Virtual Network Embedding in Software-Defined Networks

Leonardo Richter Bays, Luciano Paschoal Gaspary
Institute of Informatics – Federal University of Rio Grande do Sul (UFRGS)
{lrbays,paschoal}@inf.ufrgs.br
Reaz Ahmed, Raouf Boutaba
David R. Cheriton School of Computer Science – University of Waterloo
{r5ahmed,rboutaba}@uwaterloo.ca

***Abstract*—Research on network virtualization has been active for a number of years, during which a number of virtual network embedding (VNE) approaches have been proposed. These approaches, however, neglect important operational requirements imposed by the underlying virtualization platforms. In the case of SDN/OpenFlow-based virtualization, a crucial example of an operational requirement is the availability of enough memory space for storing flow rules in OpenFlow devices. In this paper, we advocate that VNE must be performed with some knowledge of the underlying physical networks, otherwise the deployment may suffer from unpredictable or even unsatisfactory performance. Considering SDN/OpenFlow-based physical networks as an important virtualization scenario, we propose an approach based on VNE and OpenFlow coordination for proper deployment of virtual networks (VNs). The proposed approach unfolds in the following main contributions: *(i)* a virtual infrastructure abstraction that allows a service provider to represent the details of his/her VN requirements in a comprehensive manner; *(ii)* a privacy-aware compiler that is able to preprocess this detailed VN request in order to obfuscate sensitive information and derive computable operational requirements; and *(iii)* a model for embedding requested VNs ensuring their feasibility at the physical level. The results obtained through our evaluation demonstrate that taking such operational requirements into account, as well as accurately assessing them, is of paramount importance to ensure the correct behavior of VNs hosted on top of the virtualization platform.**

I. Introduction

Research on network virtualization has been active for a number of years. During this period, several approaches for embedding VNs on top of physical infrastructures have been proposed [1]–[6]. Most approaches consider a similar set of VN requirements, such as CPU, memory, and bandwidth guarantees, as well as location constraints. Some approaches take into account additional aspects such as virtual router image transfer and instantiation overheads, network survivability, or communication security. In contrast, relevant operational requirements related to the instantiation of VNs on different virtualization platforms are neglected. This simplification enables the streamlining of the optimization models and heuristics used in these approaches. Moreover, it renders them generic enough to be applied to a number of different scenarios. However, by not taking into account operational requirements of the underlying virtualization platforms, the mappings produced by these VNE approaches may *(a)* not be feasible in practice, *(b)* be unable to properly fulfill SLA requirements, or *(c)* fail to use infrastructure resources in an efficient manner. In this work, we focus on ensuring the feasibility of VNE mappings on a multi-tenant, SDN/OpenFlow-based network environment so as not to put at risk satisfactory performance and/or network predictability of embedded VNs.

Software-Defined Networking (SDN) offers a promising platform for network virtualization. In addition to slicing physical resources among customers [7]–[12], SDN-based environments provide abstractions that allow different virtualization functionality, including the instantiation of arbitrary virtual topologies [8]–[12] and the use of overlapping address spaces [10]–[12]. In the case of SDN/OpenFlow-based virtualization, a crucial example of an operational requirement that may render VNE-provided mappings inadequate in real environments is the unavailability of enough memory space for storing flow rules in OpenFlow devices. If this critical issue is not taken into account by the VNE algorithm, OpenFlow devices may not be able to accommodate all flow rules required by the virtual routers assigned to them. As a consequence, these devices would need to frequently contact the controller in order to handle incoming packets. High rates of controller intervention, in turn, could hinder network performance predictability and potentially render the multi-tenant environment unstable.

Figure 1 depicts an example of mappings that would be considered valid by a standard VNE approach but could ultimately lead to performance issues in practice. In this example, three VN requests (VN1–3) are embedded on top of a physical network. In this example, physical routers support up to 4,000 flow rules each, while routers in each VN request require either 2,000 or 3,000 flow rules to be installed on the physical routers hosting them. Moreover, a number of flow rules must also be installed on "auxiliary routers" – *i.e.*, routers that are not part of the VN request directly, but are necessary in order to create physical paths to host virtual links between such routers. As one can observe, the computed mappings exceed the capacity of some of the physical routers – namely, PR2 (which is hosting virtual routers B and D), PR4 (hosting virtual router F and an auxiliary router in the path between virtual routers B and C), and PR5 (hosting virtual routers E and H).

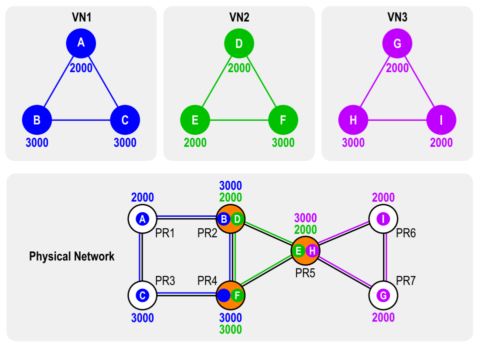


Fig. 1. Example of mappings generated by a standard VNE approach that exceed the flow table capacity of physical devices.

The example illustrated above not only underscores that VNE must be performed with some knowledge of the underlying physical networks but also sheds light to the importance of going further in terms of expressing what a VN needs from the physical network. Applications running on top of VNs have distinct traffic patterns and are often subject to different policies or network functions (*e.g.*, load balancing, access control, or deep packet inspection). The specification of the expected behavior of a VN, translated into an estimate of flow table usage and communicated to the infrastructure provider, would potentially lead to both a more accurate orchestration of VN embeddings and, ultimately, an overall better quality of service.

In this paper, we propose a VNE approach that is aware of operational requirements related to the instantiation of VNs on top of an SDN/OpenFlow environment. The central idea of the proposed approach is the specification, by VN requesters, of VN requests enriched with information about how (to be) provisioned networks will be used (*e.g.*, important application flows, network functions/policies to which packets will be subjected to, etc.). These VN specifications are then used to derive operational requirements, still at the customer's end. The resulting specifications – reflecting requesters willingness (or not) to disclose information about the VNs – are sent to the InP, which will ultimately correctly embed the requested VNs favoring incoming requests with well defined operational requirements. We consider a number of pieces of information that, if known in advance by the InP, can lead to improved allocation of network resources and, in turn, to improved network utilization. The main contributions of this paper are threefold: *(i)* an abstraction model for expressing requirements related to internal VN policies and traffic patterns; *(ii)* a strategy for accurately deriving the number of flow rules needed to instantiate VNs based on the aforementioned requirements; and *(iii)* a VNE method that leverages this information in order to correctly and efficiently allocate resources in an SDN/OpenFlow-based virtualization environment.

The remainder of this paper is organized as follows. In Section II we discuss existing VNE approaches. In Section III we introduce our proposed solution and describe its main elements. In Section IV we describe the evaluation we carried out and present and discuss the obtained results. Last, in Section V we present final remarks and perspectives for future work.

II. Related Work

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

Yu *et al.* [1] present a heuristic-based VNE approach. The algorithm embeds virtual routers and links in separate phases, and prioritizes VNs with largest revenue. This approach takes into account CPU and location constraints for routers, and bandwidth constraints for links.

Chowdhury *et al.* [2] propose two optimization models, one being a relaxed version of the other. Routers and links are embedded in distinct phases; however, improved coordination between these phases is achieved by preselecting router mappings taking into account their location constraints in order to facilitate link mapping. Similarly to the work of Yu *et al.*, CPU, location, and bandwidth requirements are considered by the proposed optimization models. Moreover, link delay is used to determine how far a virtual router may be embedded from its preferred location.

Cheng *et al.* [3] introduce the concept of "node ranking", in which virtual and physical nodes are ranked according to their own capacity and the capacities of their neighbors. Two embedding algorithms are proposed, one mapping routers and links in distinct stages while the other performs both simultaneously. The algorithms take into account CPU and bandwidth constraints but do not include location constraints.

Alkmim *et al.* [4] present two VNE approaches based on optimization models. One employs a traditional Integer Linear Programming (ILP) model, while the other employs a relaxation technique in order to reduce running times. The authors focus on constraints related to overheads incurred when transferring and instantiating virtual router software images. As such, in addition to CPU, location, bandwidth, and link delay, the size of virtual router images (and the memory needed to support them), the locations in which they are stored, and the time needed to transfer and instantiate them are also taken into account.

Bays *et al.* [5] propose both an optimization model and a heuristic algorithm for virtual network embedding focusing on privacy. Both approaches take into account throughput capacity and location requirements of routers as well as link bandwidth. Additionally, a number of security related constraints are considered, namely which physical routers are capable of supporting the necessary security protocols, overheads associated with cryptographic operations, and which VNs may not share physical resources.

Last, Demirci *et al.* [6] focus on embedding VNs on top of an SDN substrate. More specifically, the issue of controller placement is tackled in addition to virtual router and link mapping. The authors devise two different embedding strategies. The first one aims at balancing the load on physical elements, while the other aims at minimizing communication delay between virtual routers and controllers. The authors consider bandwidth capacity constraints, in addition to controller location requirements. Embedding is performed in an offline manner, assuming all requests are known in advance.

As highlighted in this section, previous work in the area of VNE does not take into account operational requirements related to the instantiation of VNs on top of SDN/OpenFlow substrates. Although the work of Demirci *et al.* [6] is more closely aligned to the approach proposed in this paper, the authors do not take into account hard capacity constraints of SDN routers, only attempting to minimize overall resource usage. As previously explained, this may lead these approaches to generate mappings that are ultimately impossible to instantiate in practice. Moreover, we are not aware of previous attempts to enable customers to represent the needs of their VNs in a level of detail they are comfortable with while simultaneously allowing infrastructure providers to leverage a "distilled" version of this information to ensure no operational constraints are broken.

III. Proposed Solution

Next, we present our proposed solution for coordinating VNE and SDN infrastructures. First, we briefly explain the characteristics of SDN environments considered in this paper and provide an overview of our proposal. Right after, we detail each of its main components.

A. Multi-tenant Infrastructure Model and Network Virtualization Paradigm Considered

The approach proposed in this paper targets an SDN/OpenFlow-based network environment in which a number of customers (service providers) request and, if possible, are granted virtual infrastructures. More specifically, we focus on correctly provisioning the VNs that interconnect the elements of these infrastructures.

An example of a multi-tenant network virtualization environment is depicted in Figure 2. VN requests are received by the infrastructure provider and processed by a VN embedder. Accepted VN requests are ultimately instantiated on top of the physical infrastructure through a network hypervisor, following the mappings produced by the VN embedder. The controller associated with each VN (represented as black boxes with the letter C in the figure), in turn, is hosted within a virtual machine, and communication channels are established between it and the network hypervisor. The hypervisor intermediates flow rule instantiation and monitoring actions sent by VN controllers in order to enforce properties such as isolation at the physical level.

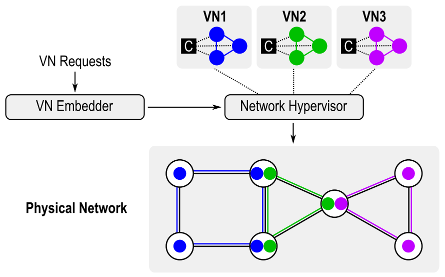


Fig. 2. Multi-tenant OpenFlow/SDN-based network virtualization model considered in this paper.

In recent years, a number of architectures for enabling network virtualization on top of OpenFlow-based SDNs have been proposed. These architectures have evolved from simpler hardware-based flow table slicing to more complex flow level virtualization. While the former achieves better performance, the latter allows a significantly higher degree of flexibility. As an example of such flexibility, recent flow level virtualization approaches [10]–[12] enable tenants to request arbitrary topologies that are not restricted to a subset of the physical network (topology virtualization) as well as to use overlapping address spaces (address space virtualization).

In this work, we consider an SDN virtualization platform capable of providing the aforementioned features, such as OpenVirteX [12]. However, it is worth mentioning that our VNE/SDN coordination approach may be adapted to interface with other virtualization platforms, even ones that do not follow the flow-level virtualization model. Moreover, we focus on VN embedding (*i.e.*, not including host embedding), assuming end hosts are located on the premises of the customer.

B. Overview of our Proposed VNE and SDN Coordination Approach

As previously explained, VN mappings generated by standard VNE algorithms may violate operational requirements when an infrastructure provider attempts to instantiate the requested VNs on a real physical substrate. In order for the VNE algorithm to take such requirements into account, they would have to be part of the VN request provided by the customer. However, we believe it is unreasonable to expect customers to be aware of operational requirements that affect the environment on a physical level. Moreover, customers may be averse to disclosing too much information regarding the internal behavior of their network. Therefore, the main goals of our approach are to: *(i)* allow the customer to represent the needs of his/her VN in a detailed manner; *(ii)* preprocess this detailed representation, removing sensitive information and deriving data regarding the operational requirements associated with this particular request; and *(iii)* embed the requested VNs ensuring both feasibility and adequate performance by making use of this "distilled" information.

Figure 3 depicts the components of our proposed approach. The customer first creates a Tenant Infrastructure Graph (TIG), which represents not only virtual routers and links (and their capacity requirements) but also elements (such as hosts and end users) connected to the network, the traffic patterns among them, and the network functions that will be applied to each traffic flow. As some of this information may be considered sensitive by the customer, the TIG is preprocessed by a Privacy-aware Compiler running on the customer's premises. This Privacy-aware Compiler allows customers to only reveal as much information about their networks as they want, while still generating an enhanced VN request that aids the VNE process in order to ensure feasible, high-quality VN deployments. The preprocessed request is then sent to the InP, which makes use of our SDN/OpenFlow-aware Embedder in order to properly embed and deploy (although the latter is out of the scope of this paper) the customer's network. The main elements of our proposal – Tenant Infrastructure Graphs, the Privacy-aware Compiler, and the SDN/OpenFlow-aware Embedder – will be further explained in the following subsections.

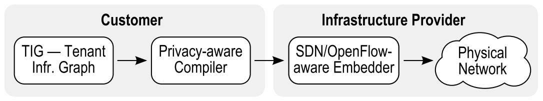


Fig. 3. Overview of our proposed approach, depicting its main elements and the information flow between them.

C. Specification of Infrastructure Resources

We now proceed to a detailed explanation of the process that is carried out on the customer's end in order to request a VN.

1) Tenant Infrastructure Graph (TIG) – A Detailed Abstraction of a Virtual Network and its Communication Patterns: The TIG enables customers to represent the needs of their requested VNs with a high level of detail. In addition to the information contained in a standard VN request (*e.g.*, network topology and capacity and location requirements), a TIG also represents: *(i)* elements such as end hosts (represented as network prefixes or individual addresses) connected to the network; *(ii)* the communication patterns among such elements; and (iii) network functions/policies each communication pattern must be subjected to.

Figure 4 depicts an example of a TIG. Each cloud represents a group of application servers executing a common task (*e.g.*, within the same tier). Globes represent groups of external users (*e.g.*, network administrators or end users accessing applications running on the customer's premises through the to be deployed VN). Each of these elements has some information associated to it – namely, the number of instances of each group of application instances and the number of network prefixes of each group of external users. Last, circles in the graph represent virtual routers, and colored edges represent communication patterns (*i.e.*, traffic flows) among network elements.

Each group of edges represented with the same color and style in Figure 4 denotes a distinct traffic flow. As an example, the solid edges represented in blue interconnect end users to externally accessible applications running on the customer's premises (*e.g.*, the front-end of a two-tier web application), while the dotted green edges interconnect the front-end to the application database back-end. Dashed red edges interconnect databases running in different locations for synchronization/replication purposes, while the dashed and dotted yellow pattern provides an administrator access to all applications.

In addition to forwarding packets, routers may need to perform other functions specific to each traffic flow. Some of these functions – Load Balancing (LB), Quality of Service (QoS), and Access Control (AC) – are represented in Figure 4. In order to discriminate between different network flows

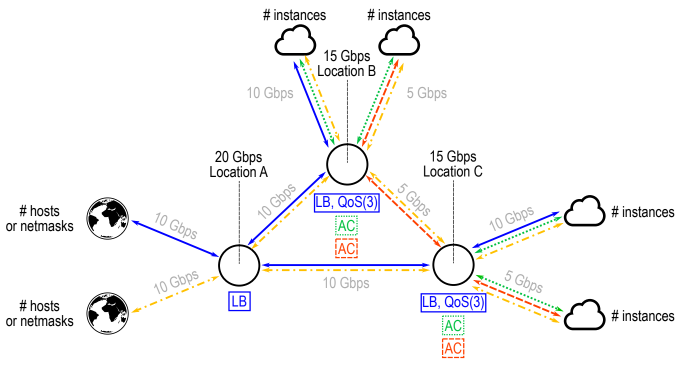


Fig. 4. Tenant Infrastructure Graph representing elements connected to a VN and the communication patterns among them.

and apply the appropriate functions to each, routers use a number of packet header fields (or combinations of fields[1]). In order to apply load balancing, for example, both the source and destination of a packet should be taken into account. For the purpose of QoS, in turn, the Differentiated Services Code Point (DSCP) header field may be used. The TIG represents these (combinations of) fields as sets of Traffic Discriminators (TDs). Moreover, each TD contains a number of entries – *i.e.*, the number of different values a given (combination of) header field(s) may be set to. In the aforementioned QoS example, the DSCP field may be set to a value between 0 and 63. Therefore, the number of entries in a traffic discriminator that uses this field may be anywhere between 2 (if only two different QoS classes are used) and 64 (if all possible classes are used).

Through the information represented in this graph, it is possible to accurately derive the number of flow rules each router in a VN will need in order to ensure its correct and optimal operation. Moreover, as shown in Figure 4, standard VNE constraints (router throughput, link bandwidth, and location requirements) are also represented in a TIG.

2) Privacy-aware Compiler: While a TIG enables the representation of operational constraints associated with the instantiation of each VN on the physical infrastructure of an InP, it also exposes information that the customer may consider sensitive. Therefore, TIGs are expected to be preprocessed by a Privacy-aware Compiler on the customer's end before being sent to an InP.

As previously mentioned, the TIG represents distinct communication patterns within elements of the requested VN. A number of flow rules will ultimately need to be installed on each router in order to ensure the correct operation of each traffic flow. The number of necessary flow rules depends on a number of pieces of information, namely: *(i)* the number of network applications and/or network prefixes of external users associated with each flow; *(ii)* the number of traffic discriminators associated with each router for handling each network flow; and *(iii)* the number of entries of each traffic discriminator.

The left side of Figure 5 shows an example TIG populated with numerical values. The communication pattern represented in solid blue interconnects a number of users (comprising 100 network masks) to a number of applications through the customer's VN hosted on the physical infrastructure of the InP (10 app instances connected to virtual router b and 10 connected to router c). The dotted green patterns interconnect each group of 10 application instances to a group of 5 database servers. The dashed red pattern, in turn, interconnects both groups of database (server) instances. Last, the dashed and dotted yellow pattern interconnects all network services to a specific external network mask (used by a network administrator to manage all network services).

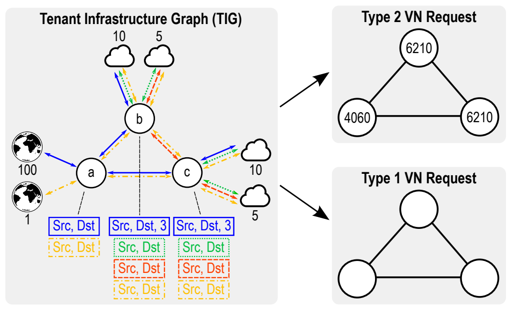


Fig. 5. Possible outputs of the Privacy-aware Compiler for a given TIG.

If no network functions need to be applied to a particular communication pattern, its flow table requirements are calculated by adding up the total number of source-destination pairs (including all applications and network masks) that are part of this traffic. The pattern represented in dashed and dotted yellow on the TIG shown in Figure 5 interconnects a single external user (or network mask) to all (30) applications running within the network, adding up to a total of 60 source-destination pairs (considering both directions of each possible communication flow). Therefore, this flow requires a total of 60 rules on each router it traverses (a, b, and c). This is also the case for the flow represented in solid blue when traversing router a. This flow interconnects 100 external network prefixes to 20 network applications, which – accounting for all possible combinations in both directions – adds up to a total of 4,000 source-destination pairs (and, therefore, 4,000 rules to be installed in a). If additional traffic discriminators are used, they enter the calculation as multiplying factors – the number of flow rules is multiplied by the number of entries in each discriminator. As an example, if the DSCP field is used as a discriminator with 3 possible QoS values (*i.e.*, 3 different traffic classes), the number of flow rules is multiplied by 3. In the example shown in Figure 5, packets that belong to the solid blue communication pattern traversing routers b and c are subjected to this traffic discriminator. Therefore, the total number of source-destination pairs (2,000) connected (directly or indirectly) to each of these routers is multiplied by the number of entries in the respective traffic discriminator (3), adding up to a total of 6,000 flow rules to be installed on routers b and c.

After being processed, the TIG is compiled into a VN request which will be shared with the InP. This request may contain more or less information according to what the customer is willing to reveal. The right side of Figure 5 shows the two different types of requests we consider. A "type 1" request is equivalent to a standard VN request. A "type 2" request, in contrast, includes the accurate number of flow rules required by each router. While not represented in this figure, standard VN requirements – namely, the throughput capacity of routers, bandwidth capacity of links, and location constraints – are also considered for both types. We envision that a larger gradient of VN request types could be considered. As an example, an intermediate level between our "type 1" and "type 2" requests could contain estimates for flow table requirements rather than exact values. We intend to further explore this aspect in future work.

[1]The source or destination of a packet, for example, may be composed of a combination of the IP address, MAC address, network protocol, and port fields.

D. SDN/OpenFlow-aware Embedder

The SDN/OpenFlow-aware Embedder is run by the InP, receiving VN requests that have been preprocessed by the Privacy-aware Compiler and embedding them on a physical substrate. It has been modeled as an Integer Linear Program (ILP), and its formulation is presented next. 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). Moreover, subscript letters represent indices associated to variables or paths.

Topologies: The topology of each VN request, as well as that of the physical network, are represented as a directed graph $N = (R, L)$. Bidirectional links are represented by pairs 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: The capacity of physical routers is measured in terms of throughput. The capacity of a physical router i is expressed as T_i^P. Likewise, $T_{r,i}^V$ denotes the throughput required by virtual router i from VN r. Likewise, the bandwidth capacity of a physical link (i, j) is represented as $B_{i,j}^P$, and the bandwidth requirement of a virtual link (k, l) from VN r is represented as $B_{r,k,l}^V$.

Locations: All physical routers are associated with a location identifier – an integer number stored in set S^P. This enables customers to demand some of their virtual routers to be instantiated in specific geographic locations. If a virtual router has a location requirement, it is stored in set S^V.

Flow Table Usage: As previously explained, the flow table requirements of VN requests are calculated by the Privacy-aware Compiler based on a given TIG and added to the generated request. The flow table capacity of a physical router is divided in two – one part (the majority of the available flow table space) will be used for requests with specific flow table requirements (type 2), while the remaining capacity will be used for type 1 requests. The flow table capacity of a physical router i reserved for type 2 requests is represented as F_i^P, while the remaining capacity reserved for type 1 requests is represented as $\mathcal{F}_i^P$. As for virtual routers, those that belong to type 2 VN requests have their flow table requirement represented as $F_{r,j}^V$, while the estimated flow table requirement for type 1 requests is represented as $\mathcal{F}_{r,j}^V$.

Previous Mappings: As VN requests are handled in an online manner, the mappings of previously embedded VNs must be taken into account and preserved while processing new incoming requests. Mappings of previously embedded routers and links are stored in sets $E_{i,r,j}^R$ and $E_{i,j,r,k,l}^L$, respectively.

The variables of the ILP model indicate the optimal placement of routers and links on the substrate.

- $A_{i,r,j}^R \in \{0, 1\}$ – Router allocation, indicates whether virtual router j from VN r is embedded on physical router i.
- $A_{i,j,r,k,l}^L \in \{0, 1\}$ – Link allocation, indicates whether virtual link (k, l) from VN r is embedded on physical link (i, j)

Next, we present the objective function of our SDN/OpenFlow-aware Embedder and its constraint sets (C1–C9). The objective function aims at minimizing overall flow table occupation – *i.e.*, the aggregated number of flow table entries needed to instantiate incoming VN requests. The calculation of flow table usage will be presented in further detail after all constraint sets are listed and explained.

Objective:

$$Minimize \sum_{(i,j)\in L^P, r\in N^V, (k,l)\in L^V} \frac{(\min_{(F_{r,k}^V, F_{r,l}^V)} + \min_{(\mathcal{F}_{r,k}^V, \mathcal{F}_{r,l}^V)}) A_{i,j,r,k,l}^L (1 - A_{i,r,k}^R)}{2} + \sum_{i\in R^P, r\in N^V, k\in R^V} (F_{r,k}^V + \mathcal{F}_{r,k}^V) A_{i,r,k}^R$$

Subject to:

$$\sum_{j\in R^P, r\in N^V, (k,l)\in L^V} \frac{\min_{(T_{r,k}^V, T_{r,l}^V)} A_{i,j,r,k,l}^L (1 - A_{i,r,k}^R)}{2} + \sum_{r\in N^V, k\in R^V} T_{r,k}^V A_{i,r,k}^R \le T_i^P \quad \forall i \in R^P \quad \text{(C1)}$$

$$\sum_{r\in N^V, (k,l)\in L^V} B_{r,k,l}^V A_{i,j,r,k,l}^L \le B_{i,j}^P \quad \forall (i,j) \in L^P \quad \text{(C2)}$$

$$\sum_{j\in R^V} A_{i,r,j}^R \le 1 \quad \forall i \in R^P, r \in N^V \quad \text{(C3)}$$

$$\sum_{j\in R^P, r\in N^V, (k,l)\in L^V} \frac{\min_{(F_{r,k}^V, F_{r,l}^V)} A_{i,j,r,k,l}^L (1 - A_{i,r,k}^R)}{2} + \sum_{r\in N^V, k\in R^V} F_{r,k}^V A_{i,r,k}^R \le F_i^P \quad \forall i \in R^P \quad \text{(C4)}$$

$$\sum_{j\in R^P, r\in N^V, (k,l)\in L^V} \frac{\min_{(\mathcal{F}_{r,k}^V, \mathcal{F}_{r,l}^V)} A_{i,j,r,k,l}^L (1 - A_{i,r,k}^R)}{2} + \sum_{r\in N^V, k\in R^V} \mathcal{F}_{r,k}^V A_{i,r,k}^R \le \mathcal{F}_i^P \quad \forall i \in R^P \quad \text{(C5)}$$

$$\sum_{i\in R^P} A_{i,r,j}^R = 1 \quad \forall r \in N^V, j \in R^V \quad \text{(C6)}$$

$$\sum_{j\in R^P} A_{i,j,r,k,l}^L - \sum_{j\in R^P} A_{j,i,r,k,l}^L = A_{i,r,k}^R - A_{i,r,l}^R \quad \forall r \in N^V, (k,l) \in L^V, i \in R^P \quad \text{(C7)}$$

$$jA_{i,r,k}^R = lA_{i,r,k}^R \quad \forall (i,j) \in S^P, r \in N^V, (k,l) \in S^V \quad \text{(C8)}$$

$$A_{i,r,j}^R = E_{i,r,j}^R \quad \forall (i,r,j) \in E^R \quad \text{(C9)}$$

$$A_{i,j,r,k,l}^L = E_{i,j,r,k,l}^L \quad \forall (i,j,r,k,l) \in E^L \quad \text{(C10)}$$

Constraint sets C1 and C2 ensure, respectively, that the throughput capacity of physical routers and that the bandwidth capacity of physical links is not exceeded. C3 prevents multiple virtual routers from a single VN request from being mapped to the same physical router. C4 and C5 ensure that the flow table capacity of physical routers is not exceeded. Constraint set C4 deals with type 2 requests, while C5 handles type 1 requests. C6 guarantees that all routers in an incoming VN request are mapped to physical routers. C7 ensures that each virtual link is mapped to a physical path whose end-points match the physical routers hosting the end-points of this link. C8 ensures all virtual routers with location requirements are mapped to physical routers in the correct location. Last, constraint sets C9 and C10 preserve the mappings of previously embedded VNs.

For the sake of clarity, our objective function, as well as constraint sets C1, C4, and C5, are shown in non-linear form. However, in practice, they are linearized by replacing the multiplication $A^L_{i,j,r,k,l}(1 - A^R_{i,r,k})$ with an auxiliary variable $Z_{i,j,r,k,l} \in \{0, 1\}$ and adding constraint sets C11, C12, and C13 – shown below – to the model. Moreover, function $\min(a, b)$ returns the lowest number between a and b and can be defined as $\frac{1}{2}(a + b - |a - b|)$.

$$Z_{i,j,r,k,l} <= A^L_{i,j,r,k,l} \quad \forall(i,j) \in L^P, r \in N^V, (k,l) \in L^V \quad \text{(C11)}$$

$$Z_{i,j,r,k,l} <= (1 - A^R_{i,r,k}) \quad \forall(i,j) \in L^P, r \in N^V, (k,l) \in L^V \quad \text{(C12)}$$

$$Z_{i,j,r,k,l} >= (1 - A^R_{i,r,k}) + A^L_{i,j,r,k,l} - 1 \quad \forall(i,j) \in L^P, r \in N^V, (k,l) \in L^V \quad \text{(C13)}$$

In order to properly account for flow table usage, the objective function must not only consider explicit flow table requirements of virtual routers, but also the flow rules that must be installed on auxiliary routers through which virtual links traverse. The number of rules that must be installed on the auxiliary routers used by a virtual link (k, l) corresponds to the lowest number between the flow table requirements of virtual routers k and l. In the objective function, the first summation refers to the flow table constraints of auxiliary routers, while the second one refers to that of physical routers hosting virtual routers of each network. Constraint sets C1, C4, and C5 employ the same strategy in order to compute the throughput and flow table usage of auxiliary routers.

IV. Evaluation

We now proceed to a performance evaluation of our proposed VNE and SDN coordination approach. Experiments were performed on a machine with an Intel Core i5 4278U CPU, 8 GB of RAM and Operating System Mac OS X 10.10.5. The previously introduced ILP model was implemented and run using the IBM ILOG CPLEX Interactive Optimizer 12.4.

A. Workloads

In order to evaluate our proposal, we developed a simulator that creates virtual topologies according to a series of parameters. Each virtual topology is then converted to a VN request in the format required by our ILP model. The simulator is run for 500 rounds, generating a new request on each one[2]. If accepted, VNs remain embedded for 25 rounds before being removed.

Fixed parameters: In all experiments, physical and virtual topologies are generated with BRITE using the Barabási-Albert model [13]. Generated physical networks contain 100 routers. Physical routers have a throughput capacity of 150 Gbps, while physical links have a bandwidth capacity of 30 Gbps. The flow table of each device is capable of storing up to 16,000 rules. Physical routers are uniformly distributed among 16 geographic locations.

Each generated VN request contains 5 routers. Virtual router throughput and link bandwidth requirements are, respectively, 50 Gbps and 10 Gbps. Each VN has two edge routers with randomly generated location requirements. 50% of the generated VNs are type 1 requests, while the remaining 50% are type 2 requests. Flow table requirements of type 2 requests are set to 3,000 rules per router, while those of type 1 requests (which are not known) are treated in different ways according to the experiment being performed.

Variable parameters: We performed a number of different experiments with variations regarding flow table space reserved for type 2 requests as well as how flow table requirements are considered. The first three experiments – henceforth referred to as "Flow-70/30", "Flow-80/20", and "Flow90/10" – reserve 70%, 80%, and 90% of each physical router flow table space for type 2 requests. The remaining flow table space is reserved to accommodate type 1 requests. As we do not know their requirements, a minimal set of 1,500 rules is reserved for each router from these networks. The idea here is to deliberately allocate little table space to these virtual routers[3].

In the fourth experiment, all flow table space is used to embed type 2 requests, while accepted type 1 requests are supported on a "best effort" manner. More specifically, type 1 requests are embedded as long as their other capacity requirements (throughput and bandwidth) can be fulfilled, with no flow table space guarantees. This experiment is referred to as "Flow-100/0'. In the last experiment, no flow table requirements are considered by the embedding model. This experiment behaves similarly to an environment running a traditional VNE algorithm and is carried out to assess the impact of our proposed approach. This experiment is referred to as "NoFlow".

B. Results

We first analyze the overall acceptance rate in all experiments, shown in Figure 6. The acceptance rates achieved throughout experiments Flow-70/30, Flow-80/20, and Flow-90/10 were, respectively, 70.2%, 64.2%, and 55.8%. As the residual flow table space for type 1 requests decreases, more VNs of this type (which require less resources and represent half of all generated requests) are rejected, leading to lower acceptance rates. Although the overall acceptance rate is lower, this, in turn, favors the acceptance of a higher amount of type 2 requests. This is a desirable outcome for the InP in order to prevent under- or overestimation of resources (as type 2 requests contain precise flow table occupation requirements), potentially leading to improved resource usage (in the case of overestimation) and/or lower rates of controller intervention (if flow table requirements are underestimated). The acceptance rates of individual types of requests and their effect on the rate of controller interventions will be further analyzed in the remainder of this section.

The acceptance rates observed in experiments Flow-100/0 and NoFlow were higher than those of other experiments – 72.6% and 73%, respectively. This is due to the former not reserving any flow table space for type 1 requests and the latter disregarding flow table requirements entirely. While seemingly a positive result at first glance, this is likely to result in severe underestimation of resources needed to adequately support the embedded VNs, potentially leading to high rates of controller intervention. We emphasize that both Flow-100/0 and NoFlow are used as baseline scenarios, *i.e.,* the best results one would achieve in terms of accepted requests at the cost of compromising network predictability and, in extreme cases, its technical feasibility.

Next, Figure 7 depicts the acceptance rate of each type of request in all evaluation scenarios. In experiments Flow-70/30, Flow-80/20, and Flow-90/10, the acceptance rates of type 1 requests were, respectively, 73.7%, 57.7%, and 30.83%. Acceptance rates of type 2 requests observed for the same experiments were of 67.18%, 69.77%, and 82.04%, respectively. These increasing acceptance rates of type 2 requests are a desirable outcome for InPs, as they would likely desire to prioritize the embedding of this type of request. This happens as a result of the fine-tuning of the amount of flow entries reserved for each type of request, leading to an acceptance rate of over 80% for type 2 requests in the most extreme scenario (Flow-90/10). InPs may fine-tune these reservations as desired

[2]On average, each request was optimally mapped in less than 5 seconds.

[3]The amount of flow table space reserved for each router in type 1 requests may be fine-tuned as desired by the InP.

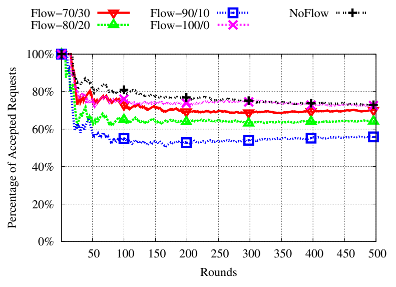


Fig. 6. Overall acceptance rate in all experiments.

in order to allow more or less of each type of request to be embedded and potentially minimize issues caused by type 1 requests (as a result of under- or overestimation of operational requirements). Moreover, all scenarios exhibit variations in acceptance rates within the first 250 rounds. In scenario Flow-70/30, the acceptance rate of type 1 requests is initially higher than that of type 2 requests. In scenarios Flow-80/20 and Flow-90/10, in turn, acceptance rates for both types of requests either increase or decrease during the first 250 rounds. This can be attributed to the fact that the substrate is completely empty at the beginning of the experiment, requiring some time for acceptance rates to stabilize. In all cases, acceptance rates become stable towards the last 250 rounds.

In the remaining experiments (Flow-100/0 and NoFlow), acceptance rates of type 1 requests were, respectively, 73.4% and 70.4%. Acceptance rates of type 2 requests, in turn, were of 72.1% and 75.49%, respectively. As previously explained, the former does not reserve any flow table space for type 1 requests while the latter completely disregards flow table requirements. Therefore, the main causes of rejection are likely related to topological factors or throughput/bandwidth resource scarcity, leading to similar acceptance rates for both types of requests on both experiments.

Last, we analyze the potential impacts of accurately or inaccurately estimating operational requirements. To this end, we consider that VN requests in each scenario have been embedded and deployed, and we calculate the number of flow rules each router would need to have installed in order to support the embedded VNs. We first assume the 1,500 rules reserved for each router of type 1 requests were sufficient. Afterwards, we assume this lower bound was not adequate and that type 1 requests would actually require 3,000 flow rules per router. These scenarios represent extreme cases – assuming either that a minimal set of flow rules was sufficient or that actual requirements exceed this lower bound by a factor of 2. Our goal is to determine in which cases the number of necessary flow rules would exceed the capacity of physical routers. As previously mentioned, if physical devices are not able to accommodate all necessary flow rules, they would need to frequently contact the controller in order to handle incoming packets. This, in turn, would likely degrade the performance of physical devices, hindering the quality of service experienced by customers.

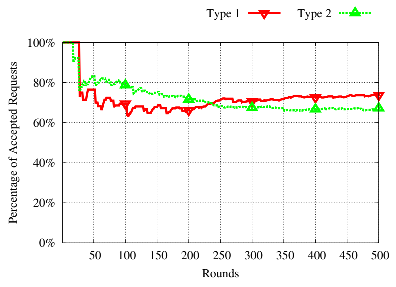
Figure 8 depicts the average number of flow rules that would exceed the capacity of physical routers in each experiment, considering the scenarios just described. Assuming the flow table usage of type 1 requests fits within the minimal provided flow table space, no exceeding rules were observed in scenarios Flow-70/30, Flow-80/20, and Flow-90/10. The ILP model used in these experiments takes into account flow table constraints for both types of requests, ensuring that the capacity of physical devices will not be exceeded as long as the reserved flow table space is sufficient. In experiment Flow-100/0, in turn, the average number of exceeding flow rules was 10,732, all belonging to type 1 requests. This is due to this experiment disregarding flow table requirements for this type of request, embedding them in a "best effort" manner. In experiment NoFlow, which mirrors the behavior of a standard VNE algorithm without flow-related constraints, the average number of exceeding flow rules was significantly higher, with both types of VN requests contributing to flow table saturation. More specifically, an average of 27,593 exceeding rules were observed (2,331 incurred by type 1 requests and 25,262 incurred by type 2 requests). The substantial numbers of exceeding flow rules observed in scenarios Flow-100/0 and NoFlow highlight the importance of considering this operational constraint in the embedding process.

Last, assuming the flow table usage of type 1 requests exceeds the minimal provided flow table space (a likely scenario in practice), all experiments exhibit flow table saturation. In experiments Flow-70/30, Flow-80/20, and Flow-90/10, the average numbers of exceeding flow rules were, respectively, 2,731, 19,783, and 18,130. Although these experiments took into account flow-related constraints, assuming flow table usage exceeds the established lower bounds led to significant saturation – particularly in scenarios Flow-80/20 and Flow-90/10, in which less resources were reserved for this type of request. The highest numbers of exceeding flow rules were observed in experiments Flow-100/0 and NoFlow – which, as previously explained, either do not reserve flow table space for type 1 requests or disregard flow constraints entirely. The average amount of exceeding flow rules observed in experiment Flow-100/0 was of 57,970 – all incurred by type 1 requests. In experiment NoFlow, an average of 22,886 exceeding flow rules were incurred by type 1 requests and 25,262 by type 2 requests, adding up to a total of 48,148. These results evidence that, in addition to the aforementioned importance of considering operational constraints, accurately determining flow table requirements plays a crucial role in ensuring the feasibility of supporting the embedded VNs. Regarding the former, it is important to note that, with the exception of scenario NoFlow, all rules of type 2 requests were properly installed in physical routers. Therefore, these VNs will be able to operate with minimal controller interventions, minimizing potential negative impacts on quality of service (as controller intervention may increase latency by up to twice the round-trip time between the switch and the controller [14]).

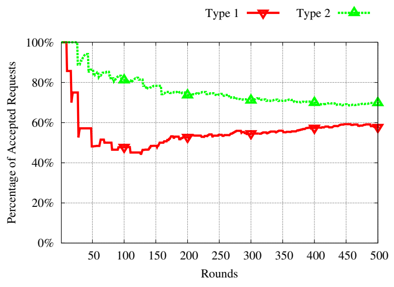
V. Conclusions

Although a substantial body of work exists in the area of virtual network embedding, existing approaches do not take into account relevant operational constraints related to the instantiation of VNs on different embedding platforms. In the case of Software-Defined Networking, which offers a promising platform for network virtualization, memory space for storing flow rules is often limited, becoming a crucial operational constraint that may render VNE-provided mappings unsuitable for real environments due to unpredictable or even unsatisfactory VN performance. At the same time, demanding information regarding such operational constraints from customers may be unrealistic as they may be either not aware of how their VN affects the InP's substrate at the physical level or unwilling to share detailed information about the inner working of their VNs.

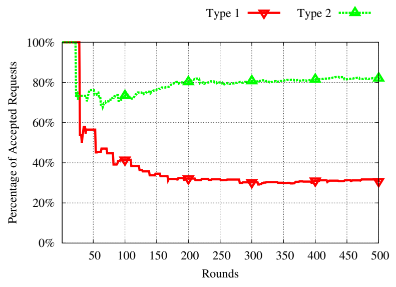
Based on this reasoning, we proposed an abstraction model for expressing requirements related to internal VN policies and traffic patterns. This model – the Tenant Infrastructure Graph – is built in a way that is familiar to customers in a network virtualization environment. Moreover, it is preprocessed on the customer's end by a Privacy-aware Compiler in order to derive information that is valuable to the InP and, at the



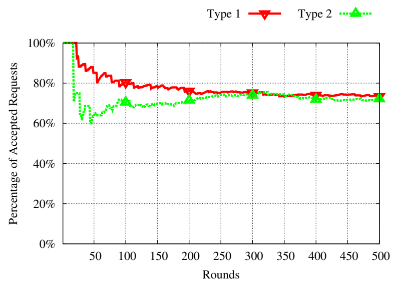
(a) Flow-70/30



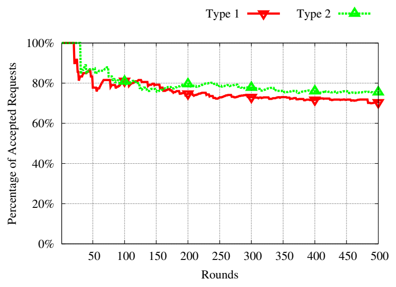
(b) Flow-80/20



(c) Flow-90/10

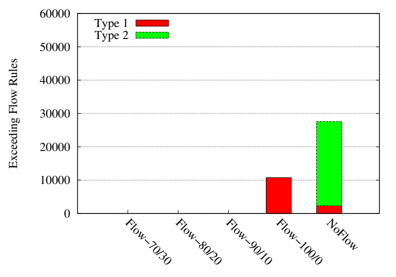


(d) Flow-100/0

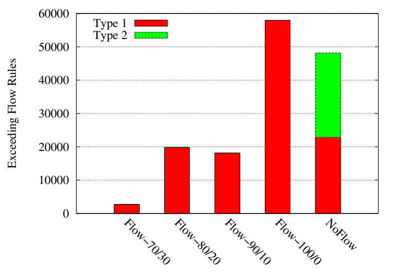


(e) NoFlow

Fig. 7. Acceptance rate of requests per TIG type in all experiments.



(a) Assuming flow table space allocated for type 1 requests was sufficient.



(b) Assuming flow table space allocated for type 1 requests was insufficient.

Fig. 8. Number of flow rules exceeding the capacity of physical routers.

same time, remove sensitive data that the customer may not be willing to disclose. The output of this compiler is then forwarded to the InP, which can employ the SDN/OpenFlow-aware Embedder to ensure embedded VNs do not break any operational constraints of its network virtualization platform.

Through a comprehensive evaluation, we demonstrated that taking the aforementioned operational constraints into account is of paramount importance to maintain a desired level of quality of service. Neglecting such constraints may render the environment unable to cope with a substantial number of flow rules that are crucial to ensure correct VN behavior. As physical devices become unable to store all necessary flow rules internally, they need to frequently contact the controller in order to route incoming packets, which, in turn, may lead to significant performance degradation for VNs hosted on such devices. In our experiments, assuming the flow table space reserved for requests with unknown requirements was sufficient, the proposed approach was able to eliminate controller intervention due to flow table saturation. Additionally, assuming the actual requirements of such requests exceeded the allocated space by a factor of 2, the number of exceeding flow rules was still reduced by 40.8% on average (compared to a traditional VNE approach). Moreover, the reduction of acceptance rates due to the added constraints was limited to, on average, 9.6%. Our proposed approach enables InPs to accurately assess operational constraints and correctly embed incoming requests without violating these constraints. Moreover, by adjusting the ratio of flow table space dedicated to different types of incoming requests, the InP may choose to which degree requests that include all the necessary information are favored in detriment of requests that do not (and that, therefore, rely on estimation of necessary resources in order to be embedded). Perspectives for future work include: *(i)* taking into account other SDN/OpenFlow-related operational constraints; *(ii)* considering a larger gradient of VN request types with varying amounts of information; and *(iii)* including a negotiation phase between the customer and the InP, providing alternatives for cases in which available resources are not sufficient to embed requested VNs.

Acknowledgments

This work has been supported by the Brazilian National Council for Scientific and Technological Development (CNPq).

References

[1] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: substrate support for path splitting and migration," *SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 17–29, Mar. 2008.

[2] M. Chowdhury, M. R. Rahman, and R. Boutaba, "Vineyard: Virtual network embedding algorithms with coordinated node and link mapping," *IEEE/ACM Transactions on Networking*, vol. 20, no. 1, pp. 206–219, Feb. 2012.

[3] X. Cheng, S. Su, Z. Zhang, H. Wang, F. Yang, Y. Luo, and J. Wang, "Virtual network embedding through topology-aware node ranking," *SIGCOMM Computer Communication Review*, vol. 41, no. 2, pp. 38–47, Apr. 2011.

[4] G. P. Alkmim, D. M. Batista, and N. L. S. Fonseca, "Mapping virtual networks onto substrate networks," *Journal of Internet Services and Applications*, vol. 3, no. 4, pp. 1–15, 2013.

[5] L. R. Bays, R. R. Oliveira, L. Buriol, M. P. Barcellos, and L. Gaspary, "A heuristic-based algorithm for privacy-oriented virtual network embedding," in *IEEE Network Operations and Management Symposium (NOMS)*, May 2014, pp. 1–8.

[6] M. Demirci and M. Ammar, "Design and analysis of techniques for mapping virtual networks to software-defined network substrates," *Computer Communications*, vol. 45, pp. 1 – 10, 2014.

[7] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer," *OpenFlow Switch Consortium, Technical Report*, 2009.

[8] Z. Bozakov and P. Papadimitriou, "Autoslice: automated and scalable slicing for software-defined networks," in *ACM Conference on Emerging Networking Experiments and Technologies*. ACM, 2012, pp. 3–4.

[9] R. Doriguzzi Corin, M. Gerola, R. Riggio, F. De Pellegrini, and E. Salvadori, "Vertigo: network virtualization and beyond," in *European Workshop on Software Defined Networking*. IEEE, 2012, pp. 24–29.

[10] E. Salvadori, R. D. Corin, A. Broglio, and M. Gerola, "Generalizing virtual network topologies in openflow-based networks," in *IEEE Global Telecommunications Conference*. IEEE, 2011, pp. 1–6.

[11] D. Drutskoy, E. Keller, and J. Rexford, "Scalable network virtualization in software-defined networks," *IEEE Internet Computing*, vol. 17, no. 2, pp. 20–27, 2013.

[12] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, G. Parulkar, E. Salvadori, and B. Snow, "Openvirtex: make your virtual sdns programmable," in *Workshop on Hot topics in Software Defined Networking*. ACM, 2014, pp. 25–30.

[13] R. Albert and A.-L. Barabási, "Topology of evolving networks: Local events and universality," *Physical Review Letters*, vol. 85, pp. 5234–5237, Dec 2000.

[14] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: Scaling flow management for high-performance networks," *SIGCOMM Computer Communication Review*, vol. 41, pp. 254–265, 2011.