visited on 07/10/2019).
- Di, Hao, Vishal Anand, Hongfang Yu, Lemin Li, Dan Liao, and Gang Sun (2013). “Quality of service aware virtual network mapping across multiple domains”. In: *Globecom Workshops (GC Wkshps), 2013 IEEE*. IEEE, pp. 476–481.
- Dietrich, David, Amr Rizk, and Panagiotis Papadimitriou (2015). “Multi-provider virtual network embedding with limited information disclosure”. In: *IEEE Transactions on Network and Service Management* 12.2, pp. 188–201. ISSN: 1932-4537. DOI: 10.1109/TNSM.2015.2417652.
- Dutertre, Bruno (2014). “Yices 2.2”. In: *Computer Aided Verification*. Ed. by Armin Biere and Roderick Bloem. Cham: Springer International Publishing, pp. 737–744. ISBN: 978-3-319-08867-9.



- Fajjari, I., N. Aitsaadi, G. Pujolle, and H. Zimmermann (Dec. 2011). “VNR Algorithm: A Greedy Approach for Virtual Networks Reconfigurations”. In: *IEEE*, pp. 1–6. ISBN: 978-1-4244-9268-8 978-1-4244-9266-4 978-1-4244-9267-1. DOI: 10.1109/GLOCOM.2011.6134006. URL: <http://ieeexplore.ieee.org/document/6134006/> (visited on 05/03/2018).
- Fischer, Andreas, Juan Felipe Botero Vega, Michael Duelli, Daniel Schlosser, Xavier Hesselbach Serra, and Hermann de Meer (2011). “ALEVIN - A framework to develop, compare, and analyze virtual network embedding algorithms”. In: *Open-Access-Journal Electronic Communications of the EASST*, pp. 1–12. URL: <https://upcommons.upc.edu/handle/2117/15170> (visited on 02/13/2019).
- Fischer, Andreas, Juan Felipe Botero, Michael Till Beck, Hermann de Meer, and Xavier Hesselbach (2013). “Virtual network embedding: a survey”. In: *IEEE Communications Surveys & Tutorials* 15.4, pp. 1888–1906. ISSN: 1553-877X. DOI: 10.1109/SURV.2013.013013.00155.
- Fischer, Andreas, Ramona Kühn, Waseem Mandarawi, and Hermann de Meer (2016). “Modeling Security Requirements for VNE algorithms”. In: *Proceedings of the 10th EAI International Conference on Performance Evaluation Methodologies and Tools*. 10th EAI International Conference on Performance Evaluation Methodologies and Tools. Taormina, Italy: ACM. ISBN: 978-1-63190-141-6. DOI: 10.4108/eai.25-10-2016.2266673. URL: <http://eudl.eu/doi/10.4108/eai.25-10-2016.2266673> (visited on 02/13/2019).
- Fulton, Jim (2018). *ZODB: ZODB, a Python object-oriented database, ver. 5.4.0*. URL: <http://www.zodb.org/> (visited on 10/01/2019).
- Ganzinger, Harald, George Hagen, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli (2004). “DPLL(T): Fast Decision Procedures”. In: *Computer Aided Verification*. Ed. by Rajeev Alur and Doron A. Peled. Red. by Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Dough Tygar, Moshe Y. Vardi, and Gerhard Weikum. Vol. 3114. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 175–188. ISBN: 978-3-540-22342-9 978-3-540-27813-9. DOI: 10.1007/978-3-540-27813-9\_14. URL: [http://link.springer.com/10.1007/978-3-540-27813-9\\_14](http://link.springer.com/10.1007/978-3-540-27813-9_14) (visited on 10/06/2019).
- Hagberg, Aric (2018). *Software for complex networks, ver. 2.2*. URL: <https://pypi.org/project/networkx/> (visited on 10/01/2019).
- Haider, Aun, Richard Potter, and Akihiro Nakao (2009). “Challenges in Resource Allocation in Network Virtualization”. In: *20th ITC Specialist Seminar*. Vol. 18, p. 20. URL: [http://www.itcspecialistseminar.com/paper/itcss09\\_Haider.pdf](http://www.itcspecialistseminar.com/paper/itcss09_Haider.pdf) (visited on 08/17/2016).

- Heizmann, Matthias, Aina Niemetz, Giles Reger, and Tjark Weber (July 13, 2018). *Competition-Wide Scoring for the Main Track*. SMT-COMP 2018. URL: <http://smtcomp.sourceforge.net/2018/results-competition-main.shtml> (visited on 07/11/2019).
- Hellkamp, Marcel (2018). *bottle: Fast and simple WSGI-framework for small web-applications, ver. 0.12.13*. URL: <http://bottlepy.org/> (visited on 10/01/2019).
- Houidi, Ines, Wajdi Louati, Walid Ben Ameer, and Djamal Zeglache (Mar. 2011). “Virtual network provisioning across multiple substrate networks”. In: *Computer Networks* 55.4, pp. 1011–1023. ISSN: 13891286. DOI: 10.1016/j.comnet.2010.12.011. URL: <http://linkinghub.elsevier.com/retrieve/pii/S1389128610003786> (visited on 06/20/2017).
- Katz, Guy, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer (2017). “Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks”. In: *Computer Aided Verification*. Ed. by Rupak Majumdar and Viktor Kunčak. Vol. 10426. Cham: Springer International Publishing, pp. 97–117. ISBN: 978-3-319-63386-2 978-3-319-63387-9. DOI: 10.1007/978-3-319-63387-9\_5. URL: [http://link.springer.com/10.1007/978-3-319-63387-9\\_5](http://link.springer.com/10.1007/978-3-319-63387-9_5) (visited on 07/10/2019).
- Kolliopoulos, S. G. and C. Stein (Oct. 1997). “Improved approximation algorithms for unsplittable flow problems”. In: *Proceedings 38th Annual Symposium on Foundations of Computer Science*. Proceedings 38th Annual Symposium on Foundations of Computer Science, pp. 426–436. DOI: 10.1109/SFCS.1997.646131.
- Leelipushpam, P. G. Jeba and J. Sharmila (2013). “Live VM Migration Techniques in Cloud Environment - A Survey”. In: *2013 IEEE Conference on Information Communication Technologies (ICT)*. 2013 IEEE Conference on Information Communication Technologies (ICT), pp. 408–413. DOI: 10.1109/CICT.2013.6558130.
- Li, J., N. Zhang, Q. Ye, W. Shi, W. Zhuang, and X. Shen (Dec. 2017). “Joint Resource Allocation and Online Virtual Network Embedding for 5G Networks”. In: *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*. GLOBECOM 2017 - 2017 IEEE Global Communications Conference, pp. 1–6. DOI: 10.1109/GLOCOM.2017.8254072.
- Liffiton, Mark H., Alessandro Previti, Ammar Malik, and Joao Marques-Silva (Apr. 1, 2016). “Fast, flexible MUS enumeration”. In: *Constraints* 21.2, pp. 223–250. ISSN: 1572-9354. DOI: 10.1007/s10601-015-9183-0.
- Liu, S., Z. Cai, H. Xu, and M. Xu (2014). “Security-aware virtual network embedding”. In: *2014 IEEE International Conference on Communications (ICC)*. 2014 IEEE International Conference on Communications (ICC), pp. 834–840. DOI: 10.1109/ICC.2014.6883423.

- Ludwig, Arne (2016). “Dynamic aspects of network virtualization”. PhD thesis. URL: <https://www.depositonce.tu-berlin.de/handle/11303/5397> (visited on 08/23/2016).
- Mano, Toru, Takeru Inoue, Dai Ikarashi, Koki Hamada, Kimihiro Mizutani, and Osamu Akashi (2016). “Efficient virtual network optimization across multiple domains without revealing private information”. In: *IEEE Transactions on Network and Service Management* 13.3, pp. 477–488. ISSN: 1932-4537. DOI: 10.1109/TNSM.2016.2582179.
- Mccarthy, J. (1962). “Towards a Mathematical Science of Computation”. In: *In IFIP Congress*. North-Holland, pp. 21–28.
- Mehraghdam, Sevil, Matthias Keller, and Holger Karl (2014). “Specifying and placing chains of virtual network functions”. In: *Cloud Networking (Cloud-Net), 2014 IEEE 3rd International Conference on*. IEEE, pp. 7–13. URL: <http://ieeexplore.ieee.org/abstract/document/6968961/> (visited on 05/30/2017).
- Moura, Leonardo de and Nikolaj Bjorner (Nov. 2009). “Generalized, efficient array decision procedures”. In: *2009 Formal Methods in Computer-Aided Design*. 2009 9<sup>th</sup> International Conference Formal Methods in Computer-Aided Design (FMCAD). Austin, TX: IEEE, pp. 45–52. DOI: 10.1109/FMCAD.2009.5351142. URL: <http://ieeexplore.ieee.org/document/5351142/> (visited on 08/19/2019).
- Moura, Leonardo de and Nikolaj Bjørner (2008). “Z3: An Efficient SMT Solver”. In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by C. R. Ramakrishnan and Jakob Rehof. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 337–340. ISBN: 978-3-540-78800-3.
- Nonde, L., T. E. H. El-Gorashi, and J. M. H. Elmirghani (May 2015). “Energy Efficient Virtual Network Embedding for Cloud Networks”. In: *Journal of Lightwave Technology* 33.9, pp. 1828–1849. ISSN: 0733-8724. DOI: 10.1109/JLT.2014.2380777.
- Papagianni, C., A. Leivadreas, S. Papavassiliou, V. Maglaris, C. Cervelló-Pastor, and Á Monje (2013). “On the optimal allocation of virtual resources in cloud computing networks”. In: *IEEE Transactions on Computers* 62.6, pp. 1060–1071. ISSN: 0018-9340. DOI: 10.1109/TC.2013.31.
- Pointurier, Yvan (2016). *Project SENDATE-TANDEM – CELTIC-NEXT*. URL: <https://www.celticnext.eu/project-sendate-tandem/> (visited on 09/02/2019).
- Rabbani, M. G., R. P. Esteves, M. Podlesny, G. Simon, L. Z. Granville, and R. Boutaba (2013). “On Tackling Virtual Data Center Embedding Problem”. In: *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), pp. 177–184.

- Rahman, M. R. and R. Boutaba (2013). “SVNE: Survivable Virtual Network Embedding Algorithms for Network Virtualization”. In: *IEEE Transactions on Network and Service Management* 10.2, pp. 105–118. ISSN: 1932-4537. DOI: 10.1109/TNSM.2013.013013.110202.
- Ricci, Robert, Chris Alfeld, and Jay Lepreau (Apr. 1, 2003). “A solver for the network testbed mapping problem”. In: *ACM SIGCOMM Computer Communication Review* 33.2, p. 65. ISSN: 01464833. DOI: 10.1145/956981.956988. URL: <http://portal.acm.org/citation.cfm?doid=956981.956988> (visited on 07/15/2019).
- Riggio, R., F. De Pellegrini, E. Salvadori, M. Gerola, and R. Doriguzzi Corin (2013). “Progressive Virtual Topology Embedding in OpenFlow Networks”. In: *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), pp. 1122–1128.
- Rost, Matthias and Stefan Schmid (Jan. 9, 2018). “NP-Completeness and Inapproximability of the Virtual Network Embedding Problem and Its Variants”. In: *arXiv:1801.03162 [cs]*. arXiv: 1801.03162. URL: <http://arxiv.org/abs/1801.03162> (visited on 01/14/2019).
- Russell, Stuart J., Peter Norvig, and Ernest Davis (2010). *Artificial intelligence: a modern approach*. 3rd ed. Prentice Hall series in artificial intelligence. Upper Saddle River: Prentice Hall. 1132 pp. ISBN: 978-0-13-604259-4.
- Shuyuan Zhang, Abdulrahman Mahmoud, Sharad Malik, and Sanjai Narain (Oct. 2012). “Verification and synthesis of firewalls using SAT and QBF”. In: *2012 20th IEEE International Conference on Network Protocols (ICNP)*. 2012 20th IEEE International Conference on Network Protocols (ICNP). Austin, TX, USA: IEEE, pp. 1–6. ISBN: 978-1-4673-2447-2 978-1-4673-2445-8 978-1-4673-2446-5. DOI: 10.1109/ICNP.2012.6459944. URL: <http://ieeexplore.ieee.org/document/6459944/> (visited on 07/10/2019).
- Wang, Y., P. Chau, and F. Chen (2015). “A Framework for Security-Aware Virtual Network Embedding”. In: *2015 24th International Conference on Computer Communication and Networks (ICCCN)*. 2015 24th International Conference on Computer Communication and Networks (ICCCN), pp. 1–7. DOI: 10.1109/ICCCN.2015.7288361.
- Yu, Minlan, Yung Yi, Jennifer Rexford, and Mung Chiang (2008). “Rethinking virtual network embedding: substrate support for path splitting and migration”. In: *ACM SIGCOMM Computer Communication Review* 38.2, pp. 17–29.
- Zhang, Q., M. F. Zhani, M. Jabri, and R. Boutaba (2014). “Venice: Reliable Virtual Data Center Embedding in Clouds”. In: *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*. IEEE INFOCOM 2014 - IEEE Conference on Computer Communications, pp. 289–297. DOI: 10.1109/INFOCOM.2014.6847950.

## Bibliography

---

## Chapter 5 Appendices

### Demonstration A.1

Let  $f(n)$  be the number of solutions for a star of  $n$  nodes, with the substrate given in Figure 5.5. We show by induction that  $f(n) = 3 \times 5^n$ . We denote as  $N_i$  where  $i \in (1, \dots, n)$  the nodes in the graph. The node  $N_1$  is the center of the star. We denote as  $m_k^n$  a solution for a star of  $n$  nodes where  $k$  is an index identifying the solution.

**Basis.** Same as Demonstration 5.1.

**Induction step.** Let  $m_k^n$  be a solution for a star of  $n$  nodes. Let  $m_q^{n+1}$  be a solution for a star of  $n + 1$  nodes such that:

$$\begin{aligned} \forall i \in (1, \dots, n), & \quad m_q^{n+1}(N_i) = m_k^n(N_i) \\ \forall i \in (2, \dots, n), & \quad m_q^{n+1}((N_1, N_i)) = m_k^n((N_1, N_i)) \end{aligned}$$

Then we need to embed  $N_{n+1}$  and  $(N_1, N_{n+1})$ . Let  $d = m_q^{n+1}(N_1)$  be the domain hosting  $N_1$ .

If  $m_q^{n+1}(N_{n+1}) = d$ , then  $m_q^{n+1}((N_1, N_{n+1})) = (d, (d, d), d)$  by Definition 3.3. We then have described one (1) solution from  $m_k^n$ .

If  $m_q^{n+1}(N_{n+1}) \neq d$ , then we can choose among the two (2) other domains. Let  $d' = m_q^{n+1}(N_{n+1})$ ,  $d' \neq d$ . Let  $d''$  be the remaining domain,  $d'' \neq d'$ ,  $d'' \neq d$ . As the substrate topology is a ring, we have two (2) paths that join  $d$  to  $d'$ . Those paths are  $(d, (d, d'), d')$  and  $(d, (d, d''), d'', (d'', d'), d')$ . They can be chosen for the mapping of  $(N_1, N_{n+1})$ . We then have described four ( $4 = 2 \times 2$ ) other solutions from  $m_k^n$ .

Then  $f(n + 1) = (1 + 4) \times f(n) = 5 \times f(n)$ .  $\square$

### Demonstration A.2

Let  $f(n)$  be the number of solutions for a complete directed graph of  $n$  nodes, with the substrate given in Figure 5.5. We show that  $f(1) = 3$ ,  $f(2) = 27$ ,  $f(3) = 673$  and  $f(4) = 41985$ . We denote as  $N_i$  where  $i \in (1, \dots, n)$  the nodes in the graph. We denote as  $m_k^n$  a solution for a complete directed graph of  $n$  nodes where  $k$  is an index identifying the solution.

**Case  $n = 1$ .** Same as Demonstration 5.1.

**Case  $n = 2$ .** Let  $m_k^1$  be a solution for a complete directed graph of 1 node. Let  $m_q^2$  be a solution for a complete directed graph of 2 nodes such that:

$$m_q^2(N_1) = m_k^1(N_1)$$

Then we need to embed  $N_2$ ,  $(N_1, N_2)$  and  $(N_2, N_1)$ . Let  $d = m_q^2(N_1)$  be the domain hosting  $N_1$ .

If  $m_q^2(N_2) = d$ , then  $m_q^2((N_1, N_2)) = m_q^2((N_2, N_1)) = (d, (d, d), d)$  by Definition 3.3. We then have described one (1) solution from  $m_k^1$ .

If  $m_q^2(N_2) \neq d$ , then we can choose among the two (2) other domains. Let  $d' = m_q^2(N_2)$ ,  $d' \neq d$ . Let  $d''$  be the remaining domain,  $d'' \neq d'$ ,  $d'' \neq d$ . As the substrate topology is a ring, we have two (2) paths that join  $d$  to  $d'$ . Those paths are  $(d, (d, d'), d')$  and  $(d, (d, d''), d'', (d'', d'), d')$ . They can be chosen for the mapping of the two (2) edges  $(N_1, N_2)$  and  $(N_2, N_1)$ . We then have described eight ( $8 = 2 \times 2 \times 2$ ) other solutions from  $m_k^1$ .

Then  $f(2) = (1 + 8) \times f(1) = 9 \times 3 = 27$ .

**Case  $n = 3$ .** We have 3 nodes to embed in 3 domains. Some nodes can be hosted in the same domain. We enumerate the different cases as follows:

- If all nodes are hosted in distinct domains.
  - We have 1 way to map 3 nodes into 3 clusters.
  - The number of node mappings is an arrangement of 3 clusters in 3 domains:  $3 \times 2 \times 1$ .
  - The number of link mappings per node mapping is  $2^6$  as 6 directed edges are not mapped to any self-loops.
  - Result:  $1 \times (3 \times 2 \times 1) \times 2^6 = 384$  solutions.

- If two nodes are hosted in the same domain.
  - We have 3 way to map 3 nodes into 2 clusters.
  - The number of node mappings is an arrangement of 2 clusters in 3 domains:  $3 \times 2$ .
  - The number of link mappings per node mapping is  $2^4$  as 4 directed edges are not mapped to any self-loops.
  - Result:  $3 \times (3 \times 2) \times 2^4 = 288$  solutions.
- If all nodes are hosted in the same domain.
  - We have 1 way to map 3 nodes into 1 cluster.
  - The number of node mappings is an arrangement of 1 cluster in 3 domains: 3.
  - But neither *Domain1* nor *Domain3* can host the whole 3-complete directed graph, so the arrangement is 1.
  - The number of link mappings per node mapping is  $2^0$  as 0 directed edges are not mapped to any self-loops.
  - Result:  $1 \times (3 - 2) \times 2^0 = 1$  solution.

The number of link mappings per node mapping is  $2^k$  where  $k$  is the number of directed edges that are not mapped to any self-loops because by Definition 3.3 an edge mapped to a self-loop has one candidate and the substrate topology is a ring, so we have two candidate paths otherwise.

At the intra-domain level, contrary to chains in Demonstration 5.1, a 3-complete directed graph is not bipartite, so we cannot respect eligible paths rule (R4) with only 2 substrate hosts, which is the number of hosts in *Domain1* and *Domain3*. Then, some cases must be rejected.

Then  $f(3) = 384 + 288 + 1 = 673$ .

**Case  $n = 4$ .** We have 4 nodes to embed in 3 domains. Some nodes can be hosted in the same domain. We enumerate the different cases as follows:

- If all nodes are hosted in distinct domains.
  - Less domains than nodes: not applicable.
- If two nodes are hosted in the same domain and the others in distinct domains.



- We have 6 ways to map 4 nodes into 3 clusters with one cluster of two nodes.
- The number of node mappings is an arrangement of 3 clusters in 3 domains:  $3 \times 2 \times 1$ .
- The number of link mappings per node mapping is  $2^{10}$  as 10 directed edges are not mapped to any self-loops.
- Result:  $6 \times (3 \times 2 \times 1) \times 2^{10} = 36864$  solutions.
- If we group nodes into two clusters of two nodes each.
  - We have 3 way to map 4 nodes into 2 clusters of two nodes each.
  - The number of node mappings is an arrangement of 2 cluster in 3 domains:  $3 \times 2$ .
  - The number of link mappings per node mapping is  $2^8$  as 8 directed edges are not mapped to any self-loops.
  - Result:  $3 \times (3 \times 2) \times 2^8 = 4608$  solutions.
- If three nodes are hosted in the same domain and the other in distinct domain.
  - We have 4 way to map 4 nodes into 2 clusters with one cluster of three nodes.
  - The number of node mappings is an arrangement of 2 clusters in 3 domains:  $3 \times 2$ .
  - But neither *Domain1* nor *Domain3* can host a 3-complete directed graph, so the arrangement is  $2 \times 1$ .
  - The number of link mappings per node mapping is  $2^6$  as 6 directed edges are not mapped to any self-loops.
  - Result:  $4 \times (2 \times 1) \times 2^6 = 512$  solutions.
- If all nodes are hosted in the same domain.
  - We have 1 way to map 4 nodes into 1 cluster.
  - The number of node mappings is an arrangement of 1 cluster in 3 domains: 3.
  - But neither *Domain1* nor *Domain3* can host the whole 4-complete directed graph, so the arrangement is 1.

- The number of link mappings per node mapping is  $2^0$  as 0 directed edges are not mapped to any self-loops.
- Result:  $1 \times (3 - 2) \times 2^0 = 1$  solution.

At the intra-domain level, contrary to chains in Demonstration 5.1, a 3-complete directed graph is not bipartite, so we cannot respect R4 with only 2 substrate hosts, which is the number of hosts in *Domain1* and *Domain3*. Then, some cases must be rejected.

Then  $f(4) = 36864 + 4608 + 512 + 1 = 41985$ . □



---

## Acronyms

**BER** bit error rate. 40, 43, 137

**CPE** Common Platform Enumeration. 10, 11

**CSP** Constraint Satisfaction Problem. 25–27, 31, 33, 34, 51, 78

**CVE** Common Vulnerabilities and Exposures. 10

**CVSS** Common Vulnerability Scoring System. 10

**DPLL** Davis–Putnam–Logemann–Loveland. 32, 33

**DPLL(T)** DPLL Modulo Theories. 33

**ILP** Integer Linear Programming. i, 6–8, 20, 25, 27–30, 33, 48, 51, 52

**InP** Infrastructure Provider. i, 1, 2, 8–13, 16, 19–25, 28, 34, 35, 37–39, 42, 43, 46, 48, 51, 54, 73, 77, 78, 80, 84, 85, 91, 95, 99, 102, 104, 111, 112, 114, 117, 118, 144

**ISP** Internet Service Provider. 8, 20, 22

**NFV** Network Function Virtualization. i, 6, 7, 17–19, 34, 114

**R1** node mapping rule. 49–51, 53, 65, 83

**R2** link mapping rule. 49–51, 53, 65

**R3** eligible substrate hosts rule. 48–51, 53, 83

**R4** eligible paths rule. 49–51, 53, 55, 82, 83, 131, 133

**R5** node-link mapping coordination rule. 49–51, 54

- R6** capacity rule. 46, 49–51, 54, 55, 82, 83
- SAT** Boolean Satisfiability. i, 27, 31–33
- SDN** Software-Defined Network. i, 7, 9, 17, 18, 34, 50
- SFC** Service Function Chain. 19, 20, 114
- SFCE** Service Function Chain Embedding. 6, 7, 19
- SMT** Satisfiability Modulo Theories. i, ii, 27, 30, 31, 33, 34, 38, 48, 51, 57–61, 65, 78, 83–85, 89, 91, 93, 96, 114, 118
- SN** substrate network. 12, 40
- SVNE** Survivable Virtual Network Embedding. 6, 8, 15
- VDCE** Virtual Data Center Embedding. 6–8, 12
- VM** virtual machine. 12, 50
- VN** virtual network. 7–9
- VNE** Virtual Network Embedding. i, 2–9, 11–25, 27–31, 33, 34, 37, 39, 48, 49, 51, 52, 75, 77, 78, 84, 91, 94, 107, 110, 117, 147, 149
- VNER** Virtual Network Embedding Reconfiguration. 6, 8, 14
- VNF** Virtual Network Function. 19, 20
- VNP** Virtual Network Provider. 8, 24, 77, 84, 91, 117

---

## Table of all Notations

| Symbol                          | Meaning   | Pages  |
|---------------------------------|---|--|
| $\mathbf{a}, \dots, \mathbf{z}$ | Font and format used to reference virtual node examples.  | 29, 30, 40, 41, 50, 52, 56, 57, 70, 73–75, 97, 98, 108, 114, 147 |
| $\mathbf{A}, \dots, \mathbf{Z}$ | Font and format used to reference physical node examples.   | 29, 30, 41, 50, 52, 53, 73, 74, 104                              |
| $e = (e_h, e_t)$                | A directed edge $e$ , where $e_h$ is the head node, and $e_t$ is the tail node.   | 30, 47, 53–55, 66, 79–82, 88, 104, 105, 108, 113, 129, 130, 141  |
| $\cdot a$                       | The aggregation function of a link attribute $a$ . See Definition 3.2.  | 47, 48, 81, 113  |
| $!bw$                           | Required bandwidth for a virtual link. Modelled as a binding link attribute. It serves also for the available bandwidth of a physical link.                 | 42, 108  |
| $!ber$                          | Maximal authorized bit error rate of a virtual link. Modelled as non-biding link attribute. It serves also for the bit error rate (BER) of a physical link. | 42–44, 104   |

Table of all Notations

| Symbol                | Meaning  | Pages   |
|-----------------------|--|---|
| <b>!col-resources</b> | Virtual resources (of the same request) that must be collocated with the requesting virtual resource. Modelled as a collocation attribute.   | 68  |
| <b>!exc-requests</b>  | Requests (of the same tenant) whose virtual resources cannot be collocated with the requesting virtual resource. Modelled as an exclusion attribute.   | 41, 55, 66–68, 82–84, 109, 112, 113                         |
| <b>!exc-resources</b> | Virtual resources (nodes or links) of the same request that cannot be collocated with the requesting virtual resource. Modelled as an exclusion attribute.   | 41, 55, 66, 67, 82–84, 97                                   |
| <b>!exc-tenants</b>   | Tenants whose virtual resources of any of their requests cannot be collocated with the requesting virtual resource. Modelled as an exclusion attribute.  | 41, 42, 66, 67  |
| <b>!loc</b>           | Authorized locations for a virtual node. Modelled as a non-biding node attribute, derived from a powerset. It serves also for the physical location of a physical node.                                  | 40, 42, 56, 57, 71, 74, 75, 97, 99, 102, 108, 109, 112, 147 |
| <b>!mem</b>           | Required memory for a virtual node. Modelled as a binding node attribute. It serves also for the available memory of a physical node.  | 42–45, 97, 108  |
| <b>!ven</b>           | Authorized vendors along a virtual <b>link</b> . Modelled as non-biding link attribute, with aggregation function and derived from a powerset. It serves also for the vendor of a physical <b>node</b> . | 40, 42, 47, 48, 80, 99, 102, 108                            |

Table of all Notations

| Symbol               | Meaning  | Pages  |
|----------------------|--|--|
| $a$                  | An attribute. For easier referencing, all examples of attributes in this document are using the following format: !<name>.   | 39–48,<br>54–57,<br>62–68,<br>70–75,<br>80–84, 97,<br>99–102,<br>104,<br>107–109,<br>111–114,<br>118,<br>137–140,<br>142, 143,<br>145, 147,<br>149 |
| $Active_L$           | The set of all active link attributes.   | 64, 70, 72,<br>73  |
| $Active_N$           | The set of all active node attributes.   | 62, 63, 70,<br>72, 73  |
| $Along_L(j)$         | The set of links along path $j$ in $P^S$ .   | 53, 54, 66,<br>82, 88  |
| $Along_N(j)$         | The set of nodes along path $j$ in $P^S$ .   | 53, 144  |
| $Binding_L$          | The set of binding link attributes.  | 47, 54, 66   |
| $Binding_N$          | The set of binding node attributes.  | 47, 54, 66   |
| $Collocate_a(i, i')$ | A Boolean function which is true when virtual resources $i$ and $i'$ in $L^R$ or $N^R$ of request $R$ must be collocated in substrate resources $j$ and $j'$ with the same offer in $a$ , and false otherwise. | 70–73  |
| $Collocate_L(i, i')$ | A Boolean function which is true when virtual links $i$ and $i'$ in $L^R$ of request $R$ must be collocated in the same substrate link, and false otherwise.   | 64, 68–70  |



Table of all Notations

| Symbol                                | Meaning  | Pages  |
|---------------------------------------|--|--|
| $Collocate_N(i, i')$                  | A Boolean function which is true when virtual nodes $i$ and $i'$ in $N^R$ of request $R$ must be collocated in the same substrate node, and false otherwise.   | 64, 68–70  |
| $d_i[a]$                              | Demand in attribute $a$ of virtual resource $i \in N^R \cup L^R$ . It is a value in $V(a)$ .   | 39, 41–44, 46, 54–57, 62, 65, 68, 81–84, 108, 140        |
| $D_a(i)$                              | Function equivalent of $d_i[a]$ .  | 62–67  |
| $D$                                   | Reference to a domain $D$ in the intra-domain level.   | 81, 85, 87, 89–91, 141                                   |
| $Exclude_a(i, i')$                    | A Boolean function which is true when virtual resources $i$ and $i'$ in $L^R$ or $N^R$ of request $R$ cannot be collocated in substrate resources $j$ and $j'$ with the same offer in $a$ , and false otherwise. | 70–72  |
| $Exclude_L(i, i')$                    | A Boolean function which is true when virtual links $i$ and $i'$ in $L^R$ of request $R$ cannot be collocated in the same substrate link, and false otherwise.   | 64, 66–68  |
| $Exclude_N(i, i')$                    | A Boolean function which is true when virtual nodes $i$ and $i'$ in $N^R$ of request $R$ cannot be collocated in the same substrate node, and false otherwise.   | 64, 66–68  |
| $\hat{H}^S = H^S \setminus \{Out^S\}$ | The set of inner hosts of substrate $S$ .  | 82, 144  |
| $H^S$                                 | The set of hosts in substrate $S$ . It is a subset of $N^S$ .  | 52–55, 62, 63, 66, 67, 69, 72, 73, 87, 88, 140, 141, 143 |

Table of all Notations

| Symbol   | Meaning  | Pages  |
|--|--|--|
| $i$  | A virtual resource in $N^R$ or $L^R$ , depending on the context.   | 29, 30, 39–43, 46, 47, 52–55, 62–70, 72, 73, 81–85, 87–91, 112, 113, 139, 140, 142–145 |
| $Ideal_L(j)$                                   | Tell if a substrate link $j$ is an ideal link. See Definition 3.1.   | 64, 67–69, 72, 73  |
| $Ideal_N(j)$                                   | Tell if a substrate node $j$ is an ideal node. See Definition 3.1.   | 64, 67, 69, 72, 73   |
| $InterdomainLinks_D^U$                         | Inter-domain links connected to domain $D$ .   | 90   |
| $j$  | A physical resource in $H^S$ or $L^S$ , depending on the context.  | 29, 30, 39–44, 46–48, 52–55, 62, 64–69, 72, 73, 82–85, 87–91, 118, 139–145             |
| $\hat{L}^S = L^S \setminus \{(Out^S, Out^S)\}$ | The set of real links of $S$ . The real links encompass the links inside the substrate as well as the links connecting it to neighbor substrates, when applicable. | 83   |
| $L^R$  | The set of links in request $R$ .  | 28–30, 52–55, 62, 64, 66–70, 72, 73, 81–84, 87, 88, 90, 139–144                        |

Table of all Notations

| Symbol        | Meaning  | Pages  |
|---------------|--|--|
| $L^S$         | The set of links in substrate $S$ .  | 28–30, 47,<br>52, 54, 55,<br>62, 64,<br>66–69, 72,<br>73, 81, 82,<br>84, 88, 113,<br>118,<br>141–143,<br>145 |
| $l_{ij}$      | A Boolean variable which is true ( $\top$ ) when allocating a substrate link $j$ in $L^S$ for a virtual link $i$ in $L^R$ , and false ( $\perp$ ) otherwise.                               | 29, 30, 52,<br>54, 55, 65,<br>82–85, 88,<br>90, 142  |
| $L_{ij}$      | The set of all $l_{ij}$ .  | 87   |
| $M_a$         | The absorbing value of attribute $a$ .   | 42, 45–48,<br>74, 81, 109,<br>118  |
| $m_a$         | The default value of attribute $a$ .   | 42, 45–48,<br>74, 104, 109,<br>118   |
| $m(x)$        | A model, which is a function assigning an unknown $x$ to a value.  | 85, 87–91  |
| $memorized_N$ | A function that memorize for each virtual node $i$ from each already embedded requests the substrate host in which it is embedded.   | 54, 66   |
| $memorized_P$ | A function that memorize for each virtual link $i$ from each already embedded requests the substrate path in which it is embedded.   | 54, 66   |
| $\mu_L$       | A function that assigns each virtual link $i$ in $L^R$ from any request $R$ in <i>Requests</i> to a set of $j$ in $L^S$ in a certain substrate $S$ .                                       | 62, 65, 66,<br>68, 69, 72,<br>73   |
| $\mu_N$       | A function that assigns each virtual node $i$ in $N^R$ from any request $R$ in <i>Requests</i> to a substrate host $j$ in $N^S$ in a certain substrate $S$ . <add note about memorization> | 54, 56, 57,<br>62, 64–67,<br>69, 72, 73  |

Table of all Notations

| Symbol                        | Meaning  | Pages  |
|-------------------------------|--|--|
| $\mu_P$                       | A function that assigns each virtual link $i$ in $L^R$ from any request $R$ in <i>Requests</i> to a substrate path $j$ in $P^S$ in a certain substrate $S$ . | 54, 62, 65–67  |
| $N^R$                         | The set of nodes in request $R$ .  | 28–30, 52–55, 62, 64, 66–70, 72, 73, 82, 87, 88, 90, 139–143 |
| $N^S$                         | The set of nodes in substrate $S$ .  | 28–30, 52, 140, 142, 143, 145                                |
| $n_{ij}$                      | A Boolean variable which is true ( $\top$ ) when allocating a substrate host $j$ in $H^S$ for a virtual node $i$ in $N^R$ , and false ( $\perp$ ) otherwise. | 29, 30, 52–55, 64, 65, 82, 85, 87, 88, 90, 91, 143           |
| $N_{ij}$                      | The set of all $n_{ij}$ .  | 87   |
| <i>NewRequest</i>             | The new-coming request, which we try to embed.   | 52, 54, 66, 85, 90, 91, 144                                  |
| <i>NonBinding<sub>L</sub></i> | The set of non-binding link attributes.  | 47, 54, 66   |
| <i>NonBinding<sub>N</sub></i> | The set of non-binding node attributes.  | 47, 54, 66   |
| $o_j[a]$                      | Offer in attribute $a$ of substrate resource $j \in N^S \cup L^S$ . It is a value in $V(a)$ .  | 39, 40, 43, 44, 46–48, 54, 56, 57, 62, 65, 81, 113, 143      |
| $O_a(j)$                      | Function equivalent of $o_j[a]$ .  | 62–66, 72, 73, 75, 112, 113                                  |
| $Out^S \in H^S$               | The node representing the outside of substrate $S$ . It is also a host in $H^S$ .  | 74, 79–82, 90, 140, 141, 144                                 |

Table of all Notations

| Symbol                              | Meaning  | Pages  |
|-------------------------------------|--|--|
| $P^S$                               | The set of loop-free host-to-host paths (cf. Definition 3.3) in the substrate $S$ .  | 53–55, 62, 66, 80–82, 87, 88, 90, 118, 143, 144                          |
| $\hat{P}^S$                         | The set of loop-free paths that are going out through the outside node $Out^S$ in substrate $S$ . It is defined as:<br>$\{j \in P^S \mid Tips_j \in (\hat{H}^S)^2, Out^S \in Along_N(j)\}$ | 80–82, 87  |
| $p_{ij}$                            | A Boolean variable which is true ( $\top$ ) when allocating a substrate path $j$ in $P^S$ for a virtual link $i$ in $L^R$ , and false ( $\perp$ ) otherwise.                               | 53, 54, 65, 82, 85, 87, 88, 91, 144                                      |
| $P_{ij}$                            | The set of all $p_{ij}$ .  | 87   |
| $R$                                 | A request in $Requests$ , as a <b>directed</b> graph.  | 28–30, 41, 42, 52–55, 62, 64, 66–70, 72, 73, 81–84, 87, 88, 113, 139–144 |
| $Requests \setminus \{NewRequest\}$ | The already embedded requests.   | 54, 66   |
| $Requests_\tau$                     | The requests from tenant $\tau$ .  | 42, 55, 67, 82–84  |
| $Requests$                          | The request set. We embed one request at a time.   | 42, 52–55, 62, 64, 66–70, 72, 73, 82–84, 109, 142–144                    |
| $Resources^P$                       | The set of all physical resources which are provided by all the Infrastructure Providers (InPs).   | 41   |
| $Resources^R$                       | The set of all virtual resources in the request $R$ .  | 41, 42, 113  |

Table of all Notations

| Symbol                    | Meaning  | Pages  |
|---------------------------|--|--|
| $Resources^V$             | The set of all virtual resources which are requested by all the tenants taken together.  | 41, 43   |
| $S$                       | The substrate, as a <b>directed</b> graph.   | 28–30, 47,<br>52–55,<br>62–64,<br>66–69, 72,<br>73, 79–84,<br>88, 113, 118,<br>139–145 |
| $SelfLoops^S \subset L^S$ | The set of self-loops (edges such that their head is their tail) in the substrate $S$ .  | 84   |
| $\tau$                    | A tenant.  | 42, 55, 67,<br>82–85, 91,<br>144   |
| $Tips_j = (j_h, j_t)$     | The nodes at the tips of path $j$ in $P^S$ . The node at the beginning of the path is denoted $j_h$ (head), and the node at end of the path is denoted $j_t$ (tail). Both nodes are in $N^S$ . | 53, 54, 66,<br>82, 88, 144   |
| $U$                       | Reference to the inter-domain level.   | 85, 87,<br>89–91, 141  |
| $V(a)$                    | The set of all values that can be taken by attribute $a$ .   | 39, 40, 42,<br>45–47,<br>62–64, 74,<br>99, 100, 109,<br>118, 140,<br>143, 149          |
| $x_{ij}$                  | A Boolean variable which is true ( $\top$ ) when allocating a substrate resource $j$ for a virtual resource $i$ , and false ( $\perp$ ) otherwise.   | 39, 41–44,<br>46, 53–55,<br>85   |

## Table of all Notations

---

---

## List of Figures

|     |   |     |
|-----|---|-----|
| 2.1 | Flow Conservation Rule Illustration . . . . .   | 29  |
| 3.1 | Virtual Network Embedding (VNE) illustration . . . . .  | 49  |
| 3.2 | Flow Conservation Rule Limit . . . . .  | 51  |
| 3.3 | Attribute-specific Exclusion Example for the !loc Attribute . .   | 71  |
| 3.4 | Attribute-specific Exclusion Example for the !loc Attribute, under Limited Information Disclosure . . . . . | 74  |
| 3.5 | Undesired Result from Attribute-specific Exclusion Model, under Limited Information Disclosure . . . . .    | 75  |
| 4.1 | Example of Intra-Domain Graph . . . . .   | 79  |
| 4.2 | Embeddings Generation Sequence Diagram . . . . .  | 86  |
| 4.3 | Embedding Selection Sequence Diagram . . . . .  | 90  |
| 5.1 | Use case request . . . . .  | 97  |
| 5.2 | Declaring resource <b>d</b> on prototype . . . . .  | 98  |
| 5.3 | Description of resource <b>d</b> on prototype . . . . .   | 98  |
| 5.4 | Proposed embeddings . . . . .   | 98  |
| 5.5 | The use case substrate, with its sub-domains . . . . .  | 103 |
| 5.6 | Computation Time Trend for Chain, Full-mesh and Star Topology   | 106 |
| 5.7 | Computation Time Trend with 0-Solution Request . . . . .  | 107 |
| 5.8 | Tested Requests in Figure 5.7 . . . . .   | 108 |
| 5.9 | Example of Embedding from Alaluna et al. . . . .  | 110 |



## List of Figures

---

---

## List of Tables

|     |   |     |
|-----|---|-----|
| 2.1 | Classification of VNE Research Papers . . . . .   | 7   |
| 2.2 | Classification of Objectives of VNE Research Papers . . . . .                                       | 9   |
| 2.3 | Classification of Substrates of VNE Research Papers . . . . .                                       | 12  |
| 2.4 | Classification of Request Handling Methods of VNE Research Papers . . . . .                         | 13  |
| 2.5 | Truth Table of $C_1$ . . . . .  | 27  |
| 2.6 | Flow Conservation Analysis for Figure 2.1 . . . . .   | 29  |
| 3.1 | Building Blocks of Some Tolerant Attributes . . . . .   | 42  |
| 3.2 | Flow Conservation Analysis for Figure 3.2 . . . . .   | 52  |
| 3.3 | Building Blocks of <b>!domain</b> Attribute . . . . .   | 74  |
| 5.1 | Overview of Python modules in use . . . . .   | 96  |
| 5.2 | Compute before Store Strategy for Attribute $a$ Such That $V(a) = \text{Powerset}(Set_a)$ . . . . . | 100 |
| 5.3 | Compute on Demand Strategy for Attribute $a$ Such That $V(a) = \text{Powerset}(Set_a)$ . . . . .    | 100 |
| 5.4 | Data Structure for Mutable Attribute At Time $t_i$ With $Set_a(t_i) = \{X, Y, Z\}$ . . . . .        | 102 |
| 5.5 | Building Blocks of State of the Art Security-oriented Attributes                                    | 109 |

**Titre :** Incorporation de réseaux virtuels dans une infrastructure multidomaine sous exigences de sécurité appliquée au slicing réseau en 5G

**Mots clés :** réseau virtuel; multidomaine; 5G; slice; sécurité; SMT

**Résumé :**

La 5G apporte un nouveau concept, le network slicing (découpage du réseau en tranches). Cette technologie permet de généraliser le modèle économique des MVNO à des entreprises qui ont besoin d'opérer un réseau, sans que cela ne soit leur cœur de métier. Chaque tranche (slice) est un réseau virtuel de bout en bout, dédié et personnalisé, au-dessus d'une infrastructure partagée ; cette infrastructure elle-même être fournie par l'interconnexion de fournisseurs d'infrastructure: nous parlons dans ce cas d'infrastructure multi-domaine.

L'objectif de cette thèse est d'étudier l'allocation de ces tranches dans une telle infrastructure multi-domaine. Le problème est connu comme l'incorporation de réseau virtuel (Virtual Network Embedding (VNE)). Il s'agit d'un problème NP-difficile. Pratiquement, le problème VNE recherche à quelles ressources physiques associer un ensemble d'éléments virtuels. Les ressources physiques décrivent ce qu'elles peuvent offrir. Les éléments virtuels décrivent ce qu'ils exigent. La mise en relation de ces offres et de ces demandes est la clé pour résoudre le problème VNE.

En l'espèce, nous nous sommes intéressés à la modélisation et à la mise en place d'exigences de sécurité. En effet, nous nous attendons à ce que les acteurs à l'initiative des tranches appartiennent à des sphères éloignées des télécommunications. Or de la même façon qu'ils connaissent peu ce domaine, nous pouvons nous attendre à ce que leurs besoins, notamment de sécurité, s'expriment d'une façon sans précédent dans le contexte des tranches.

Cette thèse présente un algorithme capable de traiter des exigences variées selon un modèle extensible fondé sur un solveur de satisfiabilité appliqué à des théories décidables (Satisfiability Modulo Theories (SMT)). Comparée à la programmation linéaire (Integer Linear Programming (ILP)), plus commune dans le domaine des VNE, cette formulation permet d'exprimer les contraintes à satisfaire de façon plus transparente, et d'auditer l'ensemble des contraintes. De plus, ayant conscience que les fournisseurs d'infrastructure sont réticents à exposer les informations relatives à leurs ressources physiques, nous proposons une résolution limitant cette exposition. Ce système a été implémenté et testé avec succès au cours du doctorat.

**Title :** Multidomain Virtual Network Embedding under Security-oriented Requirements applied to 5G Network Slices

**Keywords :** virtual network; multidomain; 5G; slice; security; SMT

**Abstract :**

5G brings a new concept called network slicing. This technology makes it possible to generalize the business model of MVNOs to companies in need to operate a network, without it being their core business. Each slice is an end-to-end, dedicated and customized virtual network, over a shared infrastructure; this infrastructure itself is provided by the interconnection of infrastructure providers: we refer to this case as a multi-domain infrastructure.

The objective of this thesis is to study the allocation of these slices in such a multi-domain infrastructure. The problem is known as Virtual Network Embedding (VNE). It is an NP-hard problem. Practically, the VNE problem looks for which physical resources to associate a set of virtual elements. Physical resources describe what they can offer. Virtual elements describe what they require. Linking these offers and requests is the key to solve the VNE problem.

In this thesis, we focused on modeling and implementing security requirements. Indeed, we expect that the initiators of the slices belong to areas distant from telecommunications. In the same way that they know little about this field, we can expect that their needs, especially in security, are novel in the slice context.

This thesis presents an algorithm able to handling various requirements, according to an extensible model based on a Satisfiability Modulo Theories (SMT) solver. Compared to Integer Linear Programming (ILP), more common in the VNE field, this formulation allows to express the satisfaction constraints in a more transparent way, and allows to audit all the constraints.

Moreover, being aware that infrastructure providers are reluctant to disclose information about their physical resources, we propose a resolution limiting this disclosure. This system has been successfully implemented and tested during the Ph.D.