UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA


# SECURE AND DEPENDABLE VIRTUAL NETWORK EMBEDDING

**Luís Xavier Mimoso Ferrolho**

**MESTRADO EM ENGENHARIA INFORMÁTICA**
Especialização em Arquitetura, Sistemas e Redes de Computadores

Dissertação orientada por:
Prof. Doutor Fernando Manuel Valente Ramos
e co-orientada pelo Prof. Doutor Nuno Fuentecilla Maia Ferreira Neves

2016

# Agradecimentos

Começo por agradecer aos meus orientadores Prof. Doutor Fernando Manuel Valente Ramos e Prof. Doutor Nuno Fuentecilla Maia Ferreira Neves por me terem acompanhado ao longo deste ano. O bom ambiente e boa disposição com que se trabalhou, bem como a motivação dada contribuíram para que a concretização deste projeto fosse atingida mais facilmente. Neste mesmo contexto, quero também deixar um agradecimento ao Prof. José Figueira pelo esclarecimento de dúvidas mais técnicas relacionadas com este trabalho, bem como ao Max Alaluna pelas trocas de ideias ao longo do ano que me foram ajudando também a delinear melhor algumas fases do projeto. Agradeço ainda ao projeto SUPERCLOUD onde esta dissertação esteve inserida.

Deixo um grande agradecimento à minha família (pais e irmã) por terem sempre acreditado em mim e por todo o apoio dado, e principalmente por todos os esforços e sacrifícios que fizeram para que tudo isto se pudesse concretizar. Sem eles, certamente nada disto seria possível.

Deixo também um agradecimento especial à minha namorada, Isabel, por todo o apoio, toda a compreensão, motivação, amor e amizade ao longo de todo este tempo. Obrigado pela enorme paciência que tiveste.

Ao pessoal que conheci na faculdade, o Joel, o Super, o Brito e o Rebelo, com os quais desenvolvi uma grande amizade, um muito obrigado. Tive a oportunidade de partilhar praticamente todos os meus dias convosco durante este tempo. Por todas as dores de cabeça que partilhámos por causa de projetos, por todas as tardadas e noitadas a trabalhar, pelos almoços e pelas gargalhadas que existiram todos os dias.

Agradeço ainda ao grupo de amigos de longa data, Matias, Diogo Teixeira, Gustavo, João Redondo e Rui Poeiras com os quais partilho boas memórias de todos estes anos. Espero poder continuar a partilhar bons momentos convosco. E que nos continuemos a encontrar na acolhedora sede durante os fins-de-semana! A eles junto também a Rita, por tudo o que aturou de nós todos ao longo de todo este tempo.

Um sincero obrigado a todos aqueles que, de alguma forma, contribuíram para a minha evolução como pessoa e como profissional.

*A todos os que me ajudaram a chegar aqui.*

# Resumo

A virtualização de redes tornou-se uma técnica poderosa que permite que várias redes virtuais, criadas por diferentes utilizadores, operem numa infraestrutura partilhada. Com o avanço de tecnologias como Redes Definidas por Software[1], a virtualização de redes ganhou um novo ímpeto e tornou-se uma funcionalidade central em ambientes de computação em nuvem.

Um dos grandes desafios que a virtualização de redes apresenta é como utilizar de forma eficiente os recursos oferecidos pelas redes físicas dos fornecedores de infraestruturas, nomeadamente os nós - entidades de uma rede com capacidade computacional - e ligações – entidades de uma rede que transportam dados entre pares de nós. De forma a resolver este problema, vários trabalhos da área de virtualização de redes têm sido desenvolvidos [1]. Em particular, têm sido propostos algoritmos que encontram formas eficazes para decidir onde mapear os nós e as ligações virtuais na rede física. Estes algoritmos podem assumir uma de três aproximações diferentes: soluções exatas, que resolvem pequenas instâncias do problema e encontram soluções ótimas para a localização dos recursos virtuais na rede física; soluções baseadas em heurísticas, que se focam em obter um bom resultado, próximo do ótimo, em pouco tempo; e meta-heurísticas, que usam técnicas específicas independentes do problema para achar um resultado próximo do ótimo.

Tipicamente o objetivo destes algoritmos é achar estes mapeamentos tendo em conta determinadas métricas, como qualidade de serviço, custos económicos ou confiabilidade. Neste contexto, uma das métricas menos exploradas é a garantia da segurança das redes virtuais, um tema que é cada vez mais importante, especialmente em ambientes de computação em nuvem.

As plataformas de virtualização propostas recentemente dão aos utilizadores a liberdade para especificarem de forma arbitrária as topologias virtuais para as suas redes e esquemas de endereçamento. Estas plataformas têm sido desenvolvidas considerando apenas um provedor de nuvem, forçando os clientes a confiarem que este provedor mantém os seus dados e cargas de trabalho seguros e disponíveis. Infelizmente, existem evidências de que problemas nestes ambientes ocorrem, tanto de natureza maliciosa (ataques causados através de algum elemento corrompido na rede) como benigna (falhas em elementos

[1]Paradigma que separa os planos de controlo e de dados/encaminhamento de uma rede.

individuais da rede, ou falhas causadas, por exemplo, por catástrofes, afetando vários elementos da rede em simultâneo) [2]. Deste modo, nesta tese defendemos que a segurança e a confiabilidade são dois fatores críticos e, por isso, devem ser considerados durante o processo de mapeamento das redes virtuais.

Nesse sentido, neste trabalho definimos um problema denominado Mapeamento de Redes Virtuais Seguro e Confiável, e construímos um algoritmo que resolve este problema num ambiente constituído por várias nuvens (i.e., múltiplos provedores de recursos físicos). Ao considerar-se um ambiente como este, evita-se que o cliente fique restringido a apenas um provedor, aumentando a possibilidade de a sua rede e o seu serviço resistirem a falhas em elementos da rede física ou interrupções numa nuvem, através da replicação dos serviços por diversas nuvens. A segurança das redes virtuais também é melhorada na medida em que os serviços mais sensíveis podem ser colocados em nuvens que oferecem maiores garantias de segurança.

O problema em si tem como principal objetivo mapear redes virtuais sobre a rede física, distribuída potencialmente por diferentes nuvens, utilizando a menor quantidade de recursos, e satisfazendo, ao mesmo tempo, os seguintes requisitos: (i) cada nó e ligação virtual é mapeado na rede física satisfazendo os requisitos de capacidade de computação e de largura de banda, respetivamente, e também os requisitos de segurança e confiabilidade associados; (ii) cada nó virtual é mapeado num nó físico cuja localização satisfaz os requisitos do primeiro (isto é, se por exemplo um nó virtual procura uma nuvem que forneça um nível de máxima segurança, o nó físico que será alocado tem de pertencer a uma nuvem com essa característica); (iii) a rede virtual está protegida contra erros na rede física ou disrupção numa nuvem, de modo a cumprir os requisitos de confiabilidade.

O algoritmo que apresentamos nesta tese cobre todos os requisitos deste problema, juntando, pela primeira vez, as propriedades segurança e confiabilidade. Adicionalmente, esta solução considera um ambiente de múltiplos domínios (neste caso, múltiplas nuvens), de maneira a eliminar eventuais limitações que surgem quando se usa um único provedor de nuvem. A solução criada é uma solução exata, desenvolvida através de uma técnica de otimização de programação inteira mista, e tem como objetivo minimizar os custos de mapeamento de redes virtuais, cobrindo sempre os seus requisitos de segurança e confiabilidade. Nesta solução são definidas diversas restrições que têm de ser cumpridas para que uma rede virtual possa ser mapeada sobre uma rede física.

O nosso algoritmo oferece vários níveis de segurança e confiabilidade que podem ser escolhidos na definição das redes virtuais, nomeadamente associados aos nós e às ligações que as compõem. O cliente pode escolher arbitrariamente que níveis deseja para cada recurso virtual, para além de poder especificar também a topologia da sua rede e os requisitos de capacidade de computação e largura de banda para os nós e ligações, respetivamente. Sumariamente, nesta tese consideramos que são suportados vários níveis de segurança para os nós e ligações virtuais, que vão desde segurança por omissão, isto é,

garantias mínimas de segurança, até à inclusão de mecanismos criptográficos que garantem maior segurança. Em relação à confiabilidade, os clientes podem optar por adicionar redundância aos seus recursos virtuais de modo a tolerar falhas. Quando é requisitada redundância, os clientes podem escolher, para cada nó virtual, se desejam a respetiva reserva adicional na mesma nuvem onde se encontra o nó primário, tolerando apenas falhas locais, ou localizada noutra nuvem, com o intuito de aumentar a probabilidade de a sua rede virtual sobreviver a uma disrupção de uma nuvem.

Na nossa solução, as nuvens são também distinguidas entre si consoante o nível de confiança que fornecem ao cliente. Podem ser consideradas nuvens públicas (pertencentes a provedores), privadas (pertencentes aos próprios clientes), entre outras. A definição de diferentes tipos de nuvem dá a possibilidade ao cliente de escolher as nuvens consoante a sensibilidade da sua informação.

Nesta tese é ainda apresentada uma interface de programação de aplicações, que fornece como funcionalidade o mapeamento de redes virtuais segura e confiável, e que pode ser utilizada por plataformas de virtualização que tenham em conta ambientes de múltiplos domínios [3].

Quanto aos resultados, quando segurança e confiabilidade são requisitadas pelas redes virtuais, os mesmos mostram que existe um custo adicional (já esperado) para fornecer estas propriedades. No entanto, um ligeiro ajuste no preço dos recursos permite aos fornecedores de infraestruturas que fornecem segurança e confiabilidade obter um lucro semelhante (ou superior) ao dos fornecedores que não fornecem este tipo de propriedades. Os resultados mostram ainda que o nosso algoritmo se comporta de maneira similar ao algoritmo mais utilizado para mapeamento de redes virtuais, D-ViNE [4, 5], quando os requisitos de segurança e confiabilidade não são considerados.

Apesar de serem uma boa base para novos trabalhos na área, as soluções exatas não escalam (este tipo de soluções apenas consegue resolver problemas num tempo razoável se estes forem de pequena escala). Deste modo, como trabalho futuro, o primeiro caminho a tomar será o desenvolvimento de uma heurística que garanta as propriedades de segurança e confiabilidade.

**Palavras-chave:** Virtualização de Redes, Mapeamento de Redes Virtuais, Segurança, Confiabilidade, Computação em Nuvem

# Abstract

Network virtualization is emerging as a powerful technique to allow multiple virtual networks (VN), eventually specified by different tenants, to run on a shared infrastructure. With the recent advances on Software Defined Networks (SDN), network virtualization – traditionally limited to Virtual Local Area Networks (VLAN) – has gained new traction.

A major challenge in network virtualization is how to make efficient use of the shared resources. Virtual network embedding (VNE) addresses this problem by finding an effective mapping of the virtual nodes and links onto the substrate network (SN). VNE has been studied in the network virtualization literature, with several different algorithms having been proposed to solve the problem. Typically, these algorithms address various requirements, such as quality of service (QoS), economic costs or dependability. A mostly unexplored perspective on this problem is providing security assurances, a gap increasingly more relevant to organizations, as they move their critical services to the cloud. Recently proposed virtualization platforms give tenants the freedom to specify their network topologies and addressing schemes. These platforms have been targeting only a datacenter of a single cloud provider, forcing complete trust on the provider to run the workloads correctly and limiting dependability. Unfortunately, there is increasing evidence that problems do occur at a cloud scale, of both malicious and benign natures. Thus, in this thesis we argue that security and dependability is becoming a critical factor that should be considered by VNE algorithms.

Motivated by this, we define the secure and dependable VNE problem, and design an algorithm that addresses this problem in multiple cloud environments. By not relying on a single cloud we avoid internet-scale single points of failures, ensuring the recovery from cloud outages by replicating workloads across providers. Our solution can also enhance security by leaving sensitive workloads in more secure clouds: for instance, in private clouds under control of the user or in facilities that employ the required security features.

The results from our experiments show that there is a cost in providing security and availability that may reduce the provider profit. However, a relatively small increase in the price of the richer features of our solution (e.g., security resources) enables the provider to offer secure and dependable network services at a profit. Our experiments also show that our algorithm behaves similarly to the most commonly used VNE algorithm when security and dependability are not requested by VNs.

**Keywords:** Network Virtualization, Virtual Network Embedding, Security, Dependability, Cloud Computing

# Contents

# List of Figures

# List of Tables

# Chapter 1 - Introduction

## 1.1 Network Virtualization

Network virtualization has emerged as a powerful technique to allow multiple networks, each customized to a particular service, application or user community, to run on a common substrate. Introduced as a way to evaluate new protocols and services, network virtualization was originally used in research testbeds [6] and has more recently started being applied in distributed cloud computing environments. In addition, it is seen as a tool to overcome the resistance of the current Internet to fundamental changes, mainly due to the existence of multiple stakeholders with conflicting goals and policies.

The basic entity of network virtualization is the virtual network (VN). A VN is a collection of virtual nodes and virtual links forming a virtual topology that is mapped/embedded[1] onto physical nodes and physical links belonging to a physical/substrate network[2] (Figure 1.1).

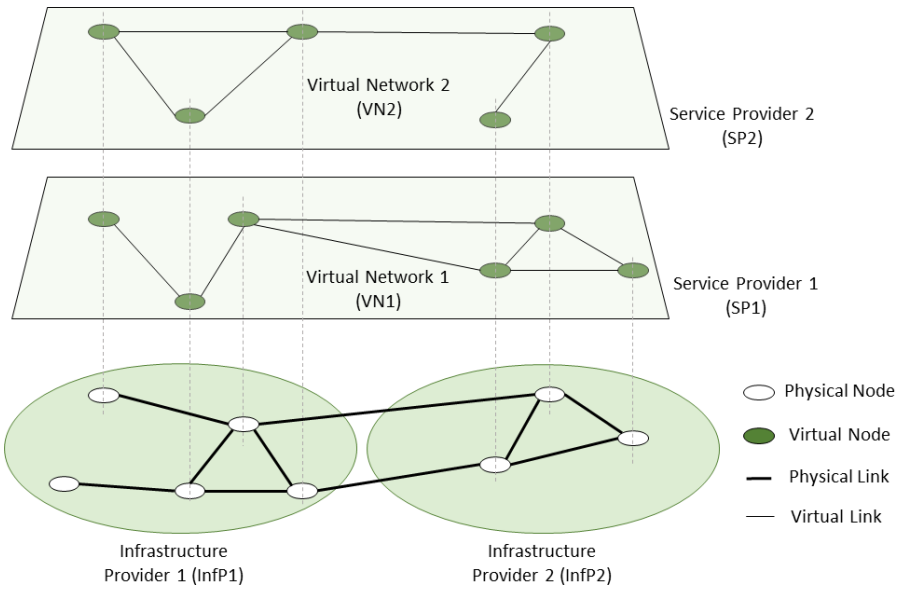



**Figure 1.1:** Network virtualization environment, with the SN at the bottom and one or more VNs on top. VNs are mapped onto the physical network. VNs can create a VN hierarchy, on which some architectural attributes can be inherited.

[1]The words 'map' and 'embed' are used interchangeably throughout this thesis.
[2]The words 'physical' and 'substrate' are used interchangeably throughout this thesis.

Network virtualization is defined by decoupling the roles of the traditional Internet service providers (ISPs) into two different players [7]: the Infrastructure Provider (InP), who deploys and manages the physical infrastructure; and the Service Provider (SP), who creates and manages virtual networks, and deploys customized end-to-end services. The SP accomplishes these tasks through the Virtual Network Provider (VNP), which assembles virtual resources from one or more InPs, and the Virtual Network Operator (VNO), which installs, manages and operates the VN according to the needs of the SP.

The overall goal of enabling multiple heterogeneous VNs to coexist together on a shared physical infrastructure can be subdivided into several smaller objectives. Network virtualization tries to achieve flexibility in order to make SPs able to use arbitrary network topologies, routing or forwarding functions. It also tries to achieve manageability by modularizing the network management tasks through the separation between the SPs and the InPs. Scalability is another important objective, since the InPs should try to maximize the number of coexisting VNs in their network without degrading the performance of any one of them. Other goals that network virtualization tries to achieve are, for instance, programmability and heterogeneity. Programming network elements makes possible the SPs to implement customized protocols and deploy diverse services, and VNs are more flexible and manageable.

In order to take network virtualization to its full potential, several challenges have been addressed over time:

- Interfacing - Every InP has to provide an interface, following some standard, so that SPs can express their requirements;
- Resource and topology discovery - There must be ways to discover resources and topologies so that InPs understand the topology of the networks they manage;
- Resource allocation - Efficient allocation and scheduling of physical resources among multiple VNs;
- Virtualization of the allocated physical resources;
- Naming and addressing with different, and often incompatible, addressing requirements in different VNs;
- Mobility management - Support mobility in core network elements (routers and switches) through migration techniques;
- Failure handling and security - Handle failures and guarantee isolation between coexisting VNs.

## 1.2 Virtual Network Embedding

In this dissertation we will focus on three of the above challenges, namely resource allocation, failure handling and security. We address these challenges by focusing on the problem of embedding VNs in a SN, a problem known as VNE. This problem consists in mapping a new VN, with virtual nodes and links that have certain constraints, onto specific physical nodes and links in the SN, ensuring that the constraints are fulfilled while optimizing some objective.

Since VNE deals with the allocation of virtual resources of two types (nodes and links), it can be split into two sub-problems [1]: Virtual Node Mapping (VNoM), where virtual nodes are allocated to physical nodes, and Virtual Link Mapping (VLiM), where virtual links interconnecting the virtual nodes are mapped to paths connecting the corresponding nodes in the physical network.

In the definition of the problem, each resource, either virtual or physical, has one or more parameters. Node and link parameters are attributes that refer to nodes and links, respectively. These parameters are of extreme importance in order to obtain a valid embedding. An example of a virtual node parameter is the CPU capacity demand, i.e., the computational capacity that this node should have to run some workload. An example of a physical node parameter is the CPU capacity, i.e., the processing capabilities the node can provide to one or more virtual nodes that are mapped on it. The same stands for virtual and physical links: virtual links may demand a certain bandwidth, while physical links have some maximum bandwidth that can be used by the virtual links mapped on it.

VNE consists of finding the optimal or near-optimal way to solve the two sub-problems - VNoM and VLiM - with respect to a certain objective. There are many different objectives that are pursued by VNE algorithms. One example is Quality of Service (QoS), where Virtual Network Requests (VNRs) are installed and operated according to a set of QoS constraints defined by the SP. Another is to maximize the economical profit of the InP. Typically, the more VNRs that can be accepted, the more profit the provider will obtain. In order to achieve this objective, VNE algorithms should try to minimize the resources spent by the SN to map the VNR (known as the embedding cost). A final common objective is dependability. Since failures may occur during the virtual network lifetime, it is important to guarantee that the workload remains operational despite any problems that may occur. To achieve this, there are VNE approaches that setup backup nodes and links to be used in case of faults.

For large problem sizes (large SN and VNRs) the time to find the optimal solution of embedding can become too high, since the VNE problem is computationally intractable [1]. Taking this into account, there are three types of approaches that have been used to solve VNE:

- Exact solutions - Propose algorithms to solve small instances of the problem and

to return optimal results. These optimal VNE solutions can be achieved through Integer Linear Programming (ILP) or Mixed Integer Linear Programming (MILP) (a variant of the first one). These techniques can be used to formulate the VNE problem including the VNoM and VLiM sub-problems. These formulations are then executed in solvers (e.g., GLPK [8] and CPLEX [9]), in order to obtain the result. This is the approach we follow in our work;

- Heuristic solutions - Algorithms that normally find good solutions while keeping execution time low. Since network virtualization deals with dynamic environments where VNRs arrival time is not known in advance, it is important to avoid delays in the processing of new VNs. However, these algorithms may suffer from a limitation where they can get stuck in a local optimum value that can be distant from the real optimum result;

- Meta-heuristic solutions - Algorithms that use specific techniques that are independent from the problem to find near-optimal solutions with a certain measure of quality in a reasonable time.

Normally, the VNO uses the embedding algorithms to decide which virtual resources to request from the VNP, which in turn, instantiates them by using the InPs substrate resources.

## 1.3 The Supercloud Concept

Cloud computing has evolved and gained reputation over the years [10], having revolutionized many areas, such as military, scientific research, and financial systems. However, current private and public clouds normally have difficulties to efficiently respond to the requirements of new applications. In particular, those where the robustness and availability despite cloud failures or cloud-based attacks is paramount. Building applications using resources from multiple clouds is an approach to tackle these limitations. In addition, it would give other benefits, such as users could connect to the nearest data center in order to obtain low latency and highest bandwidth, and services could migrate their virtual resources to providers with lower prices.

Despite the benefits, the use of a multiple cloud providers solution is complex, mainly due to the heterogeneity and the lack of interoperability between them. Different cloud providers use different virtual machine (VM) image formats, and it is not possible to live migrate a VM from one type of hypervisor to another. Networking between VMs in different clouds is also not supported. Therefore, applications are locked-in to a specific cloud, which makes the use of resources from multiple cloud providers difficult.

The Supercloud concept, also referred as cloud-of-clouds, is a “meta-cloud” that uses resources from several private and public clouds. In order to solve the heterogeneity be-

tween the virtual devices and VM images of different clouds, Supercloud makes use of nested virtualization [11, 12]. Nested virtualization allows a hypervisor to run other hypervisors with their associated VMs. This technology has many potential uses in cloud environments. An Infrastructure-as-a-Service provider can give a user the ability to run a user-controller hypervisor as a VM. This way the cloud user can manage its own VMs directly with his favorite hypervisor. Furthermore, nested virtualization enables live migration of hypervisors and their guest VMs as a single entity. This characteristic can be useful for load balancing and disaster recovery.

Figure 1.2 shows a multiple clouds environment that is inherent to the Supercloud concept. In this environment, each user's VM may use resources from different cloud providers, e.g., virtual network 1 includes resources from cloud providers *a* and *b*, while virtual network 2 is mapped to resources from cloud providers *a* and *c*. In this figure, it is also possible to observe the entity that deals with the heterogeneity between different cloud hypervisors and VM images, the Supercloud network hypervisor.

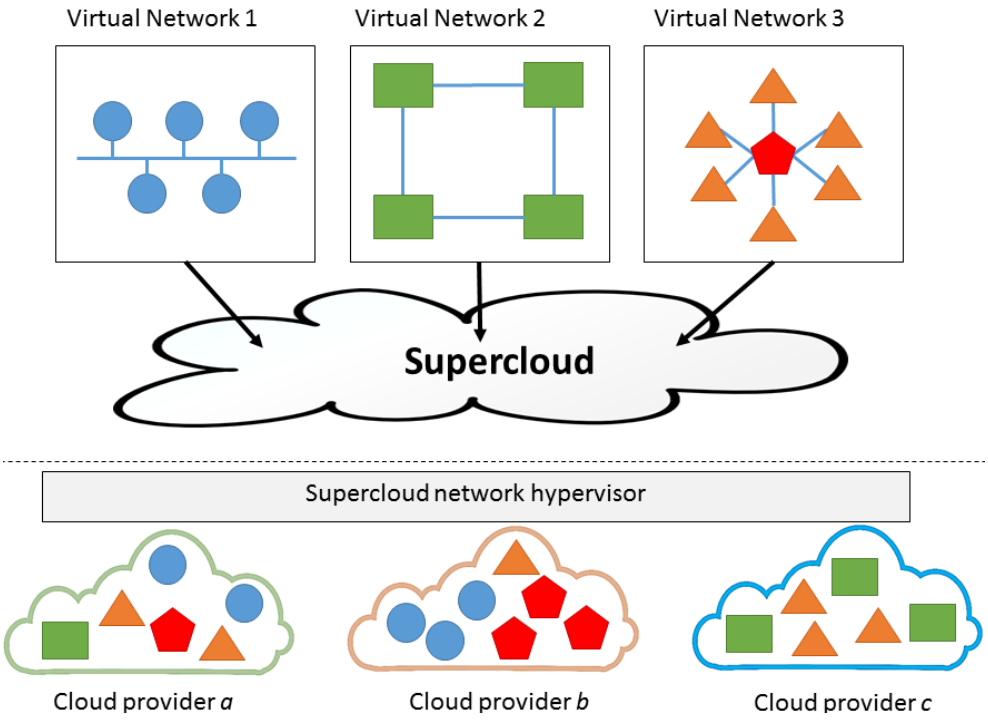



**Figure 1.2:** Multiple clouds environment representation. The tenants' VNs are mapped into resources belonging to different cloud providers.

## 1.4 Motivation

With the advent of network virtualization platforms, cloud operators now have the ability to extend their cloud computing offerings with virtual networks. To shift their workloads to the cloud, tenants trust their cloud providers to guarantee that their workloads are secure and available. However, there is increasing evidence [2] that problems do occur, of

malicious kind where resources may be compromised (e.g., caused by a corrupt cloud insider), or benign when services can not be provided (e.g., a cloud outage), causing important security and dependability concerns.

Embedding VNs in an environment with multiple clouds brings benefits that may attenuate these problems: cloud users are no longer limited to just one provider, having a wider range of resources available to map their VNs, allowing the placement of the most critical services in the clouds where the user has the highest trust. In addition, VNs can be made more robust and have a higher probability to survive to cloud failures if backups are set up at several providers.

Thus, there is a need for VNE algorithms that take into account security and dependability requirements of users, while taking advantage of the benefits provided by a multi-cloud environment.

## 1.5 Goals and Contributions

In this thesis, we argue that security and dependability are becoming critical factors that should be considered by VNE algorithms. In this sense, the main contribution of this work is the design of a secure and dependable VNE algorithm, through a MILP formulation. Our solution considers dependability constraints that allocate additional computing and communication resources during the process of embedding (to tolerate possible failures), and security constraints that require appropriate security mechanisms in place. We also assume a multiple cloud provider model (e.g., one based on nested virtualization) where different types of clouds may coexist: both private, belonging to the tenant, and public, belonging to cloud providers. By not relying on a single cloud provider, we avoid internet-scale single points of failures, tolerating cloud outages by replicating workloads across clouds. In addition, we can enhance security by leaving sensitive workloads in the tenant's private clouds or in trustworthy public facilities.

To the best of our knowledge, this is the first proposal that puts together security and dependability in a formulation of the VNE problem, further extending these requirements to a multiple clouds environment. The main contributions can be summarized as following:

- A MILP solution for the secure and dependable VNE problem in a multiple clouds environment;
- A simple application programming interface (API) that can be used by virtualization platforms to enable the definition of security and dependability attributes;
- An evaluation of the solution comparing its behavior against the most common VNE algorithm (namely, D-ViNE [4]). We assess:

- The acceptance ratio, i.e., the ratio of VNs that are served during a certain period of time;
- The average embedding cost, i.e., the average cost of allocating resources for a certain VN;
- The generated revenue over a period of time;
- The average node utilization, i.e., how much total CPU capacity is allocated to all VNs over time;
- The average link utilization, i.e., how much total bandwidth capacity is allocated to all VNs over time.

In summary, the experimental results show that there is a profit decrease for the providers when security and dependability are taken into account. A relatively small increase of the price of these features enables the provider to obtain a similar profit to the profit of providers that do not offer security and dependability. Our experiments also show that our algorithm behaves similarly to the most commonly used VNE algorithm when security and dependability are not requested by VNs.

We note that this thesis contributed to a section in a deliverable document to SUPERCLOUD project (D4.2 Specification of Self-Management of Network Security and Resilience) [13], and also resulted on an abstract [14] that was presented at NSDI'16 (Symposium on Networked Systems Design and Implementation) poster session.

## 1.6 Planning

Figure 1.3 shows the activities that we have performed in order to produce this dissertation. The scheduling deviated slightly from the initially proposed work plan (Figure 1.4) for multiple reasons:

1. *Algorithm proposal* - The proposal of the algorithm was delayed since it was necessary to first understand better the practical implementations (mainly exact and heuristic solutions) of other works in the field. This helped us to define which type of solution we would use;
2. *Implementation phase* - The implementation phase also took some time, since we had to learn the language used in GLPK (GNU MathProg modeling language [8]) to model our solution. Small tests to ensure the correctness of the solution also delayed this phase. During this period, we also worked on an abstract [14] that was presented at NSDI'16 poster session;

3. *Evaluation* - The evaluation phase was delayed because the writing of a simulator that reproduces the online arrival of VNRs took more time than was planned. The need for repeat the simulations due to modifications to the algorithm as it evolved and the addition of other experiments also delayed this phase. We also had to understand the structure of the inputs for D-ViNE. The construction of an API for network virtualization platforms that takes into account dependability and security also contributed to the delay of this phase;

4. *Dissertation* - The delay to complete the writing of the dissertation was caused by all the above delays. We worked on this phase in parallel with the evaluation phase.

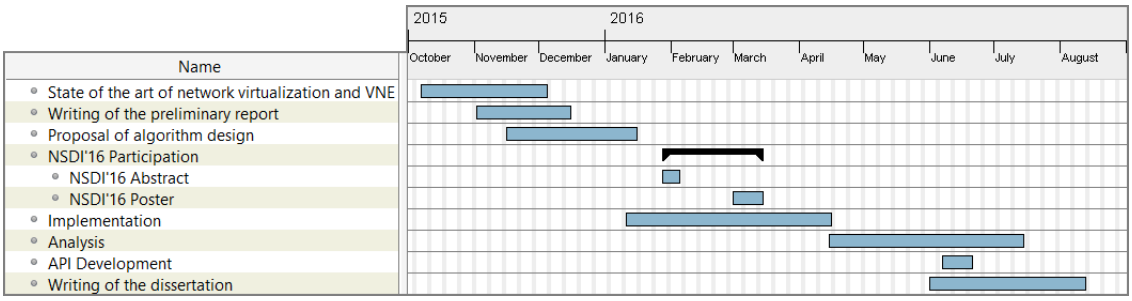



**Figure 1.3:** The activities developed and respective durations to produce this dissertation.

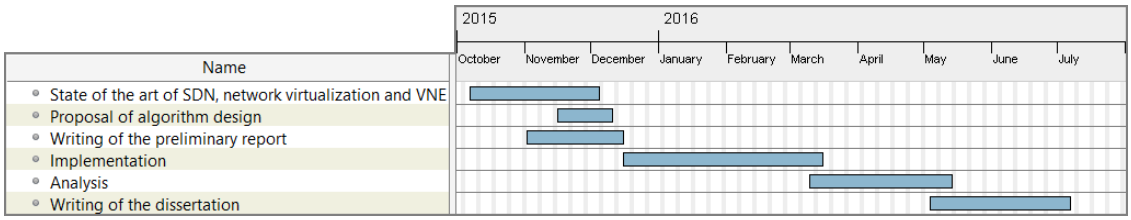



**Figure 1.4:** The activities (and respective durations) anticipated to produce this dissertation.

## 1.7 Thesis Organization

Chapter 2 covers the related work. This is an extensive chapter: we present some network virtualization platforms that are interrelated and that include VNE algorithms in their designs (Section 2.1), and we present VNE algorithms, their evolution and the different types of objectives they try to achieve (Section 2.2).

Chapter 3 defines the Secure and Dependable VNE problem and presents an exact solution to address it. An explanation of the API developed is also presented.

Chapter 4 explains our evaluation method and shows the results obtained from our simulation. A discussion about which alternative techniques can be used in order to improve our results is also shown.

Finally, Chapter 5 summarizes this work, and concludes.

# Chapter 2 - Related Work

## 2.1 Network Virtualization

Network virtualization is not a novel idea [15]. Some well-known technologies are closely related with this concept, such as virtual local area networks (VLANs), virtual private networks (VPNs), and overlay networks. In short, VLANs are groups of logically networked hosts with a single broadcast domain, regardless of their physical connectivity; VPNs are specialized VNs connecting multiple distributed sites through tunnels over shared or public networks; overlay networks are typically implemented in the application layer and they are virtual computer networks which create virtual topologies on top of the physical topologies of other networks.

In recent years, with the technologies advance in networking - namely with SDN - and with the development of datacenters design, network virtualization experienced major improvements. Traditionally, datacenters used dedicated servers to run applications, but this approach resulted in poor server utilization and high operational cost, where management of the resources was hard and time-consuming. Server virtualization, which allows multiple VMs to be co-located on a single physical machine (giving a software abstraction of a server to users), solved these problems, although achieving a transparent management of the resources remained a challenge. In practical environments, datacenters have a multi-tenant nature, that is, single physical datacenters are shared by many tenant users (customers, developers, applications/services). Besides the challenge of resources management, server virtualization alone can not solve all issues: different workloads require different services and network topologies; and virtualized workloads need to operate in the same address space as the physical network. Consequently, the operators can not move their VMs to arbitrary locations, can not authorize VMs to execute their own Internet Protocol (IP) address management schemes (common requirements in datacenters) and can not change the addressing type.

Network virtualization has helped to overcome these challenges by allowing the creation of VNs, each with independent service models, topologies and addressing architectures, on top of a shared physical network, and by permitting the configuration and management of these VNs through global abstractions, reducing the management complexity.

In this section we present network virtualization platforms that are related with our work. All of them are related with the evolution of datacenters, namely multi-tenant datacenters (MTDs), and were constructed in the context of SDN [16], a networking paradigm that separates the control plane from the data (forwarding) plane, centralizes the network control, and defines open, programmable interfaces. In Section 2.1.1, we present NVP, and in Section 2.1.2, we present OVX. Lastly, as in our work we target a multi-cloud scenario, in Section 2.1.3, we present a multi-tenant multi-datacenter network virtualization platform.

### 2.1.1 NVP

NVP [17] is a network virtualization solution for MTDs. It is the result from the advances in datacenter network design, software forwarding, programming languages, and SDNs.

In MTDs, the hosts are connected by a physical network. Each host has multiple VMs supported by the host's hypervisor. A hypervisor on a host provides the right virtualization abstractions to VMs and it has an internal software virtual switch (which is implemented with Open vSwitch (OvS) [18]) that accepts packets from these local VMs and forwards them either to another local VM or over the physical network to another host hypervisor.

The architecture of NVP is built around a network hypervisor. A network hypervisor is a software layer located between the physical forwarding infrastructure and the tenant control planes responsible for providing the right network virtualization abstractions, and has essentially two abstractions: control abstraction, where tenants define a set of logical network data plane elements that they can control; and packet abstraction, that makes packets sent by endpoints see the same service as on a "native" network.

To achieve these abstractions, in NVP design, logical datapaths are implemented in the virtual switches of each host hypervisor and the network hypervisor creates tunnels between every pair of host hypervisors (as such, the physical network only sees IP packets between physical hosts). Additional support is required to provide multicast or broadcast services with the tunnels. To achieve this, NVP constructs a simple overlay multicast using additional physical forwarding elements, known as service nodes. This feature is only used by distributions that need multicast services, otherwise service nodes are not used. NVP also makes use of a logically centralized SDN controller to configure the hosts virtual switches, e.g., modify their flow tables[1], with the appropriate logical rules as tenants show up in the network. Figure 2.1 presents a general architecture of NVP.

Host hypervisors and physical gateways provide location and topology information to the controller. The controller is configured by the SPs and the forwarding state is pushed from the controller to the virtual switches through OpenFlow [19]. This design choice

[1]The flow table in a virtual switch contains a set of flow entries. A flow entry consists of match fields that are matched against packets when they arrive to the switch, and a set of actions that are applied when there is a matching. The match fields can be any field of a packet header and the actions can be forward the matched packet to a given port, forward the matched packet to a SDN controller, or simply drop it.

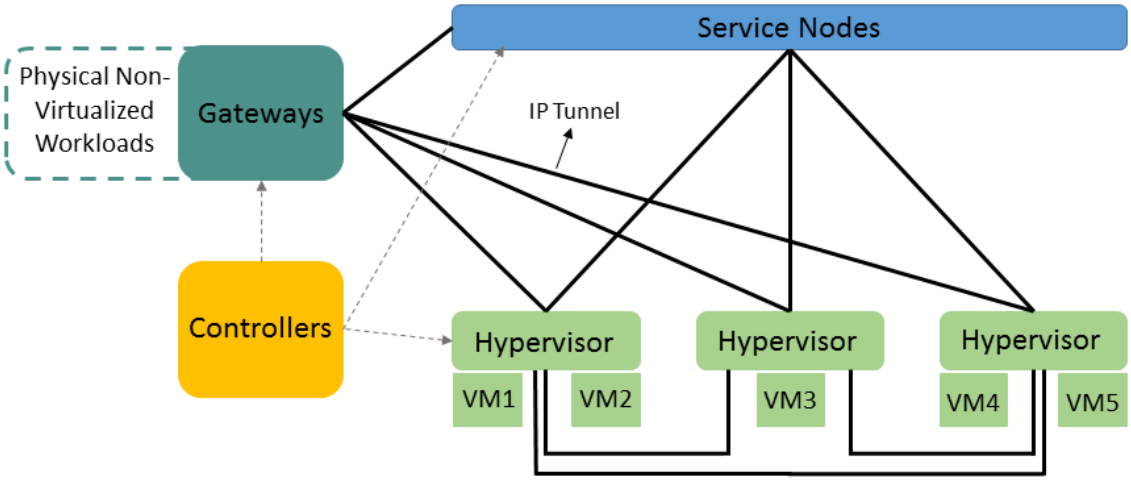



**Figure 2.1:** General architecture of NVP. Controllers manage the forwarding state of all hypervisors, gateways and service nodes. Gateways connect the logical networks with workloads on non-virtualized servers and service nodes are used for logical multicast or broadcast.

triggers other two challenges: software switching at the end hosts needs to be fast (that is, OvS in the end hosts must classify each incoming packet against its entire flow table in software, and that can be a slow process); and the controller computation (computation of the logical datapaths and tunnels) has to scale. The first challenge is solved by making NVP perform exact-match flows in the kernel module of OvS. The first packet of each flow in OvS is sent from its kernel module to the userspace program, where it is matched against the full flow table. Then, the userspace program installs exact-match flows into a flow table in the kernel, which contains a match for every part of the flow. Future packets in this same flow can then be matched entirely by the kernel, saving the time that would be spent by switching between kernel and user spaces. The second challenge is solved by decomposing controller computation into two layers. The upper layer consists in logical controllers, that parallelize the computational workload, i.e., the computation of flows and tunnels for logical datapaths. The bottom layer consists in physical controllers that communicate with hypervisors, gateways and service nodes.

The controller is responsible for every forwarding state and its dissemination to the virtual switches. In order to reduce the recomputing costs, ensure consistency between different orders of events, and handle network changes, the authors of NVP also developed a declarative language for the controller.

Despite the importance of this platform, the work does not mention any type of VNE algorithm to embed VNs that arrive to the system. On the contrary, OpenVirteX [20], which we will describe next, is a network virtualization platform that shows a concrete VNE module in the system design.

### 2.1.2 OVX

OVX [20] is another network virtualization platform that has SDN at its core. In this platform, operators can create and manage virtual SDNs. Tenants are free to specify the topology and addressing scheme of their virtual SDN, and run their own network operating system (NOS) (system that creates the network view, and is capable to communicate with forwarding elements, send control orders, get state information from them, among other features) to control it.

OVX is built based on the design of FlowVisor [21], another virtualization platform, and behaves as an OpenFlow [19] controller proxy between an operator's network and the tenant's NOSs. The topologies of tenants' VNRs can be arbitrary. In OVX, a network embedder module is considered. First, the user specifies all the topology information in a request and sends it to the embedder, which generates a mapping (that is, which decide in which physical elements the virtual elements will be located) using the information from OVX, who has a global view of the physical network. Then, the mapping result is passed to OVX, which is responsible to instantiate the request on the physical topology. The authors do not propose a new embedding algorithm: the network embedder module can use any VNE algorithm depending on the objectives of the SP and the InP. This separation of OVX from the VNE is important since it reduces the complexity of the system, and also reduces the probability of errors in the request mappings, since it uses well-studied VNE works.

To publish a virtual topology, OVX resolves Link Layer Discovery Protocol (LLDP) (protocol used by network devices for advertising their identity, capabilities, and neighbors in Ethernet) messages coming from the tenants' NOS. When an LLDP message arrives at a virtual switch element with a certain outport specified in its body, OVX knows where the other end of that link is. Therefore, OVX forges an LLDP response packet and sends it back to the tenants' NOS, creating the illusion of a link at the NOS.

The address scheme can be defined as the tenant wishes, which increases the probability of existing overlapping IP address blocks in the same physical network. To differentiate hosts, OVX generate globally unique tenant identifiers (IDs) for each tenant, and for each host, a physical address that encodes the host's membership using the tenant ID. Collisions of addresses are avoided by installing flow rules to rewrite addresses at the edge switches of the network: from tenant-assigned address to physical IP address at the ingress edge, and vice-versa at the egress edge, as illustrated in Figure 2.2.

Although these two works have shown that network virtualization is possible, both of them are confined to one datacenter controlled by a single cloud operator. This limitation can be a barrier since many critical applications are being moved to the cloud.

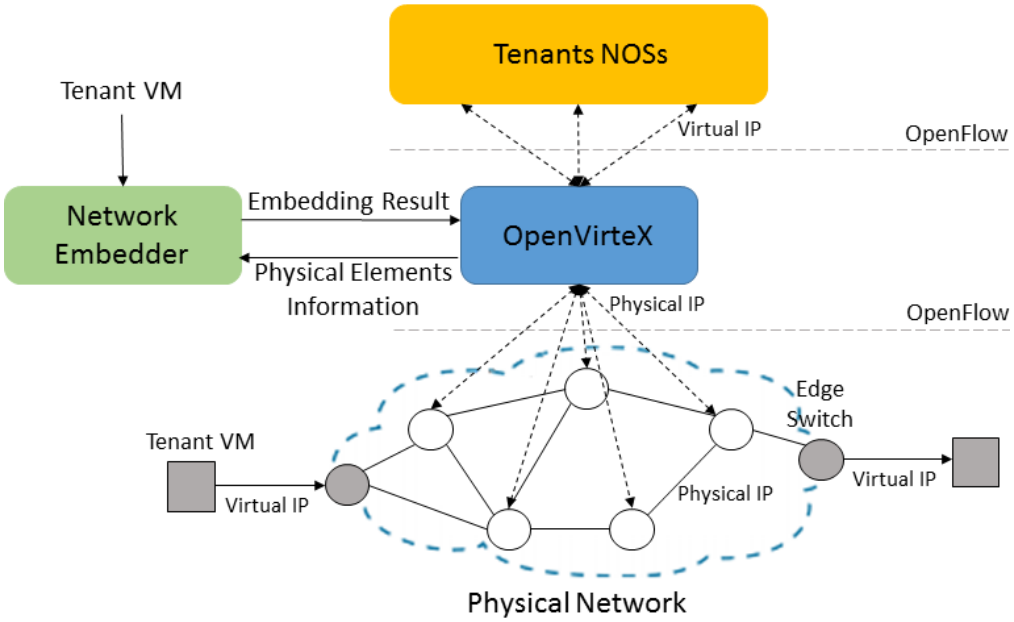



**Figure 2.2:** OVX system architecture. The IP translation is done at the edge switches for data packets and between OVX and the NOS for control packets.

### 2.1.3 Multi-tenant multi-datacenters network virtualization platform

In [3], a network virtualization platform that tackles the challenge of using multiple clouds is presented. Using resources from several cloud providers can potentiate some benefits: by spreading the services across different providers, a tenant can be made immune to any single datacenter or cloud availability zone outage, and thus, it is possible to state that this approach ensures greater dependability; user costs can be decreased due to dynamic pricing plans from the several cloud providers by moving its VMs to less costly locations; and performance and QoS also can be increased by bringing services closer to clients or by migrating VMs that need to closely cooperate during a certain time.

The network virtualization platform proposed takes into account network infrastructures from public cloud providers, on which this platform has a limited control, and private clouds belonging to the tenants. Considering this setting, the platform tries to fulfill three requirements: remote and flexible control over network elements, full network virtualization, and network snapshot and migration. The first requirement is achieved in different ways for the two types of clouds considered. For public clouds, since the control is limited, the platform has an additional virtualization layer on top of the cloud hypervisor to provide virtualization between multiple tenants. Private clouds include the proposed network hypervisor, with OvSs running on bare metal, as is shown in Figure 2.3.

The second requirement is fulfilled by ensuring that the network hypervisor guarantees isolation between tenants, while enabling them to use their desired addressing schemes and topologies. The network hypervisor runs on top of the SDN controller to map the physical to virtual events by intercepting the message flows between the physical net-

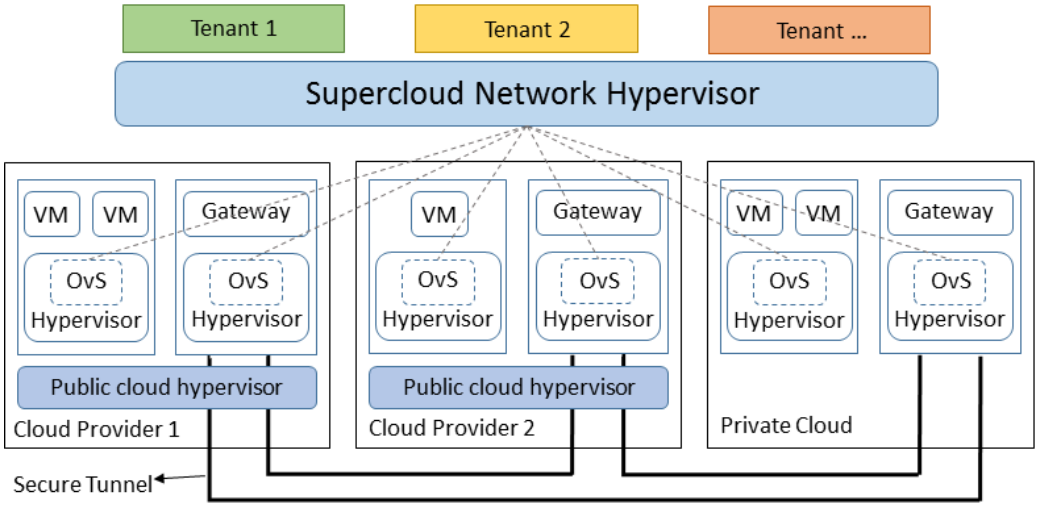



**Figure 2.3:** Network virtualization architecture.

work and the users' applications. This, along with flow rule redefinition at the edge of the network (like OVX), allows isolation between tenants' networks. For addressing virtualization, the traffic that originates from tenant VMs is all tagged. The first 16 bits of the media access control (MAC) address are used as tenant ID. For topology abstraction LLDP messages are all intercepted. By intercepting all topology-related messages the SDN controller can offer arbitrary virtual topologies to tenants.

In order to accomplish the third requirement, the platform uses well-studied snapshot techniques and further extends them to deal with the multiple datacenters environment. For migration, the idea is to clone one or more switches at a time, and then move the VMs associated. This leads to two copies of the same switch to co-exist. To avoid inconsistencies the actions that a switch can take are limited during the migration period. Furthermore, to also respect packet orders, packets from the respective flow are temporarily sent to the controller until it is guaranteed that the rules are installed in both switch replicas.

As in OVX, this platform can make use of a VNE algorithm to decide the mapping between the virtual elements of a tenant's VN and the physical elements (in [3] no such algorithm is presented). The solution we propose in this thesis is the perfect fit for this platform, since it considers a multiple domain environment, namely public and private clouds, and it gives dependability guarantees so that tenants' VNs survive to clouds outages. Furthermore, the security guarantees given by our solution are also an added value to the network virtualization platform.

## 2.2 Virtual Network Embedding

The VNE problem consists in efficiently mapping a VN, with virtual nodes and links, onto SN resources (physical nodes and links). In Figure 2.4 we present an example of

how VNRs can be mapped to a physical network. The virtual nodes in VNR 1 demand 10 units of capacity (CPU) and the virtual links demand 10 – (1,2) and (1,3) – and 12 – (2,3) – units of capacity (bandwidth). The virtual nodes 1, 2 and 3 are mapped to the substrate nodes A, B and E, respectively, respecting the capacity constraints. The virtual link (1,2) is mapped to the substrate links (A,B). The virtual link (1,3) is mapped to the substrate links (A,H) and (H,E). The virtual link (2,3) is mapped to the substrate links (B,C) and (C,E). The virtual resources from VNR 2 are mapped to the SN resources in a similar way as the VNR 1. It is important to note that a substrate resource can map virtual resources from different VNs at the same time, since the sum of capacity demands from the virtual resources do not exceed the capacity available in the substrate resource, e.g., the substrate link (C,E) is part of the path of virtual links (2,3) from VNR 1 and (99,100) from VNR 2.

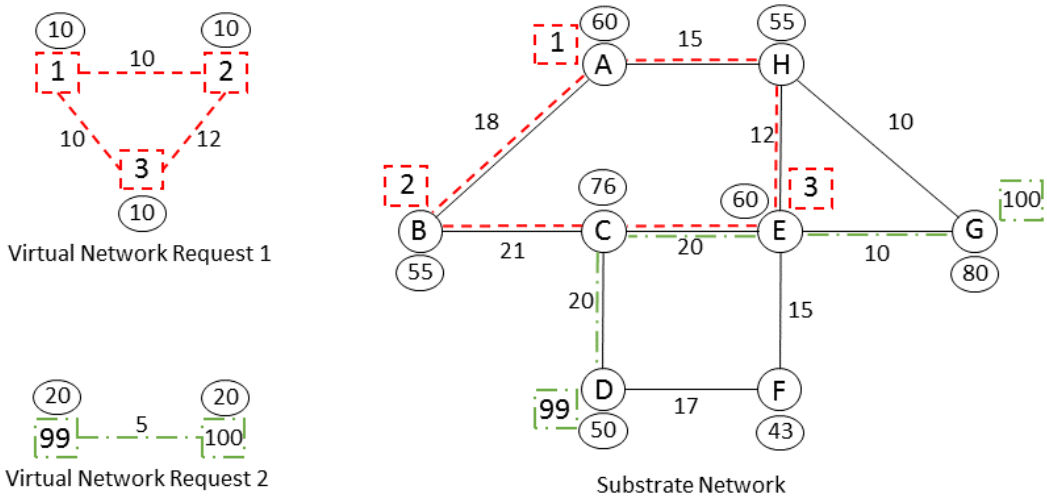



**Figure 2.4:** Embedding of multiple VNRs onto a shared SN. The circular shapes near the nodes in all networks are node parameters, i.e., CPU demand for virtual nodes and CPU available for substrate nodes. The numbers near the links are link parameters, i.e., bandwidth demand for virtual links and bandwidth available for substrate links. It is important to note that virtual links can be allocated to multiple substrate links (a path) in the SN.

In this section we present the major research contributions to the VNE problem. We divide the existing works in basic VNE algorithms and in sophisticated VNE algorithms. In basic VNE algorithms (Section 2.2.1), we show the early algorithms that were developed in the field, classifying each one accordingly with their method to solve the subproblems of the VNE problem, namely, VNoM (mapping of the virtual nodes onto physical nodes) and VLiM (mapping of the virtual links connecting the virtual nodes onto paths that connect the corresponding nodes in the physical network) problems. We classify them into uncoordinated VNE algorithms, where there is no coordination between the VNoM and VLiM phases; two-stage coordinated VNE algorithms, where VNE is achieved through two stages and the VNoM phase tries to obtain a result that optimizes the VLiM phase, achieving better performance; and one-stage coordinated VNE algorithms, where virtual links are mapped at the same time as virtual nodes (in a single

stage). More complex VNE algorithms with specific objectives, such as to achieve energy efficiency, to achieve dependability, to solve the VNE problem across multiple domains, among others, are presented in the sophisticated VNE algorithms section (Section 2.2.2).

### 2.2.1 Basic Virtual Network Embedding Algorithms

In this section we show the VNE algorithms that first addressed this problem. Currently, they are still being used as baseline for many other VNE algorithms. Most of these works try to increase the performance and average revenue, or to reduce the embedding costs. The first algorithms try to address some challenges that make the VNE problem hard [22]: Node and link constraints (the combination of these constraints make the embedding problem computationally difficult to solve), admission control (substrate resources are limited and some VNRs must be rejected or postponed to avoid violating the resources guarantees for existing VNs), online requests (VNRs arrive dynamically and stay in the network for an arbitrary period of time) and diverse topologies (handling arbitrary topologies is difficult).

#### 2.2.1.1 Uncoordinated Algorithms

In [22], M. Yu et al. present an efficient and simple embedding algorithm that addresses the challenges listed above and maximizes the long-term average revenue from accepting VNRs.

The algorithm for VNE is solved in two independent phases. In the VNoM phase a greedy algorithm is employed. It chooses, for each virtual node from a certain VN, a set of eligible substrate nodes and then assigns one of them based on its amount of available resources. The aim is to assign the virtual nodes with higher demands to the substrate nodes with better resources. Then, in the VLiM phase, there are two different ways for mapping the links: Single path mapping using the k-shortest path algorithm [23] when each virtual link must be mapped to a single path in the SN; or Path splitting / Multiple path mapping when each virtual link demand can be carried by several paths in the SN, enabling better resource utilization by harnessing the small pieces of available bandwidth and consequently accepting more VNRs.

#### 2.2.1.2 Two-Stage Coordinated Algorithms

The algorithm presented in [22] does a clear separation between the node and the link mapping phases. However, mapping the nodes without considering its relation to the link mapping stage restricts the solution space and may result in poor performance.

N. Chowdhury et al. in [4] introduce better coordination between the two phases by proposing two new VNE algorithms: D-ViNE (Deterministic VNE) and R-ViNE (Ran-

domized VNE). In these algorithms the authors also minimize the embedding cost and add a new set of node constraints – geographical location for substrate and virtual nodes and a non-negative distance per VNR, indicating how far a virtual node of the VNR can be of its demanded location.

The algorithms take online VNRs as input, i.e., VNRs arrive dynamically, and map them onto the SN one at a time. In order to coordinate the two phases, the substrate graph is augmented with a set of meta-nodes, one per virtual node, each connected to a cluster of candidate substrate nodes obeying location and capacity constraints. A representation of this phase is presented in Figure 2.5. Over the augmented graph, the algorithms execute a MILP formulation that tries to embed the virtual nodes and links. The MILP main goal is to solve the VNE trying to minimize the embedding cost. Since solving MILP formulations is known to be computationally intractable (working only for small instances), the authors present a Linear Programming (LP) relaxation. In this approach, the relaxation is solved and virtual nodes are mapped to real substrate nodes by rounding the obtained solution from the relaxation either deterministically (D-ViNE) or randomly (R-ViNE). Once all the virtual nodes have been mapped, the algorithms perform one of the two solutions proposed in [22] for the VLiM phase.

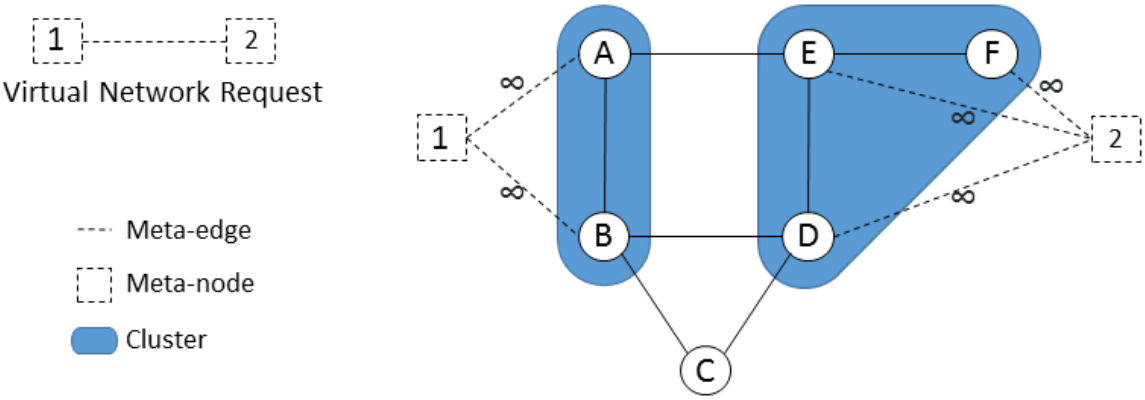



**Figure 2.5:** Augmented substrate graph with meta-nodes and meta-edges for a VNR. For each virtual node, it is created a meta-node that connects to a cluster of candidate substrate nodes, obeying location and capacity constraints. The connection between the meta-nodes and the substrate nodes is done through meta-edges with infinite bandwidth.

In [5] the same authors improve the previous work to a generalized window-based VNE algorithm (WiNE) to evaluate the effect of look-ahead on VN embedding. WiNE discretizes time into consecutive time windows, and instead of making individual embedding decisions for each VN arrival, it batches the requests to process them at the end of a window period. In WiNE, each VNR comes with an additional information denominated maximum waiting period, which sets a deadline on how long WiNE can defer making an embedding decision for that request.

At the end of each window period, all the requests in that window are decreasingly

ordered by revenue, so that the maximum revenues are considered first for embedding, maximizing the overall revenue.

Addressing a different problem from the dynamic arrival of VNRs, in [24], X. Cheng et al. consider that one of the most important parameters of a network node (substrate or virtual) is its position inside the topology. Topological attributes of a node have a direct impact on the success and efficiency of the embedding. Taking this into account, the authors created a node ranking approach, called NodeRank, to measure the topology incidence of a node. This approach is inspired by the PageRank algorithm used by Google's search engine to measure the popularity of web pages based on Markov random walks.

Based on NodeRank, the authors devised RW-MaxMatch to find possible embedding solutions to VNRs. In a pre-stage, the node rank for each node in a VNR and for each node in the residual SN are computed. Then, in the first stage, the algorithm maps the virtual nodes with the highest ranks to the substrate nodes with the highest ranks. In the second stage it embeds virtual links using one of the solutions presented in [22]: shortest path algorithm if path splitting is not supported by the substrate network or multi-commodity flow otherwise.

Most of the algorithms listed until here do not take workload fluctuation into consideration. Predicting workload in a system (or network) that targets users all over the world is extremely difficult. To cope with a peak workload on demand, SPs often over-purchase physical resources, which may lead to a considerable waste of resources for a normal workload and to a loss of some upcoming costumers due to inefficient resource utilization for InPs.

To solve this problem, S. Zhang et al. in [25] re-examine the VN mapping problem proposing a VN mapping framework, ORSTA, which is based on Opportunistic Resource Sharing and Topology-Aware node ranking.

In Opportunistic Resource Sharing, the workload in a VN is modeled as the combination of a basic sub-workload, which always exists, and a variable sub-workload, which occurs with some probability. Multiple variable sub-workloads from different VNs are allowed to share some common resources to achieve efficient resource utilization, while for a basic sub-workload the required resources are allocated as usual.

To facilitate the efficient embedding of VNs, the authors take topology into consideration and also design a Markov Chain-based substrate node ranking algorithm, MCRank, to measure the importance of each substrate node. MCRank is also inspired by PageRank Google's algorithm. To calculate the importance of a substrate node, MCRank takes both residual resources and topology into consideration.

In ORSTA, the algorithm starts computing the residual resources for each substrate node and link, and running the MCRank algorithm. After the arrival of a VNR, in the VNoM phase, all virtual nodes are sorted in a decreasing order of their CPU constraints and are placed in a queue. Then, each virtual node in the sorted queue is mapped to the

unused substrate node with the highest MCRank. In the VLiM phase, each virtual link is mapped to the shortest path between its end hosts. At the end, after a VNR is successfully embedded, ORSTA updates the residual resources and the MCRank of each substrate node and link.

Although two-stage coordinated algorithms have a better coordination between the two mapping phases, they may lead sometimes to unnecessary consumption of bandwidth resources of the SN. For instance, sometimes virtual nodes that are just separated by one-hop in a VN are mapped to a longer path in the SN, wasting its bandwidth resources.

#### 2.2.1.3 One-Stage Coordinated Algorithms

One stage coordinated VNE algorithms are another approach to solve the VNE problem. These solutions try to save bandwidth resources of the SN. Solving the VNE in one single stage implies that virtual links are mapped at the same time as virtual nodes. More precisely, when the first virtual node pair is mapped, the virtual link between them is also mapped.

In [26] a one-stage coordinated algorithm is presented. The authors propose a backtracking algorithm, known as vnmFlib, based on a subgraph isomorphism search method that maps the nodes and links at the same time.

An isomorphism of graphs $G$ and $H$ is a bijection between the vertex sets of $G$ and $H$, $f : V(G) \rightarrow V(H)$, such that any two vertices $i$ and $j$ of $G$ are adjacent in $G$ if and only if $f(i)$ and $f(j)$ are adjacent in $H$ too. Figure 2.6 shows an example of an isomorphism of graphs G and H.

In short, to map a VN, vnmFlib as to find an isomorphic subgraph of the VN in the physical network with the minimum number of hops in the substrate paths.

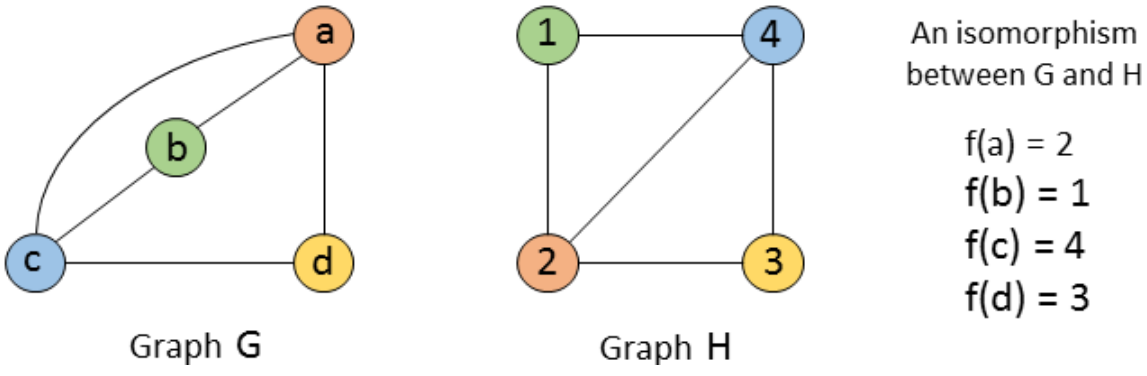



**Figure 2.6:** An example of an isomorphism of graphs.

In [24], the authors also present RW-BFS, a one-stage backtracking VNE algorithm based on Breadth-First Search (BFS). As most of the algorithms of this type, this solution

was designed to solve the unnecessary consumption of bandwidth resources of the SN that a two-stage embedding algorithm may cause.

After computing the NodeRank values for all nodes, substrate or virtual, RW-BFS constructs a BFS tree of the VN, whose root node is the virtual node with the largest NodeRank value. The remaining virtual nodes are sorted in non-increasing order in each level of the tree according to their NodeRank values. Then, for each virtual node, it is built a candidate substrate node list. The substrate nodes in these lists are also sorted by their NodeRank values in non-increasing order. After this, VNE is performed by going through the BFS tree and mapping each virtual node in the first feasible substrate node of its list and, at the same time, mapping the virtual links incident to that virtual node onto the substrate shortest paths that satisfy the bandwidth demands. If there is no suitable mapping for a virtual node $v$, RW-BFS backtracks to the previous one, re-map it to another substrate node and continues to map $v$.

### 2.2.2 Sophisticated Virtual Network Embedding Algorithms

The algorithms described previously were very important to the initial development of the VNE field. However, these algorithms only focus on capacity requirements of VNs, i.e., they only focus on CPU and bandwidth capacity requirements. As new technologies and paradigms appeared, the need to consider more complex requirements has grown. Thus, in this section, we present VNE algorithms that try to achieve more complex objectives, namely energy efficiency, fault-tolerance, security, and also inter-domain environments.

#### 2.2.2.1 Energy efficiency

Most of the previous works presented solve the VNE only looking for the minimization of the embedding cost, i.e., the total resources spent by the SN to embed a VNR, to increase the probability of embedding the next VNR and, as a consequence, minimize the percentage of rejected VNRs.

In order to minimize the energy cost, the authors of [27] introduce a VNE energy aware problem (VNE-EA), where the goal is to allocate the set of VNRs in a reduced group of physical network equipment, and propose a MILP to solve it. In this work, the substrate nodes and links that are not used to realize the mapping of the VN can be deactivated by switching them off to minimize the overall energy consumption of the SN. The proposed model is based on cost-based formulations, similar to [5].

#### 2.2.2.2 Dependability

Due to the shared nature of virtualization and the probability of failures to occur, dependability (i.e., ability to tolerate failures and recover from them) is another complex

objective that has gained importance. To guarantee dependability, redundant computing resources and communication bandwidth need to be allocated for each VN.

In [28], H. Yu et al. study the VN mapping problem where they focus on the failure recovery of facility nodes while minimizing the total amount of computing and bandwidth resources. A facility node is a node with computing capability that can fail if any of its components fails (computer(s), disks, memory, among others). The objective is to restore a VN node mapped onto a facility node that failed by migrating it and its associated virtual connections to a backup facility node.

The authors propose two MILP solutions for dependability, namely 1-redundant scheme and K-redundant scheme, and a heuristic solution. The idea of the 1-redundant scheme is to take the original non-survivable VN graph and transform it to a graph with redundancy by adding one additional VN node. If any of the critical VN nodes fails, the failed VN node and its connections are migrated to that redundant node. Hence, the redundant node should have redundant VN links with all neighbors of all critical nodes. In some cases, especially when resources are limited, establishing multiple VN links from the backup node to each neighbor of all critical nodes is not cost-efficient. It can be difficult or even impossible to support such mapping. The K-redundant scheme solves this problem. This approach is a flexible solution, where in a design phase a K-redundant reliable VN graph is created, in which it is permitted each critical node to have a corresponding backup node. The backup nodes are connected to all neighbors of their corresponding critical VN node. Figure 2.7 shows a K-redundant VN graph, where it is possible to observe backup nodes for the critical nodes and the respective redundant links to their neighbors. In the heuristic, the idea is to use D-ViNE [4] to find the working mapping for the original VNR and use the result to obtain the backup solution through a simplified MILP.

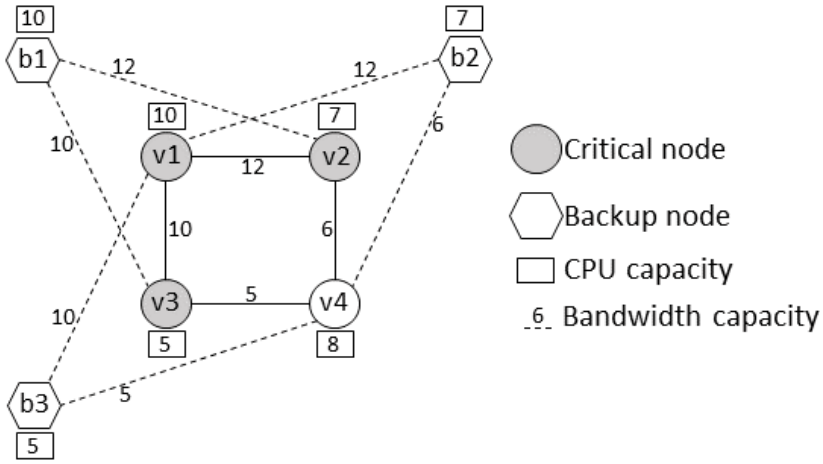



**Figure 2.7:** Representation of a K-redundant VN graph.

In [29], Qian Hu et al. study a similar problem but with one new factor: location-awareness. In network planning or design, location is critical and can affect the system

performance. When embedding VNs, SPs may place constraints on the location of the substrate nodes in order to ensure QoS metrics (e.g., response time, end-to-end delay). Location constraints may also be posed due to geographic requirements of the SP (e.g., the major customers of a service are more concentrated in a certain region). In this sense, to solve this problem defined as Location-constrained survivable network embedding, the authors present an ILP solution that yields a joint optimal solution for both the active and backup traffic accommodation, and also present a heuristic algorithm for larger scales. In both the solutions, the goal is to guarantee the protection against single facility node failures using the minimum quantity of resources at the SN.

In [30], Rahman et al. formulate the survivable virtual network embedding (SVNE) problem to incorporate single substrate link failures in VNE and propose an efficient hybrid policy heuristic to solve it. The main objective of SVNE is to maximize long term business profit. The hybrid heuristic that solves SVNE consists in three separated phases. In the first phase, before any VNR arrives, the InP pre-computes possible backup paths for each substrate link using a path selection algorithm, e.g., k-shortest path algorithm [23]. Therefore, each substrate link have a set of candidate backup detours. The second phase starts when a VNR arrives. In this phase, the InP performs a node embedding using existing heuristics, e.g. [4], and a multi-commodity flow based link embedding through a linear program. The last phase is invoked when a substrate link failure happens. In this event a reactive backup detour optimization solution is invoked which reroutes the affected bandwidth along candidate backup detours selected in the first phase.

In [31], Guo et al. also tackle substrate link failures, postulating that resource efficiency from the perspective of an InP is extremely important. Therefore, they argue that backup resources can be reused by several VNs enabling more space to accept more incoming VNRs. In this sense, the authors propose two shared backup network provision schemes for VNE: Shared-On-Demand approach ($SOD\_BK$) and Shared Pre-Allocation approach ($SPA\_BK$). In short, in the first approach, backup resources are allocated on demand, i.e. upon the arrival of each VNR. In the second one, backup resources are pre-allocated during the configuration phase before any VNR arrives to protect against any potential link failure. The authors focus on link embedding procedure, assuming that node embedding is done firstly using existing approaches, e.g., [22].

Both the approaches presented are formulated through linear programs. In $SOD\_BK$ the objective is to minimize the total resources reserved for both primary flows and restoration flows as well as balance the load to avoid a bottleneck link separating the substrate topology. In this approach, for the primary flows of a VNR, no bandwidth sharing is possible since they need to be operating all the time. The restoration flows, that protect against different substrate link failures, can share the bandwidth to minimize the resources utilization for backup purpose. In $SPA\_BK$ the objective is to maximize the total protected bandwidth of the SN for mapping primary flows, since this approach pre-

allocates the backup bandwidth for each substrate link during network pre-configuration phase and it is impossible to predict the arrival and demand pattern of the future VNRs.

Also looking for the minimum quantity of SN resources used, Yeow et al. [32] defend that backup resources should be pooled and shared across multiple VNs. In this sense, they propose an Opportunistic Redundancy Pooling (ORP) mechanism to leverage the properties of the VN and achieve a $n : k$ redundancy architecture (where $k$ redundant resources can be backups for any of the $n$ primary resources), and share the backups across multiple VNs. ORP ensures VNs do not connect to more redundant nodes than necessary in order to keep the number of redundant links low. Figure 2.8 demonstrates the concept of polling backup nodes for multiple VNs. Backup nodes can be used by several VNs. In this figure, it can be seen as VN1 lending some of its backup nodes to other VNs. This approach presents some advantages such as no degradation for large number of $n$, the possibility for VNs of arbitrary reliability requirements to be pooled together, and flexibility in adding VNs and in removing them.

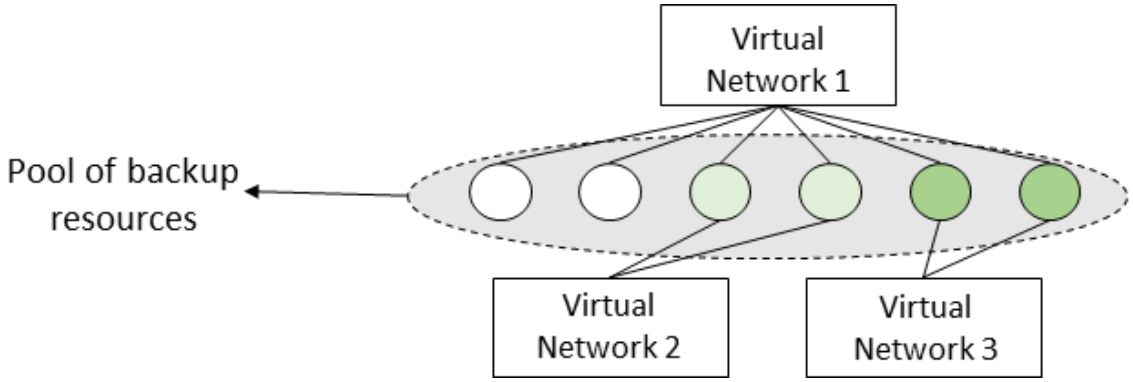



**Figure 2.8:** Pooling backup nodes. The backup nodes can be used by several VNs.

To balance the trade-off between service resource consumption and service resiliency, an efficient resource allocation approach is required. In [33], Y. Chen et al. present Pardalis, another algorithm with resiliency guarantees. By exploiting a heuristic VN mapping scheme and a restoration path selection scheme based on intelligent bandwidth sharing, the algorithm simultaneously makes cost-effective usage of network resources and protects VN services against network failures.

The algorithm solves two problems: the VN mapping and the selection of restoration paths. To solve the first problem, the algorithm tries to use the minimum number of substrate nodes, always respecting the requirements of virtual nodes. Once all the virtual nodes have been mapped to the SN, a primary path is determined for each virtual link. In this work, the authors adopt shortest path algorithms to find a path with minimum bandwidth usages for simplicity, but they could also chose a multi-commodity flow approach. For the second problem, the authors focus on finding a restoration path that is disjoint from its primary path. To achieve this, for each virtual link $l$, the algorithm removes the

substrate links that are on the primary path. Then, it determines the additional bandwidth required to be reserved in the event of any substrate link failure along the primary path. After this, a shortest path is found and denoted as the restoration path for $l$. By also exploiting restoration bandwidth sharing, the total amount of bandwidth consumed by the VN services is reduced.

In [34], H. Yu et al. consider another type of failure: regional failures. A regional failure happens when a geographical region of nodes and links fail due to events such as natural disasters or intentional attacks. Such a failure affects facility nodes, which in turn affect the virtual infrastructure (VI) nodes mapped onto them, and also the links which connect the VI nodes.

Thus, in this work, the authors adopt an approach where there is a backup solution associated with each regional failure scenario and try at the same time to minimize the redundant allocated resources. The authors formulate the survivable VI mapping (SVIM) problem as an optimization problem and propose two heuristic algorithms to solve this problem: separate optimization with unconstrained mapping (SOUM) and incremental optimization with constrained mapping (IOCM).

The main idea is to start with a working mapping of the VI, and then for each failure scenario, derive backup mappings of the VI. The authors present a non-survivable VI mapping algorithm (NSVIM), which satisfies VI requests without any dependability requirement.

The algorithms SOUM and IOCM are built on the top of NSVIM. In SOUM, the problem is decomposed into several separated NSVIM problems, one involving the working mapping and the others involving the backup mappings. In this algorithm, when a regional failure happens, all the VI requests are remapped. The IOCM algorithm also starts considering the working mapping and the regional failures one by one. The major difference from SOUM is that it only finds additional facility nodes to remap the affected VI nodes after a regional failure, instead of remaping all the VIs.

In [35], Khan et al. discuss some challenges that are introduced by dependability in VNE and are not considered by other works namely, that the failure characteristics and repair times are unpredictable, and reserving the full demand of a virtual link as backup is expensive (since resources reserved as backup may never be used). Shared backup schemes, that solve the previous challenge, do not guarantee the full requested bandwidth of a virtual link during failure, and path splitting, although it mitigates the impact of failures, introduces overhead (such as packet redirection, increased routing table size, and packet reordering). In order to address these challenges, the authors propose SiMPLE, an algorithm that presents a multi-path embedding strategy by exploiting the path diversity in the SN, with the goal of finding a trade-off between maximizing dependability and minimizing redundant resources and path splitting overhead. In SiMPLE, a virtual link is split into three disjoint substrate paths and half of the bandwidth demanded by the virtual

link is allocated to each one of them. In this way, two paths are used to carry the primary flow, and the third one is used as backup. Due to the fact that these paths are disjoint, at most one of them is affected by a single failure of a substrate link. Thus, if a substrate path is affected by a failure, the two unaffected paths deliver the demand bandwidth. In Figure 2.9 it is possible to observe that with SiMPLE at least 50% of backup bandwidth is saved in contrast to the full backup schemes. This idea can be extended to a higher number of splits. This work presents an ILP model of SiMPLE (to find optimal solutions in small scale networks), and an heuristic that produces near-optimal solutions. The basic idea of the heuristic is to assume that node embedding is done beforehand and then, for each VN, compute the first k link-disjoint shortest paths using Dijkstra's shortest path algorithm.

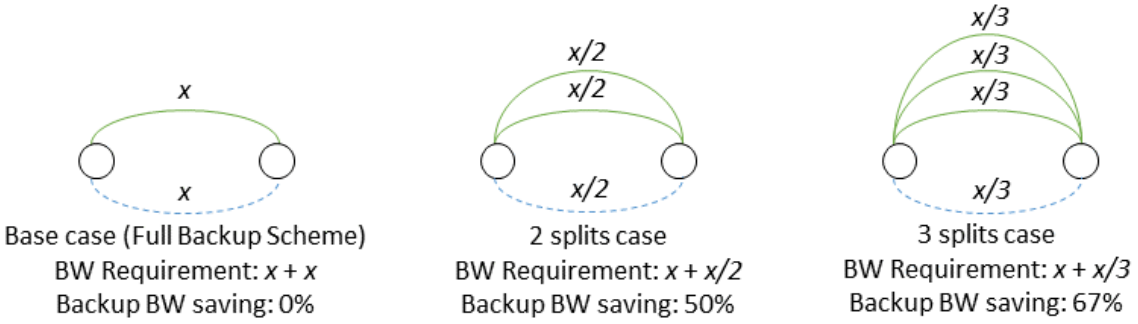



**Figure 2.9:** Embedding general idea of SiMPLE algorithm.

In [36], the authors address a different form of dependability than traditional SVNE: the Connectivity-aware VNE (CoViNE) problem. Here, the goal is to find a VNE that can ensure connectivity in a VN topology in presence of multiple link failures in the SN. The discovery of an embedding that survives to $k$ link failures is done by augmenting the VN topology to be $k + 1$ edge connected, i.e., $k + 1$ edge-disjoint virtual paths exist between each pair of virtual nodes, and then finding which virtual links need to be embedded disjointly so as to maintain $k + 1$ edge-disjoint paths between each pair of virtual nodes after the embedding. Finally the VN is embedded onto the SN obeying to the disjointness requirement while minimizing the total cost of embedding.

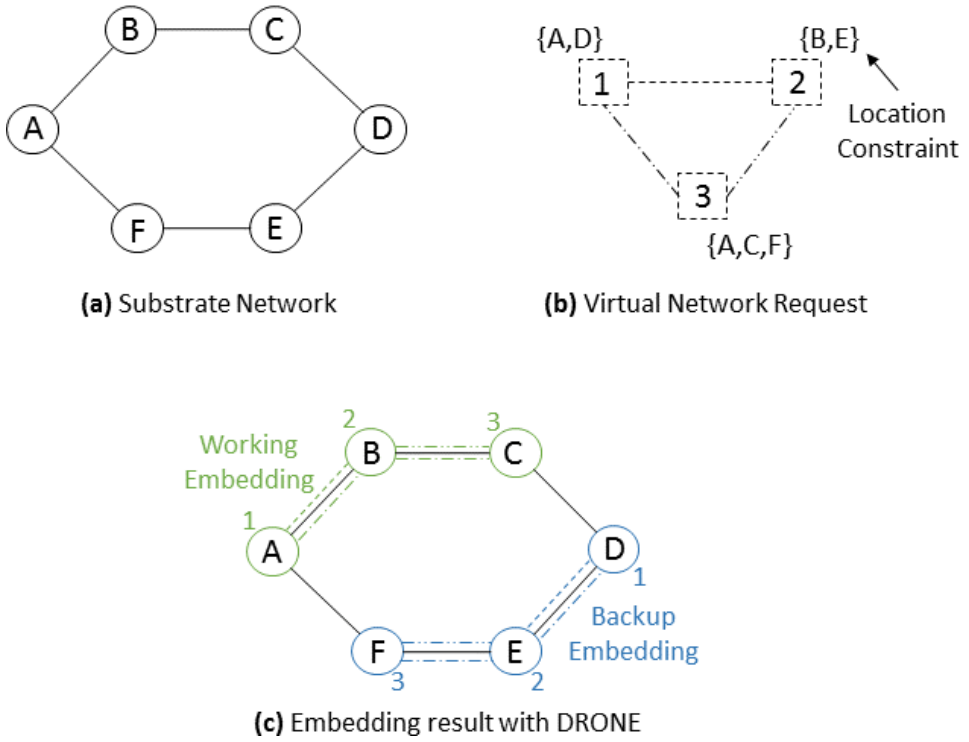
The authors present two approaches to the problem: the first uses a heuristic to compute the disjointness constraint and an ILP model for the VNE; the second uses heuristics for both steps, being capable of solving larger instances of the problem.

In [37], Chowdhury et al. study an extreme case for the VN protection: to provide dedicated backup resources for each virtual node and virtual link in a VNR, an approach also known as 1+1 - protection scheme. In network virtualization, this type of scheme is motivated by use cases from Transport-Software Defined Networking (T-SDN): customer VNs carry high volume and high speed traffic, and usually have agreements with the InP for recovery from physical failures within tens of milliseconds. To satisfy such agreements, the InP needs to provision dedicated backup resources for the VNR, which can be used for immediate recovery from a physical failure. In this sense, the authors formulate

the problem of 1+1 - Protected Virtual Network Embedding (ProViNE) with the objective of minimizing resource provisioning cost in the SN, while protecting each node and link in a VNR with dedicated backup resource in SN.

In this work, the authors also proposed Dedicated Protection for VNE (DRONE), a suite of solutions 1+1 ProViNE (namely, OPT-DRONE - an ILP formulation -, and FAST-DRONE - a heuristic solution), which guarantees a VN to survive under a single physical node failure.

The basic idea in 1+1 ProViNE is, given a SN $G$ and a VN $G'$, ensure the location constraints of the virtual nodes from $G'$ and find two disjoint embeddings of the nodes and links of $G'$ on $G$ such that nodes of $G'$ have two disjoint embeddings on $G$ and links of $G'$ have two disjoint embedding paths on $G$. Figure 2.10 shows an embedding result of this solution.




**(a)** Substrate Network

**(b)** Virtual Network Request

**(c)** Embedding result with DRONE

**Figure 2.10:** Embedding result with DRONE algorithm. It is possible to observe that the working part of the algorithm is completely disjoint from the backup part, respecting at the same time the location constraints of the virtual nodes.

In our algorithm we consider failures in the substrate nodes and links, and also cloud outages. In order to tolerate these failures we opted for a backup scheme where the working and the backup embeddings are disjoint. This technique avoids the possibility of a failure in the working embedding of a VN also affecting its backup embedding. Our backup scheme guarantees that nodes always have a backup if any failure occurs, in contrast to backup pooling schemes, where there is the possibility of a backup node not being available when a failure occurs.

#### 2.2.2.3 Quality of Service

Satisfying QoS requirements of an application in overlay mapping is also important. Thus, a key challenge related to overlay mapping lies in meeting link-related QoS requirements of an application. Most works assume unlimited hops of the substrate network in connecting each pair of virtual nodes. However, for applications having hop-related constraints (like latency), each hop in a path affects the performance of the application. Another key challenge is related to the violation of QoS constraints due to changes in the network behavior. While QoS requirements of an application can be satisfied at the time of boot strapping, the dynamic behavior of the underlying shared substrate makes it difficult to maintain the QoS requirements satisfied for the lifetime of the application.

In this sense, J. Shamsi and M. Brockmeyer [38] try to achieve two main goals in their work: to provide a solution for overlay mapping that satisfies stringent hop-related QoS requirements of an application under dynamic network conditions; and to devise an efficient scheme to improve overlay dependability against changing network conditions and to minimize overlay reconfiguration in case of QoS failures.

To this end, the authors describe QoSMap, an overlay mapping tool that utilizes link-specific QoS constraints from the application to compute high quality paths. It considers direct substrate paths in order to meet per hop stringent QoS requirements and indirect paths that serve as backup in case of QoS failure in the direct path. This algorithm can be utilized looking to many QoS constraints (and combinations of constraints) including latency, bandwidth, jitter, and congestion.

The idea of the algorithm is simple. From a list of unmapped overlay nodes, QosMap selects a node with highest degree requirements and finds all possible underlay nodes that fulfill the degree criteria. It then selects an underlay node based on three factors: number of backup routes a node can provide to the overlay mapped so far; whether the node is already included as an intermediate node for backup path; and the quality of a node (the quality is computed by calculating the average quality over all the egress and ingress paths). If at any moment, QoSMap cannot find an underlay node that meets the degree requirements of an overlay node, then it backtracks to the preceding level and selects a different underlay node for the preceding overlay node.

The work in [39] extends [38] by trying to achieve higher QoS and by utilizing fail-aware reconfiguration techniques in order to extend networks lifetime.

In [40], Zhang et al. propose an ILP model to solve the overlay mapping of both nodes and links with the dual goal of achieving QoS (achieve the best delay performance) and resilience performance through finding effective backup paths for overlay links. The requirements considered in this work are the location of substrate nodes, the QoS and the overlay topology.

The mapping process starts with a subset of substrate nodes being chosen as candi-

dates to host the overlay nodes of the request. Thus, a simplified logical topology with this subset is constructed to facilitate the mapping of the overlay request and reduce the computational complexity of the overlay mapping model. Then, the overlay nodes and links are mapped to the correspondent substrate resources using the objective function of the ILP model, which aims to minimize the delay of the mapped overlay and to minimize the number of substrate nodes that are used only as backup.

An enhanced ILP model is also presented to further provide effective resilience. To achieve this, two aspects are considered: the working and backup paths are chosen in such a manner that they avoid link overlap in the SN; and the minimization of overlaps will minimize the impact of a single substrate link failure on the mapped links of the VN. This enhanced ILP model has the same objectives as the previous one, plus the goal of maximizing the diversity between each pair of mapped overlay links, to reduce the chance of more than one overlay link being affected by a substrate link failure.

Although achieving QoS is not our main objective, in our work we try to embed the VNs with the minimum number of hops in the substrate paths. This way, minimizing the number of hops in substrate paths may reduce their effect on the performance of an application.

#### 2.2.2.4 Security

Most of the previous works focus on optimizing the use of resources with regard to performance and guaranteeing dependability. However, considering security in the VNE problem is also important and has not been investigated in depth so far.

In [41], Fischer et al. present a start position paper where they discuss the security problems in network virtualization. They propose additional constraints that should be considered for a secure VNE: a physical host attacking one of its VM; a VM attacking its physical host (known as VM outbreak); or a VM attacking another VM (side-channel attacks).

Solving these issues by installing additional software in the VMs or in the physical host is either difficult or impossible. An appropriate solution to these problems will, therefore, take into account the mapping of virtual resources onto physical machines, minimizing the risk exposure of both the VMs and the physical machines. A first step is to assign a security level and a security demand to both virtual and physical resources. Hence, the authors define three additional constraints that have to be considered for a secure VNE: a virtual resource should not be mapped on physical resources that have a lower security level than the security demand of the virtual resource; a physical resource should not be used to host virtual resources that have a lower security level than the security demand of the physical resource; and a virtual resource should not be co-hosted on the same physical resource together with another virtual resource having a lower security level than the se-

curity demand of the first resource. With the addition of these constraints, it is important to highlight that the run-time of a VNE algorithm to solve an embedding may increase.

In this work, only the security of virtual nodes is considered. Because links also suffer from security threats (e.g. adversaries can influence the physical links in a negative way, like replay attacks), Liu et al. [42] also take them into account. The authors define four security constraints: the three presented in [41] plus an additional one: a virtual link with a certain security demand should be hosted by a substrate path with an adequate security level. An example of a virtual network mapping with security constraints is presented in Figure 2.11. The embedding presented in the figure is possible because the security demands of the physical resources are covered by the security provided by the virtual resources, and the security demands of the virtual resources are covered by the security provided by the physical resources. As an example, virtual node 1 can be mapped onto substrate node A because the last one provides a security level higher than the security level demanded by virtual node 1 (L4 of A $>$ D3 of 1) and vice versa (L4 of 1 $>$ D2 of A). It could not be mapped, for instance, onto substrate node B because the security level provided by B is lower than the security demanded by 1 (L2 of B $<$ D3 of 1).

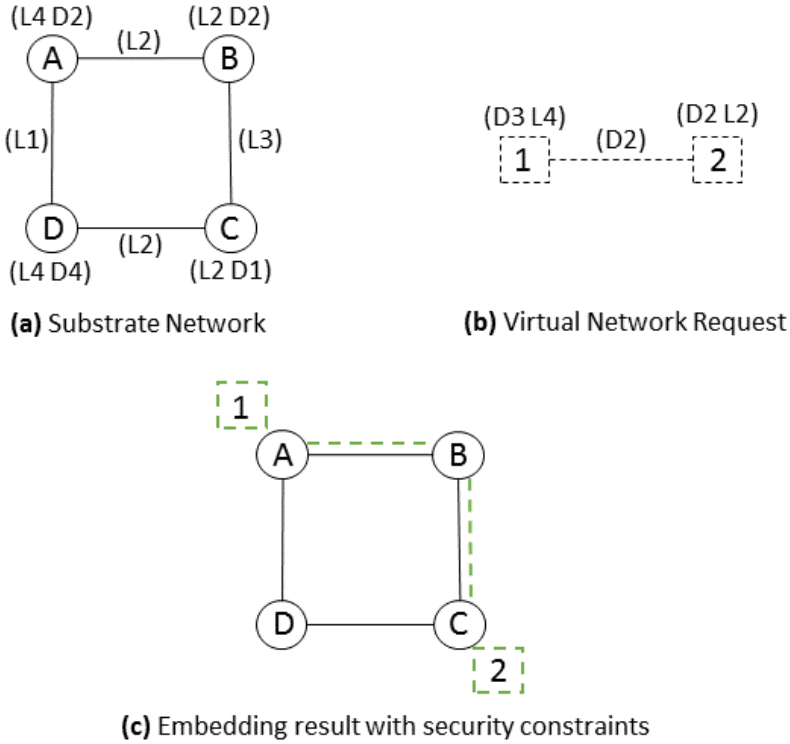



**Figure 2.11:** VNE taking into account security constraints. $D$ stands for security demand and $L$ stands for security provided. For a successful embedding, $L$ of substrate nodes should cover $D$ of virtual nodes, and $L$ of virtual nodes should cover $D$ of substrate nodes. For the links, it is only necessary to ensure that $L$ of a substrate path always cover $D$ of a virtual link.

In [42], the authors propose a two-stage security-aware virtual network algorithm. First, they design a heuristic to estimate for each substrate node its availability to host a

given virtual node. Based on this estimated value, they are able to sort the substrate nodes in a reasonable order.

The node mapping phase tries to properly embed all virtual nodes of a VNR to the substrate, while ensuring high revenue and low cost during the second stage of link mapping. The goal of the link mapping phase is to get the substrate path between virtual nodes with the lowest cost instead of smallest number of paths hops. To this end, the authors designed a Path Cost Coefficient that takes into account link security demand.

To be applied in real-time scenarios, dealing with new-coming requests, an algorithm framework is designed to be called once in every fixed time interval. First, the algorithm scans all of the online VNRs. Then, the requests are sorted in descending order of their revenues. At last, the sub-algorithms of both node and link mapping are called in turn, to try embedding the awaiting VNRs with the maximum revenue.

In [43], L. Bays et al. propose a VNE model that optimizes physical resource usage while meeting security requirements whenever feasible. The solution proposed by the authors is an ILP model that aims to minimize the physical bandwidth consumed by virtual links, thus minimizing cost and preserving bandwidth for future allocations.

In this model, each VNR has a set of security requirements associated with it, which aims to provide one of three distinct levels of confidentiality to communications within networks: end-to-end cryptography, where the end points of a VN must be mapped to physical routers that are able to provide this feature, i.e., these end points must support protocol suites such as IPSec; point-to-point cryptography, where packets are encrypted in their entirety, protecting both the payload and the header (the packets need to be decrypted and reencrypted on each hop in order to be properly routed); and non-overlapping networks, where a VNR may demand that its virtual routers and links do not share physical routers or paths with one or more other VNs. This case can be used to protect highly sensitive information from competitors.

The security guarantees provided by our algorithm are based in [41, 42], i.e., in defining security demands and security levels for the networks resources. These security levels are defined taking into account the different types of secure protocols that are normally used, e.g., the protocols presented in [43]. Our proposal is further extended by considering dependability and multiple clouds environments.

#### 2.2.2.5 Multiple Infrastructure providers

The majority of the VNE works focus only on a single InP. However, sometimes VNs must be provisioned across heterogeneous administrative domains belonging to multiple InPs to deploy and deliver services end to end.

In [44], M. Chowdhury et al. address the conflict of interests between SPs and InPs. On the one hand, each InP normally strives to optimize the allocation in its equipment

by getting requests with higher revenue while offloading unprofitable work onto their competitors. On the other hand, the SPs are interested in satisfying their demands while minimizing their expenditure. In this sense, the authors present PolyVINE – a policy-based end-to-end VNE framework – that embeds VNs across multiple InPs in a global manner while allowing each concerned InP to enforce its local policies at the same time. PolyVINE introduces a distributed protocol that coordinates the participating InPs and ensures competitive embedding pricing for the SPs. The VN assignment in inter-domain environment is decomposed into three major components: partitioning the VNR into $k$ subgraphs to be embedded onto $k$ SNs, establishing inter-connections between the $k$ subgraphs using inter-domain paths, and embedding each subgraph in each InP SN using an intra-domain algorithm.

From an InP point of view, there are three major stages in embedding each end-to-end VNR:

- Local embedding - An InP must decide whether to reject or accept a VNR. It can reject it in case of possible policy violations or if it fails to profitably embed any part of the request. To decide which part of a VNR to embed, the InP can use existing intra-domain VNE algorithms;

- Forwarding – If an InP only embeds part of a VNR, it has to send the rest of the request to other InPs. This is done through either recursive or iterative forwarding;

- Back-propagation – The VNR proceeds from one InP to the next, until there are no available InPs to send the request or the VNR has been satisfied. In case of success, a message carries back the embedding details and the corresponding price. At each step of this back-propagation, the sender InP can select mappings based on several criteria (e.g. lower price). As VNEs follow paths back to the SP, the prices are accumulated and the SP ends up with multiple choices.

Figure 2.12 shows a final result of embedding a VN through multiple InPs. After the execution of the embedding algorithm it is possible to observe that the nodes of a VN are mapped onto different InPs, and the virtual links are mapped onto physical paths that include inter and intra-domain links.

In [45], T. Lee et al. focus on the provisioning of computing resources, a management issue of cloud computing. They believe that an efficient provisioning algorithm is the first step to fully utilizing cloud computing infrastructures. In this sense, the authors introduce a graph clustering based resource provisioning algorithm to achieve their goal.

The provisioning algorithm is based on graph isomorphism detection, graph partition, and graph clustering algorithms. The graph isomorphism detection is used for the node and link mapping of VNs or partitioned VNs. Graph partitioning is used for partitioning the VN to reduce the provision cost. Graph clustering algorithms are used because the

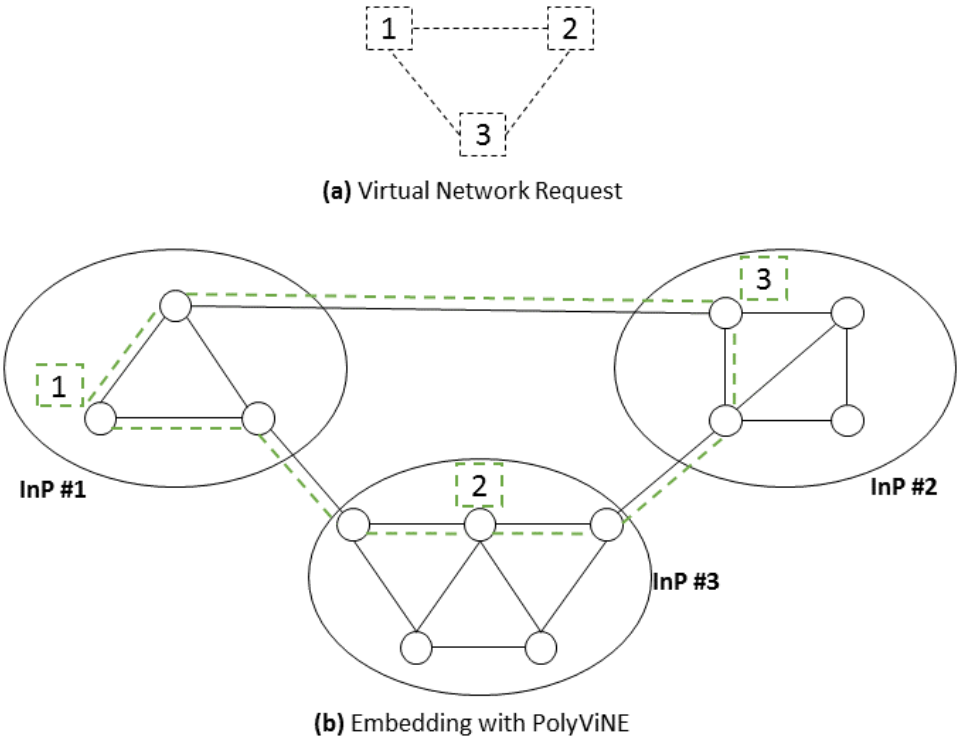



**Figure 2.12:** General embedding result when a multiple domain environment is considered.

user can ask some parts of its VN to be provisioned in different SNs, and the cost of provisioning partitioned VNs is cheaper than provisioning the entire VN as a whole.

In this work, after the SN manager analyze the VN provided by the VN customer and identify the partitioned sub-VNs, every domains manager runs the isomorphism detection algorithm to find a suitable mapping for each sub-VN, and computes a vector cost for embedding each sub-VN. Then, it is determined which domain manager embeds which sub-VN. Starting from the sub-VN that requires the most amount of resources, it is allocated to the SN with the least amount of available resources that can accommodate the request.

Besides the provisioning of computing resources, how to organize and discover virtual resources effectively is also an important research issue in network virtualization environment, especially across multiple InPs with conflicting goals. In [46], Lv et al. propose the design of a virtual resource organization and discovery framework. The proposed framework is composed by a Cluster Index Server (CIS) and local Management Nodes (MNs). The MN of each InP is responsible for maintaining and classifying the local virtual resources and it adopts a hierarchical conceptual clustering approach to organize this information. The CIS aggregates and organizes this information of multiple InPs, accordingly to the root attributes of the virtual resources.

Because assuming a single InP scenario is not very realistic, the authors also present a VNE scheme that considers multiple InPs based on the framework. Here, the VN user submits the VN description including the topology, resource attributes and QoS constraints

to the VNP. The VNP searches the virtual resource discovery framework with the VN description. Then, if the virtual resources returned by the framework belong to multiple InPs, the VNP should coordinate multiple InPs to implement the inter-InP virtual link embedding. Meanwhile the VNP delivers parts of the VNR and the corresponding information of the virtual resources candidates to different InPs. Each InP completes the intra-InP VNE in its administrative domain.

In [47], Leivadeas et al. explore inter-domain resource mapping in a networked cloud environment also through a hierarchical framework. To deal with the complexity and scalability of the resource mapping problem, the authors propose a request partitioning approach with the use of an Iterated Local Search meta-heuristic and a network cloud mapping approach. In the proposed hierarchical framework, the virtual resource mapping has two phases: a Request Partitioning Phase and an Embedding Phase.

In the Request Partitioning Phase, every cloud broker (entity responsible for providing cloud IaaS based on user's requirements) that receives an incoming request, groups the requested virtual resources according to their functional attributes, creating Virtual Resource Sets (VRSs), and sends them to cloud service providers. These entities match physical resources candidates to the received VRSs and respond to the cloud broker with a resource provisioning cost per VRS. The cost is related to resource availability in the cloud. The cloud broker calculates the most-effective request partitioning and sends the corresponding partial requests to the selected cloud service providers for further processing. The Embedding Phase is the phase where the actual mapping takes place. Here, upon receiving a partial request, the cloud service provider embeds it to its substrate resources using an appropriate intra-domain VNE algorithm.

The authors introduce the networked cloud mapping algorithm as an intra-domain VNE algorithm. Here, the physical network graph is augmented by introducing one pseudo node for each virtual node and connecting it to all the physical nodes. With the substrate augmented, a MILP solution is solved to find an appropriate embedding to the virtual nodes. Once the node mapping has been successfully completed, the links mapping is solved as a multi-commodity flow problem. Alternatively, a shortest path algorithm can be applied.

In this section we presented several works that propose solutions for the VNE problem. The algorithms normally try to achieve different objectives: some try to minimize the costs of embedding, others try to embed VNs in a way to minimize the energy spent in the SN, among other objectives. Our proposal differentiates from these works by the fact that it provides security and dependability together, and further extend these properties to multi-cloud environments.

# Chapter 3 - Secure and Dependable VNE

The analysis to the state of the art of VNE presented before shows that there is a good amount of works focusing on dependability. However, there is still a lack of works about secure VNE. We believe we are the firsts putting together dependability and security, and extending them to a multiple clouds environment in VNE.

In this chapter, we define the secure and dependable VNE problem (Section 3.1), we describe what are the attributes that define a VN and a SN (Section 3.2), and we explain the MILP formulation we propose (Section 3.3). Finally, we explain the simple API we developed to be used as a module of the multi-tenant multi-datacenters network virtualization platform described in [3].

## 3.1 Problem Description

The multi-cloud environment considered in our work increases the flexibility, dependability, and security of the network virtualization solution. This increase in the options offered to users makes the problem different from the VNE problems considered to this date. In our environment, when a user wants to instantiate a VN, besides the processing capacity, i.e., CPU, for its nodes and the bandwidth resources for its links, it may also include as requirements security and dependability demands for nodes and links. To fulfill these requirements, we give users the possibility to choose specific security and dependability levels for its VN (namely, to its virtual nodes and virtual links). Figure 3.1 gives an example of typical security and dependability levels that can be chosen by the user for its virtual resources.

In the example, virtual nodes can be embedded onto machines with three different types of security: normal containers[1], normal VMs, or secure VMs (e.g., VMs that include trusted components, such as a TPM[2]). Users can also choose the security requirements for their virtual links: default security, an intermediate security level (e.g., where authenticity and integrity are guaranteed), or a higher security level (e.g., where authenticity, integrity, and confidentiality are guaranteed). Finally, the user can choose the type of cloud where

[1]The difference between virtualization and containerization is that containers have applications isolated directly on top of the host operating system. In virtualization, there is an hypervisor to guarantee isolation

[2]TPM is a specialized chip on an endpoint device that stores encryption keys specific to the host system for hardware authentication.

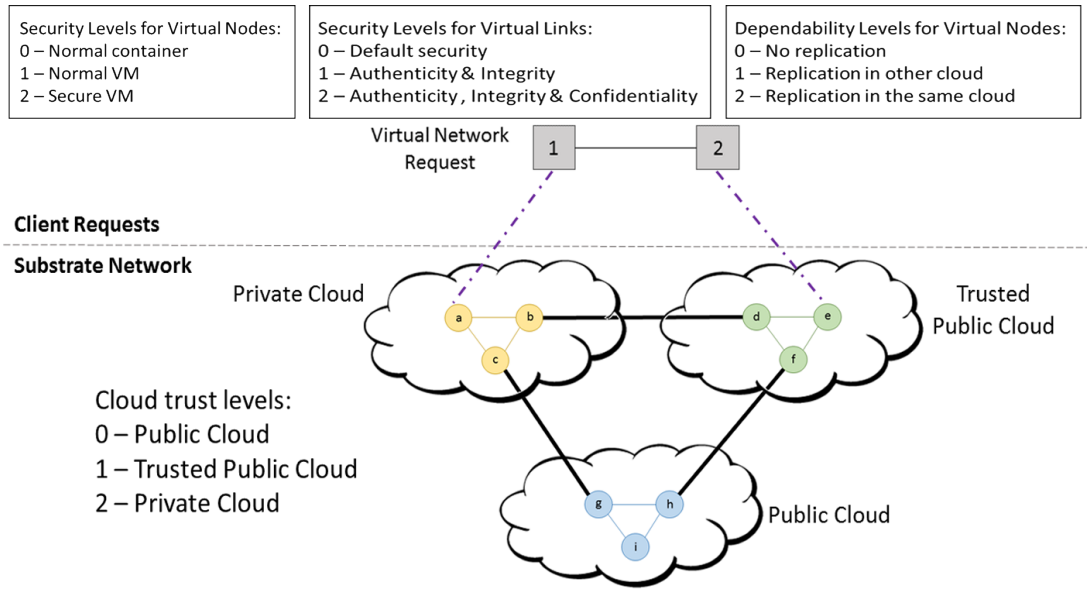



**Figure 3.1:** Possible levels of security and dependability the user can choose for its virtual network.

they want their virtual nodes to be located. In the example, we define three types of clouds: public clouds (belonging to cloud providers), trusted public clouds (belonging to cloud providers that are considered more trustworthy), and private clouds (belonging to the tenant, assumed to be the most secure option). With these options, the user may choose private clouds or trusted public clouds for more sensitive workloads, while leaving the others in public clouds, to scale out.

Besides security, tenants can also require backups for their virtual resources, for fault-tolerance. Providing dependability is important in the sense that many failures of devices and links occur every day in datacenters and repair times are unpredictable [48]. To ensure dependability additional physical resources need to be allocated in the SN to the user's VN. In addition, when replication is required, tenants can choose the location of each backup node, further improving dependability. To avoid cloud outages (caused by a natural disaster or a malicious attack), they may choose replication for virtual nodes in different clouds. Note that when dependability is required substrate paths between the backup nodes of a VN need to be set up.

With the features of our solution described, we now define the Secure and Dependable Virtual Network Embedding (SecDep VNE) problem:

> Given the virtual network $G^V$ with resources requested and corresponding security and dependability requirements, and substrate network $G^S$ with resources to serve incoming VNRs, can $G^V$ be mapped to the substrate network with the minimum resources while satisfying the following? (i) Each virtual node and link is mapped to the sub-

strate network meeting the CPU capacity and bandwidth constraints, respectively, and also security and dependability constraints, namely node security type, node location, node backup, and link security type; (ii) Each virtual node is mapped to a substrate node that is located in a cloud that covers its cloud type requirements; (iii) The virtual network is protected against faults in the substrate network or cloud outages, when backups are required by the user.

Our model handles the SecDep VNE problem, trying to map a VN onto a SN while respecting all the requirements and constraints. When a VNR arrives, our solution tries to find the best mapping for the VN while reducing the costs of embedding it (i.e., reduce the total quantity of substrate resources allocated to it). If there is no possible solution to embed the incoming VN, then the VN is rejected. When a solution is found, the quantity of resources demanded by the VN is allocated. In our formulation, after the embedding of a VN, the SN is augmented with the virtual nodes that were embedded. These virtual nodes have meta-links with the substrate nodes on which they are mapped. The meta-links allow a certain virtual node to be mapped onto a certain substrate node. In Figure 3.2(c), we illustrate the result of embedding the VNR presented in 3.2(b) onto the SN presented in 3.2(a) (admitting that all the constraints are guaranteed). Virtual node 1 has a meta-link with substrate node A, and virtual node 2 has one with E, which are their primary nodes. They also have meta-links with substrate nodes B and D, the backup nodes that are used when a failure occurs. In this figure it is also possible to observe that the substrate paths (both working and backup) correspond to more than one substrate link (e.g., the substrate links (A,F) and (F,E) are assigned to the working path).

## 3.2 Network Model

In this section we describe the characteristics and attributes that define a physical and a virtual network. More precisely, we describe each attribute of the resources of the networks (its nodes and links).

### 3.2.1 Substrate Network

We model the substrate network as a weighted undirected graph. It is denoted by $G^S = (N^S, E^S, A_N^S, A_E^S)$ , where $N^S$ is the set of substrate nodes, $E^S$ is the set of substrate links, $A_N^S$ is the set of attributes of substrate nodes, and $A_E^S$ is the set of attributes of substrate links.

$A_N^S$ contains the following attributes for substrate nodes:

$$A_N^S = \{\{cpu^S(n), sec^S(n), cloud^S(n)\} | n \epsilon N^S\}$$

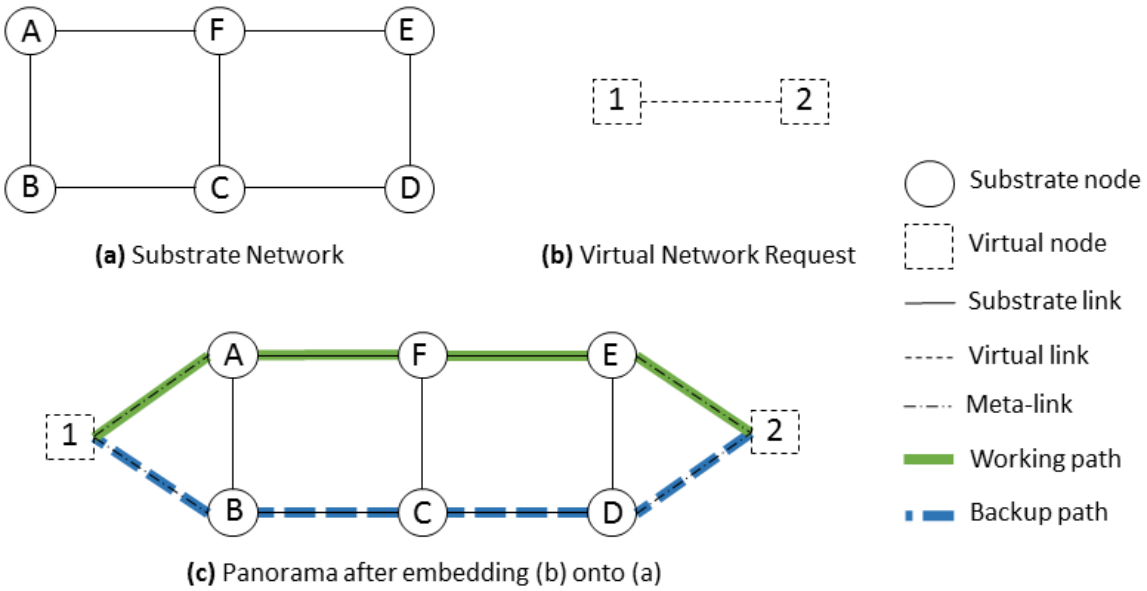



**Figure 3.2:** (a) Simple substrate network. (b) Simple virtual network request that arrives to be embedded. (c) Example of an embedding result of (b) onto (a) after the execution of our MILP formulation (supposing that all constraints are achieved - resource capacity, security, cloud type and dependability constraints).

- $cpu^S(n)$ - Total CPU capacity of the substrate node *n*. This attribute can take any values greater or equal to 0;
- $sec^S(n)$ - Security provided by the substrate node *n*. This attribute can take any positive value (including zero). In the example given in Figure 3.1, if substrate node *n* is a normal container, then $sec^S(n) = 0$. If it is a normal VM, $sec^S(n)$ = 1. Lastly, if *n* is a secure VM, then $sec^S(n) = 2$;
- $cloud^S(n)$ - Defines in which type of cloud the substrate node *n* is located. This attribute can take any positive value (including zero). In the example, if *n* is located in a public cloud, then $cloud^S(n)$ = 0. If it is located in a trusted public cloud, then $cloud^S(n)$ = 1. Lastly, if *n* is located at a private cloud, then $cloud^S(n)$ = 2.

$A_E^S$ contains the following attributes for substrate links:

$$A_E^S = \{\{bw^S(l), sec^S(l)\} | l \epsilon E^S\}$$

- $bw^S(l)$ - Total bandwidth capacity of the substrate link *l*. This attribute can take any values greater or equal to 0;
- $sec^S(l)$ - Security provided by the substrate link *l*. This attribute can take any positive value (including zero). In the example, $sec^S(l) = 0$ if substrate link *l* only provides simple security mechanisms defined as default. If link *l* supports protocols

that provide authenticity and integrity guarantees, then $sec^S(l) = 1$. Lastly, if $l$ supports the use of protocols that provide authenticity, integrity and confidentiality guarantees, then $sec^S(l) = 2$.

Figure 3.3 shows an example of a SN. Substrate nodes A and D are in the same cloud (a trusted public cloud) and B and C are in a public cloud. Links (D,A) and (B,C) are intra-domain links; links (A,B) and (C,D) are inter-domain links. It is also possible to observe the diversity of the resource attributes. Some nodes are more secure (e.g., node A, a secure VM), some have weaker computing or bandwidth capacities, such as node C or link (A,B).

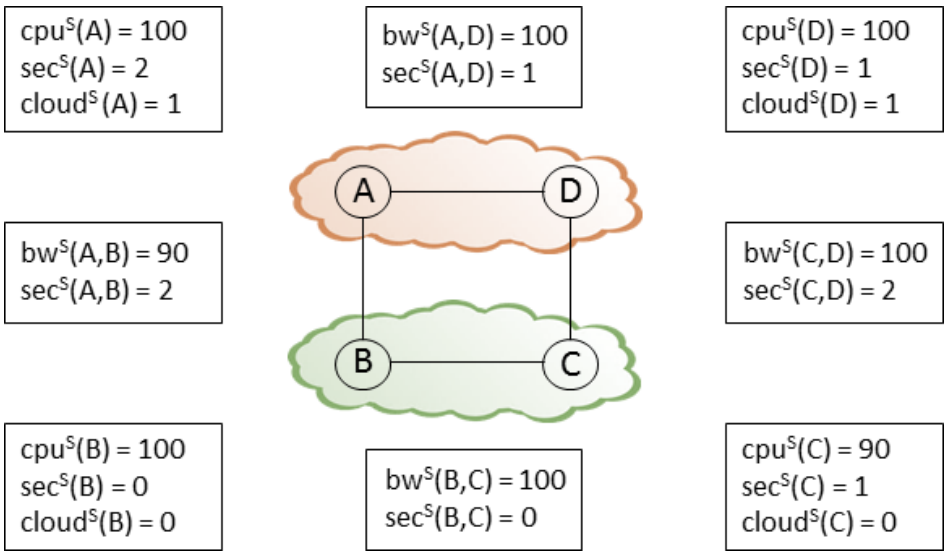



**Figure 3.3:** Example of a SN $G^S$ where $N^S = \{A, B, C, D\}$ and $E^S = \{(A, B), (B, C), (C, D), (D, A)\}$. The sets $A_N^S$ and $A_E^S$ (i.e., the attributes of the substrate resources) are presented in the figure.

### 3.2.2 Virtual Network Requests

VNRs are defined by the clients of the system. Similar to the substrate network, the VNRs are also modeled as weighted undirected graphs. Each virtual network request is denoted by $G^V = (N^V, E^V, Time^V, Dur^V, A_N^V, A_E^V)$, where $N^V$ is the set of virtual nodes, $E^V$ is the set of virtual links, $Time^V$ is the arrival time of the VNR, $Dur^V$ is the time period for which the VN is valid, $A_N^V$ is the set of attributes of substrate nodes, and $A_E^V$ is the set of attributes of substrate links.

$A_N^V$ contains the following attributes demanded by virtual nodes:

$$A_N^V = \{\{cpu^V(n), sec^V(n), cloud^V(n), dep^V(n)\} | n \epsilon N^V\}$$

- $cpu^V(n)$ - CPU capacity demanded by the virtual node *n*. This attribute can take any value greater than 0;

- $sec^V(n)$ - Security demanded by the virtual node *n*. This attribute can take any positive value (including zero), similar to the SN case;
- $cloud^V(n)$ - Defines the type of cloud on which the virtual node *n* should be mapped. This attribute can take any positive value (including zero), similar to the SN case;
- $dep^V(n)$ - Defines where the backup of virtual node *n* should be mapped. This attribute can take any positive value (including zero). Considering the same example, if replication is not needed for the VNR , then $dep^V(n)$ = 0. If virtual node *n* should have a backup located in other cloud (e.g., in order to survive to a cloud outage), then $dep^V(n)$ = 1. Finally, if *n* should have a backup in the same cloud, then $dep^V(n)$ = 2.

$A_E^V$ contains the following attributes demanded by virtual links:

$$A_E^V = \{\{bw^V(l), sec^V(l)\}|l\epsilon E^V\}$$

- $bw^V(l)$ - Bandwidth capacity demanded by the virtual link *l*. This attribute can take any value greater than 0;
- $sec^V(l)$ - Security demanded by the virtual link *l*. This attribute can take any positive value (including zero), again similar to the SN case.

Figure 3.4 shows an example of a VNR. Similar to Figure 3.3, here it is also possible to observe the diversity of the virtual resources. The user requires virtual node 1 to be mapped onto a substrate node that is located in a trusted public cloud. In addition, it requires its backup to be located in another cloud with the same level of security. Virtual node 2 can be mapped onto a substrate node that is located in a public cloud, and its backup can be located in that same cloud. Virtual link (1,2) can be mapped onto one or more substrate links, but requires all constituent links to have an equal or greater security level than the one required. In this case, the physical links where virtual link (1,2) will be mapped should have $sec^S = 1$ at least.

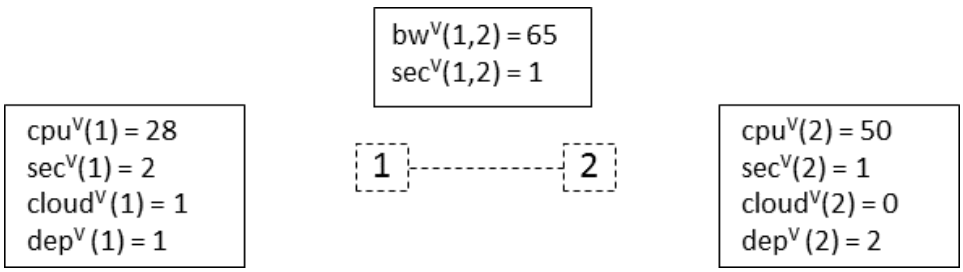



**Figure 3.4:** Example of a VN $G^V$ where $N^V = \{1, 2\}$ and $E^V = \{(1, 2)\}$. The sets $A_N^V$ and $A_E^V$, i.e., the demands of virtual nodes and links, are presented in the figure.

It is important to note that substrate resources with higher security (or dependability) levels can map virtual resources that have lower or equal security level demands. For instance, a substrate node with $sec^S = 2$ and $cloud^S = 2$ (located in a private cloud) can map either one of the virtual nodes presented in Figure 3.4, since it provides the highest level of security demanded by virtual nodes.

### 3.2.3 Measurement of Substrate Network Resources

The residual capacity (or available capacity) of a substrate node, $R_N(n^S)$, is defined as the available CPU capacity of the substrate node $n^S \epsilon N^S$.

$$R_N(n^S) = cpu^S(n^S) - \sum_{\forall n^V \uparrow n^S} cpu^V(n^V),$$

where $n^V \epsilon N^V$ and $x \uparrow y$ denotes that the virtual node *x* is hosted on the substrate node *y*.

Similarly, the residual capacity of a substrate link, $R_E(e^S)$, is defined as the total amount of bandwidth available on the substrate link $e^S \epsilon E^S$.

$$R_E(e^S) = bw^S(e^S) - \sum_{\forall e^V \uparrow e^S} bw^V(e^V),$$

where $e^V \epsilon E^V$ and $x \uparrow y$ denotes that the flow of the virtual link *x* traverses the substrate link *y*.

Since a virtual link can be mapped onto multiple substrate links, i.e., mapped onto a substrate path, it is also important to define the available bandwidth capacity of a substrate path *P*. This can be defined as the minimum available bandwidth among all the substrate links belonging to *P*.

$$R_E(P) = \min_{e^S \epsilon P} R_E(e^S)$$

### 3.2.4 Objectives

The main goal of VNE is to maximize the profit of the provider. For this purpose, and similar to [22, 4], we define the revenue of accepting a VNR as:

$$\mathbb{R}(G^V) = \lambda_1 \sum_{e^V \epsilon E^V} bw^V(e^V)\ sec^V(e^V) + \lambda_2 \sum_{n^V \epsilon N^V} cpu^V(n^V)\ sec^V(n^V)\ cloud^V(n^V),$$

where $\lambda_1$ and $\lambda_2$ are weights that denote the relative proportion of each revenue component to the total revenue. We will address these weights later in Section 3.3.2. Although the revenue give us an idea of how much an InP will gain by accepting a certain VNR, it is

not useful if we do not know the cost the InP will incur for embedding that request. Therefore, it is also necessary to define the cost. The cost of embedding a VNR can be defined as the sum of total substrate resources allocated to that VN. Normally, the numerical cost of embedding a VNR is equal or higher than the revenue generated by that request. This is due to the fact that virtual links may be embedded to one or more physical links. In our work, the cost may also increase if the VNR requires higher security or dependability for its virtual nodes and links. We define the cost of embedding a VNR as:

$$\mathbb{C}(G^V) = \lambda_1 \sum_{e^V \epsilon E^V} \sum_{e^S \epsilon E^S} f_{e^S}^{e^V} \; sec^S(e^S) + \lambda_2 \sum_{n^V \epsilon N^V} \sum_{n^S \epsilon N^S} cpu_{n^S}^{n^V} \; sec^S(n^S) \; cloud^S(n^S),$$

where $f_{e^S}^{e^V}$ denotes the total amount of bandwidth allocated on the substrate link $e^S$ for virtual link $e^V$ and $cpu_{n^S}^{n^V}$ denotes the total amount of CPU allocated on the substrate node $n^S$ for virtual node $n^V$ (either working or backup parts). $\lambda_1$ and $\lambda_2$ are weights that denote the relative proportion of each cost component to the total cost.

## 3.3 MILP Formulation

We have developed a MILP formulation to solve the SecDep VNE problem. In this section we start by explaining the variables used in our formulation, the objective function, and finally the constraints that were defined to solve the problem.

### 3.3.1 Variables

In Table 3.1 the variables that are used in our MILP formulation are showed (and explained). Briefly, $wf_{u,v}^{i,j}$, $bf_{u,v}^{i,j}$, $wl_{u,v}^{i,j}$, $bl_{u,v}^{i,j}$ and $rl_{u,v}$ are related to working and backup links; $wn_{i,v}$, $bn_{i,v}$ and $rn_v$ are related to working and backup nodes; $wc_{i,c}$ and $bc_{i,c}$ are related to the embedding location of virtual nodes.

| Symbol | Meaning |
|---|---|
| $wf_{u,v}^{i,j}$ | The amount of working flow, i.e., bandwidth, on the physical link (*u*,*v*) for the virtual link (*i*,*j*) |
| $bf_{u,v}^{i,j}$ | The amount of backup flow, i.e., backup bandwidth, on the physical link (*u*,*v*) for the virtual link (*i*,*j*) |
| $wl_{u,v}^{i,j}$ | Denotes whether the virtual link (*i*,*j*) is mapped onto the physical link (*u*,*v*). (1 if (*i*,*j*) is mapped on (*u*,*v*), 0 otherwise) |
| $bl_{u,v}^{i,j}$ | Denotes whether the backup of virtual link (*i*,*j*) is mapped onto the physical link (*u*,*v*). (1 if backup of (*i*,*j*) is mapped on (*u*,*v*), 0 otherwise) |
| $rl_{u,v}$ | The reserved backup resources on a physical link (*u*,*v*), i.e., the total quantity of bandwidth that is allocated for backup flows. |
| $wn_{i,v}$ | Denotes whether virtual node *i* is mapped onto the physical node *v*. (1 if *i* is mapped on *v*, 0 otherwise) |
| $bn_{i,v}$ | Denotes whether virtual node *i*'s backup is mapped onto the physical node *v*. (1 if *i*'s backup is mapped on *v*, 0 otherwise) |
| $rn_v$ | The reserved backup resource on a physical node *v*, i.e., the total quantity of CPU that is allocated to backups. |
| $wc_{i,c}$ | Denotes whether virtual node *i* is mapped on cloud *c*. (1 if *i* is mapped on *c*, 0 otherwise) |
| $bc_{i,c}$ | Denotes whether virtual node *i*'s backup is mapped on cloud *c*. (1 if *i*'s backup is mapped on *c*, 0 otherwise) |

**Table 3.1:** List of all the variables used in our MILP formulation.

### 3.3.2 Objective Function

The objective function of our formulation has three goals: to minimize 1) the sum of all computing costs, 2) the sum of all communication costs, and 3) the overall number of hops of the substrate paths for the virtual links.

Since we have different objectives, and these objectives are measured in different units, we have to unify them. Thus, in our formulation we consider a weighted sum function with three different weights, $\lambda_1$, $\lambda_2$, and $\lambda_3$, which should be reasonably parameterized for each objective, in order to mitigate the differences between the units.

The first and the second parts of Eq. 3.1 are the sum of all working and backup link bandwidth costs, respectively. The third and the fourth parts are the sum of all working and backup computing node costs. In this function, the level of security provided by the physical resources is considered. The parameter $\alpha$ is a weight for each physical link that assumes some value defined previously and that depends if (*u*,*v*) is an inter-cloud connection (link between two clouds) or intra-domain link (link inside a cloud). This is due to the expectation that virtual links that use links connecting two clouds (inter-domain links) to have higher cost (monetary, delay, or other). Also, mapping a VN onto substrate resources that provide higher security or dependability requirements are expected to increase costs. The fifth and last parts of the objective function achieve the third goal presented above.

$$
\begin{aligned}
min \quad & \lambda_1 \sum_{(i,j)\epsilon E^V} \sum_{u,v\epsilon N^S} \alpha_{u,v} \quad wf_{u,v}^{i,j} \quad sec^S(u,v) \\
& + \lambda_1 \sum_{u,v\epsilon N^S} rl_{u,v} \quad sec^S(u,v) \\
& + \lambda_2 \sum_{i\epsilon N^V} \sum_{v\epsilon N^S} cpu^V(v) \quad wn_{i,v} \quad sec^S(v) \quad cloud^S(v) \\
& + \lambda_2 \sum_{v\epsilon N^S} rn_v \quad sec^S(v) \quad cloud^S(v) \\
& + \lambda_3 \sum_{(i,j)\epsilon E^V} \sum_{u,v\epsilon N^S} wl_{u,v}^{i,j} \\
& + \lambda_3 \sum_{(i,j)\epsilon E^V} \sum_{u,v\epsilon N^S} bl_{u,v}^{i,j} \qquad (3.1)
\end{aligned}
$$

Intuitively, with our formulation when a VNR arrives to our system the embedder will try to match the resources to requests in such a way that it saves the more "powerful" resources (e.g., those with higher security levels) to the virtual resources that require them explicitly. For instance, virtual nodes with $sec^V = 1$ will be mapped onto substrate nodes with $sec^S = 2$ if and only if there are no substrate nodes with $sec^S = 1$ available.

### 3.3.3 Typical Constraints

In this section we define the constraints typically defined in most VNE MILP formulations.

**Domain Constraints** The following constraints define the values space for each variable defined in our MILP formulation.

$$wf_{u,v}^{i,j} \geq 0, \forall u,v\epsilon N^S, (i,j)\epsilon E^V \qquad (3.2)$$
$$bf_{u,v}^{i,j} \geq 0, \forall u,v\epsilon N^S, (i,j)\epsilon E^V \qquad (3.3)$$

$$wl_{u,v}^{i,j}\epsilon \{0,1\}, \forall u,v\epsilon N^S, (i,j)\epsilon E^V \qquad (3.4)$$
$$bl_{u,v}^{i,j}\epsilon \{0,1\}, \forall u,v\epsilon N^S, (i,j)\epsilon E^V \qquad (3.5)$$

$$wn_{i,v}\epsilon \{0,1\}, \forall i\epsilon N^V, v\epsilon N^S \qquad (3.6)$$
$$bn_{i,v}\epsilon \{0,1\}, \forall i\epsilon N^V, v\epsilon N^S \qquad (3.7)$$

$$rl_{u,v} \geq 0, \forall u,v\epsilon N^S \qquad (3.8)$$
$$rn_v \geq 0, \forall v\epsilon N^S \qquad (3.9)$$

$$wc_{i,c} \epsilon \{0,1\}, \forall i \epsilon N^V, c \epsilon C \tag{3.10}$$

$$bc_{i,c} \epsilon \{0,1\}, \forall i \epsilon N^V, c \epsilon C \tag{3.11}$$

Eq. 3.2 and 3.3 ensure that the bandwidth allocated for a virtual link (i,j) in a substrate link (u,v), either for the working or backup parts, never takes negative values.

Eq. 3.4, 3.5, 3.6 and 3.7 ensure, respectively, that the variables $wl_{u,v}^{i,j}$, $bl_{u,v}^{i,j}$, $wn_{u,v}$ and $bn_{u,v}$ take either the value 0 or 1.

Eq. 3.8 and 3.9 ensure that the bandwidth reserved for backup on substrate links and the capacity (e.g., CPU) reserved on substrate nodes, also for backup, never take negative values.

Eq. 3.10 and 3.11 ensure variables $wc_{i,c}$ and $bc_{i,c}$ only take the value 0 or 1.

**Note:** The set C present in these restrictions (and later) is the set of existent clouds.

**Link Mapping for Working Traffic** Constraints 3.12, 3.13 and 3.14 refer to the working flow conservation conditions, which denote that the network flow to a node is zero, except for the source node and the sink node, respectively (i.e., no flow appears or disappears in any node, unless it is a source or a sink node).

$$\sum_{u \epsilon N^S \cup N^V} wf_{u,v}^{i,j} - \sum_{u \epsilon N^S \cup N^V} wf_{v,u}^{i,j} = 0, \forall (i,j) \epsilon E^V, v \epsilon N^S \tag{3.12}$$

$$\sum_{v \epsilon N^S} wf_{i,v}^{i,j} - \sum_{v \epsilon N^S} wf_{v,i}^{i,j} = bw^V(i,j), \forall (i,j) \epsilon E^V \tag{3.13}$$

$$\sum_{v \epsilon N^S} wf_{j,v}^{i,j} - \sum_{v \epsilon N^S} wf_{v,j}^{i,j} = -bw^V(i,j), \forall (i,j) \epsilon E^V \tag{3.14}$$

Eq. 3.15 and 3.16 guarantee that the working flow of a virtual link $(i,j)$ always departs from the correspondent working node of *i* and arrives to the correspondent working node of *j*.

$$wn_{i,v} \quad bw^V(i,j) = wf_{i,v}^{i,j}, \forall v \epsilon N^S, (i,j) \epsilon E^V \tag{3.15}$$

$$wn_{j,v} \quad bw^V(i,j) = wf_{v,j}^{i,j}, \forall v \epsilon N^S, (i,j) \epsilon E^V \tag{3.16}$$

Figure 3.5 shows an example of how these constraints contribute to our formulation. In short, they define the values of variables $wf$. Constraints 3.12, 3.13 and 3.14 are responsible for $wf_{D,A}^{1,2} = 4$, i.e., they are responsible for the working flows in the substrate network. Eq. 3.15 and 3.16 are responsible for the working flows on meta-links. Supposing that virtual nodes 1 and 2 of a VNR are mapped onto substrate nodes A and D, respectively, we have $wn_{1,A} = 1$ and $wn_{2,D} = 1$. Therefore, $wf_{1,A}^{1,2} = 4$ and $wf_{2,D}^{1,2} = 4$. All other $wf$ and $wn$ variables, such as $wf_{1,B}^{1,2}$ or $wn_{2,C}$ are equal to 0.

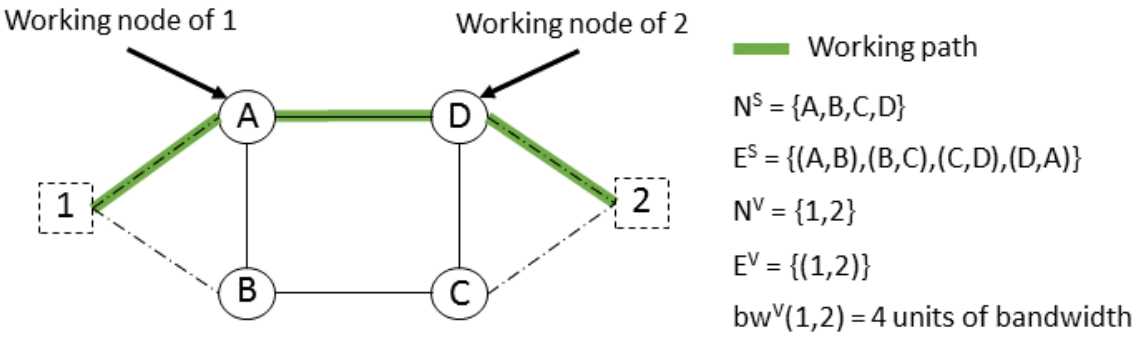



**Figure 3.5:** Example of how variables $wf$ contribute to our formulation.

**Node Capacity Constraints** Substrate nodes can map nodes from different VNRs simultaneously. They can be the correspondent working node of a virtual node $i$ from a VNR $x$ and simultaneously be the correspondent backup node of a virtual node $j$ from a VNR $y$. Considering this, for a substrate node, the total allocated capacity depends on the total capacity that is allocated for working nodes, plus the total capacity that is allocated for backup nodes, which should be less than the current capacity of the substrate node. This is represented by Eq. 3.17 and 3.18.

$$\sum_{u \epsilon N^V} bn_{u,v} \quad cpu^V(u) \leq rn_v, \forall v \epsilon N^S \tag{3.17}$$

$$\sum_{u \epsilon N^V} wn_{u,v} \quad cpu^V(u) + rn_v \leq R_N(v), \forall v \epsilon N^S \tag{3.18}$$

**Link Capacity Constraints** Like substrate nodes, substrate links also can map virtual links from different VNRs simultaneously. Eq. 3.19 and 3.20 define the allocated link capacity of a substrate link as the sum of the capacity allocated for the active flows and the reserved resources for backup. The allocated capacity of a substrate link should be less than the residual capacity of that physical link.

$$\sum_{(i,j) \epsilon E^V} (bf_{u,v}^{i,j} + bf_{v,u}^{i,j}) \leq rl_{u,v}, \forall u, v \epsilon N^S \tag{3.19}$$

$$\sum_{(i,j) \epsilon E^V} (wf_{u,v}^{i,j} + wf_{v,u}^{i,j}) + rl_{u,v} \leq R_E(u,v), \forall u, v \epsilon N^S \tag{3.20}$$

Figure 3.6 shows an example of how this set of restrictions works. Taking into account only CPU and bandwidth resources as attributes, we can observe in 3.6(c) that nodes A, B, C and D, and links (A,D) and (B,C), are sharing their capacity to two different VNs at the same time. Node A is the working node of 100 and the backup node of 1, node B is the working node of 1 and the backup of 100, node C is the working node of 2 and the backup of 99, and node D is the working node of 99 and the backup of 2. Link (A,D)

carries the data that goes from 99 to 100 and is the backup link for (1,2), i.e., if a failure occurs in B, C or (B,C) the data that goes from 1 to 2 will pass through (A,D). Link (B,C) is the working link for the virtual link (1,2) and the backup for the virtual link (99,100). This is possible because the substrate resources have sufficient capacity to map multiple virtual resources from different VNRs.

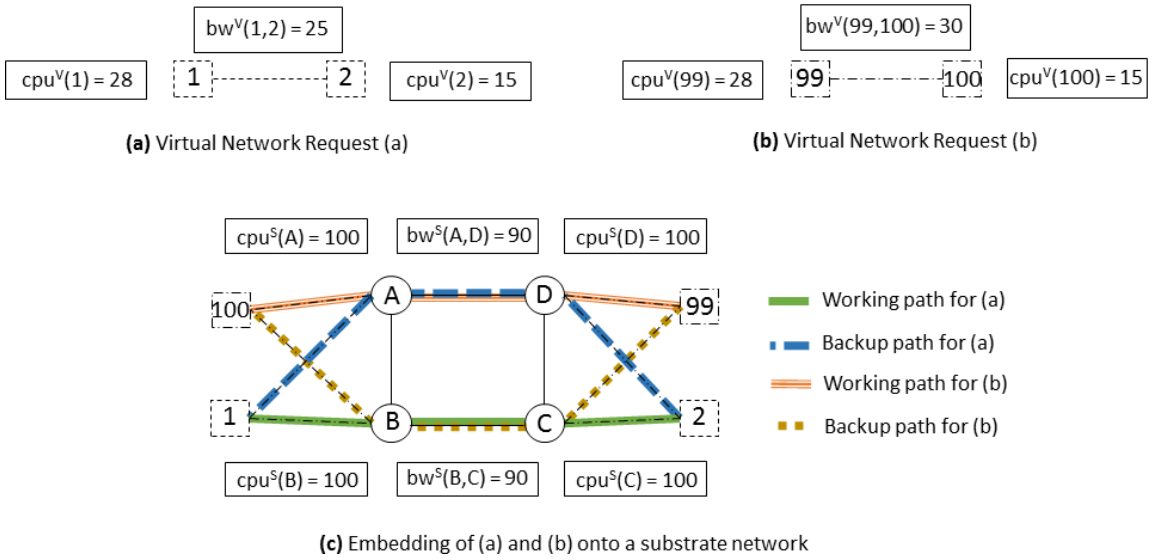



**Figure 3.6:** Allocation of substrate resources from a SN in (c) to multiple VNRs ((a) and (b)).

### 3.3.4 Security Constraints

Our VNE formulation includes security constraints, for both the nodes, links, and clouds.

**Node Security Constraints** The security restrictions for nodes are defined as:

$$wn_{u,v} \quad sec^V(u) \leq sec^S(v), \forall u \epsilon N^V, v \epsilon N^S \tag{3.21}$$

$$bn_{u,v} \quad sec^V(u) \leq sec^S(v), \forall u \epsilon N^V, v \epsilon N^S \tag{3.22}$$

Eq. 3.21 guarantees that a virtual node $u$ is only mapped to a physical node that has an equal or higher security level than $u$'s security demand.

Eq. 3.22 ensure the same as the previous one, but for backup nodes.

This ensures, returning to our initial example, that a secure VM from the physical infrastructure can map virtual nodes requesting normal containers, VMs, or secure VMs, whereas a physical container can only map virtual nodes that are looking for normal containers.

**Link Security Constraints** The following equations are related with the working and the backup link security constraints:

$$wl_{u,v}^{i,j} \quad sec^V(i,j) \leq sec^S(u,v), \forall (i,j) \epsilon E^V, u, v \epsilon N^S \tag{3.23}$$

$$bl_{u,v}^{i,j} \quad sec^V(i,j) \leq sec^S(u,v), \forall (i,j) \epsilon E^V, u, v \epsilon N^S \tag{3.24}$$

Here, it is necessary to ensure that each virtual link is mapped to one or more physical links that provide a security level equal or higher than the security demand of the virtual link. Similarly to the previous case, a physical link that provides default security can only map virtual links that demand for that low level of security, while a physical link that provides authenticity, integrity and confidentiality guarantees can map virtual links with any security demand.

**Cloud Security Constraints** Eq. 3.25 ensures that a virtual node $u$ is mapped to a certain physical node $v$ only if the cloud where $v$ is located is of a type of equal or higher security than the type of cloud demanded by node $u$. For instance, a virtual node that requires to be mapped on a public cloud may be mapped to either a public, a trusted public or a private cloud, considering again our example. On the other side, a virtual node that requires the highest level of security can only be mapped to nodes located in a private cloud. Eq. 3.26 guarantees the same for the backup nodes.

$$wn_{u,v} \quad cloud^V(u) \leq cloud^S(v), \forall u \epsilon N^V, v \epsilon N^S \tag{3.25}$$

$$bn_{u,v} \quad cloud^V(u) \leq cloud^S(v), \forall u \epsilon N^V, v \epsilon N^S \tag{3.26}$$

### 3.3.5 Dependability Constraints

Finally, we define the constraints related to fault-tolerance and dependability.

**Link Mapping for Backup Traffic** For the backup traffic, it is necessary to define the same set of flow constraints defined for working traffic, but using the variables $bf_{u,v}^{i,j}$ and $bn_{u,v}$. Constraints 3.27, 3.28 and 3.29 refer to the backup flow conservation conditions, which denote that the network flow to a node is zero, except for the source node and the sink node, respectively. $wantBackup$ is a parameter defined by the tenant and it assumes the value 1 if backups are needed or the value 0 otherwise.

$$\sum_{u \epsilon N^S \cup N^V} bf_{u,v}^{i,j} - \sum_{u \epsilon N^S \cup N^V} bf_{v,u}^{i,j} = 0, \forall (i,j) \epsilon E^V, v \epsilon N^S \tag{3.27}$$

$$\sum_{v \epsilon N^S} bf_{i,v}^{i,j} - \sum_{v \epsilon N^S} bf_{v,i}^{i,j} = bw^V(i,j) * wantBackup, \forall (i,j) \epsilon E^V \tag{3.28}$$

$$\sum_{v \epsilon N^S} bf_{j,v}^{i,j} - \sum_{v \epsilon N^S} bf_{v,j}^{i,j} = -bw^V(i,j) * wantBackup, \forall (i,j) \epsilon E^V \tag{3.29}$$

Eq. 3.30 and 3.31 guarantee that the backup flow of a virtual link $(i,j)$ always departs from the correspondent backup node of *i* and arrives to the correspondent backup node of

$j$. Normally, the backup path only carries information of a virtual link if a failure in the working substrate path has occurred.

$$bn_{i,v} \quad bw^V(i,j) = bf_{i,v}^{i,j} * wantBackup, \forall v \epsilon N^S, (i,j) \epsilon E^V \tag{3.30}$$

$$bn_{j,v} \quad bw^V(i,j) = bf_{v,j}^{i,j} * wantBackup, \forall v \epsilon N^S, (i,j) \epsilon E^V \tag{3.31}$$

Figure 3.7 shows an example of how these constraints contribute to our formulation. In short, they define the values of variables $bf$. Constraints 3.27, 3.28 and 3.29 are responsible for $bf_{B,C}^{1,2} = 4$, i.e., for the backup flows in the substrate network. Eq. 3.30 and 3.31 are responsible for the backup flows on meta-links. If we assume that virtual nodes 1 and 2 of a VNR are mapped onto substrate nodes B and C, respectively, we have $bn_{1,B} = 1$ and $bn_{2,C} = 1$. Therefore, $bf_{1,B}^{1,2} = 4$ and $bf_{2,C}^{1,2} = 4$. All other $bf$ and $bn$ variables, such as $bf_{1,A}^{1,2}$ or $bn_{2,D}$ are equal to 0.

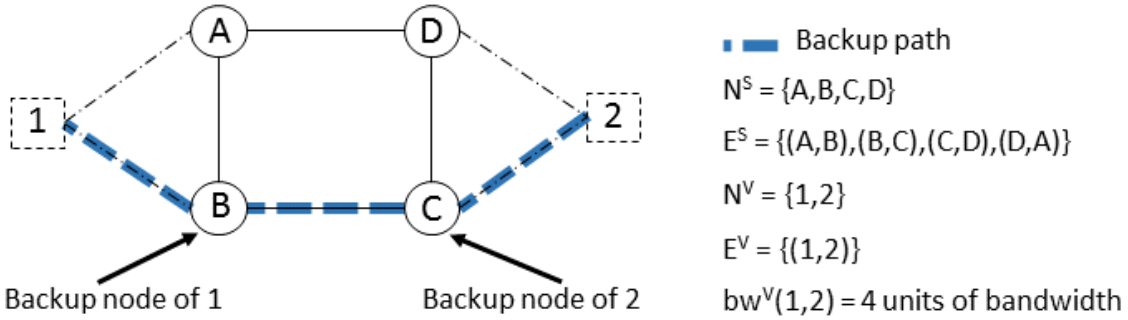



**Figure 3.7:** Example of how variables $bf$ contribute to our formulation.

The equation below guarantees that the meta-links only carry working or backup traffic to their correspondent virtual nodes. This means that, if a virtual node 1 needs to send information to virtual node 2, the data does not need to pass through the meta-links of a virtual node 3.

$$\sum_{j,k!=i \;\; j,k \epsilon N^V} wf_{i,v}^{j,k} + wf_{v,i}^{j,k} + bf_{i,v}^{j,k} + bf_{v,i}^{j,k} = 0, \forall v \epsilon N^S, i \epsilon N^V \tag{3.32}$$

**Virtual Node Mapping** Eq. 3.33 and 3.34 state that each virtual node has to be mapped to exactly one working node and $wantBackup$ backup nodes in the substrate node, i.e., if $wantBackup = 0$, virtual nodes will not have backup, if $wantBackup = 1$, virtual nodes will have backup. For the same VN, Eq. 3.35 guarantees that *(i)* two different virtual working nodes are not mapped to the same substrate node; and *(ii)* a substrate node that is the backup for a virtual node does not have any virtual working nodes on it. Eq. 3.36 guarantees that a substrate node can be the backup of a single virtual node, for

the same VN.

$$\sum_{v \epsilon N^S} wn_{u,v} = 1, \forall u \epsilon N^V \tag{3.33}$$

$$\sum_{v \epsilon N^S} bn_{u,v} = wantBackup, \forall u \epsilon N^V \tag{3.34}$$

$$\sum_{u \epsilon N^V} wn_{u,v} + bn_{z,v} \leq 1, \forall v \epsilon N^S, z \epsilon N^V \tag{3.35}$$

$$\sum_{u \epsilon N^V \setminus \{z\}} bn_{u,v} + bn_{z,v} \leq 1, \forall v \epsilon N^S, z \epsilon N^V \tag{3.36}$$

Note that we define Eq. 3.35 because we want to minimize the number of virtual resources of a VN affected if a failure occurs in a certain substrate node. Figures 3.8 and 3.9 clarify this idea. In Figure 3.8 if a failure occurs in the physical node A, and nodes 1 and 2 are mapped onto it, all the virtual links will be affected. Instead, if all the nodes of the VN are mapped onto different physical nodes and the failure occurs in node A, only the virtual link (1,3) will be affected.

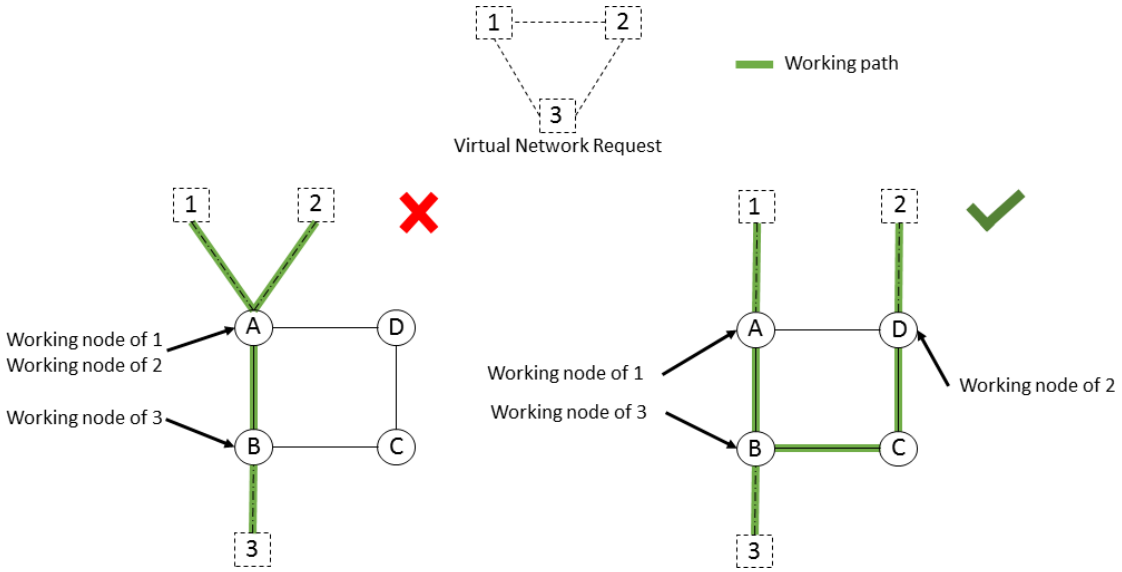



**Figure 3.8:** Example of an embedding that respects the first part of eq. 3.35.

In Figure 3.9, if a failure also occurs in the physical node A, and A is the working node for 1 and the backup node for 2, both working and backup paths will be affected. If physical nodes do not assume the working and backup function simultaneously for the same VN, this problem is solved, as we can observe in the right part of the figure. Here, if a failure occurs in node A, there will always exist a backup path through which nodes 1 and 2 can continue to communicate while A is down.

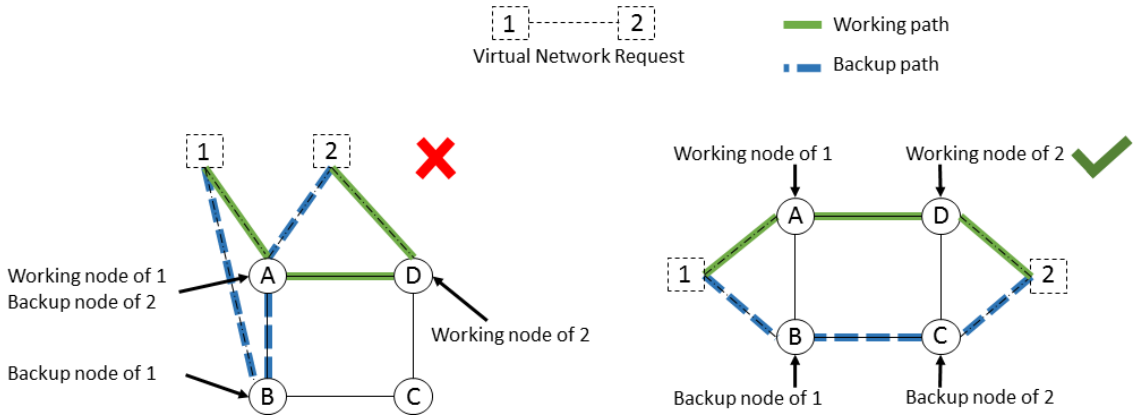



**Figure 3.9:** Example of an embedding that respects the second part of eq. 3.35.

Since in our work we allow the user to choose between having no replication, replication in the same cloud or in different clouds, it is necessary to specify these restrictions.

$$\sum_{c \epsilon C} wc_{u,c} = 1, \forall u \epsilon N^V \tag{3.37}$$

$$\sum_{c \epsilon C} bc_{u,c} = wantBackup, \forall u \epsilon N^V \tag{3.38}$$

$$wantBackup * wc_{u,c} + bc_{u,c} \leq dep^V(u), \forall u \epsilon N^V, c \epsilon C \tag{3.39}$$

$$wc_{u,c} \geq bc_{u,c} * dep^V(u) - 1, \forall u \epsilon N^V, c \epsilon C \tag{3.40}$$

Eq. 3.37 states that each virtual node is mapped on exactly one cloud. Eq. 3.38 ensures that, when a VN needs backup ($wantBackup = 1$), the backup of each virtual node is mapped to exactly one cloud. Eq. 3.39 and 3.40 are restrictions that address if a virtual node *u* and its correspondent backup will be on the same cloud or in different clouds, depending on the dependability level required by *u* ($dep^V(u)$).

Figure 3.10 represents the embedding of a VNR accordingly to the $dep^V$ of its nodes. Node 1 requires a backup in the same cloud ($dep^V$ = 2), while node 2 requires to have a backup in other cloud ($dep^V$ = 1).

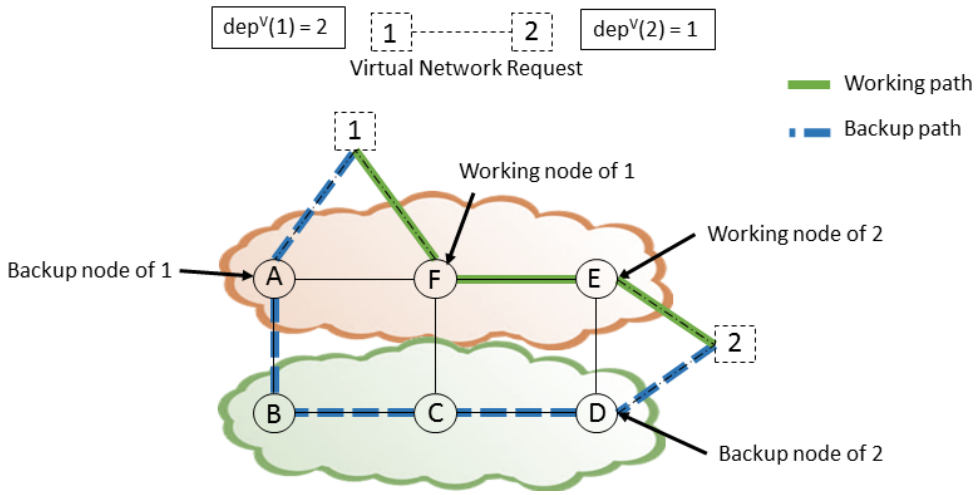



**Figure 3.10:** Example of an embedding respecting the $dep^V$ required by the virtual nodes.

Eq. 3.41 and 3.42 establish a relation between the virtual and physical nodes and the clouds (the first one related to working nodes, and the second related with the backup nodes).

$$\sum_{v \epsilon N^S} (wn_{u,v} * doesItBelong_{c,v}) \geq wc_{u,c}, \forall u \epsilon N^V, c \epsilon C \tag{3.41}$$

$$\sum_{v \epsilon N_S} (bn_{u,v} * doesItBelong_{c,v}) \geq bc_{u,c}, \forall u \epsilon N_V, c \epsilon C \tag{3.42}$$

The aim of these equations is the following. If a virtual node *u* is mapped onto a physical node *v* and *v* belongs to cloud *c* ($doesItBelong_{c,v} = 1$ if substrate node $v$ belongs to cloud $c$, 0 otherwise), then *u* is mapped on cloud *c*.

In a similar fashion, we also need restrictions to create relationships between the variables *wf*, *wl* and $wn$, and *bf*, *bl* and $bn$:

$$wn_{i,v} * bw^V(i,j) \geq wl^{i,j}_{i,v}, \forall (i,j) \epsilon E^V, v \epsilon N^S \tag{3.43}$$

$$wn_{j,v} * bw^V(i,j) \geq wl^{i,j}_{v,j}, \forall (i,j) \epsilon E^V, v \epsilon N^S \tag{3.44}$$

$$bw^V(i,j) * wl^{i,j}_{u,v} \geq wf^{i,j}_{u,v}, \forall (i,j) \epsilon E^V, u, v \epsilon N^S \cup N^V \tag{3.45}$$

Eq. 3.43 and 3.44 are constraints that ensure that if a meta-link is established between a virtual node $i$ and a physical node $v$, then it means that $i$ is mapped onto $v$. For instance, if $wl^{i,j}_{i,v} = 1$, then $wn_{i,v} = 1$. Eq. 3.45 ensures that if there is a flow between nodes $u$ and $v$ for a virtual link (i,j), then this means that (i,j) is mapped to a meta-link or a physical link whose end-points are $u$ and $v$. For example, if $wf^{i,j}_{u,v} = 1$, then $wl^{i,j}_{u,v} = 1$.

Eq. 3.46, 3.47 and 3.48 achieve the same goals as before, but for the backup:

$$bn_{i,v} * bw^V(i,j) \geq bl^{i,j}_{i,v}, \forall (i,j) \epsilon E^V, v \epsilon N^S \tag{3.46}$$

$$bn_{j,v} * bw^V(i,j) \geq bl^{i,j}_{v,j}, \forall (i,j) \epsilon E^V, v \epsilon N^S \tag{3.47}$$

$$bw^V(i,j) * bl^{i,j}_{u,v} \geq bf^{i,j}_{u,v}, \forall (i,j) \epsilon E^V, u, v \epsilon N^S \cup N^V \tag{3.48}$$

Finally, we include two binary constraints to guarantee the symmetric property of the binary variables related with links.

$$wl^{i,j}_{u,v} = wl^{i,j}_{v,u}, \forall (i,j) \epsilon E^V, u, v \epsilon N^S \cup N^V \tag{3.49}$$

$$bl^{i,j}_{u,v} = bl^{i,j}_{v,u}, \forall (i,j) \epsilon E^V, u, v \epsilon N^S \cup N^V \tag{3.50}$$

**Nodes and Links Disjointness** Since any substrate nodes and links of a working path can fail, we have to ensure that backup paths connecting the backups of the virtual nodes are disjoint from the substrate resources that are being used for the working part (otherwise a backup path can be compromised if a physical resource belonging to the working and backup part fails).

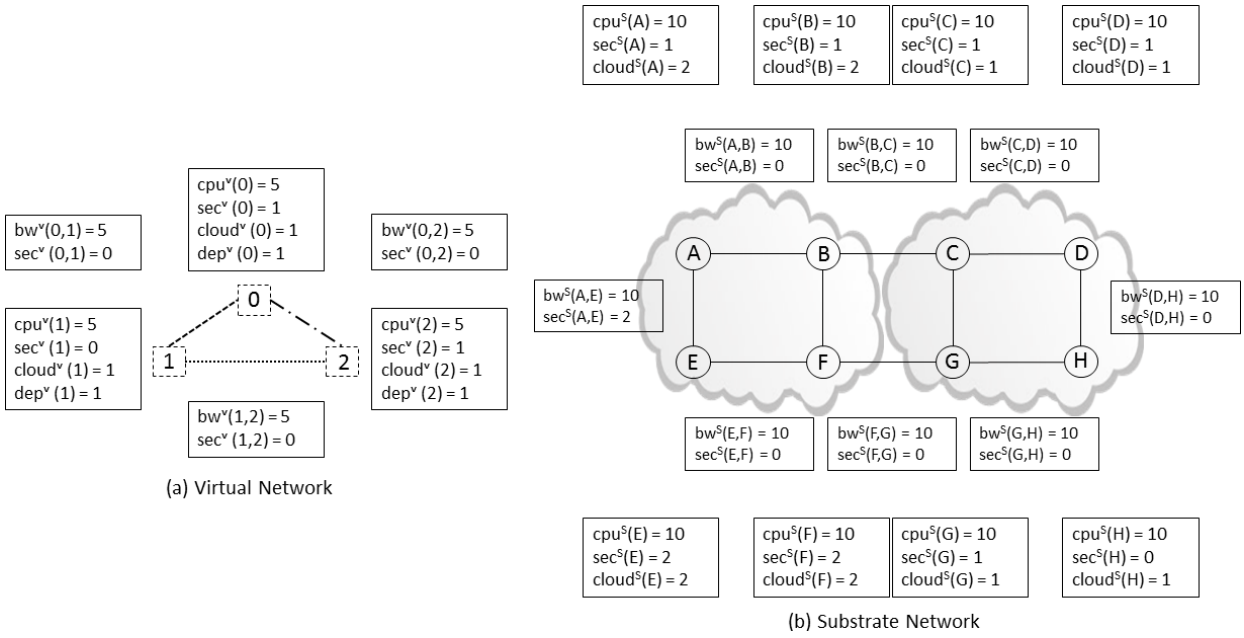
$$BigConstant * working_u >= \sum_{v \epsilon N^S} wl_{u,v}^{i,j}, \forall (i,j) \epsilon E^V, u \epsilon N^S \quad (3.51)$$

$$BigConstant * backup_u >= \sum_{v \epsilon N^S} bl_{u,v}^{i,j}, \forall (i,j) \epsilon E^V, u \epsilon N^S \quad (3.52)$$

$$backup_u = 1 - working_u, \forall u \epsilon N^S \quad (3.53)$$

Eq. 3.51 - 3.53 together with Eq. 3.35 ensure path disjointness between the working and the backup parts. As we already observed, Eq. 3.35 ensures that a substrate node mapping the working virtual node can not be a backup of any other node, and vice-versa. Relatively to the Eq. 3.51 - 3.53 we ensure that if a substrate node $u$ is an end point of a certain link that is being used as a working resource, $u$ can not be an end point of a link that is being used as a backup resource, and vice-versa. Variables $working$ and $backup$ are auxiliary variables that define if a certain physical node $u$ belongs to the working or backup part. $BigConstant$ is a constant big enough to ensure that the restriction is valid when its rightmost part is greater than 0.

To close the explanation of our algorithm, in Figure 3.11 we present a full embedding of a VN onto a SN. An embedding is only successful if all of the previous restrictions explained are fulfilled. In this figure, in addition to the CPU and bandwidth capacities, the VN requests for security for its nodes and links, and asks also for backups in another cloud. All the requirements of the VN and the characteristics of the substrate resources are presented in the figure, according with the attributes defined in Sections 3.2.1 and 3.2.2.



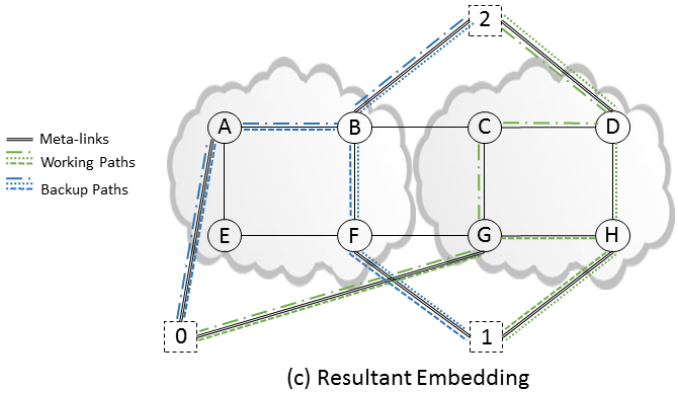



**Figure 3.11:** Example of a full embedding of a VN onto a SN with our solution.

## 3.4 A Simple API

An API through which network virtualization platforms can use our embedding algorithm is essential. Our algorithm focus on platforms that consider security and dependability as first class citizens in a multiple cloud environment, such as [3].

### 3.4.1 Usage

The API is simple to facilitate its usage. The client of the API has to construct a file with the current state information of the SN, and also with the information of the VN that he wants to embed. After this, the client has to call a method "solve" that is responsible for the embedding of the given VN onto the given SN. If an embedding of the VN is found, the result is a file with the mapping information, that should be read by the client. Otherwise, it is informed that the embedding was not successful. The reading of this file and the treatment of this information is left to the client, since different platforms may have different ways to treat this information.

Figure 3.12 shows an example of a configurations file, where it is possible to observe the information that has to be specified. The explanation of what each field means is in the figure. The file extension should be ".config".

After the execution of the SecDep VNE algorithm, a file similar to the one presented in Figure 3.13 is produced. This file presents the information of the embedding in a simplified and synthetic way. It shows the working substrate nodes where the virtual nodes were mapped, the correspondent backup nodes allocated, the working substrate path, and also the backup substrate path. In this figure it is possible to observe that virtual node 1 and 0 were embedded onto substrate nodes B and C, respectively, and they are separated by one hop. The same happens with the substrate part, where A is allocated to virtual node 0 and D is allocated to virtual node 1.

input1.config

```
Substrate

NumNodes: 4 #Number of substrate nodes
NumLinks: 4 #Number of substrate links
NumClouds: 2 #Number of clouds
NodesID: A B C D #IDs of the substrate nodes
NodesCPU: 10 10 10 10 #CPU capacity that each substrate node has
NodesSec: 2 2 1 1 #Security provided by each substrate node
NodesLoc: 0 0 1 1 #Location of substrate nodes
Links: (A,B) (B,C) (C,D) (D,A) #Substrate links
LinksBw: 10 10 10 10 #Bw capacity that each substrate link has
LinkSec: 2 2 2 2 #Security provided by each substrate link
LinksWeight: 1 1 1 1 #Weight of each substrate link
CloudsID: 0 1 #Cloud IDs
CloudSec: 2 1 #Security provided by each cloud
################################################################################
Virtual

NumNodes: 2 #Number of virtual nodes
NumLinks: 1 #Number of virtual links
NodesID: 0 1 #IDs of the virtual nodes
NodesCPUDem: 5 5 #CPU demanded by each virtual node
NodesSecDem: 2 1 #Security demanded by each virtual node
NodesLocDem: 2 1 #Location demanded by each virtual node
NodesBackupLocDem: 2 2 #Location demanded for the backup of each virtual node
Links: (0,1) #Virtual links
LinksBwDem: 5 #Bw demanded by each virtual link
LinksSecDem: 0 #Security demanded by each virtual link
Backup: 1 #Defines if backup is needed or not
```

**Figure 3.12:** Example of an input file for our SecDep VNE API.

input1.smplout

```
Working node of 0 -> B
Working node of 1 -> C
Backup node of 0 -> A
Backup node of 1 -> D
Working links for (0,1) -> (B,C)
Working path for (0,1) -> B C
Backup links for (0,1) -> (A,D)
Backup path for (0,1) -> A D

```

**Figure 3.13:** Example of an output file produced by our SecDep VNE API.

### 3.4.2 Classes and Work flow

Figure 3.14 is a class diagram showing the classes that were constructed and relationships between them. Class $Embeddor$ is the "front-end" class, i.e., the class through which the API's client initiates an embedding. As we said, to start an embedding of a VN the client has to call the method $solve$, passing it the path to the file with the configurations. The method $readConfigFile$ from the class $ConfigsReader$ will be called from $solve$. The class $ConfigsReader$ is responsible for reading the configurations file and keep all the information of the VN and the SN in its attributes. The method $readConfigFile$ makes the parse of the configurations file and puts the information in the attributes of the class.

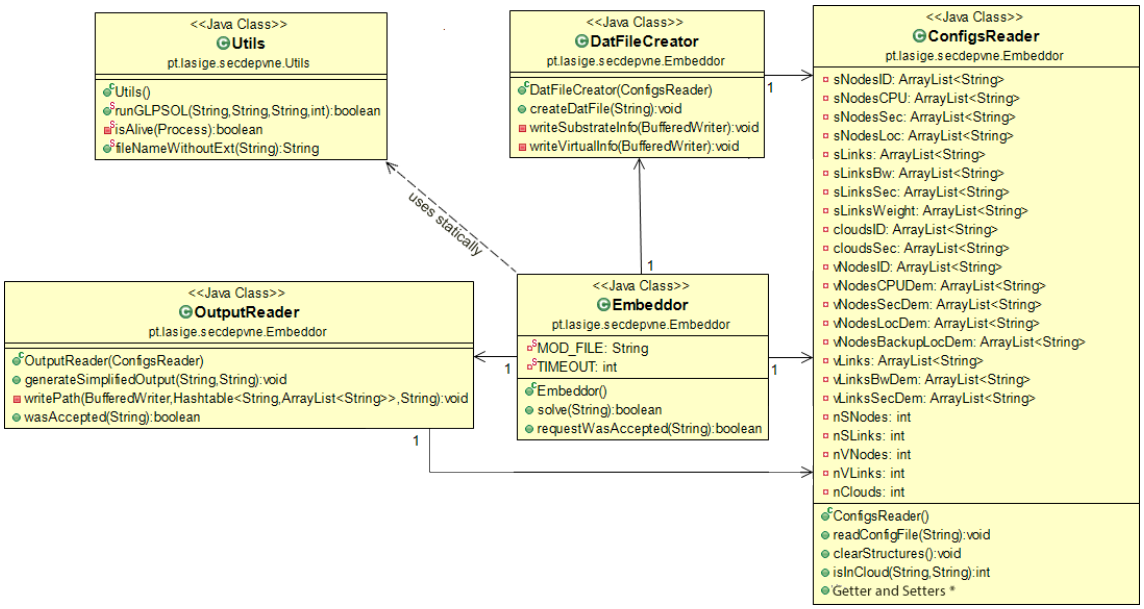



**Figure 3.14:** Class diagram of the SecDep VNE API.

After all the information is loaded, the method $createDatFile$ from the class *DatFileCreator* is called. They are responsible for the creation of a ".dat" file, with all the MILP input parameters, which will be used in the method $runGLPSOL$. Then, $runGLPSOL$ executes the MILP formulation and generates an output file with all the information resultant from the execution, e.g., final values of all the MILP variables. After the execution terminates the method $wasAccepted$ from the class $OutputReader$ is called to check if the request was embedded. If it was, $generateSimplifiedOutput$ will be called to produce a simplified output file with concise information, in order to facilitate the client obtaining the important information.

After these steps, the line of execution will end with method $solve$ returning a positive answer to the client if all has gone well, or a negative answer otherwise.

In this section we presented an API that can be used by network virtualization platforms that consider security and dependability as first class citizens in multiple cloud environments. In the following section we explain the simulation setup that we used to assess the performance of our algorithm, as well as we present the obtained results.

# Chapter 4 - Evaluation

In this chapter we present some performance results of our solution. We have implemented a simulator to reproduce an environment where VNRs with different requirements arrive over time. Section 4.1 presents the simulation setup, where we explain how the environment was set. In Section 4.2 we present the different experiments that were executed and we also present the algorithm which was the basis for comparison. Finally, in Section 4.3 we show the results of our evaluation, followed by a discussion in Section 4.4.

## 4.1 Simulation Setup

We have implemented an event simulator to evaluate the performance of our algorithm against the performance of D-ViNE [4, 5]. We have chosen D-ViNE due to its availability as open-source software and the fact that it has been considered as the baseline for most VNE work. This simulator was based on the simulator presented in [49] and, in short, it simulates the dynamical arrival of VNRs to the system. For our evaluation, the SN topology was randomly generated with 25 nodes using the GT-ITM tool [50] in (10x10) grids. Each pair of substrate nodes was randomly connected with probability between 0.1 and 0.3. The CPU and bandwidth resources of the substrate nodes and links were real numbers uniformly distributed between 50 and 100. We assumed that VNRs arrivals ($Time^V$) are modeled as a Poisson process with an average rate of 4 VNRs per 100 time units, each one having an exponentially distributed lifetime ($Dur^V$) with an average of $\mu = 1000$ time units. In each VNR, the number of virtual nodes was randomly determined by a uniform distribution between 2 and 4. Each pair of virtual nodes was randomly connected with probability between 0.1 and 0.3. The capacity requirements of the virtual nodes and links were real numbers uniformly distributed between 10 and 20.

We chose to only address a small scale environment (25 nodes to the SN and 2-4 nodes to the VNs) because optimal solutions, such as the MILP we used in SecDep VNE, do not scale for large networks and, therefore, it would not be possible to have results in a reasonable time.

For SecDep VNE we need to include the security and dependability attributes. In this simulation, the substrate nodes belonged to three clouds, each one with a different security level (public, trusted public and private). To the parameters $sec$ and $cloud$ we used the values $\{1.0, 1.1, 1.2\}$. The value 1.0 corresponds to the lower level of security for nodes, links and clouds; the value 1.1 corresponds to the intermediary level of security; and the value 1.2 corresponds to the higher level of security. We chose these values in order to better adjust the prices of the different levels of this property. The probability of substrate nodes and links having $sec^S = 1.0$ was 0.05, $sec^S = 1.1$ was 0.4, and $sec^S = 1.2$ was

0.55. The weight ($\alpha$) of all substrate links was 1. The $sec^V$ of the virtual nodes and links was distributed uniformly by the three different security types. $cloud^V$ and $dep^V$ for the virtual nodes of a request were also generated randomly. In this evaluation, we considered that $\lambda_1 = 1$, $\lambda_2 = 1$ and $\lambda_3 = 1$.

To solve the MILP solutions we used the open source MILP library GLPK [8]. The simulation ran for 50000 time units, and during this period, the MILPs tried to embed 1000 VNRs. The order of arrival of VNRs and the capacity requirements of each VNR were the same for both algorithms, in order to ensure that both dealt with similar problem instances.

## 4.2 Comparison Method

In our evaluation, we compared two algorithms that solve the VNE problem with different requirements, D-ViNE [4] and SecDep. D-ViNE requirements include only CPU and bandwidth capacities, while our algorithm adds to these requirements also security demands, cloud preferences and dependability requirements. This means the algorithms are not in an equal footing (ours has more requirements and consequently restrictions), but we chose D-ViNE to comparison because it has been considered in the VNE literature as the baseline. Furthermore, the comparison of SecDep against D-ViNE is interesting in order to understand what are the implications when we introduce security and dependability aspects to a typical embedding problem. The objective function of D-ViNE is presented in 4.1. Besides the minimization of CPU and bandwidth resources allocated to a VN (i.e., minimization of costs), the objective function of this algorithm also tries to balance the load introducing weights. In our evaluation, we considered this weights as 1, as its authors [5].

$$min \sum_{uv \epsilon E^S} \frac{\alpha_{uv}}{R_E(u,v) + \delta} \sum_i f_{uv}^i + \sum_{w \epsilon N^S} \frac{\beta_w}{R_N(w) + \delta} \sum_{m \epsilon N^{S'} \setminus N^S} x_{mw} c(m) \tag{4.1}$$

To evaluate our solution, we run six different types of experiments:

1. Solve the embedding problem where there are no security and dependability requirements for VNs and the SN;

2. Solve the embedding problem where the SN and all VNs have random security requirements for their resources, and only 10% of them requests dependability;

3. Solve the embedding problem where the SN and all VNs have random security requirements for their resources, and 30% of them requests dependability;

4. Solve the embedding problem where the SN and all VNs have random security requirements for their resources, and 50% of them requests dependability;

5. Solve the embedding problem where the SN and all VNs have random security requirements for their resources, and 100% of them requests dependability. This case is considered the worst-case in our evaluation, since all the VNRs seek for security and dependability;

6. Solve the embedding problem where there are four different types of requests: 25% of the VNRs request neither security nor dependability, 25% of the VNRs request only dependability, 25% of the VNRs request only security, and the last 25% of VNRs have both security and dependability requirements.

The notations that we used to refer the different experiments are enumerated in Table 4.1.

| Notation | Algorithm Description |
|---|---|
| D-ViNE | VNE MILP model presented in [5] |
| SecDep0 | Secure and Dependable VNE MILP model with the environment explained in Point 1. |
| SecDep10 | Secure and Dependable VNE MILP model with the environment explained in Point 2. |
| SecDep30 | Secure and Dependable VNE MILP model with the environment explained in Point 3. |
| SecDep50 | Secure and Dependable VNE MILP model with the environment explained in Point 4. |
| SecDep100 | Secure and Dependable VNE MILP model with the environment explained in Point 5. |
| MixedSecDep | Secure and Dependable VNE MILP model with the environment explained in Point 6. |

**Table 4.1:** Compared algorithms

## 4.3 Evaluation Results

We used several performance metrics for evaluation in our experiments. We have considered the same metrics as in [4]:

- VNR acceptance ratio: the percentage of requests accepted over the time;
- Average revenue: the revenue that the InP gets over the time;
- Average cost of accepting a VNR: the cost the InP will incur by embedding a request;
- Average node utilization: the load of the SN nodes over time;
- Average link utilization: the load of SN links over time.

To calculate the revenue $\mathbb{R}$ and costs $\mathbb{C}$ we used the equations presented in Section 3.2.4. We now present all results from the simulations and discuss each outcome.

1) **The embedding performance of SecDep without security and dependability requirements is similar to the embedding performance of D-ViNE.** When security and dependability are not taken into account, our algorithm performs similar to D-ViNE, as can be seen in all figures we present in this section. This is important as it shows our baseline to be identical to the most commonly used VNE algorithm.

2) **A richer set of features (namely, security and dependability) decreases the acceptance ratio.** Figure 4.1 shows that D-ViNE leads to a higher acceptance ratio over time when compared with our SecDep VNE. This was expected, as SecDep VNE is richer in terms of the features provided (security and dependability, in addition to CPU and bandwidth). Consequently, in SecDep VNE to accept a VNR the number of constraints is higher and more conditions need to be satisfied. For instance, a VNR with some virtual nodes demanding the maximum security level may be rejected by SecDep if the SN does not have enough substrate nodes available to cover that demand at the moment. Another important factor that contributes to a smaller acceptance ratio is the higher use of substrate resources due to VNRs that require dependability features, as more resources need to be allocated to these VNRs. When compared to D-ViNE, the worst-case of SecDep (SecDep100) has an acceptance ratio around 35% smaller. The scenarios SecDep50 and MixedSecDep have similar acceptance ratios, both accepting around 20% less requests when compared to D-ViNE. In general, and as expected, a secure and dependable VNE algorithm will present a lower acceptance ratio as providing security and dependability is more demanding.

3) **A richer set of features (namely, security and dependability) increases the revenue until the point when the acceptance ratio becomes too low.** Figure 4.2 illustrates the time average of generated revenue. When our algorithm is set without security and dependability requirements (SecDep0), it generates the same revenue as D-ViNE. In this figure it is also possible to observe that SecDep10 and SedDep30 generate more revenue than D-ViNE. This is due to security having a weight that is considered in the revenue function (as we expect richer resources to be more more expensive). Since the acceptance ratio in these is not far from D-ViNE, the revenue will increase. All the other cases (SecDep100, SecDep50, and MixedSecDep) generate less revenue than D-ViNE as a consequence of the smaller acceptance ratio they present.

4) **A relatively small increase in the price of security resources leads to revenues that are inline with traditional VNE algorithms.** As per the previous point, SecDep0, SecDep10, and SecDep30 generate similar or slightly better revenues than D-ViNE, so we address this point only to the other cases. As we stated earlier, a way to increase the revenue is to increase the price of each security resource. To have a better notion of the scale of the necessary increase, in Table 4.2 we show the target price security resources should have in cases SecDep50, SecDep100 and MixedSecDep, in order to achieve a total revenue similar to D-ViNE. For this analysis we change the prices of node, link and cloud securities proportionally. The column of security prices shows the price changes we made. The first line of each cell in this column refers to the price increase of the intermediary security levels, while the second one refers to the price increase of the highest security levels. The important take away from this table is the fact that it is possible to achieve a revenue inline with D-ViNE with a relatively small increase in the price of each security resource.

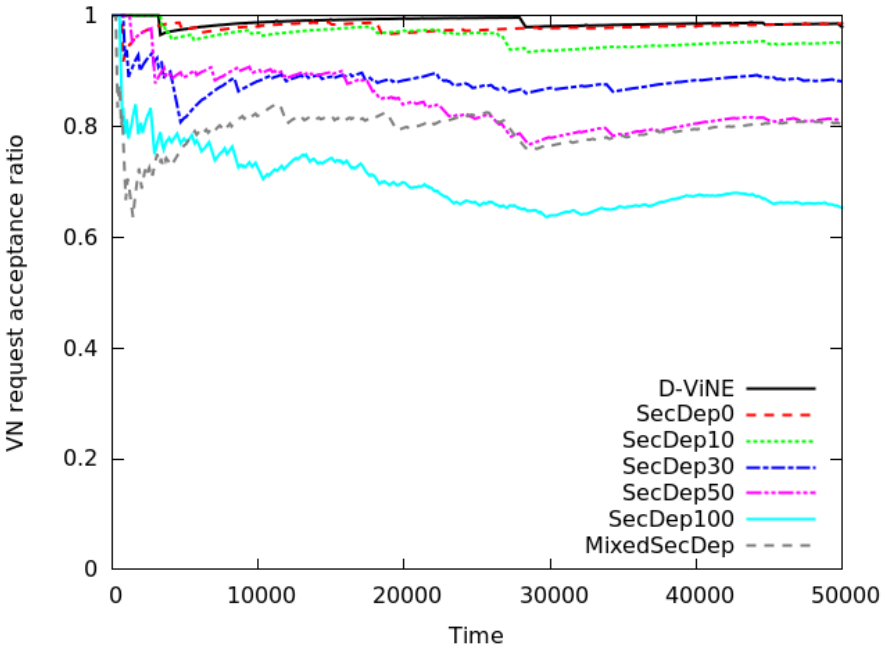



**Figure 4.1:** VNR acceptance ratio over time.

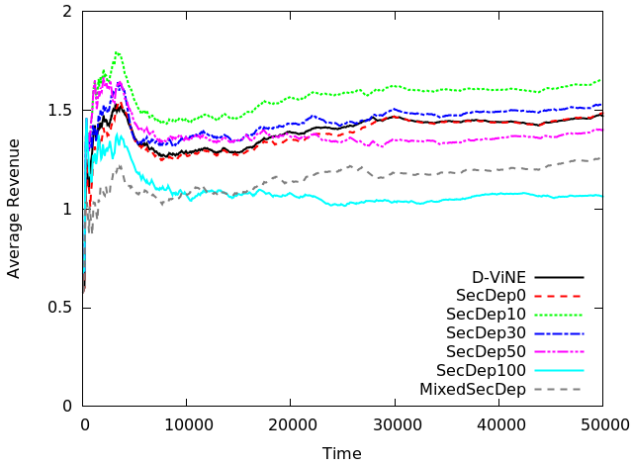



**Figure 4.2:** Time average of generated revenue.

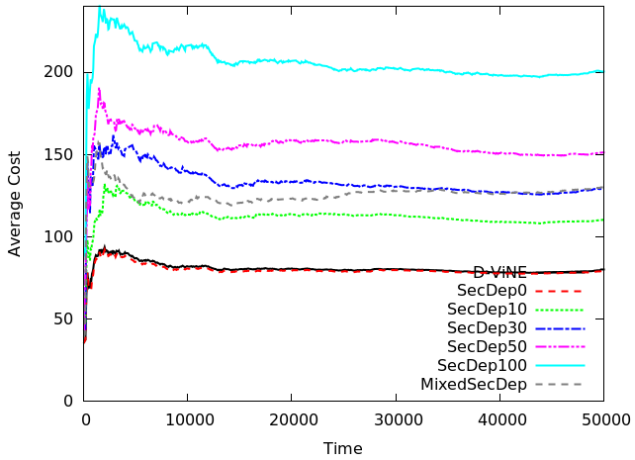



**Figure 4.3:** Average cost of accepting VNRs over time.

5) **Dependability and security requirements increase the costs of embedding a VNR.** Figure 4.3 shows that the costs of embedding a VNR with SecDep algorithm are higher when compared to D-ViNE, as expected. The cost is higher for one main reason. When backups are required by a VNR, the CPU resources demanded by a virtual node are always allocated twice, one allocation for the working node and another to the backup node. There will also be additional allocations of bandwidth resources for the virtual links, since it is necessary to have substrate paths connecting the working nodes (working paths composed by at least one working link) and substrate paths connecting the backup nodes (backup paths composed by at least one backup link).

| Algorithm | Total $\mathbb{R}$ | Security Prices (Original price → New price) | Average % Price Increasing |
|---|---|---|---|
| D-ViNE | ≈ 76796 | | |
| SecDep50 | ≈ 76773 | 1.1 → 1.1<br>1.2 →1.3 | 4.165 |
| SecDep50 | ≈ 80320 | 1.1 → 1.2<br>1.2 →1.3 | 8.71 |
| SecDep50 | ≈ 80372 | 1.1 → 1.1<br>1.2 →1.4 | 8.335 |
| SecDep100 | ≈ 77800 | 1.1 → 1.3<br>1.2 →1.8 | 34.09 |
| SecDep100 | ≈ 77931 | 1.1 → 1.4<br>1.2 →1.7 | 34.47 |
| SecDep100 | ≈ 78061 | 1.1 → 1.5<br>1.2 →1.6 | 34.845 |
| MixedSecDep | ≈ 75473 | 1.1 → 1.3<br>1.2 →1.5 | 20.19 |
| MixedSecDep | ≈ 77522 | 1.1 → 1.4<br>1.2 →1.5 | 26.135 |
| MixedSecDep | ≈ 77544 | 1.1 → 1.3<br>1.2 →1.6 | 24.355 |
| MixedSecDep | ≈ 79635 | 1.1 → 1.4<br>1.2 →1.6 | 30.3 |
| MixedSecDep | ≈ 81764 | 1.1 → 1.5<br>1.2 →1.6 | 34.845 |

**Table 4.2:** Prices increasing to achieve total revenues near to D-ViNE.

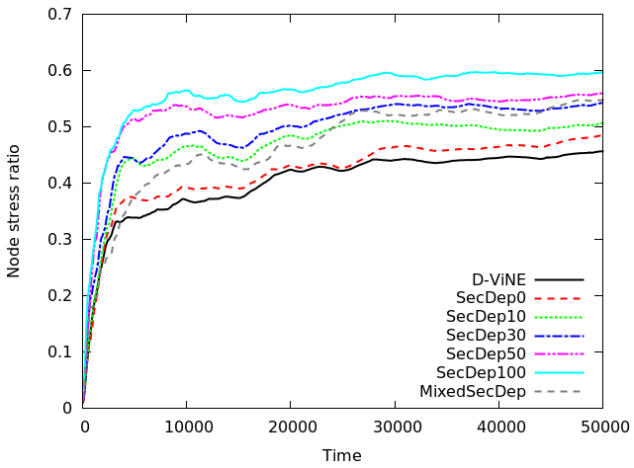



**Figure 4.4:** Average node utilization.

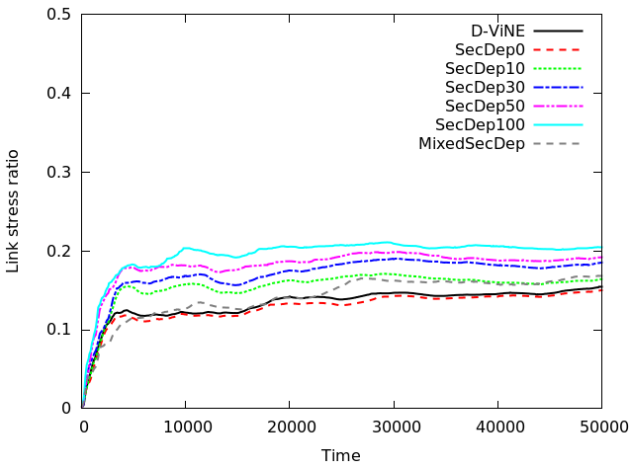



**Figure 4.5:** Average link utilization.

6) **Dependability requirements increase substrate resources utilization.** Figure 4.4 and Figure 4.5 show the average substrate node and link utilization, respectively. In

both figures we observe that more resources are allocated in the SN with SecDep VNE than with D-ViNE. The reasons are the same as in Point 5.

## 4.4 Discussion

Security and dependability have a cost that may affect the profit of the InP. However, it is possible to reduce the costs by increasing the embedding efficiency. For instance, our solution adopts a backup scheme where each virtual node has a dedicated backup, and additional links are allocated to ensure that VNs survive in case of a failure. This approach is the most conservative, and consequently the most expensive.

Some well-known techniques can be used to reduce the costs of embedding. For example, the use of a backup pooling mechanism, as in [32], can reduce the cost of embedding as only a set of substrate nodes (instead of all) would be allocated to be backup of any VNR when a failure occurs. Although this solution may reduce costs and resource utilization, it brings some disadvantages: the substrate nodes pooled may not be sufficient to all VNRs if an outage occurs. In addition, when a failure occurs making a VNR operational again takes more time, since it is necessary to search which pooled nodes can allocate resources for the failed nodes.

In a similar way, a backup bandwidth sharing technique where bandwidth resources of backup links (or backup paths) can be allocated on-demand by any VNR could also reduce the costs and resources utilization. The downside is that this technique does not ensure that there are sufficient backup bandwidth resources available when a failure occurs.

All these techniques target efficiency, aiming to reduce costs. Other possibilities include increasing revenue. For instance, the acceptance ratio of our algorithm could be improved by, instead of refusing a VNR when there are no resources available at the moment of its arrival, postponing the request to be embedded later, as proposed in [5].

# Chapter 5 - Conclusion

## 5.1 Conclusions

A major challenge in network virtualization is how to make efficient use of the shared resources. Virtual Network Embedding addresses this problem by finding an effective mapping of the virtual nodes and links onto the substrate network. For some scenarios, VNE has been studied in some detail in the network virtualization literature.

The VNE problem is traditionally formulated with the objective of maximizing network provider revenue by efficiently embedding incoming VNRs. This objective is subject to constraints, such as processing capacity on the nodes and bandwidth resource on the links. A mostly unexplored perspective on this problem is providing some security assurances, a gap increasingly more acute. With the advent of network virtualization platforms, cloud operators now have the ability to extend their cloud computing offerings with virtual networks. To shift their workloads to the cloud, tenants trust their cloud providers to guarantee that their workloads are secure and available. Unfortunately, there is increasing evidence that problems do occur, of both the malicious kind (e.g., caused by a corrupt cloud insider) or benign (e.g., a cloud outage). Security and dependability is thus becoming a critical factor that should be considered by virtual network embedding algorithms.

In this thesis we proposed a VN embedding solution that considers security and dependability as first class citizens. For this purpose, we introduced specific security and dependability constraints into the MILP formulation. The Supercloud concept we introduced allows us to extend the resiliency properties further, by assuming a multiple cloud provider model, considering the coexistence of multiple clouds: both private, belonging to the tenant, and public, belonging to cloud providers. By not relying on a single cloud provider we avoid internet-scale single points of failures, avoiding cloud outages by replicating workloads across clouds. In addition, we can enhance security by leaving sensitive workloads in the tenant's private clouds or in public trusted facilities.

The results from our experiments show that there is an (already expected) cost in providing security and availability that may reduce the InP profit. However, a relatively small increase in the price of the richer features of our solution (security resources, for instance), coupled with efficient techniques to reduce the embedding cost (e.g., pooling of backup resources) enables the provider to offer secure and dependable network services at a profit.

## 5.2 Future Work

As future work, the first direction we point is to construct a heuristic to solve this problem, since exact solutions are not feasible in a reasonable time. The heuristic can be a simple relaxation of the MILP formulation presented in this work, or can be a solution more sophisticated that looks to the network state and to the attributes of VNRs (e.g., a greedy solution).

Other important aspects to improve are, for instance, the backup scheme, with the aim of reduce costs for the cloud providers, i.e., implement backup schemes such as backup pooling or backup bandwidth sharing. Implementing a mechanism capable of postponing VNRs when there are no substrate resources available to embed is also important.

The study of advanced models of costs and revenues to cloud providers is another important research topic that needs further attention.

# Glossary

**API** Application Programming Interface.

**BFS** Breadth-First Search.

**CPU** Central Processing Unit.

**ID** Identifier.
**ILP** Integer Linear Programming.
**InP** Infrastructure Provider.
**IP** Internet Protocol.
**ISP** Internet Service Provider.

**LLDP** Link Layer Discovery Protocol.
**LP** Linear Programming.

**MAC** Media Access Control.
**MILP** Mixed Integer Linear Programming.
**MTD** Multi-Tenant Datacenter.

**NOS** Network Operating System.

**OvS** Open vSwitch.

**QoS** Quality of Service.

**SDN** Software Defined Networking.
**SN** Substrate Network.
**SP** Service Provider.

**VI** Virtual Infrastructure.
**VLAN** Virtual Local Area Network.
**VLiM** Virtual Link Mapping.
**VM** Virtual Machine.
**VN** Virtual Network.
**VNE** Virtual Network Embedding.
**VNO** Virtual Network Operator.
**VNoM** Virtual Node Mapping.
**VNP** Virtual Network Provider.
**VNR** Virtual Network Request.
**VPN** Virtual Private Network.

# Bibliography


[1] Andreas Fischer, Juan Felipe Botero, Michael Till Beck, Hermann De Meer, and Xavier Hesselbach. Virtual network embedding: A survey. *IEEE Communications Surveys & Tutorials*, 15(4):1888–1906, 2013.

[2] Cloud Security Alliance. The notorious nine cloud computing top threats in 2013.

`https://downloads.cloudsecurityalliance.org/initiatives/top_threats/The_Notorious_Nine_Cloud_Computing_Top_Threats_in_2013.pdf`, 2013.

[3] Max Alaluna, Fernando Ramos, and Nuno Neves. (Literally) above the clouds: virtualizing the network over multiple clouds. *arXiv preprint arXiv:1512.01196*, 2015.

[4] Mosharaf Chowdhury, Muntasir Rahman, and Raouf Boutaba. Virtual network embedding with coordinated node and link mapping. In *INFOCOM 2009, IEEE*, pages 783–791, April 2009.

[5] Mosharaf Chowdhury, Muntasir Rahman, and Raouf Boutaba. Vineyard: Virtual network embedding algorithms with coordinated node and link mapping. *IEEE/ACM Transactions on Networking*, 20(1):206–219, Feb 2012.

[6] Thomas Anderson, Larry Peterson, Scott Shenker, and Jonathan Turner. Overcoming the internet impasse through virtualization. *Computer*, (4):34–41, 2005.

[7] Nick Feamster, Lixin Gao, and Jennifer Rexford. How to lease the internet in your spare time. *ACM SIGCOMM Computer Communication Review*, 37(1):61–64, 2007.

[8] GLPK. Gnu linear programming kit.

`http://www.gnu.org/software/glpk/`, 2008.

[9] Inc ILOG. Ilog cplex: High-performance software for mathematical programming and optimization.

`http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/index.html`, 2008.

[10] Qin Jia, Zhiming Shen, Weijia Song, Robbert van Renesse, and Hakim Weatherspoon. Supercloud: Opportunities and challenges. *ACM SIGOPS Operating Systems Review*, 49(1):137–141, 2015.

[11] Muli Ben-Yehuda, Michael Day, Zvi Dubitzky, Michael Factor, Nadav Har'El, Abel Gordon, Anthony Liguori, Orit Wasserman, and Ben-Ami Yassour. The turtles project: Design and implementation of nested virtualization. In *OSDI*, volume 10, pages 423–436, 2010.

[12] Dan Williams, Hani Jamjoom, and Hakim Weatherspoon. The xen-blanket: virtualize once, run everywhere. In *Proceedings of the 7th ACM european conference on Computer Systems*, pages 113–126. ACM, 2012.

[13] Supercloud project.

`https://supercloud-project.eu/publications-deliverables`.

[14] Luís Ferrolho, Max Alaluna, Nuno Neves, and Fernando Ramos. Secure and dependable virtual network embedding. NSDI, 2016.

[15] Mosharaf Chowdhury and Raouf Boutaba. A survey of network virtualization. *Comput. Netw.*, 54(5):862–876, April 2010.

[16] Diego Kreutz, Fernando Ramos, Paulo Veríssimo, Christian Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, Jan 2015.

[17] Teemu Koponen, Keith Amidon, Peter Balland, Martín Casado, Anupam Chanda, Bryan Fulton, Igor Ganichev, Jesse Gross, Natasha Gude, Paul Ingram, Ethan Jackson, Andrew Lambeth, Romain Lenglet, Shih-Hao Li, Amar Padmanabhan, Justin Pettit, Ben Pfaff, Rajiv Ramanathan, Scott Shenker, Alan Shieh, Jeremy Stribling, Pankaj Thakkar, Dan Wendlandt, Alexander Yip, and Ronghua Zhang. Network virtualization in multi-tenant datacenters. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pages 203–216, 2014.

[18] Open vswitch.

`http://openvswitch.org/`.

[19] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008.

[20] Ali Al-Shabibi, Marc De Leenheer, Matteo Gerola, Ayaka Koshibe, Guru Parulkar, Elio Salvadori, and Bill Snow. Openvirtex: Make your virtual sdns programmable. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 25–30, New York, USA, 2014.

[21] Rob Sherwood, Glen Gibb, Kok-Kiong Yap, Guido Appenzeller, Martin Casado, Nick McKeown, and Guru Parulkar. Flowvisor: A network virtualization layer. *OpenFlow Switch Consortium, Tech. Rep*, pages 1–13, 2009.

[22] Minlan Yu, Yung Yi, Jennifer Rexford, and Mung Chiang. Rethinking virtual network embedding: substrate support for path splitting and migration. *ACM SIGCOMM Computer Communication Review*, 38(2):17–29, 2008.

[23] David Eppstein. Finding the k shortest paths. *SIAM Journal on computing*, 28(2):652–673, 1998.

[24] Xiang Cheng, Sen Su, Zhongbao Zhang, Hanchi Wang, Fangchun Yang, Yan Luo, and Jie Wang. Virtual network embedding through topology-aware node ranking. *SIGCOMM Comput. Commun. Rev.*, 41(2):38–47, April 2011.

[25] Sheng Zhang, Zhuzhong Qian, Jie Wu, and Sanglu Lu. An opportunistic resource sharing and topology-aware mapping framework for virtual networks. In *INFOCOM, 2012 Proceedings IEEE*, pages 2408–2416, March 2012.

[26] Jens Lischka and Holger Karl. A virtual network mapping algorithm based on subgraph isomorphism detection. In *Proceedings of the 1st ACM Workshop on Virtualized Infrastructure Systems and Architectures*, VISA '09, pages 81–88, New York, USA, 2009.

[27] Juan Felipe Botero, Xavier Hesselbach, Michael Duelli, Daniel Schlosser, Andreas Fischer, and Hermann De Meer. Energy efficient virtual network embedding. *IEEE Communications Letters*, 16(5):756–759, May 2012.

[28] Hongfang Yu, Vishal Anand, Chunming Qiao, and Gang Sun. Cost efficient design of survivable virtual infrastructure to recover from facility node failures. In *2011 IEEE International Conference on Communications (ICC)*, pages 1–6, June 2011.

[29] Qian Hu, Yang Wang, and Xiaojun Cao. Location-constrained survivable network virtualization. In *Sarnoff Symposium (SARNOFF), 2012 35th IEEE*, pages 1–5, May 2012.

[30] Muntasir Raihan Rahman, Issam Aib, and Raouf Boutaba. Survivable virtual network embedding. In *International Conference on Research in Networking*, pages 40–52. Springer, 2010.

[31] Tao Guo, Ning Wang, Klaus Moessner, and Rahim Tafazolli. Shared backup network provision for virtual network embedding. In *2011 IEEE International Conference on Communications (ICC)*, pages 1–5, June 2011.

[32] Wai-Leong Yeow, Cédric Westphal, and Ulaş Kozat. Designing and embedding reliable virtual infrastructures. In *Proceedings of the Second ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures*, VISA '10, pages 33–40, New York, USA, 2010.

[33] Yang Chen, Jianxin Li, Tianyu Wo, Chunming Hu, and Wantao Liu. Resilient virtual network service provision in network virtualization environments. In *Parallel and Distributed Systems (ICPADS), 2010 IEEE 16th International Conference on*, pages 51–58, Dec 2010.

[34] Hongfang Yu, Chunming Qiao, Vishal Anand, Xin Liu, Hao Di, and Gang Sun. Survivable virtual infrastructure mapping in a federated computing and networking system under single regional failures. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pages 1–6, Dec 2010.

[35] Md Mashrur Alam Khan, Nashid Shahriar, Reaz Ahmed, and Raouf Boutaba. SiMPLE: Survivability in multi-path link embedding. In *Network and Service Management (CNSM), 2015 11th International Conference on*, pages 210–218, Nov 2015.

[36] Nashid Shahriar, Reaz Ahmed, Shihabur Rahman Chowdhury, Md Mashrur Alam Khan, Raouf Boutaba, Jeebak Mitra, and Feng Zeng. Connectivity-aware virtual network embedding. In *2016 IFIP Networking Conference (IFIP Networking) and Workshops*, pages 46–54, May 2016.

[37] Shihabur Chowdhury, Reaz Ahmed, Md Mashrur Alam Khan, Nashid Shahriar, Raouf Boutaba, Jeebak Mitra, and Feng Zeng. Dedicated protection for survivable virtual network embedding. *IEEE Transactions on Network and Service Management*, PP(99):1–1, 2016.

[38] Jawwad Shamsi and Monica Brockmeyer. Efficient and dependable overlay networks. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–8, April 2008.

[39] Jawwad Shamsi and Monica Brockmeyer. QoSMap: Achieving quality and resilience through overlay construction. In *Internet and Web Applications and Services, 2009. ICIW '09. Fourth International Conference on*, pages 58–67, May 2009.

[40] Xian Zhang, Chris Phillips, and Xiuzhong Chen. An overlay mapping model for achieving enhanced qos and resilience performance. In *Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2011 3rd International Congress on*, pages 1–7, Oct 2011.

[41] Andreas Fischer and Hermann De Meer. Position paper: Secure virtual network embedding. *PIK-Praxis der Informationsverarbeitung und Kommunikation*, 34(4):190–193, 2011.

[42] Shuhao Liu, Zhiping Cai, Hong Xu, and Ming Xu. Security-aware virtual network embedding. In *2014 IEEE International Conference on Communications (ICC)*, pages 834–840, June 2014.

[43] Leonardo Richter Bays, Rodrigo Ruas Oliveira, Luciana Salete Buriol, Marinho Pilla Barcellos, and Luciano Paschoal Gaspary. Security-aware optimal resource allocation for virtual network embedding. In *Proceedings of the 8th International Conference on Network and Service Management*, CNSM '12, pages 378–384, Laxenburg, Austria, 2013.

[44] 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*, VISA '10, pages 49–56, New York, USA, 2010.

[45] Tae-Ho Lee, Shahnaza Tursunova, and Tae-Sang-Choi. Graph clustering based provisioning algorithm for virtual network embedding. In *2012 IEEE Network Operations and Management Symposium*, pages 1175–1178, April 2012.

[46] Bo Lv, Zhenkai Wang, Tao Huang, Jianya Chen, and Yunjie Liu. Virtual resource organization and virtual network embedding across multiple domains. In *2010 International Conference on Multimedia Information Networking and Security*, pages 725–728, Nov 2010.

[47] Aris Leivadeas, Chrysa Papagianni, and Symeon Papavassiliou. Efficient resource mapping framework over networked clouds via iterated local search-based request partitioning. *IEEE Transactions on Parallel and Distributed Systems*, 24(6):1077–1086, June 2013.

[48] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. Understanding network failures in data centers: Measurement, analysis, and implications. *SIGCOMM Comput. Commun. Rev.*, 41(4):350–361, August 2011.

[49] Vine-yard.

`http://www.mosharaf.com/ViNE-Yard.tar.gz`.

[50] Ellen Zegura, Kenneth Calvert, and Samrat Bhattacharjee. How to model an internetwork. In *INFOCOM '96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation. Proceedings IEEE*, volume 2, pages 594–602 vol.2, Mar 1996.

[51] Mosharaf Chowdhury and Raouf Boutaba. Network virtualization: state of the art and research challenges. *Communications Magazine, IEEE*, 47(7):20–26, 2009.