Computer Communications 82 (2016) 13–27

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

A toolset for efficient privacy-oriented virtual network embedding and its instantiation on SDN/OpenFlow-based substrates

Leonardo Richter Bays*, Rodrigo Ruas Oliveira, Luciana S. Buriol, Marinho Barcellos, Luciano Paschoal Gaspary*

Institute of Informatics, Federal University of Rio Grande do Sul (UFRGS), Caixa Postal 15064, 91501-970 Porto Alegre, Brazil

ARTICLE INFO

Article history:
Received 29 July 2015
Revised 1 December 2015
Accepted 13 February 2016
Available online 23 February 2016

Keywords:
Network virtualization
Virtual network embedding
Privacy
Software-defined networking

ABSTRACT

Network virtualization has become increasingly popular in recent years. It has the potential to allow timely handling of network infrastructure requests and, after instantiated, their lifecycle. In addition, it enables improved physical resource utilization. However, the use of network virtualization in large-scale, real environments depends on the ability to adequately map virtual routers and links to physical resources, as well as to protect virtual networks against security threats. With respect to security, mechanisms supporting confidentiality and privacy have become essential in light of recent discoveries related to pervasive electronic surveillance. In this paper we propose a set of tools to efficiently embed virtual networks with privacy support and to allow their real instantiation on top of SDN/OpenFlow-based substrates. This toolset unfolds into three main contributions: (a) an exact VNE model suitable for smaller networks, which also serves the purpose of determining an optimality baseline; (b) a heuristic VNE algorithm, which features precise modeling of overhead costs of security mechanisms and handles incoming requests in an online manner; and (c) a VNE to SDN/OpenFlow translation mechanism, which takes as input the outcome of the heuristic VNE algorithm and produces a set of coherent OpenFlow rules to guide the real instantiation of the mapped virtual networks. We present a detailed performance comparison between the proposed heuristic and the optimization model. The obtained results demonstrate that the heuristic algorithm is able to find feasible mappings in the order of seconds even when dealing with large network infrastructures. Finally, we demonstrate how mappings generated by our heuristic VNE algorithm may be implemented in practice as well as assess the technical feasibility of this process.

1. Introduction

Network virtualization provides a higher level of dynamicity for Infrastructure Providers (InPs), by simplifying the process of network instantiation and removal. This enables InPs to promptly create networks that are specifically tailored to the needs of distinct customers, as well as to improve physical resource utilization. Such features have rendered network virtualization increasingly popular in recent years.

Although the benefits that may be achieved through the employment of network virtualization are clear, its use in large-scale, real environments depends largely on two factors. Firstly, resource allocation must be handled in a timely and optimal manner. This is considered an NP Hard problem, due to its similarity to the multi-way separator problem [1]. Secondly, the shared use of routing devices and communication channels by a number of different entities introduces security-related concerns. Without adequate protection, users from a virtual network might be able to gain unauthorized access to data being transmitted through other virtual networks, violating the privacy of the entities that own those networks [2].

The recent discovery of pervasive electronic surveillance has highlighted privacy concerns in network infrastructures. In the context of network virtualization, which relies on the sharing of physical resources and is used in key contexts such as data centers and network providers, these concerns are even more exacerbated. For this reason, it is extremely important to consider the provision of security mechanisms in order to maintain a desired level of privacy in virtual network environments. Even though security can be implemented on a per-application basis, we deem important to provision it on the link or network layer so that it applies to all traffic and is somewhat native and transparent to users.

* Corresponding authors. Tel.: +55 51 3308 9450.
E-mail addresses: lrbays@inf.ufrgs.br (L. Richter Bays), ruas.oliveira@inf.ufrgs.br (R. Ruas Oliveira), buriol@inf.ufrgs.br (L. S. Buriol), marinho@inf.ufrgs.br (M. Barcellos), paschoal@inf.ufrgs.br (L. Paschoal Gaspary).

http://dx.doi.org/10.1016/j.comcom.2016.02.007
0140-3664/© 2016 Elsevier B.V. All rights reserved.

Although related work exists in the area of virtual network embedding, little has been done towards reconciling efficient resource mapping and satisfaction of security requirements. Early work in the area focuses on features such as path splitting, migration, and improved coordination between router and link mapping [3,4]. Subsequently, some authors have focused on taking into account a wider array of operational constraints related to the instantiation of virtual networks [5] or formulating more advanced strategies for improved resource distribution [6,7]. Last, there have also been efforts towards ensuring different degrees of survivability for embedded networks [8–10]. However, to the best of our knowledge, the issue of taking into account precise overhead costs of privacy-enabling mechanisms in the embedding process has yet to be approached.

The motivation to properly tackle the aforementioned issue is threefold. First, InPs need to be able to host a large number of virtual networks sharing the same physical substrate while preserving the confidentiality of each network, leading to an increased level of privacy. Second, while physical resources need to be efficiently utilized, the amount of resources needed to offer security provisions must be considered in order to not underestimate the capacity requirements of virtual network requests. Third, adequate mappings that meet the previously mentioned requirements must be generated as promptly as possible.

In this paper we consolidate a toolset designed to efficiently embed virtual networks with privacy support and to allow their real instantiation on top of SDN/OpenFlow-based substrates. The toolset includes three main components: an exact privacy-aware VNE model; a heuristic algorithm that is capable of sub-optimally embedding large virtual networks with privacy support in a time frame limited to the order of seconds; and a mechanism that takes as input the outcome of the VNE algorithm and translates it into a set of coherent OpenFlow rules to guide the real instantiation of the mapped virtual networks.

In comparison to our previous seminal work on this subject [11], in this paper we present a substantially improved version, in terms of the quality of the computed mappings, of our simulated annealing-based VNE algorithm. This is accomplished by employing a logarithmic cooling schedule (rather than a linear one), a first-improvement-based local search (as opposed to a random walk), and running multiple local searches per temperature change. As demonstrated in our evaluation, these changes have led to significant progress, rendering the results obtained through the heuristic method notably closer to those produced by the optimal model. Additionally, in this paper we introduce a VNE to SDN/OpenFlow translation mechanism, which is supported by a software tool that enables InP administrators to easily visualize and modify mappings produced by our virtual network embedding approaches (*i.e.*, exact model and heuristic algorithm). More importantly, the mechanism/tool ultimately generates the set of SDN/OpenFlow rules needed to instantiate the desired virtual networks taking into account the proposed mappings and any administrator-made modifications. We demonstrate how mappings generated by our heuristic VNE algorithm may be implemented in practice as well as assess the technical feasibility of this process.

The remainder of this paper is organized as follows. In Section 2 we present our heuristic method, as well as a baseline ILP model. In Section 3 we report the results of an extensive evaluation we carried out, including a detailed comparison of both the heuristic method and the optimal model. In Section 4 we propose and evaluate a mechanism to deploy the computed virtual network mappings on top of real physical networks, using an SDN-based substrate as case study. In Section 5 we discuss virtual network embedding approaches proposed in previous investigations. Last, in Section 6 we present final remarks and perspectives for future work.

2. Privacy-oriented virtual network embedding

In this section, we explain the assumptions behind our proposed privacy-oriented embedding solution, and introduce our ILP formulation and heuristic algorithm. In order to represent the scenario of virtual network embedding with a desired level of accuracy, several details were taken into consideration. We consider a scenario in which an infrastructure provider supplies virtual networks to a number of customers. In order to request the creation of a virtual network, these customers sign a Service Level Agreement (SLA) with the infrastructure provider. This SLA describes the characteristics of the requested virtual network and its security requirements, which must be honored by the provider.

We assume that the infrastructure provider will receive a series of virtual network requests over time. Therefore, these requests must be handled in an online manner, *i.e.*, individually as they arrive. If the substrate has sufficient free resources to embed a request, the output of our model or algorithm indicates the optimal or near-optimal mapping in terms of resource usage, maximizing the amount of free resources available for future requests. If the substrate is not capable of embedding a virtual network due to lack of resources, the request is denied. In practice, we envision that our proposed solution may be used either to automatically handle virtual network requests received by an infrastructure provider (communicating directly with a preexisting virtual network embedding platform) or as an “advisor” (providing candidate mappings to a human operator that may approve, deny, or change such mappings as desired). Moreover, we consider a scenario in which virtual networks are instantiated by means of full virtualization or paravirtualization techniques. Therefore, we do not consider costs of operations associated with other types of network virtualization (such as VLANs).

With regards to security, the main threat considered in this paper is the theft of information through the interception of packets being exchanged between virtual routers. This type of attack is commonly known as “sniffing” or “eavesdropping”. While any network is susceptible to this type of attack, it is an even more significant concern in network virtualization environments. As physical resources are shared among different virtual networks, a single compromised router or link may be used to obtain confidential information from multiple customers. Security mechanisms such as Virtual Private Networks (VPNs) secured by IPSec [12] are able to mitigate this threat through the use of encryption. Therefore, in this paper, we assume that data confidentiality is achieved through this technology.

2.1. ILP model

Before presenting our model, we introduce the syntax for our formulation. Capital letters represent sets or variables, and superscripts denote whether a given set or variable refers to physical (P) or virtual (V) entities, or to routers (R) or links (L). Also, each subscript represents an index associated to a variable or path. For ease of reference, the symbols used in this section are listed in Table A.1 (Appendix A).

Topologies

Virtual network requests must specify the desired topology, *i.e.*, the number of virtual routers in the network and the interconnections between these routers. Physical and virtual network topologies are represented as directed graphs $N = (R, L)$. Each vertex in R denotes a router, and each edge in L denotes a unidirectional link. Bidirectional links are represented as a pair of edges in opposite directions. Each virtual router is mapped to a single physical router, while virtual links may be mapped to either a physical link or a substrate path.

Physical and virtual capacities

Physical routers have limited throughput capacity. In other words, routers are able to handle a limited amount of traffic in terms of bits per second. The throughput capacity of a physical router i is expressed by T^P_i. Likewise, $T^V_{r,i}$ denotes the throughput required by virtual router i from virtual network r. As for physical and virtual links, bandwidth limits are represented by $B^P_{i,j}$ and $B^V_{r,i,j}$ respectively, where (i, j) denotes a link, and r, a virtual network.

Locations

We assume that a majority of customers will request virtual networks that require one or more of its routers to be hosted in specific geographical locations. Therefore, S^P associates routers and locations as follows. Pair $(i, a) \in S^P$ denotes physical router i is situated in location a. Similarly, $(j, b) \in S^V$ denotes virtual router j must be embedded on top of a router in location b. If a virtual router does not have a location requirement, it will not be stored in set S^V.

Security requirements

We believe it is highly important to offer security provisions, as (i) while requesters may freely implement their own security mechanisms, they may desire to have confidentiality guarantees on application traffic that is not encrypted (*e.g.*, HTTP, FTP, or email protocols); and (ii) standard traffic isolation may not adequately protect against eavesdropping. Additionally, such security provisions have the potential to provide a higher level of privacy to users who are not eager to go through cumbersome processes to set up security mechanisms.

To this end, the model allows each virtual network to require three varying levels of security. The first and second levels relate to cryptographic techniques. In the first level, the substrate provides end-to-end cryptography – packet payloads are encrypted and decrypted at the edges of the network –, protecting against data theft. The second level provides point-to-point cryptography – which protects packet headers as well as their payload, requiring decrypting and re-encrypting every packet on each hop –, offering additional protection against traffic analysis attacks. Set K^P_i enables the substrate network to indicate which routers support protocol suites that allow cryptographic operations, such as IPSec [12]. Likewise, set $K^V_{r,i}$ indicates whether particular virtual routers require this feature.

The third and last security level concerns isolation and allows virtual network requesters to indicate other virtual networks that must not share physical resources with their own. In practice, we envision that this would be treated at the SLA level, with each requester declaring with which (types of) institutions it does not wish to share resources. Sets of conflicting virtual networks, which are not allowed to share physical routers and links, are stored in X. More specifically, a pair $(r, s) \in X$ denotes virtual networks r and s cannot share the same physical resources.

Costs of security operations

As security does not come without a price, our model considers not only support for cryptographic protocols but also additional processing and bandwidth costs that arise from the use of cryptographic techniques. Set $W^R_{r,j}$ represents the additional processing cost a virtual router j from network r will demand from the physical router hosting it. This processing cost is modeled as a ratio based on the normal processing cost for a packet that does not require encryption or decryption. As such, the cost per packet is 1.0 if a virtual router does not require cryptographic operations. As the model considers processing overheads for individual virtual routers, this allows virtual networks to request varying cryptographic algorithms and key sizes. It also enables the consideration of different costs depending on the number of cryptographic operations that need to be performed on each packet, as: *(i)* no such operations may be required; *(ii)* only one operation – either encryption or decryption – is needed; or *(iii)* two operations – decryption and re-encryption – may be necessary. In these cases, the processing cost per packet is higher than 1.0 and proportional to the costs associated with the chosen cryptographic algorithm and key size.

As previously stated, in addition to processing overheads, the model considers bandwidth overheads. Set W^L_r represents the additional bandwidth necessary to encapsulate packets in a network r that requires cryptography (in relation to packets that do not require such encapsulation). As end-to-end and point-to-point cryptography impose different bandwidth overheads, W^L may be set to different values for each network. If a network does not require cryptography, this cost is 1.0. If it does, the cost is higher than 1.0 and reflects the overhead caused by the necessary encapsulation method.

Previous mappings

As the model handles virtual network requests in an online manner, it is necessary to consider the mappings of virtual routers and links already embedded on the substrate when a new request is received. Sets $E^R_{i,r,j}$ and $E^L_{i,j,r,k,l}$ denote the mappings of previously embedded virtual routers and links, respectively.

The variables of our model indicate where virtual routers and links are mapped on the substrate.

- $A^R_{i,r,j} \in \{0, 1\}$ – Router allocation, indicates whether the physical router i is hosting virtual router j from virtual network r.
- $A^L_{i,j,r,k,l} \in \{0, 1\}$ – Link allocation, indicates whether the physical link (i, j) is hosting virtual link (k, l) from virtual network r.

Next, we present the objective function (Formula (2.1)) and its constraints (C1–C11). The objective function aims at minimizing the bandwidth consumed by embedded virtual networks, while considering overheads introduced by security provisions.

Objective

$$min \sum_{(i,j)\in L^P} \sum_{r\in N^V} \sum_{(k,l)\in L^V} B^V_{r,k,l} W^L_r A^L_{i,j,r,k,l} \tag{2.1}$$

Subject to

$$\sum_{r\in N^V, j\in R^V} T^V_{r,j} W^R_{r,j} A^R_{i,r,j} \le T^P_i \quad \forall i \in R^P \tag{C1}$$

$$\sum_{j\in R^V} A^R_{i,r,j} \le 1 \quad \forall i \in R^P, r \in N^V \tag{C2}$$

$$\sum_{r\in N^V, (k,l)\in L^V} B^V_{r,k,l} W^L_r A^L_{i,j,r,k,l} \le B^P_{i,j} \quad \forall (i, j) \in L^P \tag{C3}$$

$$K^V_{r,j} A^R_{i,r,j} \le K^P_i \quad \forall i \in R^P, r \in N^V, j \in R^V \tag{C4}$$

$$\sum_{i\in R^P} A^R_{i,r,j} = 1 \quad \forall r \in N^V, j \in R^V \tag{C5}$$

$$\sum_{j\in R^P} A^L_{i,j,r,k,l} - \sum_{j\in R^P} A^L_{j,i,r,k,l} = A^R_{i,r,k} - A^R_{i,r,l} \forall r \in N^V, \quad (k,l)\in L^V, i \in R^P \tag{C6}$$

$$\sum_{q\in N^V,k\in R^V} A^R_{i,q,k} + \sum_{r\in N^V,l\in R^V} A^R_{i,r,l} \le 1 \quad \forall q, r \in X, i \in R^P \tag{C7}$$

$$\left\lceil \frac{\sum_{q\in N^V,(k,l)\in L^V} A^L_{i,j,q,k,l}}{|L^P|} \right\rceil + \left\lceil \frac{\sum_{r\in N^V,(o,p)\in L^V} A^L_{i,j,r,o,p}}{|L^P|} \right\rceil \le 1 \quad \forall q, r \in X, (i,j) \in L^P \tag{C8}$$

$$jA^R_{i,r,k} = lA^R_{i,r,k} \quad \forall (i,j) \in S^P, r \in N^V, (k,l) \in S^V \tag{C9}$$

$$A^R_{i,r,j} = E^R_{i,r,j} \quad \forall (i,r,j) \in E^R \tag{C10}$$

$$A^L_{i,j,r,k,l} = E^L_{i,j,r,k,l} \quad \forall (i,j,r,k,l) \in E^L \tag{C11}$$

Constraint C1 ensures that the maximum throughput capacity of each physical router is not exceeded, considering the throughput requested by virtual routers as well as any overhead costs. Constraint C2 prevents multiple virtual routers from a single virtual network from sharing a physical router. Constraint C3 ensures that bandwidth capacities of physical links will be respected, considering bandwidth overheads in a similar way to constraint C1. Constraint C4 does not allow virtual routers that require cryptographic operations to be mapped to physical routers that do not support such features. Constraint C5 guarantees that each virtual router is mapped to a physical router. Constraint C6 ensures that each virtual link is mapped to a physical path between the routers hosting its source and destination. Constraint C7 ensures virtual routers from conflicting virtual networks will not share physical routers. Constraint C8 applies the same restriction to virtual links from conflicting virtual networks. C8 is nonlinear, and it was presented in this manner for the sake of comprehension. In practice, C8 was linearized by replacing it with the following 3 constraints (using auxiliary variables $Y, Z \in \{0, 1\}$).

$$Y_{q,r,i,j} \ge \frac{\sum_{q\in N^V,(k,l)\in L^V} A^L_{i,j,q,k,l}}{|L^P|} \quad \forall q, r \in X, (i,j) \in L^P \tag{C8.1}$$

$$Z_{q,r,i,j} \ge \frac{\sum_{r\in N^V,(o,p)\in L^V} A^L_{i,j,r,o,p}}{|L^P|} \quad \forall q, r \in X, (i,j) \in L^P \tag{C8.2}$$

$$Y_{q,r,i,j} + Z_{q,r,i,j} \le 1 \quad \forall q, r \in X, (i,j) \in L^P \tag{C8.3}$$

Constraint C9 forces virtual routers with location requirements to be mapped to physical routers in the specified location. Last, constraints C10 and C11 guarantee that the mapping of previously embedded virtual routers and links, respectively, will be maintained.

2.2. Heuristic algorithm

The proposed heuristic algorithm receives the same inputs and produces outputs in the same format (*e.g.*, a list of non-null values in A^R and A^L) as the previously described ILP model. Furthermore, generated solutions are bound by the same constraints. However, instead of exploring the entire solution space searching for the optimal mapping, this algorithm employs Simulated Annealing to iteratively search for solutions, stopping after a maximum number of cycles is reached or when the best solution found meets certain quality criteria.

Simulated annealing works by first generating an initial solution, and afterwards obtaining a similar solution (called a neighbor) in each iteration. Local search strategies are then iteratively applied in an attempt to find neighbors that are better than the current solution according to an evaluation function. If the local search is able to find a better neighbor, the algorithm moves to it. Otherwise, a probability function is used to decide whether the algorithm should move to the new (worse) solution. This possibility of moving to a worse solution aims to prevent the method from getting stalled at a local optimum and potentially missing the global optimum. This probability function takes into account the current annealing temperature, which is initially high and is gradually decreased throughout the process. The purpose of this temperature is to allow more moves to worse neighbors near the beginning of the process (while the temperature is higher) and less towards the end (as it decreases). Regardless of the current solution, the best solution found is always stored separately. Algorithm 1 presents a simplified pseudocode version of our annealing-based solution, and its details are explained next. Should the reader be familiar with metaheuristic algorithms, reading of the detailed explanation of Algorithm 1 may be skipped. Algorithms 2 and 3, however, represent specific procedures

Algorithm 1 Proposed heuristic algorithm.

```
$s \leftarrow generateInitialSolution$
$c \leftarrow evaluateSolution(s)$
$s^{best} \leftarrow s;\ c^{best} \leftarrow c$
$k \leftarrow 0;\ t \leftarrow 1.0$
while $k < k^{max}$ and $c > c^{max}$ do
  for $l \leftarrow 0$ to $l^{max}$ do
    $s' \leftarrow localSearch(s)$
    $c' \leftarrow evaluateSolution(s')$
    if $probability(c, c', t) > random[0, 1)$ then
      $s \leftarrow s';\ c \leftarrow c'$
    end if
    if $c < c^{best}$ and $isFeasible(s)$ then
      $s^{best} \leftarrow s;\ c^{best} \leftarrow c$
    end if
  end for
  $t \leftarrow t \times \psi$
  $k \leftarrow k + 1$
end while
```

Algorithm 2 Function *generateInitialSolution*.

```
for $(i, r, j) \in E^R$ do
  $A^R_{i,r,j} \leftarrow E^R_{i,r,j}$
end for
for $(i, j, r, k, l) \in E^L$ do
  $A^L_{i,j,r,k,l} \leftarrow E^L_{i,j,r,k,l}$
end for
$n \leftarrow N^V$ currently being embedded
for $j \in R^V_n$ do
  $i \leftarrow$ suitable router for $j \in R^P$
  $A^R_{i,n,j} \leftarrow 1$
end for
for $(k, l) \in L^V_n$ do
  $i \leftarrow R^P$ where $R^V_{n,k}$ is mapped
  $j \leftarrow R^P$ where $R^V_{n,l}$ is mapped
  for $(p, q) \in dijkstra(i, j)$ do
    $A^L_{p,q,n,k,l} \leftarrow 1$
  end for
end for
```

Algorithm 3 Function *localSearch*.

```
$neighbors = \varnothing$
$n \leftarrow N^V$ currently being embedded
$j \leftarrow$ random $R^V$ from $N^V$ currently being embedded
$i \leftarrow R^P$ where $R^V_{n,j}$ is mapped
$pRouters \leftarrow$ suitable routers for $j \in R^P \setminus \{i\}$
for $r \in pRouters$ do
    $A'^R \leftarrow A^R$
    $A'^L \leftarrow A^L$
    $A'^R_{i,n,j} \leftarrow 0$
    for $(k,l) \in L^V_n$ do
        if $k = j$ or $l = j$ then
            $pLinks \leftarrow$ set of $L^P$ hosting $(k,l)$
            for $(p,q) \in pLinks$ do
                $A'^L_{p,q,n,k,l} \leftarrow 0$
            end for
        end if
    end for
    $A'^R_{r,n,j} \leftarrow 1$
    for $(k,l) \in L^V_n$ do
        if $k = j$ or $l = j$ then
            $s \leftarrow R^P$ where $R^V_{n,k}$ is mapped
            $t \leftarrow R^P$ where $R^V_{n,l}$ is mapped
            for $(p,q) \in dijkstra(s,t)$ do
                $A^L_{p,q,n,k,l} \leftarrow 1$
            end for
        end if
    end for
    $c' \leftarrow evaluateSolution(A'^R, A'^L)$
    if $c' < c$ then
        return $A'^R, A'^L$
    else
        add $(A'^R, A'^L)$ to $neighbors$
    end if
end for
return random $(A'^R, A'^L) \in neighbors$
```

performed by our approach (referenced in Algorithm 1). Therefore, we deem them essential in order to fully understand our solution.

Function *generateInitialSolution* (line 1) places virtual routers semi-randomly on the substrate, and allocates physical paths between these routers for each virtual link. The details of this function will be explained after the main algorithm is described. The initial solution is then evaluated by function *evaluateSolution* (line 2). This function first checks solution s against the same set of constraints as our ILP model. If s satisfies all constraints, it applies Formula (2.2) (the same calculation performed in the objective function of the ILP model) to evaluate the total amount of bandwidth it consumes.

$$e \leftarrow \sum_{(i,j)\in L^P} \sum_{r\in N^V,(k,l)\in L^V} B^V_{r,k,l} W^L_r A^L_{i,j,r,k,l} \tag{2.2}$$

In contrast, if any constraint is not satisfied, *evaluateSolution* applies a penalty to the evaluation, as shown in Formula (2.3). The penalty is calculated as a function of the number of unsatisfied constraints. In this representation, γ is a constant that defines the severity of the applied penalty, and κ is the number of unsatisfied constraints. γ should always be at least greater than 1.0, as otherwise there will be no penalty if a single constraint is not satisfied. The purpose of this penalty is to allow the comparison between feasible and unfeasible solutions. We will retake this discussion later, after explaining the inner mechanics of the algorithm.

$$e \leftarrow \gamma\kappa \sum_{(i,j)\in L^P} \sum_{r\in N^V,(k,l)\in L^V} B^V_{r,k,l} W^L_r A^L_{i,j,r,k,l} \tag{2.3}$$

The initial solution and its evaluation are then stored as the current best (in s^{best} and c^{best}, respectively – line 3). Afterwards, k, which represents the current iteration, is initialized as 0, and the temperature, t, is initialized as 1.0. (line 4).

Next, the iterative search for solutions is started. This process is composed of an outer loop (which starts at line 5) and an inner loop (which starts at line 6). The outer loop will be explained first, followed by a breakdown of the inner loop.

The purpose of the outer loop is to iteratively explore the solution space while gradually lowering the temperature in each step. It runs until a maximum number of iterations is reached ($k = k^{max}$) or the evaluation of the best found solution is equal to or better than a desired maximum ($c \leq c^{max}$). In our algorithm, c^{max} represents the maximum desired bandwidth a solution may consume in order to be accepted immediately, and is calculated as shown in Formula (2.4). The formula computes the total bandwidth required by virtual network requests and multiplies it by a constant β, which represents the maximum bandwidth overhead allowed in the mapping process. As any virtual link can be mapped to a path composed of two or more physical links, this overhead is commonly present (and also occurs in the ILP model). β may be adjusted according to the interests of the infrastructure provider, varying from a more conservative scenario in which no exceeding resource consumption is tolerated ($\beta = 1.0$) to more relaxed cases where a certain percentage of overhead is allowed ($\beta > 1.0$). If c^{max} is never reached, the iterative algorithm will continue until $k = k^{max}$.

$$c^{max} \leftarrow \beta \sum_{r\in N^V,(k,l)\in L^V} B^V_{r,k,l} W^L_r \tag{2.4}$$

In each iteration of the outer loop, the inner loop (lines 6–15) is ran, followed by a temperature update (line 16). The temperature starts as 1.0 and is exponentially decreased throughout the execution of the algorithm. This is achieved by multiplying the current temperature by a cooling factor ψ within the interval (0, 1). Last, iteration counter k is incremented by 1 (line 17).

The inner loop performs a number of local searches with the same temperature within each iteration of the outer loop. It always runs for a fixed number of iterations (l^{max}). In each iteration, a neighbor solution s' is generated by running a local search on the current solution (line 7). Similarly to *generateInitialSolution*, the details of function *localSearch* will be subsequently described. After a neighbor is generated and selected by this function, it is then evaluated by function *evaluateSolution* (line 8). As previously mentioned, function *evaluateSolution* applies a penalty to solutions with unsatisfied constraints. The purpose of this penalty is to induce the algorithm to move to solutions with less unsatisfied constraints than the current one. Moreover, it discourages moves to solutions with unsatisfied constraints once a feasible solution has been found.

Next, the algorithm calculates the probability of moving from current solution s to neighbor s' (line 9). If the evaluation of the neighbor solution is better than that of the current one, the probability function returns 1.0. In other words, if a newly generated neighbor is better than the current solution in terms of consumed bandwidth, the algorithm will always move to it. Otherwise, a probability is calculated as a function of the ratio between the bandwidth consumed by solutions s and s' (c and c', respectively) and the current temperature t. Our probability calculation, presented in Formula (2.5), is a slightly modified version of the standard calculation used in Simulated Annealing. Lower c/c' ratios

Table 1
Effect of different values of c, c', and t on the probability of moving to a worse solution.

	t		
c, c'	$t = 1.0$	$t = 0.5$	$t = 0.25$
$c = 1.0, c' = 1.5$	0.716	0.513	0.263
$c = 1.0, c' = 2.0$	0.606	0.367	0.135

influence the probability function positively, as in this case $c/c' - 1$ tends to 0. In other words, there is a higher probability of moving to a worse solution if it is only slightly worse than the current. And, as $c/c' - 1$ tends to -1 if the new solution is significantly worse, the probability will be lower. In a similar way, the probability function is also affected by the current temperature. As previously explained, the temperature is initially set to 1.0, causing a higher chance of moving to worse solutions. It is then exponentially decreased, tending to 0 towards the end of the iterative process. As a result, the chances of moving to worse solutions are increasingly lower. This means that the algorithm assumes a riskier behavior at first, and becomes more conservative at an exponential rate.

$$probability(c, c', t) \leftarrow e^{(c/c'-1)/t} \tag{2.5}$$

Table 1 shows an example of how different values of c, c', and t affect the probability of moving to a worse neighbor. When the temperature is 1.0 and c' is 50% worse than c, the probability is 71.6%. If c' is twice as bad as c, however, the probability drops to 60.6%. Considering the same situations with a temperature of 0.5, the probabilities would be, respectively, 51.3% and 36.7%. Last, with a temperature of 0.25, the probabilities would be 26.3% and 13.5%.

As the calculated probability always results in a value between 0 and 1, it is compared with a randomly generated number within the interval [0, 1) in order to decide if the algorithm should move to the new solution. A probability of 0.7, for example, is higher than 70% of all possible random values within this interval, ensuring that, statistically, the algorithm would only make such a move 70% of the time. If $probability(c, c', t) > random[0, 1)$, the move will be made, and current solution s and its evaluation c will be replaced by s' and c', respectively (line 10). Next, if the current solution is feasible and has a better evaluation than the current best, it is stored in s^{best}, and c^{best} is updated (lines 12 and 13), concluding the inner loop.

At the end of the execution of the algorithm, if the best attained solution is a feasible one (*i.e.*, it satisfies all constraints), it is then used to embed the current virtual network. If no feasible solution is found, the virtual network request is denied. Next, we present in further detail the mechanism of our initial solution and local search functions.

Bootstrapping the algorithm with an initial solution

Function *generateInitialSolution*, shown in Algorithm 2, first initializes A^R and A^L with the values from E^R and E^L, respectively (lines 1–6). This ensures that routers and links from previously embedded networks will remain mapped to the same physical network elements. Next, each router from the virtual network currently being embedded is mapped to a suitable physical router (lines 7–11). A physical router is considered suitable if it can satisfy all the requirements of the virtual router. More specifically, (i) it must have enough residual throughput capacity to host the virtual router; (ii) should the virtual router have a location requirement, the physical router must be situated in the requested location; and (iii) if the virtual router needs to encrypt and/or decrypt packets, the physical router must support cryptographic protocols. Subsequently, each link (k, l) from the current virtual network is mapped to a physical path between the physical routers where virtual routers k and l are mapped (lines 12–18). The physical path is created using Dijkstra's algorithm (line 15). When running this algorithm, the weight of each physical link is set according to Formula (2.6). In this formula, $consumed(B^P_{i,j})$ represents the ratio of consumed bandwidth in physical link (i, j) (*e.g.*, 0.0 if its resources are completely free and 1.0 if its resources are completely depleted). As a result of this function, the weight equals 1 if bandwidth resources are completely free, and quadratically increases up to 10 if some or all bandwidth resources are depleted. This weight calculation aims to favor the selection of physical paths with greater amounts of free resources.

$$weight(i, j) \leftarrow consumed(B^P_{i,j})^2 \times 9 + 1 \tag{2.6}$$

Searching locally for a better solution

Function *localSearch*, depicted in Algorithm 3, receives the current solution as a parameter and runs a first-improvement-based local search on it. As such, this function ultimately returns the first neighbor it finds that has a better evaluation than the current solution or, if no neighbors are better than the current solution, a randomly selected neighbor is returned. First, a list of neighbors is initialized as empty (line 1). Then, a router from the virtual network currently being embedded is selected in a random manner (lines 2–4). After this step, a list of suitable physical routers for the randomly selected virtual router is determined (line 5). As shown in the algorithm, this list excludes the physical router that was previously hosting the selected virtual router. Afterwards, an iterative process begins (line 6) where the selected virtual router is moved to each suitable physical router, generating a neighbor solution. In each iteration, a copy of A^R and A^L from the current solution is created (represented as A'^R and A'^L, respectively – lines 7–8). In this copy, the mapping of the randomly selected virtual router is removed (line 9), as well as the mapping of all virtual links associated with this router (lines 10–17). The virtual router is then mapped to another physical router (line 18), and all links associated with it are reconstructed using Dijkstra's algorithm (in order to point to the new physical location of the virtual router – lines 19–27). This concludes the generation of a neighbor solution. This solution is then evaluated (line 28). If its evaluation is better than that of the current solution, the function is immediately interrupted, returning this neighbor (lines 29–30). Otherwise, it is added to the list of neighbors (lines 31–32). If none of the generated neighbors is better than the current solution, a randomly selected neighbor is selected from the list and returned (line 35). As previously explained, this practice aims at preventing the heuristic from getting stalled at a local optimum, enabling it to potentially follow a different path that leads to a better solution.

3. Evaluation

In this section, we describe the workloads used for the performance evaluation, and present a detailed comparison between the heuristic-based approach and the optimal model. All experiments were performed in a machine with four AMD Opteron 6276 processors, 64GB of RAM and Operating System Ubuntu GNU/Linux Server 11.10 x86_64. The heuristic algorithm was implemented in Java, while the ILP model was implemented and run in the CPLEX Optimization Studio (version 12.3).

3.1. Workloads

The workload for each experiment is generated by a simulator developed by the authors, which randomly creates virtual network requests according to a series of parameters. The simulator is run for 1000 rounds and generates one new virtual network request per round. If accepted, requests remain embedded for 25 rounds before being deallocated.

Fixed parameters

In all experiments, physical routers have a throughput capacity of 10 Gbps. Likewise, all physical links have a bandwidth capacity of 10 Gbps. 95% of the physical routers support protocols that enable the provision of cryptographic operations. Considering modern network environments, we deem it realistic to assume that only a small number of (legacy) devices do not support such protocols. All physical routers are equally distributed among 16 locations.

In virtual network requests, router throughput and link bandwidth requirements are set to 3.333 Gbps. 35% of the requests do not demand any type of cryptography, while 35% require end-to-end cryptography, and the remaining 30%, point-to-point cryptography. Further, 5% of all requests demand that the mapping of its virtual network must not overlap with another network, randomly chosen among currently embedded networks. As there are no previous studies characterizing the proportion of different security levels used in production networks at the VN/flow level, these parameters were chosen considering a hybrid environment. Two of the nodes of each virtual network are edge routers. These routers have a randomly generated location requirement.

Physical and virtual topologies are generated with BRITE using the Barabási-Albert (BA-2) model [13]. Topologies created using this model follow characteristics widely observed in real networks, such as organic growth and preferential attachment. Thus, this model is able to emulate backbone networks with a high degree of fidelity. Moreover, the specific features of this type of topology represent an additional challenge in the process of virtual network embedding when compared to randomly generated networks. The choice to employ this topology model in our evaluation is in line with previous work in the area of virtual network embedding and related research topics [5,14,15].

Variable parameters

In addition to the aforementioned fixed parameters, experiments have a number of varying parameters. Physical networks had two sizes, namely 100 and 500 physical routers. In experiments that use a physical network of size 100, virtual networks have a total of 5 routers each. For experiments with physical networks of size 500, each virtual network contains 10 routers. The smaller infrastructure size (100) is in line with existing literature in the area of virtual network embedding, while the larger size (500) is closer to what one would observe in real mid- to large-sized backbone networks.

Another varying parameter is the key size used for cryptographic operations. In the experiments, we considered the AES cryptographic algorithm with 128 and 256-bit key sizes. In addition to being widely used in real environments, AES-128 and AES-256 are considered to provide a substantially high level of security. Both are recommended by the National Institute of Standards and Technology (NIST) of the United States Department of Commerce without a set expiration date, indicating that they should provide an adequate level of protection for the foreseeable future [16]. While no attacks targeting AES with the aforementioned key sizes are considered computationally feasible, using a key size of 256 rather than 128 elevates the computational complexity of a widely known attack (the biclique attack) from $2^{126.1}$ to $2^{254.4}$ [17]. As a tradeoff, networks that are protected by this form of encryption (*i.e.*, those that require end-to-end or point-to-point cryptography) experience throughput and bandwidth overheads, which vary according to the selected key size. The values for these overheads were set based on the characterization presented by Xenakis et al. [18], and will be explained next.

Bandwidth overheads vary depending on whether virtual networks request end-to-end or point-to-point cryptography. Considering packet sizes of 1536 bytes, the overhead imposed by IPSec encapsulation is of 10.8% for transport mode (end-to-end) and 12.5% for tunnel mode (point-to-point). Therefore, input W^L was set to 1.108 in networks that require end-to-end cryptography, and 1.125 in networks that require point-to-point cryptography.

Throughput overheads depend on the cryptographic algorithm used, key size, router CPU performance, and the number of operations a router needs to perform on each packet. As previously mentioned, the selected combinations of algorithm and key size were AES-128 and AES-256. Throughput overheads for networks requiring end-to-end or point-to-point cryptography were also set in line with benchmark results presented by Xenakis et al. [18] with routers capable of performing 100 million instructions per second (MIPS). These values are described next.

In networks that require end-to-end cryptography, edge routers need to perform one cryptographic operation on each packet – either encryption or decryption. For such routers, in networks that require AES-128, W^R was set to 1.222, while in networks that require AES-256 it was set to 1.375. This is the overhead generated by the decryption operation, which has a higher cost in relation to the encryption operation. We consider the operation which has the highest cost in order to not underestimate this overhead. The remaining routers in such networks do not need to perform any cryptographic operations.

Edge routers in networks that require point-to-point cryptography also need to perform only one cryptographic operation per packet. Therefore, the same overhead values were used as for end-to-end cryptography. However, in order to provide point-to-point cryptography, core routers need to decrypt and reencrypt each packet. For this reason, overhead costs for these routers were set as the aggregated costs of both operations. Namely, W^R was set to 1.222 in networks that require AES-128, as the overhead cost of the encryption operation is negligible, and for the decryption operation it is 22.2%. Further, it was set to 1.532 in networks that require AES-256, as the overhead costs of encryption and decryption are respectively 15.7% and 37.5%.

All experiments were carried out for both the ILP model and the heuristic algorithm. However, when attempting to use the ILP model for experiments with physical networks of size 500, individual requests took approximately 24 h to be processed. Therefore, such experiments were canceled, as the ILP model was deemed unsuitable for these workloads. Additionally, the heuristic algorithm was configured with the parameters shown below. These parameters were set based on preliminary tests as well as results obtained from experiments performed using the ILP model.

- $k^{max} = 1{,}000$ (maximum number of outer iterations);
- $l^{max} = 1{,}000$ (maximum number of inner iterations);
- $\beta = 2.0$ when using a physical network of size 100, $\beta = 3.0$ when using a physical network of size 500 (maximum bandwidth overhead tolerated in a given solution in order to terminate the iterative process before k^{max} is reached);
- $\gamma = 100$ (penalty for unsatisfied constraints);
- $\psi = 0.92$ (cooling factor).

3.2. Results

First, we analyze the average time needed by each approach to reach a solution. Fig. 1 depicts the aggregated average solution time in each experiment, which encompasses solution times observed from the beginning of the experiments until each round. For physical networks with 100 routers (represented in the legend as 100r), the heuristic algorithm takes on average 57.9 s considering the AES algorithm with key size of 128 bits, and 55.2 s considering AES-256. In contrast, the ILP model takes approximately 8.8 and 9.2 s for both cases, respectively. Although CPLEX is able to handle requests faster using the ILP model for networks of this size, the

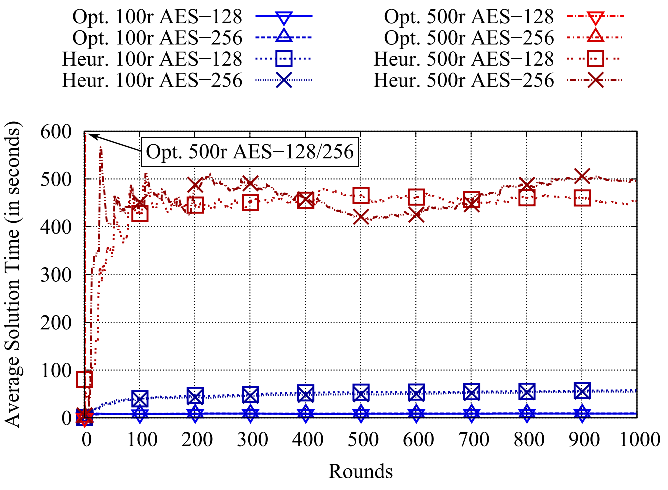


Fig. 1. Time needed to find the accepted solution in each experiment.

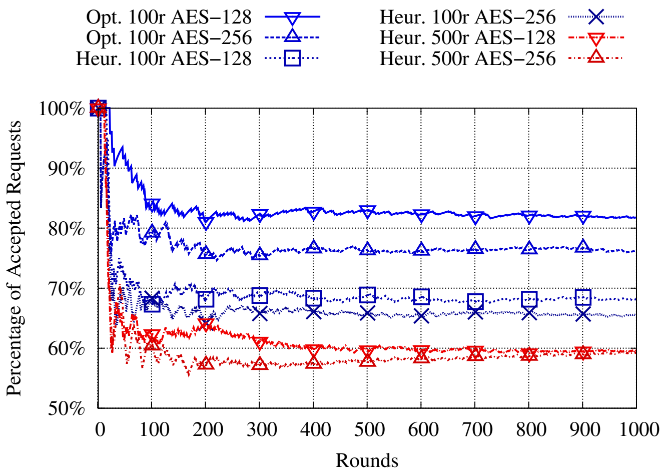


Fig. 2. Acceptance rate in all completed experiments.

heuristic algorithm is still able to embed virtual networks in the order of seconds.

Further analysis of Fig. 1 reveals that, for physical networks with 500 routers, the heuristic algorithm is able to find a solution after approximately 453 s on average considering AES-128, and 495.7 s considering AES-256. In other words, as the physical network size was multiplied by 5 and the maximum virtual network size was multiplied by 2, solution times remained in the order of a few minutes. This shows that the heuristic algorithm is able to scale to larger physical networks while exhibiting good performance. Furthermore, average solution times observed when considering different key sizes (*i.e.*, AES-128 and AES-256) on an equivalent workload remain similar, with a difference of less than 10% between them. Additionally, all aforementioned experiments exhibit lower solution times towards the beginning, as substrate resources are initially 100% free. After a number of rounds, which varies for each experiment, solution times become stable.

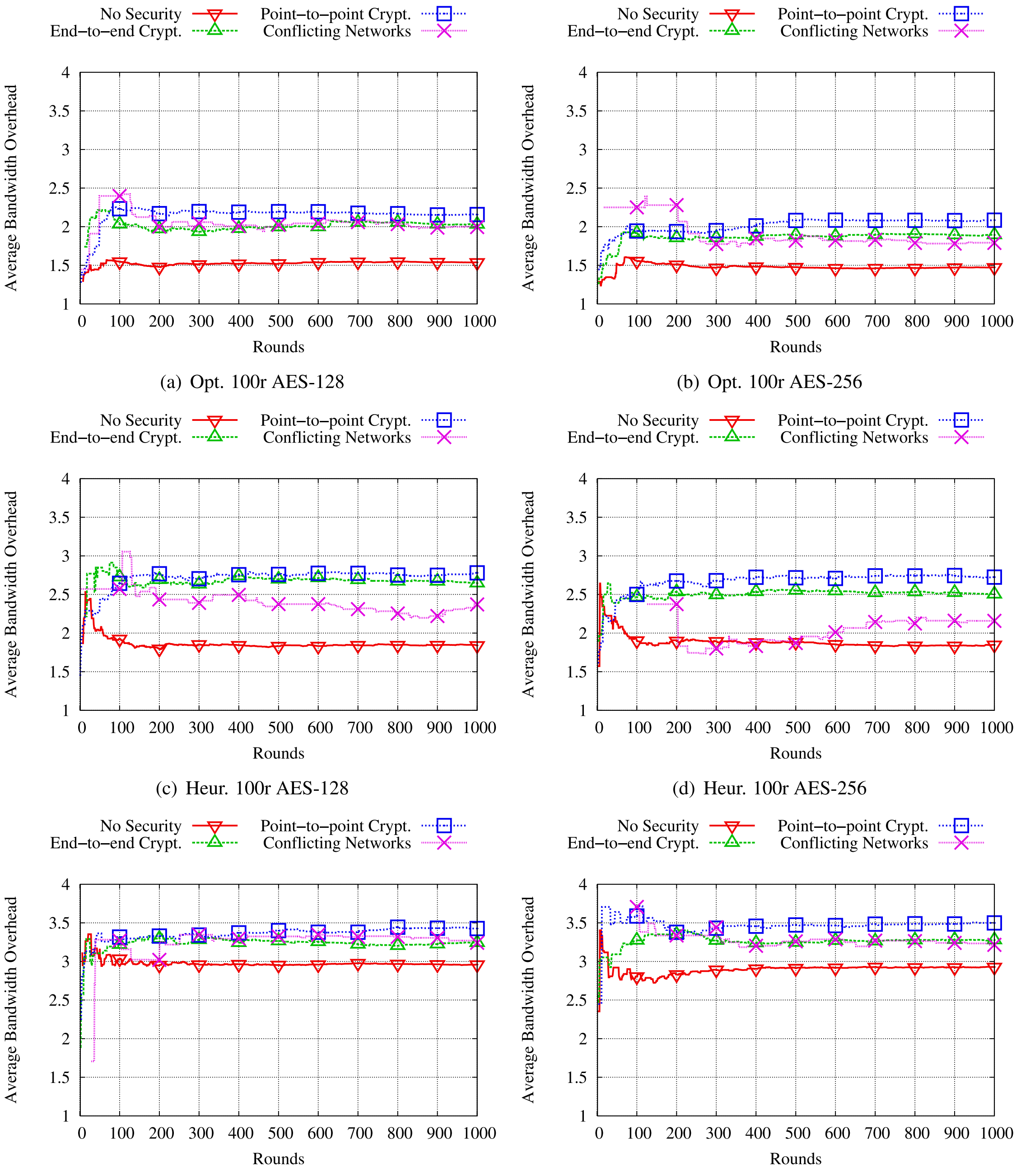
As previously stated, the ILP model was considered unsuitable for physical networks with 500 routers, taking several hours to produce an optimal solution after receiving a virtual network request. For this reason, the corresponding experiments were terminated after 10 rounds. While the graph shown in Fig. 1 was not scaled to accommodate solution times found in such experiments (as doing so would significantly compress the curves related to other experiments), the results obtained in the first 10 rounds are represented in this particular graph. The average solution time in these scenarios was approximately 24 h, producing two overlapping vertical lines next to the *Y* axis.

Next, we analyze the average acceptance rate achieved in each experiment, shown in Fig. 2. With physical networks of size 100, the ILP model achieves average acceptance rates of 81.8% and 76.2% considering virtual networks requiring AES-128 and AES-256, respectively. Using the same physical network and virtual network requests generated for the aforementioned experiments, the heuristic algorithm is able to achieve acceptance rates of 68.2% and 65.5%, respectively. As the ILP model produces optimal results in terms of minimal bandwidth usage, it is able to preserve the maximum amount of resources for subsequent allocations, therefore leading to higher acceptance rates. Lower acceptance rates achieved by the heuristic algorithm can also be explained by the fact that its parameterization in the performed experiments is somewhat permissive in terms of bandwidth overhead ($\beta = 2.0$). Better results may be achieved by decreasing β or increasing the maximum number of iterations (k^{max}), at the cost of possibly increasing solution time. Fine-tuning this parameter to its optimal value depends on a number of factors, such as the interests of the InP, the size of the infrastructure, and the amount of available resources. Therefore, it is out of the scope of this paper to stress-test this parameter.

For physical networks with 500 routers, the acceptance rates were lower than those observed in other experiments (approximately 59% for both AES-128 and AES-256). This can be explained by the increased complexity in these scenarios. While the physical network in these experiments is larger, the maximum size of virtual network requests was also increased from 5 to 10 virtual routers, which significantly increases the amount of resources demanded by virtual networks. Moreover, the number of possible mappings for each network is also significantly increased, likely demanding more iterations in order to find feasible solutions. In all experiments, the use of AES-128 leads to marginally higher acceptance rates in relation to AES-256 (up to 7.3%). Additionally, all depicted experiments exhibit greater variations towards the beginning. The acceptance rate is initially high as the substrate has enough free resources to embed all requests, and starts decreasing once resources become scarce. Eventually, acceptance rates stabilize as previously embedded networks expire. Once this happens, they are removed from the substrate and its resources are released for new requests.

The results analyzed thus far reveal that the ILP model is very well suited for physical networks with up to a hundred routers, providing optimal results within the order of seconds. In contrast, the heuristic algorithm is able to scale to larger infrastructures, producing high quality mappings in the order of minutes. Therefore, an infrastructure provider may choose the approach that better suits its environment and, additionally, parameterize the heuristic algorithm in order to favor either higher quality mappings or faster solution times.

In Fig. 3 we present the ratio between the bandwidth needed to embed each virtual network and the bandwidth requested by such network. A ratio of 1.0 indicates no overhead, which is only observed when each virtual link is mapped to a single physical link (*i.e.*, no virtual links are mapped to paths composed of multiple physical links). As expected, virtual networks with no security requirements generate the least amount of overhead. In experiments with a physical network composed of 100 routers performed with the ILP model (Fig. 3a and b), the average ratio is approximately 1.5. Using the same workload with the heuristic algorithm (Fig. 3c and d), the average ratio is approximately 1.8. In the remaining experiments (Fig. 3e and f), which use a physical network of size 500 and virtual networks with a maximum of 10 routers, this ratio is approximately 2.9.



(a) Opt. 100r AES-128

(b) Opt. 100r AES-256

(c) Heur. 100r AES-128

(d) Heur. 100r AES-256

(e) Heur. 500r AES-128

(f) Heur. 500r AES-256

Fig. 3. Ratio between the bandwidth allocated by the substrate network in order to embed each network and its requested bandwidth. Each graph shows this ratio for each type of request in one experiment.

The marginally higher bandwidth overhead ratios observed when using the heuristic algorithm can be attributed to the tolerance introduced by constant β, since in our experiments the algorithm stops searching for better solutions when the ratio is less than or equal to 2.0 on physical networks of size 100 and 3.0 on physical networks of size 500. If an infrastructure provider is able to tolerate longer solution times, possibly in the order of a few hours (in contrast to seconds or minutes, as observed in our evaluation), β may be set to lower values, potentially leading to ratios which are close to optimality. We emphasize that, in the experiments shown in this paper, this parameter was adjusted in order to favor lower execution times rather than acceptance rates.

The overhead generated by virtual networks requiring end-to-end cryptography considering a physical network of size 100 is approximately 2.0 for the ILP model (Fig. 3a and b) and 2.6 for the heuristic algorithm (Fig. 3c and d). In the remaining experiments (Fig. 3e and f), the ratio is approximately 3.3. Bandwidth overhead ratios for virtual networks requiring point-to-point cryptography in the aforementioned experiments are 2.1, 2.7, and 3.5, respectively. This shows that the use of stricter security mechanisms generates slightly higher overheads. Meanwhile, the difference in terms of bandwidth overhead generated by different key sizes has not been mentioned, as it was negligible.

As for conflicting networks, despite requiring their virtual routers and links to be mapped on different physical devices, the overhead caused by this constraint is less significant than that caused by other security constraints. In experiments with physical networks of size 100 performed on the ILP model, conflicting

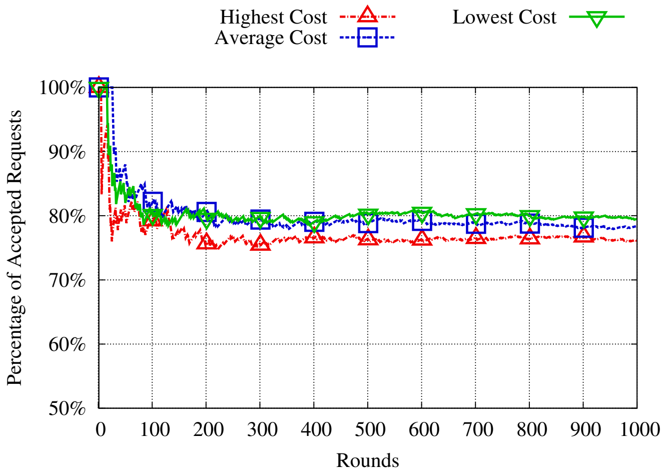
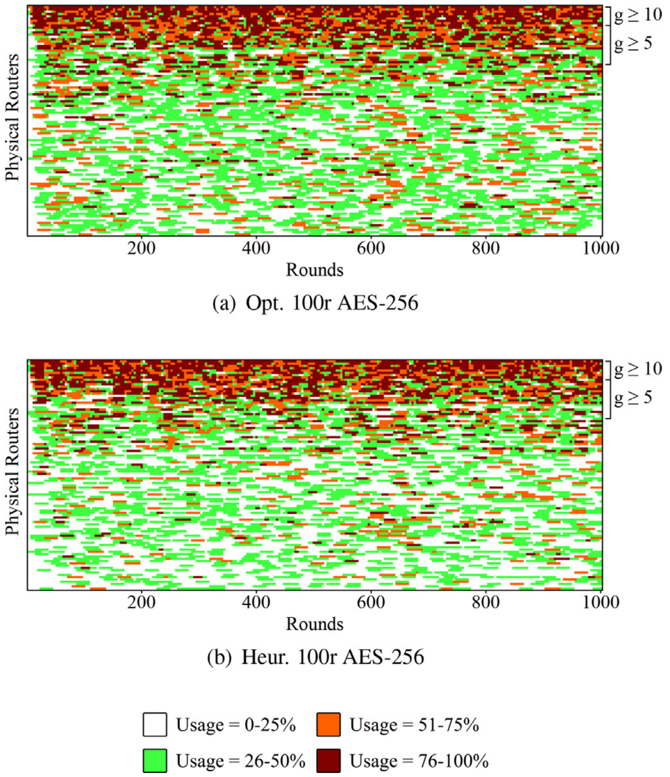


Fig. 4. Comparison of the acceptance rate observed when considering the highest, average, and lowest cost of cryptographic operations on edge routers.

networks caused average bandwidth overheads of 2.0 and 1.8. In experiments performed with the heuristic algorithm and 100 physical routers, the averages were 2.4 and 2.2, while with 500 physical routers, averages were 3.24 and 3.21. In all cases, the higher overhead values refer to experiments that consider AES-128 rather than AES-256. While it may seem counterintuitive that overheads were higher in experiments using the smaller key size, we would like to emphasize that this parameter only affects router throughput, and therefore does not affect bandwidth overheads directly. Instead, these marginally higher overheads are likely a consequence of the higher acceptance rates associated with smaller key sizes. Additionally, it is noticeable that overheads for this type of request oscillate more during the experiment than others. This happens because networks with this requirement may also demand end-to-end or point-to-point cryptography, which implies security provisions that affect bandwidth overhead in a more significant manner.

As previously mentioned, in all experiments thus far, we have taken into account the cost of the most computationally expensive operation (*i.e.*, decryption) when dealing with edge routers of networks with security requirements. This is done in order to prevent any underestimation of overhead costs. We now analyze the impact of this strategy on the observed acceptance rate in relation to considering the average cost of encryption and decryption operations as well as the cost of the least computationally expensive one. Fig. 4 shows that the impact of the strategy employed in our evaluation has a very limited impact on the acceptance rate. Using an average of both costs caused an increase in the acceptance rate from 76.2% to 78.4%, while considering only the lowest cost led to an acceptance rate of 79.5%. Bandwidth overheads in these cases are in line with the changes observed in the acceptance rate, with negligible effects. As demonstrated by these results, the impact of taking into account the highest possible cost is minimal. Therefore, we consider that the benefits of ensuring no underestimation of overhead costs will occur outweigh any potential drawbacks.



(a) Opt. 100r AES-256

(b) Heur. 100r AES-256

Fig. 5. Throughput usage of each router at the end of each round. The vertical axis represents all routers in the network, while the horizontal axis represents simulation rounds.

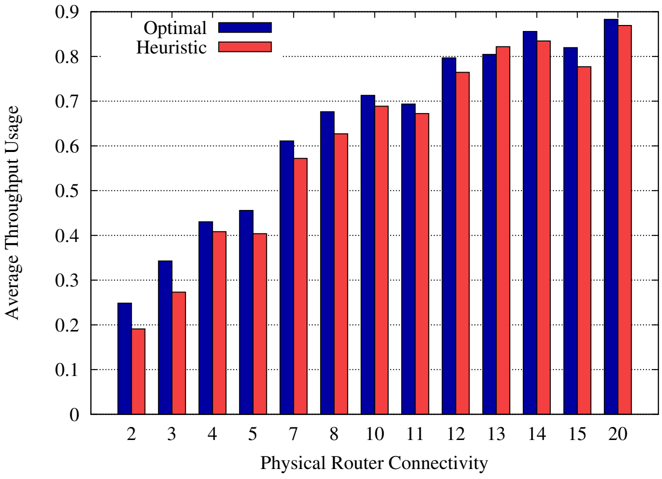
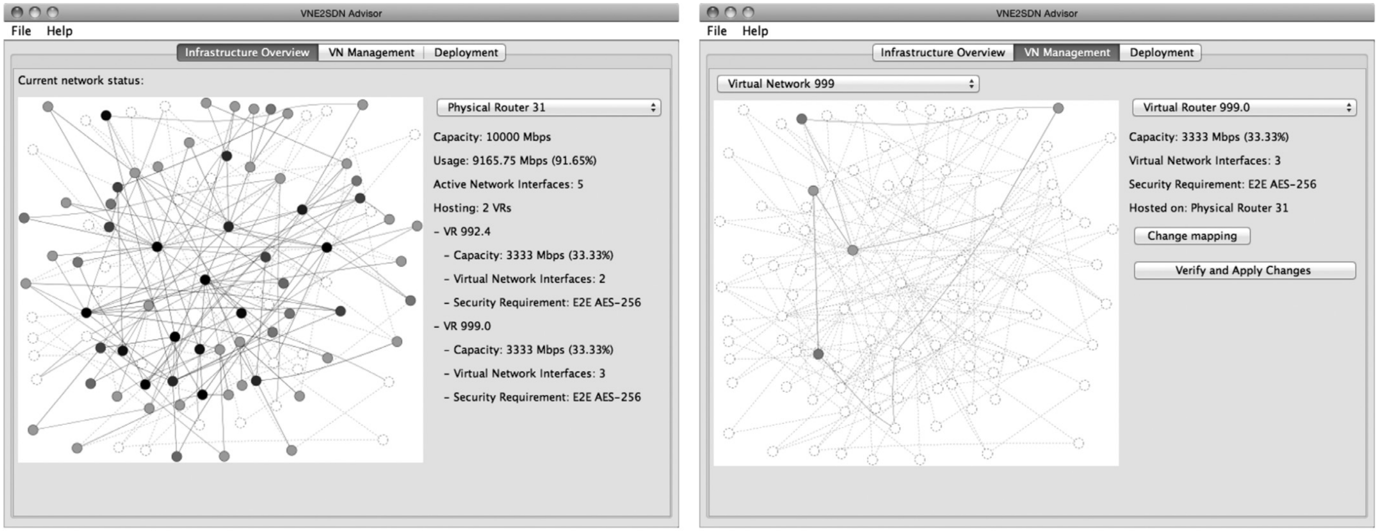


Fig. 6. Average throughput usage of routers with different connectivity degrees in experiments using a physical network of size 100 and AES-256.

Next, we analyze in further detail the characteristics of virtual network mappings generated by the optimal and heuristic methods. Fig. 5 shows the throughput usage of each router at the end of each round in experiments employing a physical network of size 100 and the AES-256 protocol. Routers are represented along the vertical axis, ordered by decreasing connectivity degree (*i.e.*, the amount of bidirectional links connected to each router – represented as *g* on right side of the figure), from top to bottom. Rounds are represented along the horizontal axis, in increasing order from left to right. When analyzing the results of both the optimal and heuristic approaches, there is a noticeable trend of higher usage of routers with higher connectivity degrees throughout the entire duration of the experiment. Moreover, both the optimal and heuristic approaches achieve very similar levels of physical resource usage. Gaps observed in this graph show that there are physical resources that remain unused, indicating that the observed rejections may be due to topological factors rather than overall resource exhaustion (This particular behavior is further analyzed by Luizelli et al. [19]). Further, patterns in the form of short horizontal lines are a result of the online arrival and subsequent removal of virtual networks (which stay embedded for 25 rounds).

Last, in order to further analyze the relationship between router usage and connectivity, Fig. 6 depicts the average throughput usage of physical routers grouped by their connectivity degree. This



Current network status:

Capacity: 10000 Mbps
Usage: 9165.75 Mbps (91.65%)
Active Network Interfaces: 5
Hosting: 2 VRs
- VR 992.4
  - Capacity: 3333 Mbps (33.33%)
  - Virtual Network Interfaces: 2
  - Security Requirement: E2E AES-256
- VR 999.0
  - Capacity: 3333 Mbps (33.33%)
  - Virtual Network Interfaces: 3
  - Security Requirement: E2E AES-256

Capacity: 3333 Mbps (33.33%)
Virtual Network Interfaces: 3
Security Requirement: E2E AES-256
Hosted on: Physical Router 31

(a) User interface presenting an overview of the current state of the physical infrastructure.

(b) User interface depicting router and link mappings of a specific virtual network.

Fig. 7. Graphical user interface of the VNE2SDN advisor software tool.

graph emphasizes the correlation between higher resource usage and connectivity degree mentioned previously. In the experiment performed using the ILP model, the average throughput usage of routers with the lowest degree of connectivity (2) is 24.8%, while that of routers with the highest degree (20) is 88.3%. Similarly, the employment of the heuristic method led to an average throughput usage of 19.1% in routers with a connectivity degree of 2 and 86.9% in routers with degree 20. In both cases, throughput usage grows following an approximately logarithmic scale in relation to the connectivity degree. These results emphasize the impact of topological aspects in the mapping process, as the presence of highly connected routers on the physical infrastructure may favor the instantiation of a greater number of virtual networks.

4. From secure virtual network mappings to their instantiation in SDN/OpenFlow networks

Thus far, we have introduced an optimization model and heuristic algorithm for efficiently embedding secure virtual networks and presented an extensive evaluation of both methods. However, the question of whether it is technically viable to take the output of the model/algorithm and process it towards the implementation of secure virtual networks in practice remains unanswered. In this section, we aim at addressing this question by introducing a mechanism that is able to automatically translate virtual network embeddings into SDN/OpenFlow rules. This mechanism is supported by a software tool that (i) enables InP administrators to easily visualize and modify mappings produced by our exact model or heuristic algorithm and (ii) generates the set of corresponding SDN/OpenFlow rules. Following a top-down approach, in the next subsections we first describe general aspects of the software implementation of our mechanism, followed by a more in-depth explanation of how virtual networks are translated to rules and a discussion of mechanisms that may be used for ensuring the required levels of security.

4.1. VNE2SDN advisor tool

The VNE2SDN advisor software tool uses the output of our optimal or heuristic approaches as its input. Such output is generated in a standardized and easy to parse manner, although it is not easily readable by humans. VNE2SDN Advisor addresses this issue by presenting a graphical representation of the physical network and indicating where each virtual network will be instantiated. Moreover, it allows the InP administrator to make changes to the suggested virtual router and link placements before they are embedded, if desired.

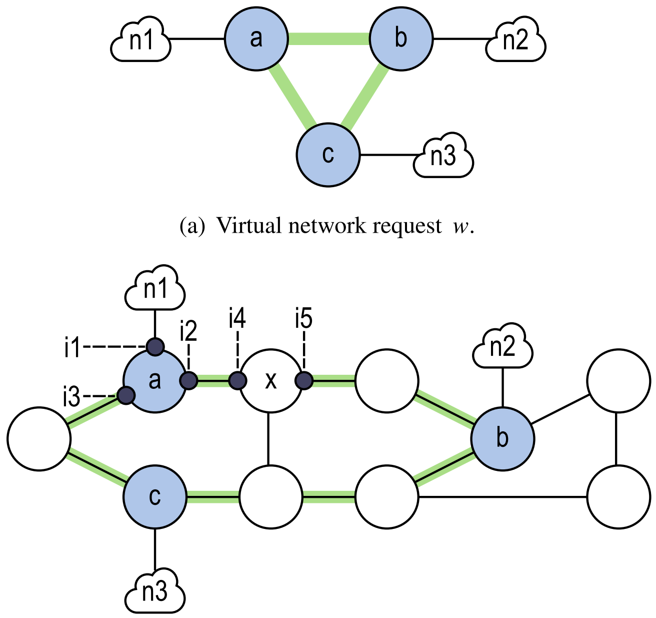
The main part of the graphical user interface (GUI) of VNE2SDN Advisor consists of three tabs. Fig. 7a depicts the first one, which displays an overview of the current state of the physical infrastructure. Unused physical routers and links are represented with dashed lines. Routers and links that are hosting virtual network elements, in turn, are represented with solid lines and their colors represent the amount of resources being used (with darker colors representing higher resource consumption). Moreover, users may view detailed information about each router or link by clicking them or manually selecting them from the drop-down list on the top right of the window. This detailed information is shown on the right side, under the drop-down list.

As previously stated, the GUI allows suggested virtual network mappings to be modified. The tab for doing such changes is presented in Fig. 7b. This window shows where virtual routers and links from a specific virtual network are hosted. After selecting a virtual router or link to see its detailed information, the user may choose to host it on a different physical router or path. The feasibility of the new mappings is checked using CPLEX when the user presses “Verify and Apply Changes”. If the user-modified mapping is not feasible, it will not be applied, and a warning message will be shown. This feasibility check is done by employing the optimization model presented in Section 2.1 with fixed values for variables A^R and A^L and is completed by CPLEX nearly instantly.

Last, the third tab (not shown in the figure) allows the user to deploy the edited or unedited virtual network mappings on the physical substrate. After the deployment is started, this tab shows its ongoing progress as OpenFlow rules are installed in each network device.

4.2. SDN/OpenFlow rule generation

The output of our optimal and heuristic approaches is generated in a standardized manner. It represents the optimal or



(a) Virtual network request w.

(b) Possible mapping of virtual network w on a substrate network.

Fig. 8. Example virtual network request with subnetworks ($n1$, $n2$, and $n3$) connected to each virtual router (a, b, and c) and a possible mapping of this request on top of a physical substrate.

near-optimal placement of virtual routers and links on the substrate as determined by each approach. As such, it determines on which physical router each virtual router should be instantiated as well as which physical path each virtual link should traverse. Based on this information, the VNE2SDN Advisor software tool creates sets of OpenFlow rules that ensure the correct behavior of each virtual network. More specifically, these rules (i) guarantee each virtual network will have the expected connectivity, (ii) limit the bandwidth of each virtual link to its maximum allowed rate, and (iii) provide isolation among different virtual networks. In this work, isolation is achieved through rules that tag packets with VLAN IDs and, additionally, rules that ensure packets in a virtual network that are not addressed to network prefixes within that network will be dropped.

The rationale behind our mechanism for automatic rule creation is that each virtual network has a number of subnetworks (each with several end hosts) connected to it. In order to ensure proper connectivity, each virtual router requires the instantiation of rules that enable the physical router hosting it to know where to route packets addressed to each network prefix within the same virtual network. Moreover, one additional rule is needed in order to drop packets that are addressed to unknown network prefixes. Therefore, for a virtual network with n subnetworks, $n + 1$ rules are needed on each physical router hosting a virtual router that belongs to this network. Additionally, as previously explained, virtual links are usually mapped to paths composed of multiple physical links. Paths hosting links from a given virtual network often pass through physical routers that are not hosting virtual routers from this network. On each of these “auxiliary routers”, n rules are installed for this virtual network, ensuring that these routers are also able to route packets to any of the network's subnetworks. Invalid packets are dropped before reaching auxiliary routers; therefore, packet drop rules do not need to be added to them. Bandwidth limits can be established in OpenFlow 1.3+ through meters, requiring n entries in the meter table of each OpenFlow-enabled physical router used by this virtual network.

Table 2
Flow table entries needed to instantiate virtual router a.

Priority	Matching fields	Actions
32,768	source = $n1$, destination = $n2$	set VLAN ID w, output:$i2$
32,768	source = $n1$, destination = $n3$	set VLAN ID w, output:$i3$
32,768	destination = $n1$	strip VLAN header, output:$i1$
0	source = $n1$	$\varnothing$

Table 3
Flow table entries added to auxiliary router x of virtual network w in order to ensure proper traffic forwarding.

Priority	Matching fields	Actions
32,768	VLAN ID = w, destination = $n1$	output:$i4$
32,768	VLAN ID = w, destination = $n2$	output:$i5$
32,768	VLAN ID = w, destination = $n3$	output:$i4$

To illustrate the previously explained strategy, consider the scenario depicted in Fig. 8. A virtual network request w (shown in Fig. 8a) with 3 virtual routers (a, b, and c) – each connected to a number of hosts sharing a network prefix ($n1$, $n2$, and $n3$, respectively) – is to be embedded on top of a physical substrate (shown in Fig. 8b). Each virtual router of this network is assigned to a physical router, while the links between these routers traverse other physical routers on the substrate.

In order to ensure the proper connectivity and isolation of virtual network w, sets of rules must be added to the physical routers hosting a, b, and c, in addition to the auxiliary routers through which virtual links will traverse. Table 2 shows the rules that would have to be added to the physical router hosting virtual router a for this purpose. The first two rules tag packets coming from $n1$ (*i.e.*, a host within the subnetwork connected to this virtual router) with a VLAN ID and forward it through the appropriate network interface ($i1$, $i2$, or $i3$). The third rule strips the VLAN tag of incoming packets addressed to hosts in $n1$ and forwards them to this subnetwork[1]. Last, the fourth rule drops packets originating from $n1$ with invalid destinations. The priority of this rule is lower than that of the other ones, ensuring that it will only match packets that are not matched by any other rule (*i.e.*, packets that are not addressed to any of the valid subnetworks of virtual network w – $n1$, $n2$, or $n3$). Table 3, in turn, shows the rules that need to be installed in auxiliary router x. The sole purpose of rules installed in auxiliary routers such as this one is to ensure that packets that belong to this virtual network will be properly forwarded between pairs of its routers. We take a conservative approach when generating rules for auxiliary routers, ensuring that all auxiliary routers have rules that enable them to forward packets to any valid subnetwork (even though, depending on the network topology, some may not be necessary – as is the case of the last rule shown in Table 3, since packets addressed to $n3$ would not reach this particular router).

By employing our translation mechanism considering the same workloads used for our evaluation in the previous section, we were able to measure the average and maximum flow table occupation after each round. Fig. 9 depicts the average number of flow table entries on physical routers. Throughout the experiment in which we used the output of the optimization model executed for physical networks of size 100, the average remains between 20 and 30 flow table entries per router. In experiments performed

[1] It should be noted that the isolation provided by VLAN tagging in this approach only applies within the virtual network, as tags are added when packets enter the network and removed when they are forwarded to the appropriate subnetwork. Therefore, additional mechanisms (*e.g.*, in the application layer) would be necessary to ensure end-host-to-end-host isolation.

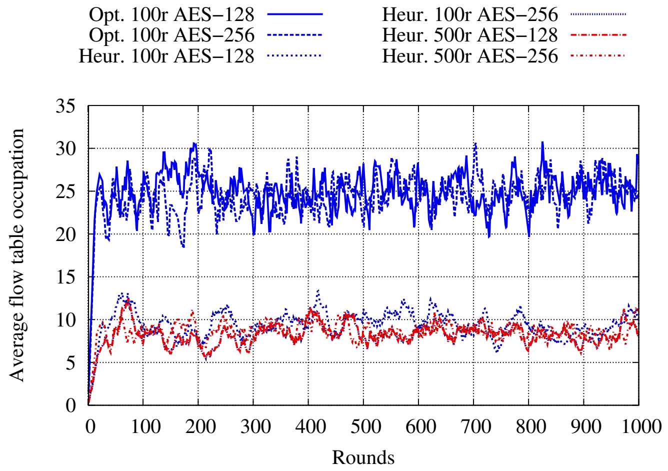


Fig. 9. Average number of flow table entries on physical routers in each round of the performed experiments.

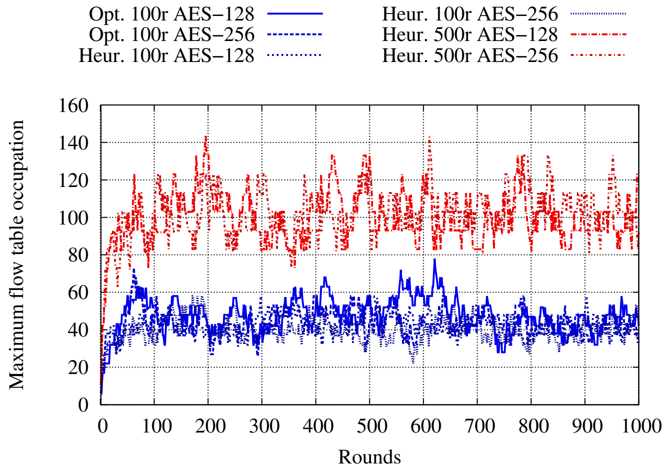


Fig. 10. Maximum number of flow table entries on physical routers in each round of the performed experiments.

considering the heuristic algorithm, the average number of entries remains between 5 and 15. Averages are lower in the latter experiments due to relatively lower acceptance rates (as the additional virtual networks embedded in the former experiments typically use longer paths in order to circumvent areas with insufficient residual resources). Further, Fig. 10 shows the highest number of flow table entries in infrastructure routers after each round (*i.e.*, the number of entries in the physical router with highest flow table occupation in each round). In experiments performed on physical networks with 100 routers, the maximum number of flow table entries per router remained between 20 and 80 (both in experiments employing the optimization model and heuristic algorithm). When considering the larger physical network with 500 routers, however, these numbers ranged from 80 to 140 throughout most of the experiments. The higher usage observed in individual routers in these scenarios is likely due to higher usage in specific parts of the infrastructure (*e.g.*, routers with high connectivity degree, similarly to the behavior observed in Fig. 6).

Considering the flow table capacity of currently available off-the-shelf OpenFlow switches (which ranges from one thousand [20] to one million entries [21]), both the average and the maximum flow table occupation values observed in the aforementioned measurements are well within acceptable ranges. Note, however, that virtual network requesters may define internal network policies and functions that may require additional flow table entries to be installed on physical routers. These include, for example, firewalls, load balancers, and traffic monitors [22]. We are currently investigating how the number of flow table entries needed to instantiate these policies/functions can be pre-calculated by the infrastructure provider and taken into account in the virtual network embedding process in order to ensure the correct operation of each virtual network.

4.3. Security

As previously explained, in our approach virtual networks may be secured by end-to-end or point-to-point cryptography protocols. Moreover, virtual network requesters may demand that their networks not share resources with certain other virtual networks.

The graphical output of our VNE2SDN Advisor software tool indicates the physical devices hosting virtual networks in which end-to-end and point-to-point cryptography must be provisioned. In practice, this is achieved through the use of the IPsec protocol suite. The transport and tunnel modes provided by IPsec are equivalent to the notions of end-to-end and point-to-point cryptography, respectively. Therefore, this protocol suite is capable of fulfilling both of the aforementioned privacy requirements.

Although we focus on the deployment of embedded virtual networks on top of an OpenFlow substrate, dedicated OpenFlow routers may not be able to perform the cryptographic operations that IPsec-protected traffic would require. Therefore, we assume that routers that have been marked as capable of supporting cryptographic protocols are hybrid routers with full IPsec support. For each virtual network that requires end-to-end or point-to-point security, IPsec is automatically set up among the routers that will host it. This is done by employing the Internet Key Exchange (IKE) protocol in order to negotiate and distribute encryption keys and establish Security Associations (SAs). Each SA is a separate logical group that contains the necessary attributes – such as the selected cryptographic algorithm, mode, and the encryption key itself – to establish secure IPsec connections among virtual routers.

The third level of security provided by our approach – non-overlapping networks – is guaranteed by the output of the developed optimization model and heuristic algorithm. Therefore, no further action is necessary to ensure that conflicting networks will not share physical resources.

These security features aim, primarily, at preventing eavesdroppers at the physical level from obtaining sensitive information being transmitted through virtual networks. In addition to this protection, the employment of cryptography also prevents data contained in network packets from being tampered with. Other types of attacks, such as Denial of Service, are not specifically targeted. However, customers that request non-overlapping networks would not be affected by such attacks if they originate from networks belonging to the (types of) institutions they specified as conflicting ones (as no physical resources are shared among their networks).

5. Related work

In this section, we discuss previous work in the area of virtual network embedding, focusing on the distinctive features of each approach.

Yu et al. [3] present a heuristic-based approach to solve the problem of virtual network embedding. The authors propose a two-phase algorithm that prioritizes virtual networks with largest revenue value. First, a greedy node mapping algorithm maps virtual routers to physical ones. Afterwards, link mapping is performed by selecting the shortest physical path between the end points of each (virtual) link with enough bandwidth capacity or, if the request accepts path splitting, by solving the multi-commodity flow problem. Furthermore, the algorithm is able to reoptimize the physical substrate by adjusting splitting ratios or remapping virtual links to different physical paths.

Chowdhury et al. [4] introduce two Mixed Integer Programming (MIP) formulations, the second being a relaxed version of the first. Both models use location constraints from virtual nodes to preselect node mappings, which, according to the authors, facilitates the mapping of virtual links. As the relaxed version does not return integer values, it employs rounding techniques to select definitive node mappings and solves the multi-commodity flow problem to map virtual links.

Butt et al. [9] devise a mechanism for considering different characteristics of substrate nodes in virtual network embeddings. Weights are assigned to substrate nodes according to how "critical" and "popular" they are. A node is considered critical if the failure of this node has the potential of partitioning the substrate network, whereas the popularity of a node is measured as the number of different virtual networks that would be affected by its failure. Highly critical and popular nodes are then avoided by the embedding strategy. The authors also present a mechanism that reoptimizes virtual network embeddings by identifying and rearranging virtual networks that contribute to physical resource fragmentation.

Rahman et al. [8] develop a heuristic-based approach that considers single substrate link failures in the virtual network embedding process. This approach preemptively calculates alternate paths that are used to reroute virtual links in the event of a physical link failure. A portion of the total available bandwidth of each physical link is used as a pre-reserved quota for backup paths.

Two virtual network embedding approaches are proposed by Alkmim et al. [5]. These approaches aim at minimizing the time needed to transfer router images from software repositories to the physical routers where such images will be instantiated. Based on a same ILP formulation, the difference between them is the method used to solve the optimization problem. The first employs the traditional branch and cut method provided by CPLEX to traverse a search tree until an optimal solution to the integer problem is found. In contrast, the other relaxes the problem by limiting the search to the root of the search tree, often resulting in a sub-optimal solution but reducing solution time.

Cheng et al. [6] present two distinct algorithms (RW-MaxMatch and RW-BFS) that take advantage of node ranking to select router mappings. Virtual and physical nodes are ranked according to their own capacity and the capacities of their neighbors in non-increasing order (*i.e.*, requests with higher requirements are mapped first). RW-MaxMatch sorts virtual and physical routers according to their ranks and matches these sorted lists to map virtual routers to physical ones. Links are mapped in a separate stage using either the k-shortest path or the multi-commodity flow algorithm, depending on whether path splitting is allowed or not. Alternatively, RW-BFS maps routers and links in a single stage. This algorithm builds a breadth-first search tree of virtual nodes sorted by their ranks. It then attempts to map each virtual node to a physical node that meets all capacity constraints. In case of failure, it is able to backtrack and remap the previous virtual node, in an attempt to solve the issue. In the experiments performed by the authors, RW-MaxMatch leads to higher average revenue than RW-BFS. However, as RW-BFS tends to use less physical resources (therefore lowering hosting costs for the InP), it produces a higher revenue to cost ratio.

Gong et al. [7] follow a similar approach than that proposed by Cheng et al., building a heuristic virtual network embedding algorithm that takes into account the "global resource capacity" (GRC) of each physical device. The GRC of a router is a weighted summation of the resources of the entire network, in which devices closer to it influence this metric to a greater extent. After the GRC of each physical and virtual router is calculated, routers are sorted in descending order, and router mappings are chosen through a greedy strategy. Subsequently, virtual links are mapped to the shortest paths with sufficient resources available.

Last, Chen et al. [10] propose a virtual network embedding strategy oriented towards survivability. The embedding process attempts to balance the bandwidth consumption of substrate links. Moreover, backup links are reserved for each virtual network, and may be reconfigured over time in order to improve efficiency. The authors present both an ILP approach and a heuristic one. Both approaches employ the load balancing strategy, while the heuristic algorithm also performs the aforementioned reconfiguration of backup resources if necessary.

Our analysis of previous work reveals a variety of methods to solve the embedding problem. As ILP approaches solved in an optimal manner tend to be too inefficient for large networks, most authors propose relaxations or heuristic-based algorithms as a means to obtain better performance at the cost of sub-optimal solutions. This performance advantage is also the main motivation behind our heuristic-based algorithm. In contrast to the related work presented in this section, we proposed an algorithm based on simulated annealing that considers not only capacity limits and location constraints but also (and deemed very important) costs associated with security techniques needed to provide data confidentiality to virtual networks. Furthermore, as presented in the previous section, we also proposed a VNE to SDN/OpenFlow translation mechanism that is able to generate a set of consistent, deployable OpenFlow rules to guide the real instantiation of the mapped virtual networks.

6. Conclusions

Reconciling efficient resource mapping and satisfaction of security requirements is of paramount importance for the use of network virtualization in real environments. In this paper, we presented both an ILP-based and a heuristic online virtual network embedding algorithm featuring precise modeling of overhead costs of security mechanisms. We reported a detailed evaluation, comparing the performance of the heuristic approach and the ILP model according to a number of metrics. Further, we presented and evaluated a mechanism for deploying virtual networks on top of SDN/OpenFlow infrastructures using the mappings produced by our approaches. Our solution allows security mechanisms to be embedded in a manner that is transparent to users, with ample support for various applications. While virtual network requesters are free to employ additional security mechanisms, this ensures that any network application will meet a minimum desired level of protection, in line with recent challenges related to privacy and trust.

Our experiments have shown that the ILP model is able to find optimal solutions in the order of seconds when considering physical networks with up to a hundred routers. However, as it is modeled to solve an NP Hard problem, it does not scale to larger network sizes. Experiments performed with this model revealed that after increasing the physical network size to 500 routers, several hours were needed to map individual virtual network requests. In contrast, the proposed heuristic algorithm is able to find feasible mappings for environments using such large networks while remaining in the order of minutes. The heuristic algorithm leads to high quality mappings, keeping low the gap between solutions produced by the heuristic approach and the ILP-based one while retaining the ability to scale to large network sizes. Additionally, the heuristic algorithm is flexible, allowing parameterizations that lead to more precise mappings if so desired, at the cost of possibly elevating solution times to the order of a few hours. To summarize, while the ILP model is capable of optimally embedding virtual networks on smaller physical infrastructures, the heuristic algorithm is better suited for larger substrate networks, being able to map virtual network requests in a timely manner.

Last, through the VNE2SDN Advisor software tool we were able to demonstrate the technical feasibility of materializing the suggested virtual network mappings in a real environment. Virtual networks are instantiated using a set of rules that ensures the correct operation of essential network functionality (*i.e.*, proper connectivity and isolation). As the flow table space occupied by these rules is minimal, service providers are able to make use of the remaining space in order to set up more advanced network functions and policies as desired. Moreover, the graphical user interface of VNE2SDN may be used by administrators to visualize the mappings produced by our ILP-based and heuristic algorithms and make any changes they deem necessary.

Acknowledgments

This work has been partially supported by FP7/CNPq (Project SecFuNet, FP7-ICT-2011-EU-Brazil), RNP-CTIC (Project ReVir), as well as PRONEM/FAPERGS/CNPq (Project NPRV).

Appendix A. Notation used in this paper

Table A.1 lists the symbols used to represent the inputs and variables of our optimization model and heuristic algorithm throughout the paper. A more detailed explanation of each symbol can be found in Section 2.

Table A1
Notation used in this paper.

Decision variables	
$A^R_{i,r,j} \in \{0, 1\}$	Indicates whether physical router i is hosting virtual router j of virtual network r
$A^L_{i,j,r,k,l} \in \{0, 1\}$	Indicates whether physical link (i, j) is hosting virtual link (k, l) of virtual network r
Input sets	
$N = (R, L)$	Directed graph representing a physical or a virtual network
N^P, N^V	Physical and virtual networks, respectively
R^P, R^V	Set of physical or virtual routers
L^P, L^V	Set of physical or virtual links
$T^P_i, T^V_{r,j}$	Throughput capacity of physical router i (T^P_i) and throughput required by virtual router j from virtual network r ($T^V_{r,j}$), respectively
$B^P_{i,j}, B^V_{r,k,l}$	Bandwidth capacity of physical link (i, j) ($B^P_{i,j}$) and bandwidth required by virtual link (k, l) from virtual network r ($B^V_{r,k,l}$), respectively
S^P, S^V	Geographical location of a physical or virtual router. Pair $(i, a) \in S^P$ indicates that router i is situated in location a, while a pair $(j, b) \in S^V$ denotes virtual router j must be hosted in location b
$K^P_i, K^V_{r,j}$	Indicates, respectively, whether physical router i supports cryptography (K^P_i) and whether virtual router j from virtual network r requires this feature $K^V_{r,j}$
$W^R_{r,i}, W^L_r$	The additional cost of cryptographic operations in terms of processing ($W^R_{r,i}$) for virtual router i and bandwidth (W^L_r) in virtual network r
X	Set of pairs of virtual networks that are not allowed to share physical resources. Pair $(r, s) \in S$ indicates that virtual networks r and s cannot share the same physical infrastructure
$E^R_{i,r,j}, E^L_{i,j,r,k,l}$	Previous virtual network mappings, expressed similarly to decision variables. $E^R_{i,r,j}$ denotes virtual router embeddings and $E^L_{i,j,r,k,l}$ denotes virtual link embeddings

References

[1] D. Andersen, 2002, http://www.cs.cmu.edu/~dga/papers/andersen-assign.ps Theoretical Approaches to Node Assignment, (unpublished manuscript).

[2] L.R. Bays, R.R. Oliveira, M.P. Barcellos, L.P. Gaspary, E.R.M. Madeira, Virtual network security: threats, countermeasures, and challenges, J. Internet Serv. Appl. 6 (1) (2015) 1–19.

[3] M. Yu, Y. Yi, J. Rexford, M. Chiang, Rethinking virtual network embedding: substrate support for path splitting and migration, SIGCOMM Comput. Commun. Rev. 38 (2) (2008) 17–29. http://dx.doi.org/10.1145/1355734.1355737

[4] M. Chowdhury, M.R. Rahman, R. Boutaba, Vineyard: Virtual network embedding algorithms with coordinated node and link mapping, IEEE/ACM Trans. Net. 20 (1) (2012) 206–219.

[5] G.P. Alkmim, D.M. Batista, N.L.S. Fonseca, Mapping virtual networks onto substrate networks, J. Internet Serv. Appl. 3 (4) (2013) 1–15. http://dx.doi.org/10.1186/1869-0238-4-3

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

[7] L. Gong, Y. Wen, Z. Zhu, T. Lee, Toward profit-seeking virtual network embedding algorithm via global resource capacity, in: Proceedings of 2014 IEEE INFOCOM, 2014.

[8] M. Rahman, I. Aib, R. Boutaba, Survivable virtual network embedding, in: Proceedings of the 9th International IFIP TC 6 Networking Conference, 6091, Springer Berlin Heidelberg, Chennai, India, 2010, pp. 40–52.

[9] N.F. Butt, M. Chowdhury, R. Boutaba, Topology-awareness and reoptimization mechanism for virtual network embedding, in: Proceedings of the 9th International IFIP TC 6 Networking Conference, 6091, Springer Berlin Heidelberg, Chennai, India, 2010, pp. 27–39.

[10] Q. Chen, Y. Wan, X. Qiu, W. Li, A. Xiao, A survivable virtual network embedding scheme based on load balancing and reconfiguration, in: Proceedings of IEEE Network Operations and Management Symposium, 2014.

[11] L.R. Bays, R.R. Oliveira, L.S. Buriol, M.P. Barcellos, L.P. Gaspary, A heuristic-based algorithm for privacy-oriented virtual network embedding, in: Proceedings of IEEE/IFIP Network Operations and Management Symposium (NOMS), IEEE, Krakow, Poland, 2014.

[12] S. Kent, K. Seo, RFC 4301: Security architecture for the internet protocol, 2005. http://tools.ietf.org/rfc/rfc4301.txt.

[13] R. Albert, A.L. Barabási, Topology of evolving networks: Local events and universality, Phys. Rev. Lett. 85 (2000) 5234–5237. http://dx.doi.org/10.1103/PhysRevLett.85.5234

[14] C. Westphal, G. Pei, Scalable routing via greedy embedding, in: Proceedings of IEEE INFOCOM, 2009.

[15] F. Esposito, D.D. Paola, I. Matta, A general distributed approach to slice embedding with guarantees, in: Proceedings of IFIP Networking Conference, 2013.

[16] E. Barker, Q. Dang, Nist special publication 800–57 part 3 revision 1, NIST Special Publication 800 (57) (2015) 1–142. http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57Pt3r1.pdf

[17] A. Bogdanov, D. Khovratovich, C. Rechberger, Biclique cryptanalysis of the full AES, in: Advances in Cryptology–ASIACRYPT 2011, Springer, 2011.

[18] C. Xenakis, N. Laoutaris, L. Merakos, I. Stavrakakis, A generic characterization of the overheads imposed by IPSEC and associated cryptographic algorithms, Comput. Netw. 50 (17) (2006) 3225–3241. http://dx.doi.org/10.1016/j.comnet.2005.12.005

[19] M.C. Luizelli, L.R. Bays, L.S. Buriol, M.P. Barcellos, L. Gaspary, Characterizing the impact of network substrate topologies on virtual network embedding, in: Proceedings of the 9th International Conference on Network and Service Management, 2013.

[20] Y. Nakagawa, K. Hyoudou, C. Lee, S. Kobayashi, O. Shiraki, T. Shimizu, Domainflow: practical flow management method using multiple flow tables in commodity switches, in: Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies, CoNEXT ’13, ACM, New York, NY, USA, 2013, pp. 399–404. http://dx.doi.org/10.1145/2535372.2535406

[21] NoviFlow, NoviSwitch, 2014. http://noviflow.com/products/noviswitch/.

[22] X. Jin, J. Gossels, J. Rexford, D. Walker, Covisor: a compositional hypervisor for software-defined networks, in: Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15), 2015.