# Solving security constraints for 5G slice embedding: a proof-of-concept

François Boutigny, Stéphane Betgé-Brezetz, Gregory Blanc, Antoine Lavignotte, Hervé Debar, Houda Jmila

# Solving Security Constraints for 5G Slice Embedding: A Proof of Concept

François Boutigny[2,*], Stéphane Betgé-Brezetz[1], Gregory Blanc[2], Antoine Lavignotte[2], Hervé Debar[2], Houda Jmila[2]

**Abstract**

Network slicing is a prominent feature of 5G, which allow tenants to rent network and computing virtual resources from one or more Infrastructure Providers (InPs). Those resources are allocated according to tenants requirements, not only in terms of QoS but also in terms of security. In this paper, we build on our previous work to propose and evaluate a security-aware slice embedding implementation which enables tenants to declare security-oriented requirements, while limiting InP network information disclosure. To do so we improve our requirement model so that it becomes compatible with an Satisfiability Modulo Theories (SMT) formulation. Our implementation distinguishes two sub-problems, one for the intra-domain level (inside each InP) and one for the inter-domain level (in between the InPs). We leverage those sub-problems in a multi-level resolution algorithm to generate all multi-domain slice embeddings.

*Keywords:* virtual network embedding, network slicing, 5G security, satisfiability modulo theories, network virtualization

## 1. Introduction

Network operators 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.

The Mobile Virtual Network Operator (MVNO) business model fortunately shows that this is not mandatory. MVNOs already provide Internet access contracts to subscribers, like other mobile network operators, while owning no or a few equipment, contrary to mobile network operators. MVNOs rent equipement instead, after negotiating specific agreements on specific resources with the mobile network operators. 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 fact, they get a dedicated, customized, virtual, end-to-end network on top of a shared infrastructure, which is called "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).

In this paper, 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 the slice deployment. In this paper, we present how we solve this problem. We borrow from the literature the concept of Virtual Network Embedding (VNE), which refers to the allocation of resources from the physical network to fulfill requirements of a virtual network request (Fischer et al., a). However, it was shown to be an NP-hard problem (Fischer et al., a). 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 paper successfully covers both.

*Corresponding author
*Email addresses:* francois_boutigny@telecom-sudparis.org (François Boutigny), stephane.betge-brezetz@nokia-bell-labs.com (Stéphane Betgé Brezetz), gregory.blanc@telecom-sudparis.eu (Gregory Blanc), antoine.lavignotte@telecom-sudparis.eu (Antoine Lavignotte), herve.debar@telecom-sudparis.eu (Hervé Debar), houda.jmila@telecom-sudparis.eu (Houda Jmila)
[1]Nokia Bell Labs France
[2]SAMOVAR, Télécom SudParis, Institut Mines Télécom, CNRS, Institut Polytechnique de Paris

Throughout this paper, 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.

For the **multi-InP aspect**, Mano et al.; Dietrich et al. have identified one key property: InPs are reluctant to share information about their topologies to others. This assumption still holds for slices. In addition, they assume that InPs are interconnected in a full-mesh way. This assumption does not hold for slices: some InPs may be network operators, and others may be data-centers, or tenant-owned networks. For this reason, we design our solution so that it is able to use an InP to interconnect slice subsets located in two distant InPs.

As for **security requirements**, state-of-the-art works from Bays et al.; Liu et al.; Alaluna et al. mainly use "security levels". The advantage of this approach is that it directly fits into Integer Linear Programming (ILP) models (as levels can be mapped to integers). While ILP (or its variants) is the main tool for describing VNE as an optimization problem so far, yet, from a security viewpoint, we cannot define such an absolute scale: security levels only make sense under a certain security policy, and we cannot expect that our diverse tenants and InPs will agree on a common security level reference.

Instead, our solution can express exclusion constraints, which makes it possible to avoid untrusted resources based on their location, vendors, certifications, or installed software programs, for instance. The solution is extensible to other requirements. In addition, we are aware that not all requirements may be appropriate for a given tenant, and that InPs may not support all of them either. For this reason, our solution tolerates incomplete requirement and resource descriptions. A basic rule is as follows: if an InP does not support a certain requirement, then it is eligible only if the tenant does not require it either. This rule is enforced through the use of a default value.

Besides, we support an exclusion requirement, which enable tenants to avoid sharing physical resources with other tenants. As such, they can still get dedicated resources if they mistrust virtualization technologies and their isolation.

Our previous work (Boutigny et al.) had the following contributions.

*i)* We proposed a naive slice generation algorithm, encompassing the following contributions.

*ii)* We proposed a heuristic for multi-domain embedding. This heuristic enables domains to embed more general slice subsets than in the state of the art. If we consider that the slices are requested by tenants in the form of graphs, state-of-the-art papers limited those slice subsets to be connected subgraphs, while we considered any subgraph and any subset of edges.

*iii)* We enabled set-like requirements (useful in security for blacklisting).

*iv)* We enabled the extension of our work to other requirements by identifying the necessary building blocks.

In this paper, we address different challenges raised by the implementation of our work (Boutigny et al.).

*i)* We improve our **slice generation algorithm** by leveraging Satisfiability Modulo Theories (SMT).

*ii)* We revise our **heuristic**, to give domains more ways to assign resources to our slice subsets.

*iii)* We revise our **requirement model** with a more algebraic-oriented description.

*iv)* We solve a **requirement consistency problem** which appears when enabling tenants to process slice request declaration and slice embedding at distinct times.

This paper is organized as follows. Section 2 presents the state of the art in VNE with multi-InP support and security requirement support. Section 3 describes VNE for a single domain case, as a set of rules. Section 4 presents how we model requirements. Section 5 formulates the single domain slice embedding problem as detailed in Section 3 by using our requirement model as detailed in Section 4. Then, we build upon Section 5 to formulate our multi-level slice embedding algorithm. Section 6 adapts the single domain formulation to the intra-domain level. Section 7 adapts the single domain formulation to the inter-domain level. Section 8 presents our actual algorithm. Section 9 presents our implementation as well as a use case infrastructure. Section 10 shows our experiments and results on the use case infrastructure. We conclude this work in Section 11.

## 2. State of the Art

VNE is a long-studied problem. As explained by Fischer et al. (a), even if we are told where to map virtual nodes, finding the right link allocations corresponds to the unsplittable flow problem, which is NP-hard (Kolliopoulos and Stein). 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 2.1 enable the tenants to require security within their virtual network requests. Works in Subsection 2.2 enable the InP to protect themselves.

### *2.1. Tenant-oriented Security*

From the state-of-the-art, 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. Overall, this primitive is used to define security levels based on an absolute scale.

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. 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. focus on privacy issues for a tenant in an Internet Service Provider (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. 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. 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. 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. (c) propose to implement security requirements on the ALEVIN simulator (see Fischer et al. (b)). 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 paper, and like Fischer et al. (c), 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. 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.

Our work is distinct to the aforementioned works as follows.

Liu et al. 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. 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. 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. 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., 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. 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. 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., 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. (c), 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.

### *2.2. InP-oriented Security*

Liu et al. 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. 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.; Dietrich et al., 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.. 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 Virtual Network Provider (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.; Houidi et al. 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. 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.; Di et al..

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., the inter-domain level algorithm knows the full pricing function of each InP for each given virtual resource. For Dietrich et al., 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.; Di et al., the inter-domain level algorithm knowns the InP's border nodes and the bandwidth of the inter-domain links. Besides, for Di et al., 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.; Mano et al., as price information is also sensitive.

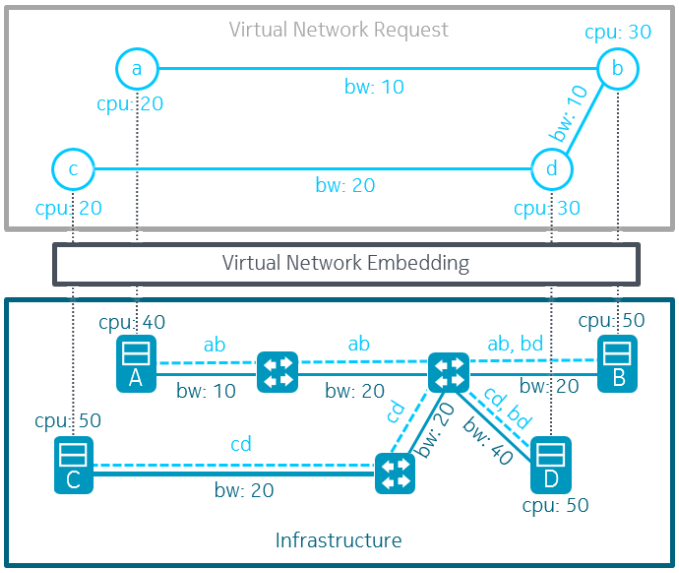



Figure 1: VNE illustration

Our work is distinct to the aforementioned works as follows.

For Dietrich et al., 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., the partial embeddings are limited to connected subgraphs 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. 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.; Chowdhury et al.. Last but not least, they limit InP embedding propositions to be connected subparts, whereas we support more general pieces.

## 3. VNE Description for Single Domain

Let us describe the VNE problem for a single domain slice embedding. The situation itself is represented in Figure 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 1, such weights are labeled as *cpu* (node weight) or *bw* (edge weight). 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, 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 eligible substrate hosts rule (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 1-to-1 mapping, as illustrated by the association between the nodes $a$ and $A$ in Figure 1.

Other node mappings are possible. As mentioned in Section 2, Bays et al. consider virtual nodes to be virtual routers, that must be instantiated in physical routers. However, the development of network hypervisors (Blenk et al.) in the Software-Defined Network (SDN) context enables a single virtual SDN switch may be mapped to multiple physical SDN switches. As such, it is a 1-to-n mapping, which we decided to not address in this paper. For this reason, we follow R3.

Link mapping is another 1-to-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 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 paper, we consider that this is still a tedious task, and we will avoid it at the physical level. This is why, in 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). It is typically this rule that becomes difficult to formulate when enabling a 1-to-n node 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.

The concept of demand (and offer) from capacity rule (R6) will be described in the next section. In Figure 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.

## 4. Attribute-based Requirement Model

Requirements are expressed in natural language. They can be technical, but also legal or security-based. 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 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 enables resources to be offered by providers to satisfy a demand from a tenant We do so by exhibiting the mathematical constraints that each attribute should follow. Such constraints indicate that whether the mapping of the virtual resource to the physical resource fulfills the tenant's requirements, or not. For our location attribute example, it is clear that offer and demand are related through the subset relation: the physical resource location must be in the authorized locations of the virtual resource.

Before introducing all the mathematical constraints, we now give a general overview of the notations used in this section. We denote as $i$ a resource as required by a tenant, and as $j$ a resource as 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 $a$. We denote as $x_{ij} \in \{\bot, \top\}$ an unknown which equals $\top$ (meaning "true") if resource $i$ is allocated to resource $j$, and $\bot$ (meaning "false") otherwise. The notations used in this paper are detailed in Table A.3, given in appendix.

In this paper, we simplify the attribute classification presented in (Boutigny et al.), to only distinguish what we call tolerant and exclusion attributes. This distinction is a compromise between providing a unified attribute model and performances.

### *4.1. Exclusion Attributes: Semantics and Typology*

*Exclusion* attributes enable tenants to forbid their virtual resources to share physical resources with some other virtual resources, be them owned by the same tenant or by other tenants. These attributes are special in the sense that their values directly refer to the resources of other tenants. Besides, exclusion by itself is a symmetric relation, as shown in Demonstration 1.

**Demonstration 1** (Exclusion symmetry)**.** *Let $X^V$ be the set of virtual resources, and $X^P$ the set of physical resources. Let Exclude be the exclusion predicate, as a binary relation. It can be expressed as:*

$$\forall (i,k) \in X^V \times X^V, i \neq k,$$
$$Exclude(i,k) = \left(\forall j \in X^P, x_{ij} \Rightarrow \neg x_{kj}\right)$$

*By contraposition, $x_{ij} \Rightarrow \neg x_{kj}$ is equivalent to $x_{kj} \Rightarrow \neg x_{ij}$, hence $Exclude(i,k) = Exclude(k,i)$.* □

We exhibit three exclusion attributes, namely $xres$ (resource exclusion), $xreq$ (request exclusion), and $xten$ (tenant exclusion). They follow respectively Equation (1), Equation (2) and Equation (3). 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 (1) treats more fine-grained values than Equation (2), which treats more fine-grained values than Equation (3).

$$x_{ij} \Rightarrow \bigwedge_{k \in d_i^{xres}} \neg x_{kj} \tag{1}$$

Equation (1) means that if $i$ is allocated to $j$, then no excluded resource $k$ can be allocated to $j$, where $d_i^{xres}$ is the set of all resources $k$ which are excluded by the requester.

$$x_{ij} \Rightarrow \bigwedge_{\substack{R \in d_i^{xreq} \\ k \in X^R}} \neg x_{kj} \tag{2}$$

Equation (2) means that if $i$ is allocated to $j$, then no virtual resource $k$ from any excluded request $R$ can be allocated to $j$, where $d_i^{xreq}$ is the set of all requests $R$ which are excluded by the requester. The set $X^R$ represents the set of resources in a request $R$.

$$x_{ij} \Rightarrow \bigwedge_{\substack{T \in d_i^{xten} \\ R \in \mathcal{R}_T \\ k \in X^R}} \neg x_{kj} \tag{3}$$

Equation (3) means that if $i$ is allocated to $j$, then no virtual resource $k$ from any request $R \in \mathcal{R}_T$ from any excluded tenant $T$ can be allocated to $j$, where $d_i^{xten}$ is the set of all tenants $T$ which are excluded by the requester. The set $\mathcal{R}_T$ represents the set of requests owned by a tenant $T$.

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 (1). Limiting co-locations 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 (2). Excluding tenants is useful when the other tenants are identified as rivals (3). 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.

### *4.2. Tolerant Attributes: Semantics and Typology*

Aside 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 paper, we provide a first list of tolerant attributes in Table 1. More specifically, Table 1a shows how tenants and InPs may view them, and Table 1b shows the building blocks of the corresponding mathematical structure. Besides, tolerant attributes are divided into binding and non-binding attributes.

$$\sum_{i \in X^R, x_{ij} = \top} d_i^a \leq o_j^a \tag{4}$$

*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 $x_{1j}$ is true. This means that the demand $d_1$ in *mem* is lower that 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 $x_{1j}$ and $x_{2j}$ 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: $d_1^{mem} + d_2^{mem} \leq o_j^{mem}$.

Table 1: Extensible List of Tolerant Attributes

a Classification and Description

| Attribute | InP Description | Tenant Description | Type |
|---|---|---|---|
| *mem* | Available memory in physical node | Required memory for virtual node | $B_N$ |
| *bw* | Available bandwidth of physical link | Required bandwidth for virtual link | $B_L$ |
| *loc* | The location of physical node | Authorized locations for virtual node | $NB_N$ |
| *ven* | The vendor of physical **node** | Authorized vendors along virtual **link** | $NB_L$, with aggregation function |
| *ber* | Bit error rate (BER) of physical link | Maximal authorized BER of virtual link | $NB_L$ |

b Building Blocks

| Attribute | $V_a$ | Ordering operator (demand $\leq$ offer) | $+$ | $m_a$ | $M_a$ |
|---|---|---|---|---|---|
| *mem* | $\mathbb{N} \cup \{\infty\}$ | Natural order | Standard addition | 0 | $\infty$ |
| *bw* | $\mathbb{N} \cup \{\infty\}$ | Natural order | Standard addition | 0 | $\infty$ |
| *loc* | Powerset of *Locations* | $\supset$ (note the direction: offer must be *in* demand) | $\cap$ | *Locations* | $\emptyset$ |
| *ven* | Powerset of *Vendors* | $\supset$ (note the direction: offer must be *in* demand) | $\cap$ | *Vendors* | $\emptyset$ |
| *ber* | Percentage | $\geq$ (note the direction) | min | 100% | 0% |

Binding attributes follow Equation (4). This latter means that the sum of the demands $d_i^a$ of all the virtual resources $i \in X^V$ 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 $a$.

$$x_{ij} \Rightarrow d_i^a \leq o_j^a \tag{5}$$

On the contrary, *non-binding* attributes result in constraints where each demand acceptance can be checked separately. It is typically the case of the 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 $x_{1j}$ is true. This means that the demand $d_1$ in *ber* is lower that 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 1a). 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 allowing 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 1a regarding the orientation of $\geq$).

For non-binding attributes, it is better to use (5)-like constraints. There is one per offer $o_j$, per demand $d_i$ and per attribute $a$.

### *4.3. Tolerant Attributes: Structure*

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 1b. 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 neutral element. We suppose that there exists in $W_a$ an element which is both its greatest 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 and the neutral element in $V_a$.

We show in Demonstration 2 that we can always extend a monoid to get the $M_a$ element, and the *mem* attribute in Table 1b, is an example of a monoid $(\mathbb{N}, \leq, +)$ we extend to get such $M_a$ element, which is denoted as $\infty$.

**Demonstration 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:*

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

*Then, we have stability of $+'$:*

$$\forall x, y \in W_a, \quad x +' y = x + y \qquad \in W_a \subset W_a \cup \{e\}$$
$$\forall x \in W_a, \quad e +' x = e + e = e \qquad \in W_a \cup \{e\}$$

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

$$\forall x, y, z \in W_a, (x +' y) +' z = (x + y) + z$$
$$= x + (y + z)$$
$$= x +' (y +' z)$$
$$\forall x, y \in W_a, (e +' x) +' y = (x +' y) +' e = e$$

*We have reflexivity of $\leq'$:*

$$\forall x \in W_a, x \leq' x \Rightarrow x \leq x$$
$$e \leq' e$$

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

$$\forall x, y \in W_a, x \leq' y \wedge y \leq' x \Rightarrow x \leq y \wedge y \leq x \Rightarrow x = y$$
$$\forall x \in W_a, x \leq' e \wedge e \leq' x \Rightarrow e = x$$

*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.* □

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 (6).

$$x_{ij} \times d_i^a = \begin{cases} d_i^a & \text{if } x_{ij} = \top \ (x_{ij} \text{ is true}) \\ m_a & \text{otherwise} \end{cases} \tag{6}$$

As a general note, we can also define the $\sum$ symbol in (4) 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 paper 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 (4) and (5) that *all* offers will be considered. For the exclusion constraint, the $\emptyset$ value guarantees through (1) 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 (4) and (5) 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 $o_j^a = M_a$, Eqs. (4) and (5) 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. We will present how and why in Sections 6 and 7.

### *4.4. Node and Link Attributes: Semantics and Typology*

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 appendix in Table A.3, we denote as $B_N$ (resp. $NB_N$) the set of binding (resp. non-binding) node attributes, and as $B_L$ (resp. $NB_L$) 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 1a, we present such a *link* attribute (from the tenant's viewpoint), *ven*, whose offer is related to the nodes at the tips of the physical link. For such attribute, in (4) and (5), we consider that $o_j^a$ is the result of some aggregation function denoted $\cdot_a$, as defined in Definition 1.

**Definition 1** (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 $h$ and $t$ the nodes at the tips of the link $j$.*

$$\forall j = (h,t) \in L^S, o_j^a := o_h^a \cdot_a o_t^a$$

*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:* $\forall x \in V_a, x \cdot_a y = y \cdot_a x$
*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 $+$.*

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 1 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 1b.

All the attributes in Table 1a 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 $\mathcal{P}(S)$, where $S$ is *finite*, because we can associate each bit to the presence/absence of a certain element of $S$. Then for $S = \{a, b, c\}$ 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 1b), is then equivalent to the predicate $(x = x \odot y)$ where $\odot$ denotes the bitwise logical "and".

## 5. Single Domain Slice Embedding Formulation

We leverage our attribute-based requirement model to express the slice embedding problem as a constraint satisfaction problem. 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 constraint satisfaction problem 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 constraint satisfaction problem, 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 paper, we do not focus on finding any 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 cautiously including some optimization features. For instance, paths are expressed using a flow conservation constraint. If we consider this constraint alone, it allows very convoluted paths, which can loop and go through the source and destination many times. Yet, the associated optimality constraint keeps the ILP solver from generating such paths. In this paper, as we leverage more theories than the integer one, it is important to be sure, before any optimization, that our problem is correctly formulated.

We already semantically describe the single domain scenario in Section 3 as a set of rules to follow. 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 $\mathcal{R}$. Each request is owned by a tenant. Each request $R$ is modeled as a directed graph $R = (N^R, L^R)$. The reader can refer to Table A.3 in appendix for all notations used in this paper. We treat one

request at a time, so we distinguish a request-to-embed, denoted $NR$, from already embedded slices.

$$\bigvee_{j \in H^S} \left( n_{ij} \wedge \bigwedge_{k \neq j} \neg n_{ik} \right) \quad \forall R \in \mathcal{R}, i \in N^R \qquad \text{(C1)}$$

$$\bigvee_{j \in P^S} \left( p_{ij} \wedge \bigwedge_{k \neq j} \neg p_{ik} \right) \quad \forall R \in \mathcal{R}, i \in L^R \qquad \text{(C2)}$$

Equation (C1) (resp. (C2)) implements node mapping rule (R1) (resp. link mapping rule (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 4. 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 $j$ (resp. a path $j$), and false (denoted $\bot$) otherwise. Especially, $j \in H^S$ complies with eligible substrate hosts rule (R3), and $j \in P^S$ complies with eligible paths rule (R4). We will give further details about $P^S$ after the description of all equations.

$$\left. \begin{aligned} p_{ij} = (n_{ux} \wedge n_{vy}) \wedge \bigwedge_{k \in L_j} l_{ik} \wedge \bigwedge_{k \in L^S \setminus L_j} \neg l_{ik} \\ \forall R \in \mathcal{R}, i \in L^R, j \in P^S, i = (u, v), E_j = (x, y) \end{aligned} \right\} \qquad \text{(C3)}$$

The $p_{ij}$ variables are derived from $n_{ij}$ and $l_{ij}$ through (C3). It defines the link-to-path variables as the conjunction of mapping link ends to path ends (cf. node-link mapping coordination rule (R5)), then mapping the link to the links along the path, and then not mapping the link to the other links in the infrastructure.

$$n_{i,m(i)} \quad \forall R \in \mathcal{R} \setminus \{NR\}, i \in N^R \qquad \text{(C4)}$$

$$p_{i,m(i)} \quad \forall R \in \mathcal{R} \setminus \{NR\}, i \in L^R \qquad \text{(C5)}$$

Equation (C4) (resp. (C5)) tells that each virtual node $i$ (resp. each virtual link $i$) of the already embedded requests should be mapped to the physical node $m(i)$ (resp. physical path $m(i)$), where $m$ is a function that memorizes the previous results of the embedding algorithm.

$$\sum_{\substack{i \in N^R \\ R \in \mathcal{R}}} n_{ij} \times d_i^a \leq o_j^a \quad \forall a \in B_N, j \in H^S \qquad \text{(C6)}$$

$$n_{ij} \Rightarrow d_i^a \leq o_j^a \quad \forall a \in NB_N, R \in \mathcal{R}, i \in N^R, j \in H^S \qquad \text{(C7)}$$

$$\sum_{\substack{i \in L^R \\ R \in \mathcal{R}}} l_{ij} \times d_i^a \leq o_j^a \quad \forall a \in B_L, j \in L^S \qquad \text{(C8)}$$

$$l_{ij} \Rightarrow d_i^a \leq o_j^a \quad \forall a \in NB_L, R \in \mathcal{R}, i \in L^R, j \in L^S \qquad \text{(C9)}$$

Equation (C6) (resp. (C7), (C8), (C9)) guarantee requirements based on tolerant attributes $B_N$ (resp. $NB_N$, $B_L$, $NB_L$). They express that the offer $o_j^a$ of a InP 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 capacity rule (R6).

Equations (C6) and (C8) are based on (4), using $n_{ij}$ and $l_{ij}$ instead of $x_{ij}$. They describe binding attributes. For their part, equations (C7) and (C9) are based on (5), using $n_{ij}$ and $l_{ij}$ instead of $x_{ij}$. They describe non-binding attributes.

$$n_{ij} \Rightarrow \bigwedge_{k \in d_i^{xres}} \neg n_{kj} \quad \forall R \in \mathcal{R}, i \in N^R, j \in H^S \qquad \text{(C10)}$$

$$n_{ij} \Rightarrow \bigwedge_{\substack{R' \in d_i^{xreq} \\ k \in N^{R'}}} \neg n_{kj} \quad \forall R \in \mathcal{R}, i \in N^R, j \in H^S \qquad \text{(C11)}$$

$$n_{ij} \Rightarrow \bigwedge_{\substack{T \in d_i^{xten} \\ R' \in \mathcal{R}_T \\ k \in N^{R'}}} \neg n_{kj} \quad \forall R \in \mathcal{R}, i \in N^R, j \in H^S \qquad \text{(C12)}$$

Equations (C10), (C11), and (C12) express node exclusion constraints. They are based respectively on (1), (2) and (3), using $n_{ij}$ instead of $x_{ij}$. They also comply with R6.

$$l_{ij} \Rightarrow \bigwedge_{k \in d_i^{xres}} \neg l_{kj} \quad \forall R \in \mathcal{R}, i \in L^R, j \in L^S \qquad \text{(C13)}$$

$$l_{ij} \Rightarrow \bigwedge_{\substack{R' \in d_i^{xreq} \\ k \in L^{R'}}} \neg l_{kj} \quad \forall R \in \mathcal{R}, i \in L^R, j \in L^S \qquad \text{(C14)}$$

$$l_{ij} \Rightarrow \bigwedge_{\substack{T \in d_i^{xten} \\ R' \in \mathcal{R}_T \\ k \in L^{R'}}} \neg l_{kj} \quad \forall R \in \mathcal{R}, i \in L^R, j \in L^S \qquad \text{(C15)}$$

Finally, Equations (C13), (C14), and (C15) express link exclusion constraints. Likewise, they are based respectively on (1), (2) and (3), using $l_{ij}$ instead of $x_{ij}$. They also comply with R6.

To give more details, the R4 is enforced through the set $P^S$ (see Table A.3 in appendix), which is the set of what we call "loop-free paths". Loop-free paths are defined by Definition 2. 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.

**Definition 2** (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 is different from target. When source equals 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* $(source, (source, source), source)$*.*

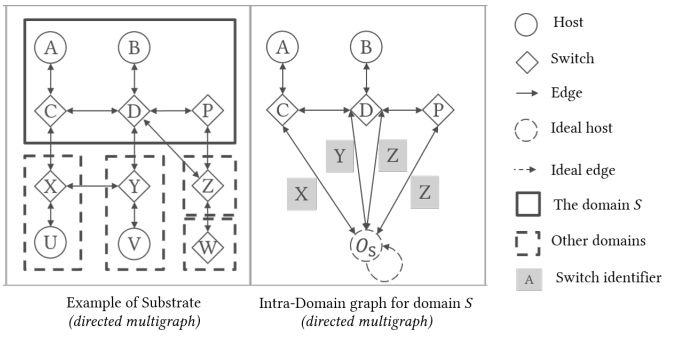



Figure 2: Example of Intra Domain Graph

## 6. Intra-Domain Level Slice Embedding Formulation

The constraint satisfaction problem as formulated in Section 5 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 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 paper, we consider two levels of cooperation, the inter-domain level and the intra-domain level. The inter-domain level is presented in Section 7.

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 that in 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 partial mapping is not easy though, because node mapping and link mapping are interrelated. In our previous work (Boutigny et al.), we modeled the partial mapping with a heuristic, which allows to abstractly embed some tenant nodes in some border nodes, where border nodes are the nodes at the boundary of the given domain. This embedding is abstract in the sense that border nodes were virtually considered as host candidates, despite not being hosts. Besides, no attribute constraint was applied for them.

Such heuristic is very limited though, because we could have two different virtual links leading to the *same* virtual node exiting the domain through two *different* border nodes. This was not possible with our previous heuristic, by definition.

Instead, we use a topological heuristic. This heuristic enables us to reuse Section 5.

Before explaining the heuristic, let us model what a domain can know. We use Figure 2 for this purpose. On the left, we present an example of a substrate, with 5 different domains, among which one is denoted $S$. As in Section 5, it is a directed, connected graph.

On the right of Figure 2, we present the intra-domain level graph, which actually represents what we assume $S$ can know, in the form of a directed, connected multigraph. Formally, we consider that any domain knows:

*i)* its own resources, as well as their attributes (in Figure 2, this corresponds to $A$, $B$, $C$ and $D$, as well as the links between them),

*ii)* the inter-domain links, as well as their attributes (in Figure 2, this corresponds to $(C, X)$ and $(D, Y)$, among others), and

*iii)* the switches to which it is connected in its neighboring domains, but *not* their attributes (in in Figure 2, this corresponds to $X$, $Y$ and $Z$).

Under these limitations of what $S$ knows, the inner topology of other domains ($(X, U)$ in Figure 2 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 $O^S$ 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 2). This follows a legitimate assumption, that there is no disjoint domain in our system.

Besides, we consider a self-loop $(O^S, O^S)$, which allows us to outsource a virtual link to the outside. The existence of $(O^S, O^S)$ 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 $OP^S$ the set of loop-free paths in $P^S$ that start from a host inside the domain, go out through $O^S$ and then end in another host inside the domain. The candidate paths we consider for our embedding are then in $P^S \backslash OP^S$.

By construction, there is a unique path in $P^S \backslash OP^S$ that contains $(O^S, O^S)$. It is the path $(O^S, (O^S, O^S), O^S)$, as explained in Definition 2. 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 $O^S$ and the self-loop $(O^S, O^S)$ is that they are ideal, which is defined in Definition 3.

**Definition 3** (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.*

By doing so, $O^S$ and $(O^S, O^S)$ 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 knownledge 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 Section 4, like the *ven* attribute. In that case, the inter-domain link attribute values are *different*, due to the presence of $O^S$. 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 1), this difference does not impact the validity of the embedding, as we show in Demonstration 3. 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 3.** *Let $d$ and $e$ be two domains. Let $S$ be the infrastructure with full knowledge about $d$ and $e$. Let $(h,t) \in L^S$ be an inter-domain link between $d$ and $e$. Let $a$ be a tolerant link attribute with an aggregation function $\cdot_a$.*

*Thanks to the aggregation function properties, as given in Def. 1, we can show that having both domains separately accepting the demand on abstract links $(h, O^d)$ and $(O^e, t)$ respectively implies to accept the demand with full knowledge on the physical link $(h,t)$.*

*By definition of outside hosts $O^d$ and $O^e$, as well as per Def. 1, we have:*

$$o^a_{(h,O^d)} = o^a_h \cdot_a M_a = o^a_h$$
$$o^a_{(O^e,t)} = M_a \cdot_a o^a_t = o^a_t$$
$$o^a_{(h,t)} = o^a_h \cdot_a o^a_t$$

*Let $R$ be a request and $i \in L^R$ a link. Let assume that each domain $d$ and $e$ separately accepts the demand $d^a_i$ for $o^a_h$ and $o^a_t$ respectively. In other words:*

$$1)\ d^a_i \leq o^a_h$$
$$2)\ d^a_i \leq o^a_t$$

*It follows:*

$$\text{translation-invariance of 1): } d^a_i \cdot_a o^a_t \leq o^a_h \cdot_a o^a_t$$
$$\text{translation-invariance of 2): } d^a_i \cdot_a d^a_i \leq o^a_t \cdot_a d^a_i$$
$$\text{idempotence of } \cdot_a\text{: } d^a_i \cdot_a d^a_i = d^a_i$$
$$\text{commutativity of } \cdot_a\text{: } d^a_i \leq d^a_i \cdot_a o^a_t$$
$$\text{transitivity of } \leq\text{: } d^a_i \leq o^a_h \cdot_a o^a_t$$

*As we have $d^a_i \leq o^a_h \cdot_a o^a_t$, we know that $S$ will also accept the demand $d^a_i$ for the link $(h,t)$.* □

We now present how this heuristic helps us to adapt the equations from Section 5. Every rule from Section 3 which is modified is explicitly cited. Those which are not cited are not modified.

Eq. (C1) still holds because $O^S$ is a host. Eqs. (C2) and (C3) only hold for $P^S \setminus OP^S$ and not $P^S$. This is to reduce recursivity: otherwise, if we enable paths going out through $O^S$, 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 (C2b), (C3b).

Eqs. (C4) and (C5) still hold because $O^S$ is a host. Eqs. (C6) and (C7) still hold because $O^S$ is an ideal node (cf. Def. 3). Eqs. (C8) and (C9) still hold because $(O^S, O^S)$ is an ideal link (cf. Def. 3). Eqs. (C10), (C11) and (C12) only hold for host $j \neq O^S$, because $O^S$ is an ideal node (cf. Def. 3. In other words, we alter capacity rule (R6) to make an exception for exclusion attributes on $O^S$, which is consistent with its abstract nature. See (C10b), (C11b), (C12b).

Equations (C13), (C14), and (C15) only hold for link $j \neq (O^S, O^S)$ because $(O^S, O^S)$ is an ideal link (cf. Def. 3). In other words, we alter R6 to make an exception for exclusion attributes on $(O^S, O^S)$, which is consistent with its abstract nature. See (C13b), (C14b), and (C15b).

$$\bigvee_{j \in P^S \setminus OP^S} \left( p_{ij} \wedge \bigwedge_{k \neq j} \neg p_{ik} \right) \quad \forall R \in \mathcal{R}, i \in L^R \tag{C2b}$$

$$\left.\begin{aligned} p_{ij} = (n_{ux} \wedge n_{vy}) \wedge \bigwedge_{k \in L_j} l_{ik} \wedge \bigwedge_{k \in L^S \setminus L_j} \neg l_{ik} \\ \forall R \in \mathcal{R}, i \in L^R, \ j \in P^S \setminus OP^S, i = (u,v), E_j = (x,y) \end{aligned}\right\} \tag{C3b}$$

$$n_{ij} \Rightarrow \bigwedge_{k \in d_i^{xres}} \neg n_{kj} \quad \forall R \in \mathcal{R}, i \in N^R, \boxed{j \in DH^S} \tag{C10b}$$

$$n_{ij} \Rightarrow \bigwedge_{\substack{R' \in d_i^{xreq} \\ k \in N^{R'}}} \neg n_{kj} \quad \forall R \in \mathcal{R}, i \in N^R, \boxed{j \in DH^S} \tag{C11b}$$

$$n_{ij} \Rightarrow \bigwedge_{\substack{T \in d_i^{xten} \\ R' \in \mathcal{R}_T \\ k \in N^{R'}}} \neg n_{kj} \quad \forall R \in \mathcal{R}, i \in N^R, \boxed{j \in DH^S} \tag{C12b}$$

$$l_{ij} \Rightarrow \bigwedge_{k \in d_i^{xres}} \neg l_{kj} \quad \forall R \in \mathcal{R}, i \in L^R, \boxed{j \in DL^S} \tag{C13b}$$

$$l_{ij} \Rightarrow \bigwedge_{\substack{R' \in d_i^{xreq} \\ k \in L^{R'}}} \neg l_{kj} \quad \forall R \in \mathcal{R}, i \in L^R, \boxed{j \in DL^S} \tag{C14b}$$

$$l_{ij} \Rightarrow \bigwedge_{\substack{T \in d_i^{xten} \\ R' \in \mathcal{R}_T \\ k \in L^{R'}}} \neg l_{kj} \quad \forall R \in \mathcal{R}, i \in L^R, \boxed{j \in DL^S} \tag{C15b}$$

## 7. Inter-Domain Level Slice Embedding Formulation

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, and use them to adapt the equations from Section 5, 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. F 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 eligible paths rule (R4) to enable two virtual nodes to be embedded in the same domain. To do so, we will use loop-free paths (cf. Def. 2), and benefit from the behavior with self-loops. Then, we keep node-link mapping coordination rule (R5) as is. Finally, we must relax capacity rule (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.

We now present how this heuristic helps us to adapt the equations from Section 5.

Eqs. (C1), (C2), (C3), (C4) and (C5) still hold because domains are hosts. Eqs. (C6) and (C7) still hold because domains are ideal nodes. Eqs. (C8) and (C9) still hold because domain self-loops are ideal links. Eqs. (C10), (C11) and (C12) are removed because domains are ideal nodes. Eqs. (C13), (C14), and (C15) only hold for inter-domain links, as domain self-loops are ideal links. See (C13c), (C14c), and (C15c).

$$l_{ij} \Rightarrow \bigwedge_{k \in d_i^{xres}} \neg l_{kj} \quad \forall R \in \mathcal{R}, i \in L^R, \boxed{j \in L^S \backslash SL^S} \tag{C13c}$$

$$l_{ij} \Rightarrow \bigwedge_{\substack{R' \in d_i^{xreq} \\ k \in L^{R'}}} \neg l_{kj} \quad \forall R \in \mathcal{R}, i \in L^R, \boxed{j \in L^S \backslash SL^S} \tag{C14c}$$

$$l_{ij} \Rightarrow \bigwedge_{\substack{T \in d_i^{xten} \\ R' \in \mathcal{R}_T \\ k \in L^{R'}}} \neg l_{kj} \quad \forall R \in \mathcal{R}, i \in L^R, \boxed{j \in L^S \backslash SL^S} \tag{C15c}$$

## 8. Multi-level Resolution Algorithm

Our algorithm is presented in Figure 3. 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 $T$ submitting a new request $NR$ to the unique *InterDomainSolver* (denoted $U$), which collaborates with the multiple *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 6), 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 7).

The generation algorithm, presented in Fig. 3 starts with an **initialization** step, where the tenant $T$ sends the new request $NR$ (step 1) to the inter-domain solver $U$, which relays it to the different intra-domain solver instances $d$ (step 3). When receiving $NR$, each solver runs independently a *genCstr* (for "generate constraints") function. The function $genCstr^d$ (resp. $genCstr^U$) is the function which generates the intra-domain level (resp. inter-domain level) mathematical constraints as per the formulation in Section 6 (resp. Section 7).

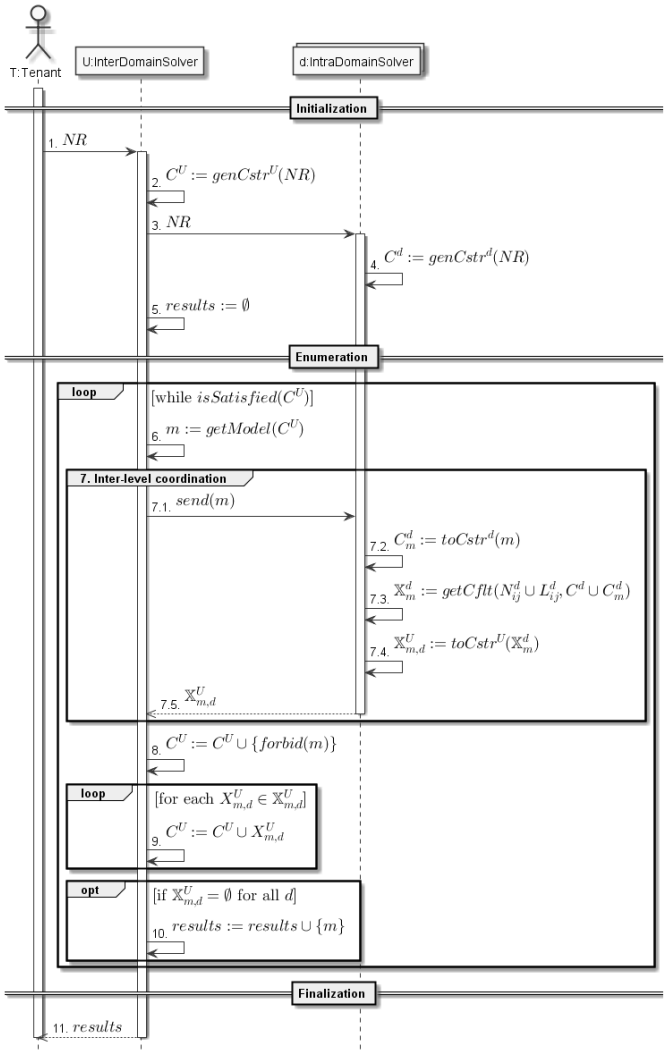



Figure 3: Embeddings Generation Sequence Diagram

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 $T$.

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

$$\bigvee_{x \in X} x \neq m(x) \tag{7}$$

$$\bigvee_{x \in N_{ij} \cup P_{ij}, m(x) = \top} \neg x \tag{8}$$

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 (7). 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 (8).

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 $j \in H^d \cup OP^d$ (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. Thus (8) is the chosen expression for *forbid*.

**Demonstration 4.** *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 5. 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:*

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

*2) $\vee$-split:*

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

*3) $\wedge$-split:*

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

*4) $j$ is indeed unique:*

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

*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 $k \neq j$, the $n_{ik}$ 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 $i = (u, v) \in L^R$. Let $j \in P^S$ such that $p_{ij} = \top$. Let $E_j = (x, y)$. It follows:*

*1) Equation* (C3) *is true:*

$$p_{ij} = (n_{ux} \wedge n_{vy}) \wedge \bigwedge_{k \in L_j} l_{ik} \wedge \bigwedge_{k \in L^S \setminus L_j} \neg l_{ik} = \top$$

*2) $\wedge$-split:*

$$(n_{ux} = \top) \wedge (n_{vy} = \top) \wedge (\forall k \in L_j, l_{ik} = \top) \wedge (\forall k \in L^S \setminus L_j, l_{ik} = \bot)$$

*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 $l_{ik}$ for $k \neq j$. 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.* □

In our algorithm, 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 $toCstr^d$ (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 $toCstr^U$, which are sent to $U$ at step 7.5.

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.

### *8.2. 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 (Liffiton et al.), 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 (Bjørner).

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 Eqs. (9), (10), and (11). Equation (9) 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, (9) 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 (11) means that mapping a tenant link $i$ on a inter-domain link from/to the domain $d$ is kept as is within the domain $d$.

$$\forall i \in N^{NR}, \qquad \exists n_{ij}^U, j = d \Rightarrow \neg n_{i,O^d}^d \tag{9}$$

$$\forall i \in N^{NR}, \qquad \exists n_{ij}^U, j \neq d \Rightarrow n_{i,O^d}^d \tag{10}$$

$$\forall i \in L^{NR}, \qquad \exists l_{ij}^U, j \in IL_d^U \Rightarrow l_{ij}^d \tag{11}$$

When generating constraints with the function $genCons^U$ (or $genCons^d$), the solver of course need to know the set of our loop-free paths $P^U$ (or $P^d$, see Table A.3 in appendix). If we assume that the infrastructure topology does not change, then this set can be considered as constant, like Yu et al. does, and produced with a depth-first search algorithm.

### *8.3. Solution Selection*

Figure 4 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 Eqs. 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 $NR$ 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 Eq. C4 and Eq. C5 at the intra-domain level.

The tenant $T$ now has a slice. Note however that how the physical resources are technically reserved and configured is out of the scope of this paper, therefore not presented in Figure 4.

## 9. Prototype and Use Case

Figure 5 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

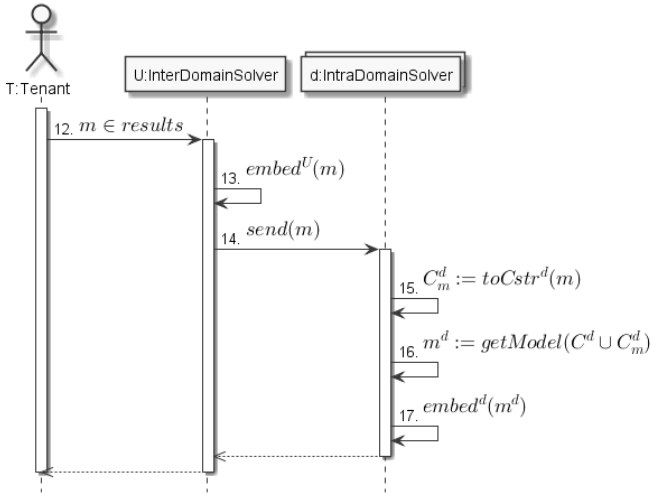



Figure 4: Embedding Selection Sequence Diagram

switch vendor $V3$. The database containing some sensitive or regulation-protected information cannot be stored in $Loc1$. Using the exclusion attribute, the tenant explicitly avoids the co-location 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 6 shows the specific form 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* (*xres* 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 7, 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.

The tenant now executes our embedding generation algorithm (see Figure 3). The tenant is then redirected to an interface similar to the one shown on Figure 8. 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 (see Figure 4).

The core language of this prototype is Python 3. In particular, we use the following Python 3 packages.

- The z3 SMT solver (de Moura and Bjørner; Bjørner and de Moura) which is available through the `z3-solver` Python package.
- The `networkx` package (Hagberg) provides graph manipulation and modelling. Particularly, we use their

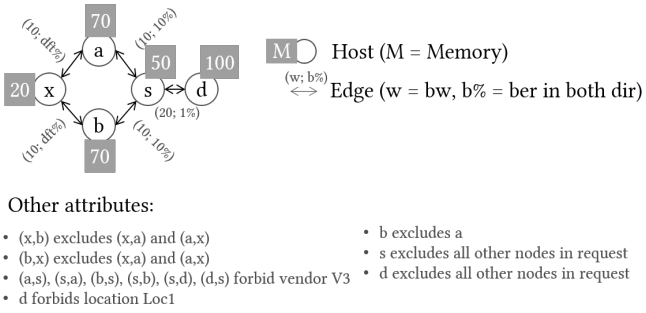



Figure 5: Use case request

simple path generation algorithm, which is based on a breadth-first search.

- The `multiprocessing` standard Python package provides domain isolation by modelling them as processes.
- The `bottle package` (Hellkamp) provides a web-based interface to tenants.
- The `ZODB` package (Fulton) provides a database for storing Python objets, especially graphs from `networkx`.

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. The very reason of this problem is that in Section 4 we do not provide any notion of mutability in our mathematical model. The following paragraphs describe how ignoring mutability may affect the data, and how we technically implement this mutability.

We want to enable $Vendors$ to be a mutable set, evolving over time. Let $Vendors(t)$ denote such time-dependent variable. At the initialization time ($t_0$), we assume that $Vendors(t_0) = \{v_X, v_Y\}$. At another (later) instant, $t_1$, we assume that $Vendors(t_1) = \{v_X, v_Y, v_T\}$.

Consider the requirement "vendor $v_X$ is not authorized" as expressed by the tenant. The tenant creates a request with such requirement at $t_0$. We know that $Vendors(t_0) = \{v_X, v_Y\}$, so we know that the vendors authorized by the tenant are in $\{v_Y\}$.

At this stage, it seems legitimate to exactly memorize only the information $\{v_Y\}$ to represent the requirement given by the tenant. However, this becomes wrong with the mutability of $Vendors$. Indeed, at $t_1$, as $Vendors(t_1) = \{v_X, v_Y, v_T\}$, the value $\{v_Y\}$ itself does *not* represent "vendor $v_X$ is not authorized" but "vendor $v_X$ *and* vendor $v_T$ are not authorized". In other words, we have semantically changed the meaning of the requirement.

We solve this issue by explicitly storing in the database a tuple $(Vendors, \{v_X\})$, where $Vendors$ is a reference to the vendor set as actually stored in the database, thanks to the object-oriented nature of `ZODB`, and $\{v_X\}$ is the set of

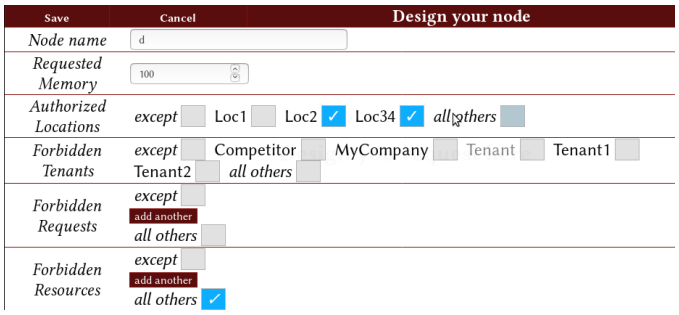



Figure 6: Declaring resource $d$ on prototype

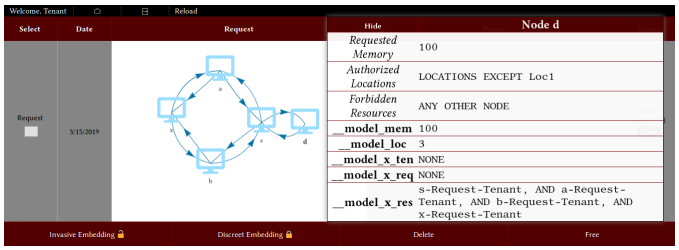



Figure 7: Description of resource $d$ on prototype

values which are explicitly given by the tenant. The exact value as used by the embedding algorithm is then recomputed each time. As such, in our example, the algorithm still interprets "vendor $v_X$ is not authorized" from the data at $t_1$. Obviously, this mechanism is even more necessary for exclusion attributes, as their value sets change whenever a new object appears.

The same problem appears when dealing with the $m_a$ value of the powerset-based attributes. For instance in Table 1b, $m_{ven} = Vendors$, where $Vendors$ is the set of all supported vendors (indeed, $V_{ven}$ is the powerset of $Vendors$). This set may change over time, leading to an ambiguity when interpreting the value as it is stored later.

Consider $Vendors' := Vendors \cup \{v_Z\}$. We do not know what better suits the tenant intent. We can still consider $m_{ven}$ as the default value and allow $v_Z$, or we can stick to allowing $v_X$ and $v_Y$.

To avoid this ambiguity, we also enable tenants to declare if they authorize future values, thanks to the *andothers* keyword.

## 10. Experimentations

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

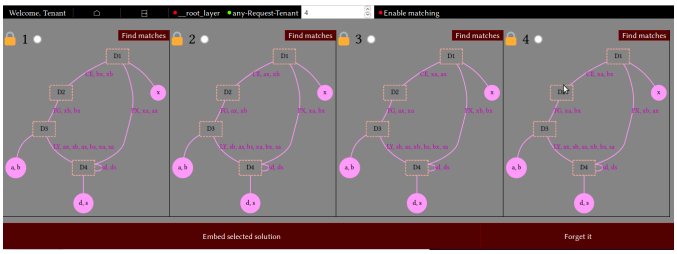



Figure 8: Proposed embeddings

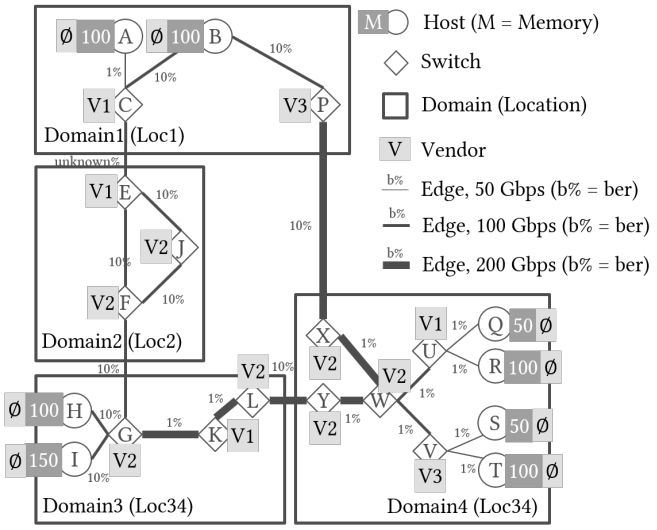



Figure 9: The use case substrate, with its sub domains

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 9. 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 distinct from the others in 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 InPs are unable to provide attribute information. As described in Section 4, the actual value is set to $m_{ber}$, which is equal to 100%.

Figure 10 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 10a 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 10b. For a chain topology, the theoretical number of embeddings is $3 \times 5^n$ where $n$ is the number of nodes (see Demonstration 5). For a star topology, the theoretical number of embeddings is $3 \times 5^n$ where $n$ is the number of nodes (see Demonstration 6 in appendix). For a full-mesh topology, the theoretical number of embeddings for $n \in (1, 2, 3, 4)$ is given in Demonstration 7 in appendix, and is respectively 3, 27, 673, 41985.

**Demonstration 5.** *Let $f(n)$ be the number of solutions for a chain of $n$ nodes, with the substrate given in Figure 9. We show by induction that $f(n) = 3 \times 5^n$. We denote as $N_i$ where $i \in (1, \cdots, n)$ 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: $m_1^0(N_1) = Domain1$, $m_2^0(N_1) = Domain3$ and $m_3^0(N_1) = Domain4$. 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:*

$$\forall i \in (1, \cdots, n), \qquad m_q^{n+1}(N_i) = m_k^n(N_i)$$
$$\forall i \in (1, \cdots, n-1), \quad m_q^{n+1}((N_i, N_{i+1})) = m_k^n((N_i, N_{i+1}))$$

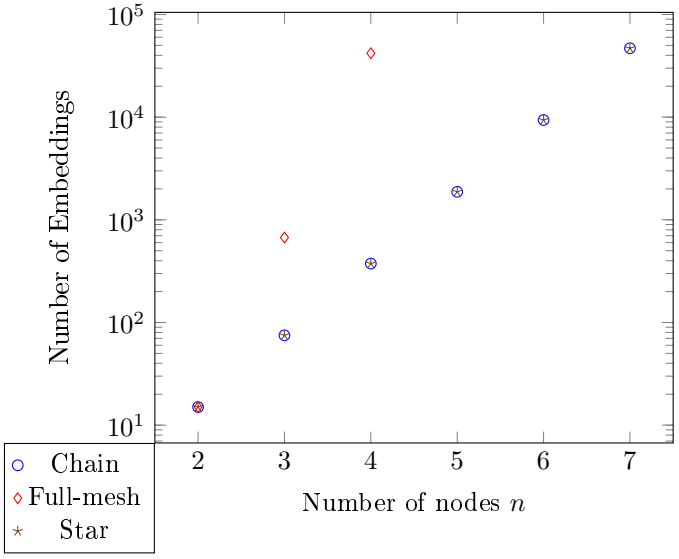
*Then we need to embed $N_{n+1}$ and $(N_n, N_{n+1})$. Let $d = m_q^{n+1}(N_n)$ be the domain hosting $N_n$.*

*If $m_q^{n+1}(N_{n+1}) = d$, then by Definition 2 we have $m_q^{n+1}((N_n, N_{n+1})) = (d, (d, d), d)$. We then have described one (1) solution from $m_k^n$.*

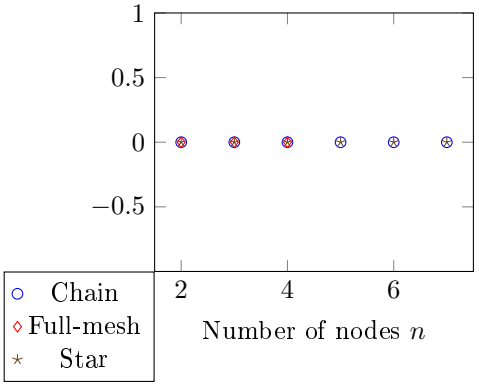
*If $m_q^{n+1}(N_{n+1}) \neq d$, then we can choose among the two (2) other domains. Let $d' = m_q^{n+1}(N_{n+1})$, $d' \neq d$. Let $d''$ be the remaining domain, $d'' \neq d'$, $d'' \neq d$. As the substrate topology is a ring, we have two (2) paths that join $d$ to $d'$. The first path is $(d, (d, d'), d')$ and the second path is $(d, (d, d''), d'', (d'', d'), d')$. 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 $f(n + 1) = (1 + 4) \times f(n) = 5 \times f(n)$.* □

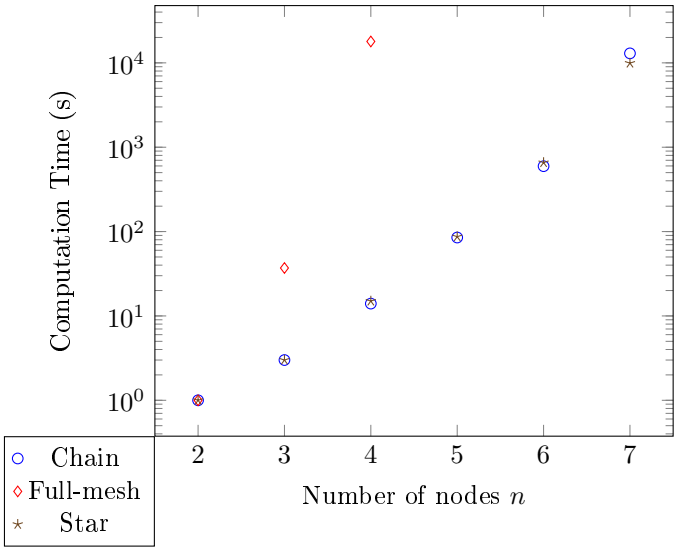
As shown in Figure 10b, there is no deviation from the expected number of embeddings for any tested request. For star and chain topologies, the experiment was conducted from $n = 2$ to $n = 7$ and repeated 50 times until $n = 5$ and 5 times for $n = 6$ and $n = 7$ due to computation time. For full-mesh topology, the experiment was conducted from $n = 2$ to $n = 4$ and repeated 5 times due to computation time.




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

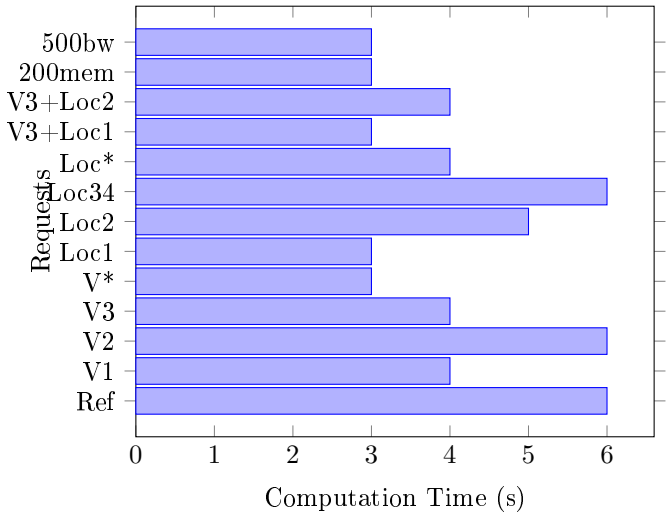



b Difference between Theoretical and Measured Numbers of Embeddings per Number of Nodes $n$ for Chain, Full-mesh and Star Topology

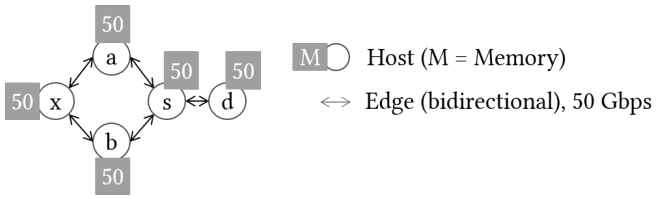



c Computation Time per Number of Nodes $n$ for Chain, Full-mesh and Star Topology

Figure 10: Computation Time Trend for Chain, Full mesh and Star Topology




a Computation Time Trend with 0-Solution Request




b The Reference Request ($Ref$)

Figure 11: Computation Time Trend with 0 Solution Request

In Figure 10c, 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.), which took at least several minutes for the $n = 3$ case.

Figure 11a 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 11b) 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 Table 2.

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 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

| Request name | Difference with $Ref$ request |
|---|---|
| $500bw$ | $d^{bw}_{(s,d)} = d^{bw}_{(d,s)} = 500$ |
| $200mem$ | $d^{mem}_{s} = 200$ |
| $V3 + Loc2$ | $d^{ven}_{(s,d)} = d^{ven}_{(d,s)} = Vendors \setminus \{V3\}$ $\wedge d^{loc}_{s} = Locations \setminus \{Loc2\}$ |
| $V3 + Loc1$ | $d^{ven}_{(s,d)} = d^{ven}_{(d,s)} = Vendors \setminus \{V3\}$ $\wedge d^{loc}_{s} = Locations \setminus \{Loc1\}$ |
| $Loc*$ | $d^{loc}_{s} = \emptyset$ |
| $Loc34$ | $d^{loc}_{s} = Locations \setminus \{Loc34\}$ |
| $Loc2$ | $d^{loc}_{s} = Locations \setminus \{Loc2\}$ |
| $Loc1$ | $d^{loc}_{s} = Locations \setminus \{Loc1\}$ |
| $V*$ | $d^{ven}_{(s,d)} = d^{ven}_{(d,s)} = \emptyset$ |
| $V3$ | $d^{ven}_{(s,d)} = d^{ven}_{(d,s)} = Vendors \setminus \{V3\}$ |
| $V2$ | $d^{ven}_{(s,d)} = d^{ven}_{(d,s)} = Vendors \setminus \{V2\}$ |
| $V1$ | $d^{ven}_{(s,d)} = d^{ven}_{(d,s)} = Vendors \setminus \{V1\}$ |

Table 2: The Other Requests

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.

This last interpretation leads us to draw the conclusion that our algorithm is dependent on the attribute values of the requests and of the domains. Consequently, the algorithm displays varying computation times depending on the attributes in use.

In general, biding attributes are expected to be more computationally expensive than non-biding attributes, because the algorithm may have to try several variable assignments before satisfying Equation (C6) (or Equation (C8)), while one variable assignment is required in Equation (C7) (or Equation (C9)).

In fact, Equation (C6) and Equation (C8) are very easy to *un*-satisfy, which is illustrated by the short times for $500bw$ and $200mem$ in Figure 11a.

As a last remark, when the aforementioned equations rely on an offer in an attribute $a$ whose value is not set by the InP, the default value (see Subsection 4.3), $m_a$, is used by the algorithm. By design, this default value drastically reduces the number of variable assignments that the algorithm may perform to satisfy Equation (C6) (or Equation (C8)), enabling us to support missing attribute data without impacting performances.

## 11. Conclusion

In this paper, 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 and the intra-domain is only known by the Infrastructure Provider (InP) owning

it. We design two Satisfiability Modulo Theories (SMT) problems for those levels, and we propose an algorithm to coordinate them in order to generate all possible solutions in an efficient manner. Besides, we support an extensible and tolerant requirement model, which enables InPs to guarantee new requirements and even blacklist-based ones, which are useful for security purpose.

Our work has some downsides though. First, even though we want InPs to limit information disclosure, the third party may end by having some knowledge about it, if it records the replies from the InPs. Second, we do not support all known requirements, like latency. The reason holds in the information disclosure limitation. Latency cannot be modelled within our requirement model, because we must validate the latency for a whole path, which may go through multiple domains.

## Acknowledgments

This work is partially supported by the Celtic-Plus SENDATE-TANDEM project (2016-2019), Nokia Bell Labs France and the Association Nationale de la Recherche et de la Technologie.

## References

Max Alaluna, Luís Ferrolho, José Rui Figueira, Nuno Neves, and Fernando Ramos. Secure virtual network embedding in a multi cloud environment. abs/1703.01313.

L. R. Bays, R. R. Oliveira, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary. A heuristic based algorithm for privacy oriented virtual network embedding. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1 8. doi: 10.1109/NOMS.2014.6838360.

Nikolaj Bjørner. Enumeration of minimal unsatisfiable cores and maximal satisfying subsets. URL `https://github.com/Z3Prover/z3/blob/master/examples/python/mus/marco.py`.

Nikolaj Bjørner and Leonardo de Moura. The z3 theorem prover, ver. 4.8.1. URL `https://github.com/Z3Prover/z3/tree/z3-4.8.1`.

A. Blenk, A. Basta, M. Reisslein, and W. Kellerer. Survey on network virtualization hypervisors for software defined networking. 18(1): 655 685. ISSN 1553 877X. doi: 10.1109/COMST.2015.2489183.

Francois Boutigny, Stephane Betge Brezetz, Herve Debar, Gregory Blanc, Antoine Lavignotte, and Ion Popescu. Multi provider secure virtual network embedding. In *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1 5. IEEE. ISBN 978 1 5386 3662 6. doi: 10.1109/NTMS.2018.8328706.

Mosharaf Chowdhury, Fady Samuel, and Raouf Boutaba. Polyvine: policy based virtual network embedding across multiple domains. In *Proceedings of the Second ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures*, pages 49 56. ACM.

Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, pages 337 340. Springer Berlin Heidelberg. ISBN 978 3 540 78800 3.

Hao Di, Vishal Anand, Hongfang Yu, Lemin Li, Dan Liao, and Gang Sun. Quality of service aware virtual network mapping across multiple domains. In *Globecom Workshops (GC Wkshps), 2013 IEEE*, pages 476 481. IEEE.

David Dietrich, Amr Rizk, and Panagiotis Papadimitriou. Multi provider virtual network embedding with limited information dis closure. 12(2):188 201. ISSN 1932 4537. doi: 10.1109/TNSM.2015.2417652.

Andreas Fischer, Juan Felipe Botero, Michael Till Beck, Hermann de Meer, and Xavier Hesselbach. Virtual network embedding: a survey. 15(4):1888 1906, a. ISSN 1553 877X. doi: 10.1109/SURV.2013.013013.00155.

Andreas Fischer, Juan Felipe Botero Vega, Michael Duelli, Daniel Schlosser, Xavier Hesselbach Serra, and Hermann de Meer. ALEVIN a framework to develop, compare, and analyze vir tual network embedding algorithms. pages 1 12, b. URL `https://upcommons.upc.edu/handle/2117/15170`.

Andreas Fischer, Ramona Kühn, Waseem Mandarawi, and Hermann de Meer. Modeling security requirements for VNE algorithms. In *Proceedings of the 10th EAI International Conference on Performance Evaluation Methodologies and Tools*. ACM, c. 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`.

Jim Fulton. ZODB: ZODB, a python object oriented database, ver. 5.4.0. URL `http://www.zodb.org/`.

Aric Hagberg. Software for complex networks, ver. 2.2. URL `https://pypi.org/project/networkx/`.

Marcel Hellkamp. bottle: Fast and simple WSGI framework for small web applications, ver. 0.12.13. URL `http://bottlepy.org/`.

Ines Houidi, Wajdi Louati, Walid Ben Ameur, and Djamal Zegh lache. Virtual network provisioning across multiple substrate net works. 55(4):1011 1023. ISSN 13891286. doi: 10.1016/j.comnet.2010.12.011. URL `http://linkinghub.elsevier.com/retrieve/pii/S1389128610003786`.

S. G. Kolliopoulos and C. Stein. Improved approximation algorithms for unsplittable flow problems. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 426 436. doi: 10.1109/SFCS.1997.646131.

Mark H. Liffiton, Alessandro Previti, Ammar Malik, and Joao Marques Silva. Fast, flexible MUS enumeration. 21(2):223 250. ISSN 1572 9354. doi: 10.1007/s10601 015 9183 0.

S. Liu, Z. Cai, H. Xu, and M. Xu. Security aware virtual network embedding. In *2014 IEEE International Conference on Communications (ICC)*, pages 834 840. doi: 10.1109/ICC.2014.6883423.

Toru Mano, Takeru Inoue, Dai Ikarashi, Koki Hamada, Kimihiro Mizutani, and Osamu Akashi. Efficient virtual network optimiza tion across multiple domains without revealing private informa tion. 13(3):477 488. ISSN 1932 4537. doi: 10.1109/TNSM.2016.2582179.

Y. Wang, P. Chau, and F. Chen. A framework for security aware vir tual network embedding. In *2015 24th International Conference on Computer Communication and Networks (ICCCN)*, pages 1 7. doi: 10.1109/ICCCN.2015.7288361.

Minlan Yu, Yung Yi, Jennifer Rexford, and Mung Chiang. Rethink ing virtual network embedding: substrate support for path split ting and migration. 38(2):17 29.

## Appendix A. Appendices

**Demonstration 6.** *Let $f(n)$ be the number of solutions for a star of $n$ nodes, with the substrate given in Figure 9. We show by induction that $f(n) = 3 \times 5^n$. We denote as $N_i$ where $i \in (1, \cdots, n)$ 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.*

***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:*

$$\forall i \in (1, \cdots, n), \qquad m_q^{n+1}(N_i) = m_k^n(N_i)$$
$$\forall i \in (2, \cdots, n), \qquad m_q^{n+1}((N_1, N_i)) = m_k^n((N_1, N_i))$$

*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 by Definition 2 we have $m_q^{n+1}((N_1, N_{n+1})) = (d, (d, d), d)$. 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 $d'$. The first path is $(d, (d, d'), d')$, and the second path is $(d, (d, d''), d'', (d'', d'), d')$. 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 $f(n+1) = (1+4) \times f(n) = 5 \times f(n)$.* □

**Demonstration 7.** *Let $f(n)$ be the number of solutions for a complete directed graph of $n$ nodes, with the substrate given in Figure 9. We show that $f(1) = 3$, $f(2) = 27$, $f(3) = 673$ and $f(4) = 41985$. We denote as $N_i$ where $i \in (1, \cdots, n)$ 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.*

***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 2. 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*

Table A.3: All Notations

| Symbol | Meaning |
|---|---|
| $\mathcal{R}$ | The request set. We embed one request at a time. |
| $NR$ | The new coming request, which we try to embed |
| $\mathcal{R}\backslash\{NR\}$ | The already embedded requests |
| $R \in \mathcal{R}$ | A request, as a **directed** graph |
| $T$ | A tenant |
| $\mathcal{R}_T$ | The requests from tenant $T$ |
| $S$ | The substrate, as a **directed** graph |
| $N^R$, $N^S$ | Nodes in request $R$ or substrate $S$ |
| $H^S \subset N^S$ | Hosts in substrate $S$ |
| $L^R$, $L^S$ | Links in request $R$ or substrate $S$ |
| $P^S$ | Loop free host to host paths (cf. Definition 2) |
| $E_j = (x, y), j \in P^S$ | The nodes at the tips of path $j$. Nodes $x, y \in N^S$ |
| $L_j, j \in P^S$ | The set of links along path $j$ |
| $n_{ij} \in \{\bot, \top\}$ | Allocating $j \in H^S$ for $i \in N^R$ |
| $l_{ij} \in \{\bot, \top\}$ | Allocating $j \in L^S$ for $i \in L^R$ |
| $p_{ij} \in \{\bot, \top\}$ | Allocating $j \in P^S$ for $i \in L^R$ |
| $m(i)$ s.t. $R \neq NR$ | The host/path to which node/link $i$ is mapped |
| $NB_N, B_N, NB_L, B_L$ | Non binding/Binding Node/Link attributes |
| $d_i^a \in V_a$ | Demand in attribute $a$ of $i \in N^R \cup L^R$ |
| $o_j^a \in V_a$ | Offer in attribute $a$ of $j \in N^S \cup L^S$ |
| **Section 4** | |
| *Locations* | The set of all physical locations |
| *Vendors* | The set of all switch vendors |
| $X^V$, $X^P$ | Virtual and physical resources |
| $X^R$ | Resources in request $R$ |
| $x_{ij} \in \{\bot, \top\}$ | Allocating $j \in X^S$ for $i \in X^R$ |
| **Section 6** | |
| $O^S \in H^S$ | The outside node |
| $DH^S = H^S \backslash \{O^S\}$ | The inner hosts |
| $DL^S = L^S \backslash \{(O^S, O^S)\}$ | The real links (intra + inter domain) |
| $N_j, j \in P^S$ | Nodes along path $j$ |
| $OP^S = \{j \in P^S \mid E_j \in (DH^S)^2, O^S \in N_j\}$ | Loop free paths going out through $O^S$ |
| **Section 7** | |
| $SL^S \subset L^S$ | The domain self loops |
| **Section 8** | |
| $x \in X$ | A variable $x$ |
| $m(x)$ | The value assigned to variable $x$ by a model $m$ |
| $N_{ij}$, $P_{ij}$ | The set of node/path allocation variables $n_{ij}/p_{ij}$ |
| $U$ | Refer to the inter domain level |
| $d$ | Refer to the domain $d$ at the intra domain level |
| $IL_d^U$ | Inter domain links connected to domain $d$ |

*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 2 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, 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, 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* $f(4) = 36864 + 4608 + 512 + 1 = 41985$. □