Towards Security-aware Virtual Network Embedding[☆]

Shuhao Liu[a], Zhiping Cai[a,c,*], Hong Xu[b], Ming Xu[a]

[a]*College of Computer, National University of Defense Technology, Changsha, Hunan, P.R. China, 410073*
[b]*Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong S.A.R., China*
[c]*School of Computer and Software, Nanjing University of Information Science & Technology, Nanjing, Jiangshu, P.R. China, 210044*

Abstract

Network virtualization is one of the fundamental building blocks of cloud computing, where computation, storage and networking resources are shared through virtualization technologies. However, the complexity of virtualization exposes additional security vulnerabilities, which can be taken advantage of by malicious users. While traditional network security technologies can help in virtualized environments, we argue that it is cost-effective to isolate virtual resources with high security demands from the untrusted ones.

This paper attempts to tackle the security issue by offering physical isolation during virtual network embedding, the process of allocating virtual networks onto physical nodes and links. We start from modelling the security demands in virtualized environments by analysing typical security vulnerabilities. A simple abstracted concept of security demands is defined to capture the variations of security requirements, based on which we formulate security-aware virtual network embedding as an optimization problem. The proposed objective and constraint functions involve both resource and security restrictions. Then, two heuristic algorithms are developed to solve this problem with splittable or unsplittable virtual links, respectively. Our simulation results demonstrate their

[☆]Some preliminary results were presented at IEEE ICC 2014.
[*]Corresponding author. Phone No.: +86-139-7519-2193
Email addresses: `liushuhao@nudt.edu.cn` (Shuhao Liu), `zpcai@nudt.edu.cn` (Zhiping Cai), `henry.xu@cityu.edu.hk` (Hong Xu), `xuming@nudt.edu.cn` (Ming Xu)

efficiency and effectiveness.

Keywords:
cloud computing, network virtualization, resource allocation, virtual network embedding

1. Introduction

Network virtualization is one of the most important technologies for cloud computing. Infrastructure-as-a-Service providers exploit virtualization technologies to enable efficient utilization of computing and storage resources. It is also considered a key technology to accelerate innovations and to provide more stable services, by enabling new protocols and topologies to be rapidly implemented upon existing network infrastructures [1]. For example, many research testbeds rely on network virtualization to experiment new architectures [2].

Making efficient use of the substrate resources in virtualization requires effective techniques for virtual network embedding [3]. Virtual networks are usually abstracted as an undirected graph, in which nodes represent Virtual Machines (VMs) and links represent virtual network paths. Virtual network embedding is essentially a resource allocation problem, where a new virtual network, with constraints on the virtual nodes and links (e.g., VM computation resource demands and dedicated link bandwidth), is mapped onto capable physical nodes and paths in the substrate network. Because of the combination of node and link constraints, and the diversity of virtual topologies, virtual network embedding is NP-hard [4], and many heuristic and meta-heuristic algorithms have been developed for specific formulations of the problem.

In this paper, we study virtual network embedding from a different perspective. We consider security, which is an important, yet largely overlooked aspect in the literature. To protect all virtual networks from potential threats and to guarantee information confidentiality and integrity, in many cases users have specific security demands and requirements that have to be satisfied. These requirements are two-fold. On the one hand, virtual networks need to be embedded onto physical nodes and links with a qualified set of protection mechanisms. For example, each VM of a virtual network must be allocated onto an end system with qualified firewalls, certain data encryption functions, etc. Security requirements intuitively make the problem even more difficult due to the additional complexity of considering network resource sharing and vulnerability of current virtualization architectures. On the other hand, studies on virtualization technologies [5] show the possibility of attacks among VMs hosted on the same substrate. We show that it is necessary to offer additional physical isolation between trusted and untrusted virtual resources. However, to the best of our knowledge, there is little work that focuses on the security aspect of virtual network embedding thus far.

In order to make virtual network embedding *security-aware*, we make three concrete contributions. First, we propose a taxonomy of abstractions to properly model the security demands of virtual networks. A concept of security level is introduced to capture the availability of different protection mechanisms in the substrate. Then, the security demand of a virtual node or link is expressed in terms of security levels, and can be satisfied with physical resources that can offer the same or a higher security level. This simple abstraction is general enough to embrace many distinct forms of security mechanisms and requirements.

Second, we develop an optimization framework for security-aware network embedding, by considering both the resources and security demands of virtual networks. We present three objective functions focusing on three different major concerns of network operators: (i) the ratio of virtual networks being successfully embedded, (ii) the long-term revenue and (iii) the revenue to cost ratio of the embedding operations, respectively. Moreover, apart from resource constraints, such as node computation capacity and link bandwidth consumptions, we propose the security constraints, based on the analysis of vulnerabilities in the current virtual network architecture.

Third, we propose two novel heuristic algorithms to solve the security-aware network embedding problem. Given the conventional embedding formulation without security constraints is NP-hard, our problem with security constraints is also NP-hard. Also, in practice with economical concerns, some virtual networks may allow to split their virtual links [6], i.e., having several physical paths to jointly satisfy its bandwidth needs. This relaxation of link mapping constraints offers operators more flexibility.

Our two algorithms are designed for practical cases where splittable links are applied or forbidden, respectively. Specifically, the algorithms are both based on the same heuristic. The notion of the heuristic is to estimate the possibility and capability of each physical node to host a given virtual node and its outgoing bandwidth resources. It involves security satisfaction and node interconnection relationship in the iterative computations. The difference between these two algorithms is the coordination between node and link mapping. One algorithm simplifies the problem by decoupling the node mapping and the link mapping completely, which is known as an *uncoordinated* mapping [1] in the line of research [7, 8, 9, 10]. We further argue that the uncoordinated method simplifies the complexity at the cost of physical resource utilization, especially in cases where virtual links are unsplittable. The other algorithm is proposed to address this issue. Using the same heuristic, it integrates node and link mappings.

Numerical simulations indicate that our algorithms are both efficient and effective. The uncoordinated algorithm works well where virtual networks allow splittable links, with quite low computation complexity. The coordinated algorithm, on the other hand, can achieve high physical resource utilization with higher execution time, even when all virtual links are unsplittable. Synthesizing them is a practical solution in large-scale real-time virtualization systems.

The rest of the paper is organized as follows. Sec. 2 summarizes related works in the literature. In Sec. 3, the security threats and requirements of virtual networks are discussed. Then, we introduce our abstraction of security demands and formulation of the security-aware embedding problems. Sec. 5 proposes our algorithms and Sec. 6 discusses the simulation results. Finally, Sec. 7 concludes the paper.

2. Related Works

Virtual Network Embedding Problems As a resource allocation problem, it basically applies to various network systems with distinct environment settings. In these cases, the formulation of the virtual network embedding problem will be slightly different with specific objectives or constraints. For example, [11] and [12] represents the early attempts for virtual network embedding in the optical and wireless domain, respectively. Soualah et al. study the problem in the cloud backbone scenario [13]. Moreover, to apply the technology to systems in practice, virtual network embedding with loosed or additional constraints are studied. Yu et al. in [6] propose a seminal algorithm that enables link splitting and migration. [14] studies a practical case where a virtual node can be mapped onto several substrate nodes. Su et al. in [10] focus on the energy-aware virtual network embedding problem. Cai et al., in [7], focus on redeploying virtual resources and minimizing the upgrading cost in the scenario of evolving networks. [15] looses the constraints in a more practical way, considering also the time-varying nature of the amount of demanded resources.

Virtual Network Embedding Algorithms A rich literature exists for virtual network embedding. Most work focuses on the general embedding problem, with a similar problem formulation. These proposed algorithms are usually heuristic or meta-heuristic [1], which can be categorized into two major lines. One line of work simplifies the problem by decoupling the node and link embedding process, such as in [6, 9, 14, 16, 17]. The other line, on the other hand, employs special tricks in modeling or designing heuristics to coordinate the node mapping and link mapping stages. The representative ones include [18, 19, 20, 21], etc.

Security in Virtual Networks Security constraints for virtual network embedding have been briefly discussed in the literature. However, to the best of our knowledge, none of the existing works has proposed an effective and applicable way of either formulating or solving the problem, as we explain below.

In [22], Fischer enumerate some security issues of end systems in a virtual network. Three security demands and constraints are concluded. Our paper additionally take link security threats into consideration. Further, we formulate the problem with the intuition of solving these problems during the process of virtual network embedding.

Bays et al. make progress in modeling the security-aware resource allocation in [23]. The authors enumerate several examples of security demands, such as link encryption and exclusiveness among virtual resources. However, a brute-force algorithm is proposed to solve this problem, because the security demands are classified into exclusive sets. We, on the contrary, abstract the security demands in a different way. Besides, the heuristic algorithm proposed herein has much lower complexity, which is practical in production environments.

3. The Security-aware Virtual Network Embedding Problem

In this section, we first study the security issues in virtualized environments. Then, the rationale for ensuring security during the process of virtual network embedding is discussed.

3.1. Virtual Network Embedding Problem

Network virtualization is a powerful tool that enables multiple users to share the same physical resources simultaneously and to exploit abstracted topologies and functions. Due to the requirement of isolation and the limitation of physical resources, the naive way of allocating resources to virtualized components may result in wasting resources. Hence, the *virtual network embedding* problem arises, which studies the way of embedding virtual components to given physical resources effectively and expediently.

Typically, virtual network embedding could be abstracted as a resource allocation problem, to find the optimal mappings between a sequence of *virtual network request*s and a given *substrate network* so that physical resources can be fully utilized. A single virtual network request is defined by its life span and virtual network topology. The life span indicates its demanding start and end time of occupying physical resources. The topology, including virtual nodes and links, is annotated with constraints, representing their demands for host physical resources. Note that a virtual node is typically in the form of an isolated VM, hosted by an end system that is usually abstracted as a substrate node. A virtual link is usually mapped to a substrate path.

Virtual network embedding problems are challenging. On the one hand, embedding operations shall be constrained according to the physical capabilities and the demands of virtual network requests, which makes it an NP-hard problem [4]. QoS constraints are one of the typical forms. On the other hand, its computation efficiency is a practical requirement for network operators due to its real-time nature. Hence, designing heuristics is important.

3.2. Security Issue in Virtualized Networks

Network virtualization can be a two-edged sword in terms of security. As an additional virtualization layer and multiple shared VMs are introduced into the architecture of end systems, more complexity is incurred by network virtualization. Network operators are able to get more flexibility of the network in trade of potential attack vectors in addition [22].

Apart from traditional vulnerabilities, virtual networks suffer from additional security issues in the following aspects: First, VMs would be easily attacked if their physical host was occupied by attackers. The VMs being attacked would not be able to defend themselves, because VMs are always supervised by their hosts in all aspects. Second, an unauthorized VM may attack its host or another VM on the same host. The attacking VM may escape from the rigid confinement created by the virtualization process [22]. The first case can be achieved after gaining administration privileges over the physical host. Third, the attackers can perform side-channel attacks [5] or negatively influence the whole system, e.g., launching Denial-of-Service attacks.

Both links and nodes suffer from additional security issues. Adversaries may influence the physical links in a negative way. For example, if some of the devices on a physical link are not managed by the operator itself, attackers may have the opportunity to perform a wide range of attacks on that link, such as replay attacks and man-in-the-middle attacks. This issue cannot be ignored, as a virtual link with high security demands may be embedded onto substrate resources without adequate protection. As a consequence, it is a necessity to consider security requirements of virtual networks during the process of embedding.

3.3. The Abstraction of Security Constraints

The concept of security level is introduced to reflect different levels of protection provided by the substrate resources. For example, a substrate node that is not accessible from unverified sources will be considered to be better protected than those nodes that are visible to anyone. As such, it should be assigned a higher security level. Another example is that physical links under the direct management of the network operation, e.g., private datacenter links, are likely to have a higher security level compared with public network links. Note that the security level is not necessarily scalar. It can also be formulated as a vector parameter, so as to capture a more complex model of security guarantees.

The variance in security levels comes from two insights. First, due to the heterogeneous customer requirements, operators often deploy heterogeneous hardware in the same substrate for the sake of cost. Second, each time of upgrade, network operators can only upgrade their machines partially, either in hardware or software, so that service is not interrupted. Therefore, a large scale substrate network usually consists of several generations of machines and virtualization softwares, and the latter generations can offer higher security guarantees. Our definition of security levels is sufficient to abstract their difference theoretically.

Security demands then could be expressed in terms of security levels. A certain demand would be satisfied by resources that have been assigned equal or higher security levels, because they are capable of a larger set of protection mechanisms. Higher security demands of a virtual network usually incur higher rate of service fees.

Based on the assumptions above, we identify four abstracted security constraints below, including constraints proposed in [22]. Note that the security

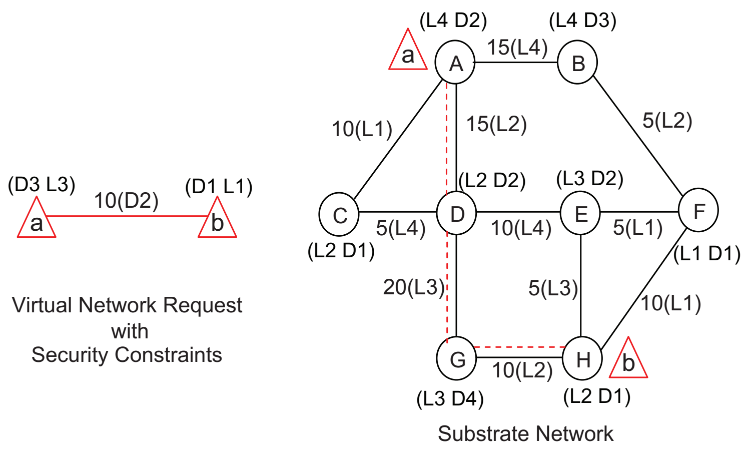

Figure 1: An example of virtual network embedding problems with security constraints.

level of a substrate path will be determined by the minimum level of all links and nodes included:

1. A substrate node should guarantee a security level that is higher than the demand of every guest node.
2. A virtual node should guarantee a security level that is higher than the demand of its host node.
3. Each virtual node should provide all other virtual nodes of the same host with an adequate security level.
4. A virtual link with a security demand d should be hosted by a substrate path with a security level that is higher than d.

Figure 1 depicts a simple example of the security-aware virtual network embedding problem. We assume an adequate computation capacity of every physical node, and we omit them in the figure to emphasize the security constraints. In the figure, `(D3, L4)` at side of a node indicates that the corresponding resource ensure a security level of 4, and require its host or guest to offer a security level not lower than 3. `5(D2)` indicates that the annotated link require 5 unit bandwidth and demand for a host path that offers a security level of at least 2.

Without the security constraints, the example virtual network request could be embedded onto any pair of physical nodes and any path with at least 10 units of bandwidth between them in the figure. However, when considering security, the mapping result is shown by the red dotted line. Virtual node a requires to be mapped to a physical node with security level 3 or higher, but offers a security level 3 only. Therefore, only nodes A, B and E are valid candidates. For a similar reason, virtual node b can only be embedded to one of substrate C, F or H. Considering the virtual link ab, which demands 10 units of bandwidth and security level 2, candidate nodes C, F (for b) are thus eliminated because none of the possible link mappings can satisfy such a demand. Finally, there exist 3 available mappings of the virtual network request ab: $ADGH$, $BADGH$ and $EDGH$. Obviously, $ADGH$ is the one with the lowest cost, because $BADGH$ occupies one more physical link AB and ED is more costly than AD due to higher security level. Above all, $ADGH$ is the optimal mapping for virtual network request ab.

4. Formulation of the Security-aware Virtual Network Embedding Problem

In this section, we formulate the security-aware virtual network embedding problem as an optimization problem. The optimizing objectives and constraints are discussed individually.

4.1. Mathematical Definitions

We present a detailed definition of the network components and parameters in a virtualized network. Based on the mathematical abstraction of the substrate and the virtual networks, formulation of the embedding operation and its utility functions are proposed.

4.1.1. Substrate Networks

A substrate network can be abstracted as a weighted undirected graph $G^S = (N^S, L^S, A_N^S, A_L^S)$, where N_S is the set of all substrate nodes and L_S is the set of all substrate links. A_N^S and A_L^S denote the attributes of substrate nodes and links, respectively. When discussing the security problem, it is reasonable to simplify the distinct attributes to the following settings:

$$A_N^S = \{\{cpu^S(n), dem^S(n), lev^S(n)\} | n \in N^S\},$$
$$A_L^S = \{\{bw^S(l), lev^S(l)\} | l \in L^S\}.$$

For a given $n \in N^S$, $cpu^S(n), dem^S(n)$ and $lev^S(n)$ denote the computation capacity (e.g., in terms of the available virtual CPUs), security demand and security level of n, respectively. Similarly, $bw^S(l)$ and $lev^S(l)$ are the available bandwidth and the security level of each substrate link l ($\forall l \in L^S$). Additionally, we use notation P^S to represent the set of all paths in G^S.

4.1.2. Virtual Network Requests

Virtual network requests are organized in order of arriving time:

$$G^V = [G_1^V, G_2^V, \ldots, G_k^V].$$

The No.i request can be described as another weighted undirected graph $G_i^V = (N_i^V, L_i^V, Time_i^V, Dur_i^V, C_{i,N}^V, C_{i,L}^V)$. It arrives at time $Time_i^V$ and lasts a time period of Dur_i^V. Similar to the description of substrate resource attributes, the virtual network requirements are represented as follows:

$$C_{i,N}^V = \{\{cpu_i^V(n), dem_i^V(n), lev_i^V(n)\} | n \in N_i^V\},$$
$$C_{i,L}^V = \{\{bw_i^V(l), dem_i^V(l)\} | l \in L_i^V\}.$$

For a given $n \in N_i^V$, $cpu^V(n), dem^V(n)$ and $lev^V(n)$ denote the demanded CPU, required security demand and security level, respectively. $bw^V(l)$ and $dem^V(l)$ are the bandwidth demand and the security demand of each virtual link l ($\forall l \in L_i^V$).

4.1.3. The Embedding Operations

Based on the descriptions above, we can define the embedding operations as a set of mappings:

$$\mathbf{M} = \{M_1, M_2, \ldots, M_i, \ldots\}.$$

For all $i \in \{1, 2, 3, \ldots\}$, M_i is the mapping of request G_i^V:

$$M_i : G_i^V \to G_i^S = (N_i, P_i, A_{i,N}, A_{i,L}),$$

where $N_i \subseteq N^S, P_i \subseteq P^S$. $A_{i,N}$ and $A_{i,L}$ represent the attributes of substrate nodes and links in G_i^S, the redundant network of G^S at $Time_i^V$, just before trying to embed G_i^V.

As a consequence, the virtual network embedding problem is to find the best $\mathbf{M}$ so that the objectives of the embedding are achieved when the constraints of the problem are satisfied. Additionally, we use $M_{i,N} : N_i^V \to (N^S, A_{i,N})$ and $M_{i,L} : L_i^V \to (P^S, A_{i,L})$ to describe the two stages of M_i, that is, node mapping and link mapping, respectively.

4.1.4. Other Functions

In order to describe our model explicitly, a two-value function $\rho(i)$ is defined to indicate whether a single request is successfully embedded or not.

$$\rho(i) = \begin{cases} 1, & \text{if request No. } i \text{ is accepted.} \\ 0, & \text{if request No. } i \text{ is denied.} \end{cases} , \forall i \in \{1, 2, \ldots, |M|\}. \tag{1}$$

Additionally, we define two sets of variables $X_i = \{x_{i,qr} | n_q \in N_i^V, n_r \in N^S\}$ and $Y_i = \{y_{i,qr} | l_q \in L_i^V, p_r \in P^S\}$ to simplify the formulation. For a given virtual network request G_i^V, we define $x_{i,qr} \in \{0, 1\}$ as an element of X_i to indicate the node relationships. If a virtual node n_q is mapped onto the substrate node n_r, then $x_{i,qr} = 1$. Otherwise, $x_{i,qr} = 0$. Moreover, $y_{i,qr} \in Y_i$ is defined in a similar way, indicating the ratio of bandwidth allocation, so we have $y_{i,qr} \in [0, 1]$. For example, $y_{i,qr} = 0.5$ means that half bandwidth of virtual link request l_q is mapped onto each substrate link of path p_r.

4.2. The Objective Functions

Operators of virtual networks always try to maximize the long-term profit of virtual network embedding operations, which involves both revenue and cost.

4.2.1. The Revenue Functions

The revenue of a mapping $M_i \in \mathbf{M}$ can be described as follows:

$$\begin{aligned} Rev(M_i) = \rho(i) \times Dur_i^V \times \Bigg[& \sum_{n_i^V \in N_i^V} dem^V(n_i^V) \times cpu^V(n_i^V) \\ & + \sum_{l_i^V \in L_i^V} dem^V(l_i^V) \times bw^V(l_i^V) \Bigg] \,. \end{aligned} \tag{2}$$

Intuitively, a request with higher security demands achieves a higher revenue. Therefore, we take both node and link security demands into account, as is shown in the equation. Note that the function $\rho(i)$ is included, since it is natural to achieve no revenue when the corresponding request is denied.

The overall revenue of virtual network embedding results $\mathbf{M} = \{M_1, M_2, \ldots, M_n\}$ is the sum of revenues of all mappings, i.e.,

$$Rev(\mathbf{M}) = \sum_{i=1}^{n} Rev(M_i)$$

.

4.2.2. The Cost Functions

The cost of a mapping $M_i \in \mathbf{M}$ can be described as follows:

$$\begin{aligned} Cost(M_i) = & \rho(i) \times Dur_i^V \times \Bigg[\sum_{n_i^V \in N_i^V} lev^S(M_{i,N}(n_i^V)) \times cpu^V(n_i^V) \\ & + \sum_{l_i^V \in L_i^V} lev^S(M_{i,L}(l_i^V)) \times len(M_{i,L}(l_i^V)) \times bw^V(l_i^V) \Bigg] \,. \end{aligned} \tag{3}$$

The function $len(p)$ in the equation indicates the number of hops through the path $p \in P^S$. Obviously, it would be a waste and would cost more to occupy substrate resources with an unnecessarily long path and an exorbitant security level. Again, the overall cost of mapping sequence $\mathbf{M} = \{M_1, M_2, \ldots, M_n\}$ is

$$Cost(\mathbf{M}) = \sum_{i=1}^{n} Cost(M_i) \,.$$

4.2.3. The Objectives

Based on the assumptions above, three different objectives of the security-aware virtual network embedding problem can be enumerated as follows:

1. to maximize the request acceptance ratio, that is, the ratio of virtual network requests being successfully embedded;
2. to maximize the long-term revenue of the network;
3. to maximize the long-term Revenue to Cost Ratio (R/C Ratio) of the network.

Generally, to evaluate the embedding results, all of the three objectives are considered with different priorities, which vary among different practical situations. In this paper, we highlight the long-term average revenue as a major consideration, since it is the prioritized objected for most network operators. which has directed our design of the heuristic in Sec. 5.

4.3. The Model of the Security-aware Virtual Network Embedding Problem

Described as an optimization problem, the security-aware virtual network embedding problem is formulated with specific objectives and constraints. The objectives are

$$\max \quad \lim_{n\to\infty} \frac{\sum_{i=1}^{n} \rho(i)}{n}, where \rho(i) = \begin{cases} 1, & \text{if}\, VNR_i \text{ accepted.} \\ 0, & \text{if}\, VNR_i \text{ denied.} \end{cases}, \tag{4}$$

$$\max \quad \lim_{T\to\infty} \frac{\sum_{i=1}^{|\mathbf{M}|} Rev(M_i)}{T}, \tag{5}$$

$$\max \quad \lim_{T\to\infty} \frac{Rev(\mathbf{M})}{T} / \lim_{T\to\infty} \frac{Cost(\mathbf{M})}{T}. \tag{6}$$

We define two arrays of variables, $x_{i,qr}$ and $y_{i,qr}$, in order to clearly formulate the constraints:

$$x_{i,qr} \in \{0,1\}, \text{if } x_{i,qr} = 1 \implies n_r^S = M_{i,N}(n_q^V),$$

$$y_{i,qr} \in [0,1], \text{if } y_{i,qr} > 0 \implies p_r^S \in M_{i,L}(l_q^V).$$

The constraints are

$$\sum_{r=1}^{|N^S|} x_{i,qr} = 1\,, \qquad \forall n_{i,q} \in N_i^V\,, \tag{7}$$

$$\sum_{r=1}^{|P^S|} y_{i,qr} = 1\,, \qquad \forall l_{i,q} \in L_i^V\,, \tag{8}$$

$$\sum_{i=1}^{|N|} x_{i,qr} cpu_i^V(n_{i,q}) \leq cpu^S(n_r)\,, \quad \forall n_{i,q} \in N_i^V, n_r \in N^S\,, \tag{9}$$

$$\sum_{i=1}^{|N|} y_{i,qr} bw_i^V(l_{i,q}) \leq \min_{l_j \in p_r} bw^S(l_j)\,, \quad \forall l_{i,q} \in L_i^V, p_r \in P^S\,, \tag{10}$$

$$x_{qr} dem^S(n_r) \leq lev^V(n_q)\,, \quad \forall n_q \in N_i^V, n_r \in N^S\,, \tag{11}$$

$$x_{qr} dem^V(n_q) \leq lev^S(n_r)\,, \quad \forall n_q \in N_i^V, n_r \in N^S\,, \tag{12}$$

$$\begin{aligned} &\max\{dem^S(n_r), \max_{x_{qr}=1} dem^V(n_q)\} \\ &\leq \min\{lev^S(n_r), \min_{x_{qr}=1} lev^V(n_q)\} \end{aligned}\,, \quad \forall n_q \in N_i^V, n_r \in N^S\,, \tag{13}$$

$$dem^V(l_q) \leq \min_{l_i \in p_r, y_{qr} > 0} lev^S(l_i)\,, \quad \forall l_q \in L_i^V, p_r \in P^S\,. \tag{14}$$

The equation (5) is the formulation of the revenue objective addressed in Sec. 4.2.3. The constraint (7) restricts each virtual node to be mapped onto a single substrate node, while (8) ensures that the bandwidth demand of each virtual link is shared by several substrate paths. The constraint (9) ensures that the host of virtual nodes can satisfy the computation demands of the guests. The constraint (10) ensures that the bandwidth of each substrate link is not over-subscribed. Constraints (11) to (14) correspond to the four security constraints, related with the constraint list in Sec. 3.3.

5. Algorithm Design

In this section, we propose two heuristic algorithms to solve the optimization problem formulated in Sec. 4.3. The embedding problem can be decomposed into a sequence of sub-problems, that is, to embed one single virtual network request onto the redundant substrate. In the following description, we first focus on solving a single sub-problem for simplicity. Subsequently, the method for combining the two algorithms is given.

Both algorithms are based on the same heuristic function, which will be given in Sec. 5.1 along with the intuition analysis. The uncoordinated two-stage algorithm (uSAV) will be presented prior to the one-stage one (cSAV) which is a variable involving node and link mapping coordination.

5.1. The Heuristic

The principle of our algorithms is to design a heuristic for each substrate node to estimate its availability of hosting a given virtual node, so as to guide the node mapping operations. Based on this estimated value, we are able to sort the substrate nodes in a reasonable order and then to perform a best-first search. Apparently, a more precise estimation will result in a better result, i.e., more optimized objectives. The heuristics not only simplify the problem, but also help to avoid bad results in the second stage. By using suitable heuristics, we are able to map the nodes of a virtual network request more efficiently and to achieve a higher success rate as well as lower mapping cost during the link mapping stage. Also, the complexity of calculating this value cannot be too high to enable it to adapt to the frequent redundant network update.

Our idea is a bit similar to the notion of NodeRanking in [8], but with more detailed refinement. Intuitively, a substrate node with more computation capacity and broader outgoing bandwidth would be more available to host virtual nodes. Offering a higher security level would also contribute, but, to avoid high cost based on equation (3), it would be the best to exactly match the security demands. For example, it is a waste to allocate a virtual node whose security demand is low onto a high-security-level substrate node. As a result, the estimated value varies with different levels of security demands, and we need to compute each for different requests.

For a given security demand k, we design two variables to properly model the factors mentioned above. In particular, let n be the substrate node we are focusing on, and $Link(n)$ denote the set of outgoing links of n, so $Link(n) \subseteq L^S$.

$\forall l \in Link(n)$, we define the uniformed bandwidth bw_u_k:

$$bw\_u(l,k) = \begin{cases} bw(l)e^{lev(l)-k}, & \text{if } lev(l) \geq k\,. \\ 0, & \text{if } lev(l) < k\,. \end{cases}$$

Similarly, the uniformed computation capacity of a substrate node n is defined by:

$$cpu\_u(n,k) = \begin{cases} cpu(n)\frac{1-(lev(n)-k)^2}{\delta}, & \text{if } lev(n) \geq k\,. \\ 0, & \text{if } lev(n) < k\,. \end{cases}$$

δ is the coefficient to ensure the positive value of the uniformed computation capacity. For example, that ensures $lev^S(n^S) \in [lev_1, lev_2]$, and node mapping requests that $dem^V(n^V) \in [dem_1, dem_2](\forall n^V \in G^V)$ is satisfied, we get $\delta \geq (lev_2 - dem_1)^2$.

As in [6], we model the intuitive estimated value of a given substrate node n at certain security demand k by using the product of its uniformed computation capacity and collective bandwidth of outgoing links, that is,

$$H^{(0)}(n,k) = cpu\_u(n,k) \sum_{l \in Link(n)} bw\_u(l,k)\,. \tag{15}$$

The result of $H^{(0)}$ is called basic estimation, depending on the related resources of a single node.

Furthermore, to make the estimation more accurate, additional information should be included. We take both virtual and substrate global topologies into consideration. Neighboring nodes interconnection factors in both virtual and physical networks are introduced after analyzing the following examples. First, assume that a virtual node a has already been embedded onto substrate node A. For virtual node b, which connects directly to a in the same virtual network request, the availability of the neighbors of A would undoubtedly get a promotion. The second case is that a substrate node B with little basic estimated value would seem to be more available, only if B had a neighboring C with a high rank of estimation. Moreover, with broader interconnection bandwidth and a higher link security level, B would be more influenced by its neighbor, because the additional cost of link BC would be less.

Thus, an iterative mechanism is introduced. We propose a variable named propagation coefficient of each substrate link, to measure the influence of neighbouring nodes on each other. For the nodes at both end of link l, $PC(l,k)$ is defined to evaluate propagation coefficient at the given security level k:

$$PC(l,k) = \frac{bw\_u(l,k)}{Max\_Prob\_BW} \, .$$

The constant `Max_Prob_BW` denotes the maximum probable bandwidth of all links. The iterative computation process of the heuristic is defined as follows:

$$H^{(t+1)}(n,k) = \lambda \sum_{(m,n)\in Link(n)} PC((m,n),k) H^{(t)}(m,k) + (1-\lambda) H^{(t)}(n,k) \, , \quad (16)$$

where $t = 0, 1, 2, \ldots, \texttt{MaxIte}-1$. λ is a bias factor. A greater λ will require more iterations until it converges, which indicates higher algorithm complexity. A smaller λ might avoid the iteration from convergence. We typically set λ to 0.15 which shows a better trade-off to ensure both convergence and low complexity in our experiments. Additionally, we take the uniformed link bandwidth into account, instead of just focusing on neighboring node resources in [8]. `MaxIte` defines the maximum number of iteration rounds, and we set it to $\lfloor \sqrt{|N^S|} \rfloor$, the square root of the number of substrate nodes, so that topology information would be able to spread out the network.

Typically, the iteration will not stop until the `MaxIte` is reached or the heuristic convergences, e.g., when $|H^{(t+1)}(n,k) - H^t(n,k)| < 0.1$. It is easy to see that if only one node and its adjacent links change their redundant resources, the heuristic recalculation will convergence fast enough. It is a key characteristic of our design, making the heuristic value update procedure less complex.

5.2. uSAV: the uncoordinated Algorithm

We propose a two-stage, uncoordinated Security-Aware Virtual network embedding (uSAV) algorithm. The two stages of the algorithm correspond to the problem decomposition of node mapping stage and link mapping stage. The separation of two stages simplifies the problem, but bad results can be avoided with our heuristic design.

5.2.1. Node Mapping Algorithm

The first stage of uSAV, node mapping algorithm, is described in Algorithm 1. The algorithm is executed to properly embed all virtual nodes of a single virtual network request G_i^V to the redundant substrate G_i^S, while ensuring high revenue and low cost during the second stage of link mapping. That is, for a given request G_i^V, Algorithm 1 aims at getting a proper mapping $M_{i,N}$ based on the heuristic $H^{(MaxIte)}$. $H^{(MaxIte)}$ is calculated by using equations (15) and (16) in terms of current security demand k. Note that if all virtual nodes are successfully allocated, the algorithm will return a state of `NODE_MAP_SUCCESS`; otherwise, it will return a state of `MAP_FAILED`.

Algorithm 1 Node Mapping Algorithm

```
1: For each possible security demand $k$, sort all substrate nodes in a candidate
   node queue $queue(k)$ in descending order of heuristic value $H$.
2: For all nodes $m \in G_i^S$, initialize their state by setting $Occupied(m) =$
   FALSE.
3: repeat
4:    Get an unmapped node $n$ randomly from $G_i^V$.
5:    $k = dem^V(n)$.
6:    if $\exists$ node $m \in queue(k)$ s.t. $Occupied(m) =$ FALSE and $dem^S(m) \leq$
      $lev^V(n)$ and $cpu^S(m) \geq cpu^V(n)$ then
7:       $Occupied(m) =$ TRUE.
8:       Map the virtual node $n$ onto the substrate node $m$.
9:    else
10:      Release all resources occupied by $G_i^V$.
11:      return MAP_FAILED.
12: until all nodes in $G_i^V$ are mapped successfully.
13: return NODE_MAP_SUCCESS.
```

5.2.2. Link Mapping Algorithm

If node mapping stage ends up with a success, we turn to map the virtual links, using Algorithm 2. As virtual nodes have become fixed in the substrate, what we need to do is to find out the available substrate paths among those hosts. Many 2-stage virtual network embedding algorithms make use of k-shortest path algorithm [24] to get the substrate path with least hops, such as [6]. Our algorithm inherits this principle, but presents the following differences: our goal is to get the substrate path between virtual nodes with the lowest cost instead of smallest number of paths hops. To this end, a variable named Path Cost Coefficient (PCC) is designed in the place of paths hops. Considering link security demand k, the Path Cost Coefficient of a substrate path $p \in P^S$ is defined by following equation:

$$PCC(p,k) = \sum_{l \in L^S, l \in p} \left[lev^S(l) - k + 1 \right].$$

The prerequisite of link mapping algorithm is that the state of current request is already `NODE_MAP_SUCCESS`. The constant `MAX_SPLIT_TIME`, which indicates the upper limit of link split times, is defined to avoid the fragmentation of splittable link mapping and to limit the algorithm complexity. The $flag$ parameter is an indicator of overall link mapping state.

5.2.3. The uSAV Algorithm

To be applied in a real-time scenario, dealing with both new-coming and suspended requests, our algorithm is designed to be called once in every fixed time interval. The time interval should be long enough for the process to finish the embedding work of the last process, and be short enough to avoid short-life-span virtual network requests being ignored.

The detailed procedure of uSAV is described in Algorithm 3. First, uSAV scans all of the online virtual network requests. Second, these requests are sorted in descending order of their revenues. This process is the preparing work of greedily deciding embedding priorities. Third, the sub-algorithms of both node and link mapping are called in turn, to try embedding the awaiting virtual

Algorithm 2 Link Mapping Algorithm

```
1:  for each unmapped virtual link $l \in G_i^V$ do
2:    $bw_r = bw^V(l), k = lev^V(l), flag = 0$.
3:    if $l$ is splittable then
4:      $split = 1$.
5:    else
6:      $split =$ MAX_SPLIT_TIME.
7:    Let $m_1, m_2 \in G^S$ be the hosts of both ends of $l$.
8:    repeat
9:      if $\exists p \in P^S$ s.t. $p : m_1 \to m_2$ has the minimum $PCC(p, k)$ then
10:       $bw^S(p) = \min_{t \in p} bw^S(t), t \in L^S$.
11:       Map the remaining resources of $l$ onto $p$.
12:       $bw_r = bw_r - bw^S(p)$.
13:       if $bw_r \geq 0$ then
14:         $flag = 1$.
15:     else
16:       $split =$ MAX_SPLIT_TIME $+ 1$.
17:   until $flag = 1$ or $split >$ MAX_SPLIT_TIME
18:   if $flag = 0$ then
19:     return MAP_FAILED.
20: return MAP_SUCCESS.
```

network request with the maximum revenue and then to refresh the redundant networks. Finally, after trying all requests, the procedure is terminated.

5.3. cSAV: the coordinated Algorithm

uSAV is an uncoordinated algorithm which decouples the node mapping and the link mapping completely. However, it is possible that the optimal node mapping results can lead to high-cost link mappings, despite how well our heuristic is designed. Things get worse when virtual links cannot be split: virtual nodes that are fixed in the substrate would compel the link mapping failure, even

Algorithm 3 The uSAV Algorithm

```
1: For all requests in the time window, set the ones that did not expire yet but
   classified with MAP_FAILED to NEW.
2: Release the substrate resources that were occupied by DONE requests. Re-
   fresh the redundant network.
3: repeat
4:   Get $G_i^V$ that has the maximum revenue from all NEW requests in the time
     window.
5:   Map the nodes of $G_i^V$ using the node mapping algorithm.
6:   if MAP_NODE_SUCCESS then
7:     Map the links of $G_i^V$ using the link mapping algorithm.
8:     if MAP_SUCCESS then
9:       Occupy the substrate resources. Refresh the redundant network.
10:      Set the state of $G_i^V$ to MAP_SUCCESS.
11:      return
12:  Set the state of $G_i^V$ to MAP_FAILED.
13:  Release the occupied resources of $G_i^V$.
14: until There is no more NEW requests in the time window
15: return
```

if the corresponding substrate nodes had rich aggregation bandwidth.

To tackle this issue, we propose another coordinated Security-Aware Virtual network embedding (cSAV) algorithm, which jointly considers node and link mappings. By construction, cSAV can lead to more optimal embedding objectives compared with uSAV, at the cost of higher time and space complexity. Algorithm 4 describes the cSAV algorithm in brief, using a single virtual network request as input for simplicity. Note that both cSAV and uSAV use the same heuristic presented in Equation (16).

Algorithm 4 The cSAV Algorithm: for a single virtual network request

```
Input: The redundant substrate network as $G^S$. The current interested virtual
    network request as $G^V$.
1: Initialize variables: $N$ for maximum back-off times; two empty stacks $S$ (for
   mapped nodes) and $Q$ (for unmapped nodes).
2: Apply the heuristic computation in the virtual network. Find the virtual
   node $m$ with the highest heuristic value $H(m)$. Let node $m' = m$.
3: repeat
4:    Try to map $m$ to $n \in G_S$, starting from substrate node with higher
      heuristic value.
5:    if No applicable substrate node $n$ to host virtual node $m$ and correspond-
      ing virtual link $mm'$ then
6:       if $N == 0$ then
7:          return MAP_FAILED
8:       else
9:          $N \leftarrow N - 1$
10:         $Q.push(m)$, $m = S.pop()$.
11:   $S.push(m)$.
12:   Update the redundant network, recalculating heuristics.
13:   for all Virtual node $m' \in Adj(m)$ in the decreasing order of $H(m')$ do
14:      $Q.push(m')$
15:   $m' = m$. $m = Q.pop()$.
16: until $Empty(Q) ==$ true
17: return MAP_SUCCESS
```

5.4. Time Complexity Analysis

uSAV: As a sub-process to embed a single virtual network request onto the redundant substrate network, both node and link mapping algorithms need to be called once. Based on the descriptions above, we conclude that node mapping is a polynomial-time algorithm. The sub-process of calculating estimated value and sorting has the time complexity of $O(|N^S|^{\frac{3}{2}})$, while node mapping is

$O(|N^S| \cdot |N^V|)$. Also, link mapping can be solved in $O(|N^S|^3 \cdot |L^V|)$. Finally, considering that the procedures of detecting request states and allocating, releasing resources have linear complexity, the uSAV algorithm is a polynomial-time algorithm in terms of $|N^S|$, $|N^V|$, $|L^V|$ and the number of virtual requests. Specifically, the time complexity of embedding a single virtual request is $O(|N^S|^{\frac{3}{2}} + |N^S| \cdot |N^V| + |N^S|^3 \cdot |L^V|)$.

cSAV: In order to calculate the heuristic, $O(|N^S|^{\frac{3}{2}})$ time is required for each update, and the substrate information will be updated once after successfully embedding a virtual node. Also, the link mapping in between will consume $O(|N^S|^3)$ amount of time. Hence, the cSAV algorithm is polynomial in time complexity in terms of $|N^S|$, $|N^V|$, $|L^V|$ and the number of virtual requests. Specifically, the time complexity of embedding a single virtual request is $O(|N^S|^{\frac{3}{2}} \cdot |N^V| + |N^S|^3 \cdot |L^V|)$. Compared with uSAV, the degree of time complexity is relatively higher, which leads to longer execution time.

6. Performance Evaluation

In this section, the performance of the two security-aware virtual network embedding algorithms, uSAV and cSAV, is evaluated. We first describe the different simulation settings, and then present the result of our evaluation. The results show that uSAV works quite well for virtual network requests whose links are splittable. cSAV works even better in all cases; however, the algorithm needs more time to execute.

6.1. Evaluation Environments

We evaluate the uSAV and cSAV algorithms with similar methodologies of previous work. Here we propose the general parameter settings in our simulation, including techniques to generate network topologies and objectives in comparison.

6.1.1. Network Settings

We generate all virtual network requests and substrate network topologies by using the GT-ITM tool [25], using values similar to ones used in [6].

The substrate is set to have 100 nodes and around 500 links, a scale that corresponds to a medium sized ISP. The computation and bandwidth capability of the substrate nodes and links are real numbers uniformly distributed between 50 and 100. The security level of both nodes and links are integer numbers uniformly distributed between 0 and 4. The security demands of substrate nodes are also integers varying from 0 to 4. Note that they are not distributed uniformly because the security demand of a single node cannot be higher than its security level. We generate the security demands as uniformly distributed integers, then normalize all demands that are unreasonably high to be equal to their corresponding security levels.

The number of nodes in each request topology is uniformly distributed between 2 and 20. The average link connectivity rate is 50%, which is determined by the α parameter of GT-ITM. The computation resource and bandwidth requirements of virtual nodes and links are real numbers uniformly distributed between 0 and 50. The virtual network requests are reconfigured by an extension to the GT-ITM tool, which arranges the topologies into a sequence and gives each of them a request time and duration. We assume that requests arrive following a Poisson process with an average arrival rate of 5 requests per 100 time units. The duration of each follows an exponential distribution with an average of 500 time units. Our simulation involves 1500 requests per instance, so that the total time of simulation would be about 30000 time units.

6.1.2. Comparison and Objectives

The three algorithms listed below are included in our evaluation to test the performances. We will compare the evaluation results of all three objectives listed in Sec. 4.2.3, based on the same hardware and software platform.

uSAV: Our uncoordinated Security-Aware Virtual network embedding algorithm, with detailed description in Algorithm 3. The two-stage algorithm is based on the heuristic H, which is calculated by using Equation (15) and (16).

cSAV: Our coordinated Security-Aware Virtual network embedding algorithm, with detailed description in Algorithm 4.

BL: The Baseline algorithm. It is an extension to the project published in [6]. We have simply revised it by adding constraints and updating the computation of embedding revenue and cost, using Equation (2) and (3).

6.2. Evaluation Results and Discussion

In our evaluation, we simulate the on-line virtual network embedding by assigning each virtual network request an initiation time. We use Link Splittable Ratio (LSR) to measure the proportion of virtual network requests which allow splittable virtual links.

6.2.1. Objectives Comparison with High LSR

We generate 6 different test sets of virtual network requests (with Link Splittable Ratio at 80%) and substrate networks to evaluate performance on all of the three optimization objectives. Figure 2 depicts that our algorithms (uSAV and cSAV) achieve much higher R/C Ratio (Figure 2 (a)), request acceptance ratio (Figure 2 (b)) and long-term average revenue (Figure 2 (c)), than the BaseLine algorithm. Compared with uSAV, cSAV can achieve better results, but the advantage (approximately 1 % in terms of request acceptance ratio) is ignorable.

Figure 3 shows the mean algorithm execution time comparison. uSAV exhibits the smaller computational complexity: it takes about 55.9% and 49.5% less time than cSAV and BL, respectively. Provided that its embedding result is near-optimal, uSAV is the most practical of the algorithms under analysis.

6.2.2. Objectives Comparison with Low LSR

With low Link Splittable Ratio (20%) settings, we generate another 6 different test sets of virtual network requests and corresponding substrate networks to evaluate the performance. Figure 4 depicts the comparison of results achieved

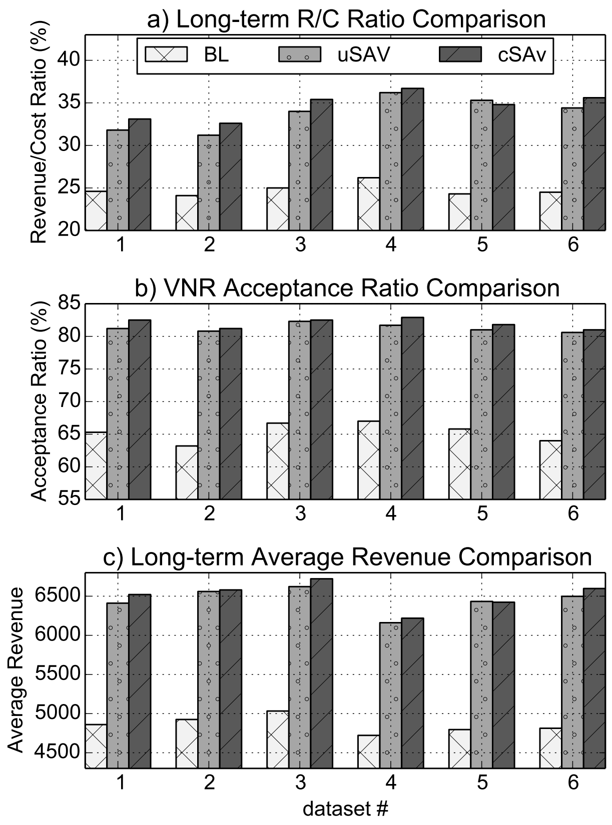

Figure 2: Simulation results of 6 data sets at high Link Splittable Ratio (80%).

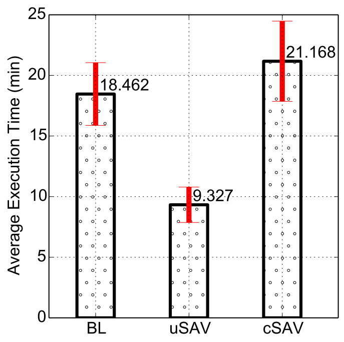

Figure 3: Average execution time comparison for 6 high Link Splittable Ratio data sets.

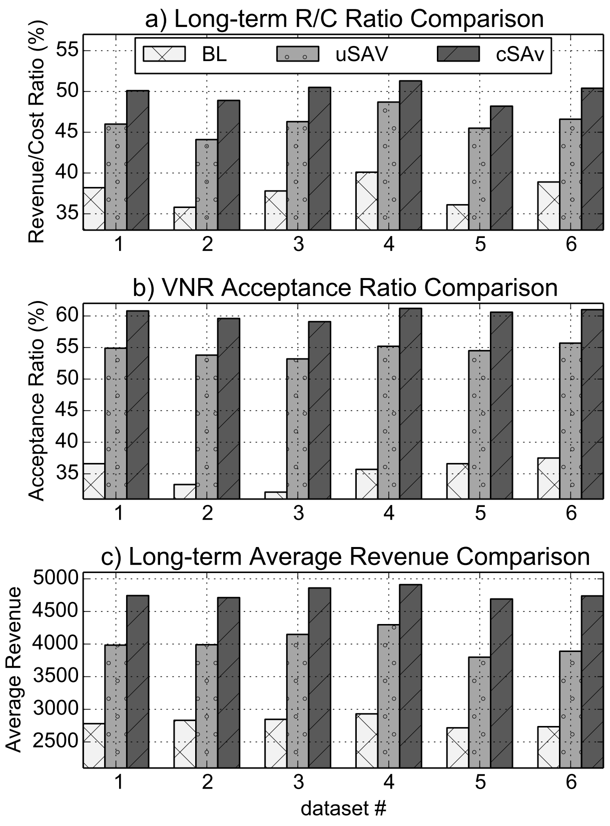

Figure 4: Simulation results of 6 data sets at low Link Splittable Ratio (20%).

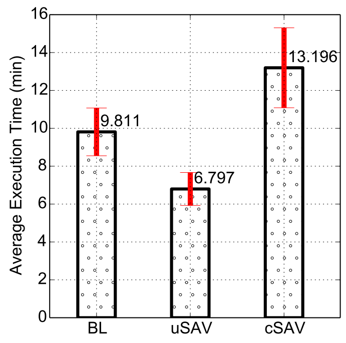

Figure 5: Average execution time comparison for 6 low Link Splittable Ratio data sets.

by the three schemes, which are different from the results in Figure 2. cSAV still achieves better resource mappings here, but the performance of uSAV is the worst, which is about 10.1% lower than cSAV in terms of long-term average R/C ratio. Nonetheless, both our algorithms still outperform the baseline significantly.

In Figure 5, the mean algorithm execution time is listed. Similar to Figure 3, uSAV convergences fast because of low complexity. However, considering its less-optimal performance, cSAV seems a better choice in low link splittable ratio cases.

6.2.3. Performance with Varied Splittable Ratio

In order to evaluate the influence of splittable link ratio on the algorithm process, we generated another 11 sets of requests, with splittable ratios ranging from 0% to 100%. The simulation results are shown in Figure 6. The figure indicates the poor performance of uSAV when few virtual link allows splitting. When less links are splittable, cSAV performs much better because of the coordination of node and link mappings. The conclusion is similar to the one of the above experiments.

Note that both uSAV and cSAV have a slightly lower request acceptance ratio and long-term average revenue when the splittable ratio is very high ($>$ 80%), as compared to medium splittable ratios ($\sim$ 60%). We think the cause

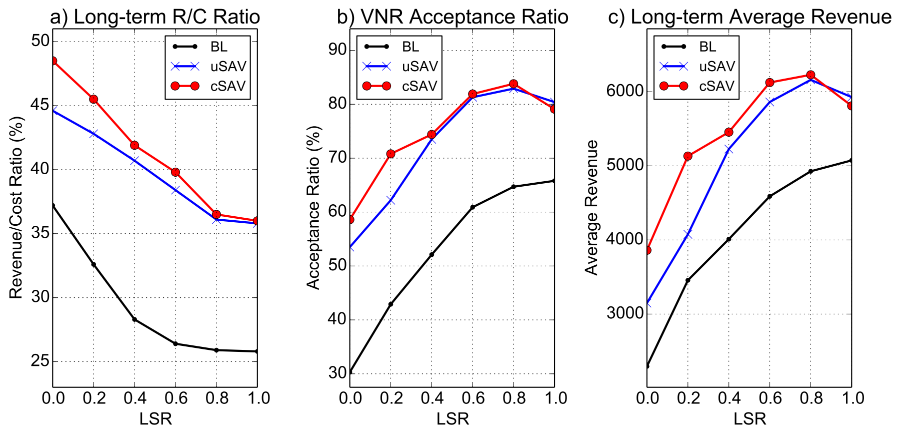

Figure 6: Algorithm performance under varied Link Splittable Ratio.

of this phenomenon is that with almost all virtual links splittable, substrate link resources can be utilized in a fragmented way. In other words, a virtual link tend to be mapped to multiple substrate paths. However, the constant `MAX_SPLIT_TIME`, which is set to 3 as in [8], limits this trend and results in mapping failures. Increasing the value of `MAX_SPLIT_TIME` might be a solution, but it is not practical as virtual link fragmentation may lead to severe out-of-order packets and difficulties in network management. Therefore, it is an acceptable result since an acceptance ratio at ∼80% is good enough in common practice.

7. Concluding Remarks

In this paper, we address the security requirements of virtual network embedding by analyzing the need for physical isolation between trusted and untrusted virtual resources. The numerical concept of security levels and security demands are proposed to properly abstract the security requirements. We formulate the problem as an optimization problem, proposing three objectives with both resource and security constraints. In our heuristic, the innovation of incurring global topologies and interconnection information is highlighted. Two algorithms are designed for virtual network requests with or without splittable links, respectively. The evaluation results demonstrate their effectiveness and practicality.

The computational complexity and effectiveness of the algorithms make of them good candidates for real deployments by network operators. A possible direction of future work is to make the parameter tuning adaptive. The algorithm parameters (e.g., number of iterations, maximum times of link split) is statically chosen to balance the complexity and the optimality. In online cases, we might be able to dynamically adapt these settings to the birth rate of virtual network requests, so as to achieve a higher revenue for network operators.

Acknowledgments

This work is supported in part by the National Natural Science Foundation of China under Grant Nos. 61379145, 61232016, U1405254, 61379144, 61170288, the PAPD fund, HKRGC-ECS 21201714 and CityU HK Start-up grant 7200366. The authors would like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of the paper.

References

[1] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, X. Hesselbach, Virtual network embedding: A survey, IEEE Communications Surveys & Tutorials (2013) 1–19.

[2] J. Carapinha, J. Jiménez, Network virtualization: a view from the bottom, in: Proc. of ACM workshop on Virtualized infrastructure systems and architectures, 2009.

[3] A. Haider, R. Potter, A. Nakao, Challenges in resource allocation in network virtualization, in: Proc. of ITC Specialist Seminar, Vol. 18, 2009.

[4] D. G. Andersen, Theoretical approaches to node assignment, Computer Science Department (2002) 86.

[5] T. Ristenpart, E. Tromer, H. Shacham, S. Savage, Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds, in: Proc. of ACM Conference on Computer and Communications Security, 2009.

[6] M. Yu, Y. Yi, J. Rexford, M. Chiang, Rethinking virtual network embedding: substrate support for path splitting and migration, ACM SIGCOMM Computer Communication Review 38 (2) (2008) 17–29.

[7] Z. Cai, F. Liu, N. Xiao, Q. Liu, Z. Wang, Virtual network embedding for evolving networks, in: Proc. of IEEE Global Communication Conference (GLOBECOM), 2010.

[8] X. Cheng, S. Su, Z. Zhang, H. Wang, F. Yang, Y. Luo, J. Wang, Virtual network embedding through topology-aware node ranking, ACM SIGCOMM Computer Communication Review 41 (2) (2011) 38–47.

[9] Y. Zhu, M. H. Ammar, Algorithms for assigning substrate network resources to virtual network components, in: Proc. of IEEE INFOCOM, 2006.

[10] S. Su, Z. Zhang, X. Cheng, Y. Wang, Y. Luo, J. Wang, Energy-aware virtual network embedding through consolidation, in: Proc. of IEEE INFOCOM Computer Communications Workshops, 2012.

[11] A. Pages, J. Perello, S. Spadaro, G. Junyent, Strategies for virtual optical network allocation, IEEE Communications Letters 16 (2) (2012) 268–271.

[12] P. Lv, Z. Cai, J. Xu, M. Xu, Multicast service-oriented virtual network embedding in wireless mesh networks, Communications Letters, IEEE 16 (3) (2012) 375–377. doi:10.1109/LCOMM.2012.012412.112364.

[13] O. Soualah, I. Fajjari, N. Aitsaadi, A. Mellouk, A reliable virtual network embedding algorithm based on game theory within cloud's backbone, in: Proc. of IEEE International Conference on Communications (ICC), 2014.

[14] S. Zhang, J. Wu, S. Lu, Virtual network embedding with substrate support for parallelization, in: Proc. of IEEE Global Communication Conference (GLOBECOM), 2012.

[15] S. Zhang, Z. Qian, J. Wu, S. Lu, L. Epstein, Virtual network embedding with opportunistic resource sharing, IEEE Transactions on Parallel and Distributed Systems 25 (3) (2014) 816–827.

[16] J. Lu, J. Turner, Efficient mapping of virtual networks onto a shared substrate, Washington University in St. Louis Technical Report (No. 2006-35) (2006) 1–13.

[17] J. F. Botero, X. Hesselbach, A. Fischer, H. De Meer, Optimal mapping of virtual networks with hidden hops, Telecommunication Systems 51 (4) (2012) 273–282.

[18] N. M. K. Chowdhury, M. R. Rahman, R. Boutaba, Virtual network embedding with coordinated node and link mapping, in: Proc. of IEEE INFOCOM, 2009.

[19] L. Wang, H. Qu, J. Zhao, Y. Guo, Virtual network embedding with discrete particle swarm optimisation, Electronics Letters 50 (4) (2014) 285–286.

[20] J. Lischka, H. Karl, A virtual network mapping algorithm based on subgraph isomorphism detection, in: Proc. of ACM workshop on Virtualized infrastructure systems and architectures, ACM, 2009.

[21] D. Yun, J. Ok, B. Shin, S. Park, Y. Yi, Embedding of virtual network requests over static wireless multihop networks, Computer Networks 57 (5) (2013) 1139–1152.

[22] A. Fischer, H. de Meer, Position paper: Secure virtual network embedding, Praxis der Informationsverarbeitung und Kommunikation 34 (4) (2011) 190–193.

[23] L. Bays, R. Oliveira, L. Buriol, M. Barcellos, L. Gaspary, Security-aware optimal resource allocation for virtual network embedding, in: Network and Service Management (CNSM), 2012, pp. 378–384.

[24] J. Y. Yen, Finding the k shortest loopless paths in a network, Management Science 17 (11) (1971) 712–716.

[25] E. W. Zegura, K. L. Calvert, S. Bhattacharjee, How to model an internetwork, in: Proc. of INFOCOM, 1996.