Policy-Compliant Virtual Network Embedding

David Dietrich Panagiotis Papadimitriou
Institute of Communications Technology, Leibniz Universität Hannover, Germany
{*firstname.lastname*}*@ikt.uni-hannover.de*

***Abstract*—Virtual network embedding algorithms that optimize the mapping of virtual network (VN) topologies onto a substrate network usually do not comply with the policies of substrate providers which may prefer to identify and embed the most profitable subset of a VN request. Such policy-based VN embedding (VNE) is required by distributed VNE architectures, such as PolyVine, or by auction-based VNE environments.**

In this paper, we introduce a policy dimension to VNE by proposing a new VNE algorithm that aims at maximizing the revenue without violating the provider's policy. In contrast to the greedy nature of most VNE techniques, our algorithm allows a provider to trade short-term revenue gains for higher revenue in the long term and cope better with evolving demands. Our simulation results corroborate the efficiency of our VNE algorithm and show the impact of diverse policy adjustments on resource utilization and generated revenue.

I. Introduction

Network virtualization has been seen as a viable path towards a diversified Internet that supports a variety of network services [1], [11]. Network virtualization carries significant benefits to service providers and Physical Infrastructure Providers (InP). More precisely, a service provider can deploy a network service within a customized virtual network (VN) that provides performance and reliability guarantees. Furthermore, the elastic provisioning in virtualized infrastructures allows existing VN allocations to be expanded on demand, obviating the need for large investment costs in advance. Network virtualization also creates new opportunities for InPs to increase their revenue, while the separation between the operations and the physical infrastructures can result in significant operational cost savings.

Recently, there has been an increasing interest in wide-area VN provisioning [11], [16], [2], [6], [4]. Wide-area VN deployments typically span multiple InPs, due to the limited geographic footprint of most network providers. Multi-provider VN provisioning entails significant challenges, primarily due to InP policies that restrict the disclosure of detailed network topology and resource information, and hinder interoperability with other parties [4]. To address this issue, network virtualization architectures have taken two different approaches. The architectures in [11], [16] rely on a layer of indirection, usually known as a VN provider, which partitions VN requests among InPs and subsequently, each InP maps his corresponding VN segment to his own substrate network. The other approach consists in relaying VN requests across InPs till the completion of VN embedding, as exemplified in PolyVine [2]. Essentially, each InP embeds the subset of the VN request which is more profitable and relays the remaining request to a neighboring InP.

ISBN 978-3-901882-58-6

Distributed VN embedding architectures, such as PolyVine, raise the requirement for policy-compliant VN embedding, i.e., algorithms that identify and embed the most profitable subset of a VN request. This is also a prerequisite for auction-based VNE architectures [14], in which each InP bids for the VN request subset whose mapping yields higher efficiency. Existing VNE techniques [3], [5], [9], [13], [15] do not fulfill this requirement, as they seek to compute the embedding of a complete VN topology, and eventually reject the VN request when this is not feasible.

In this paper, we introduce a policy dimension to the VNE problem, allowing only profitable VN embeddings according to the InP's policy. Such policy can express the balance between the generated revenue and resource efficiency as well as specific constraints that the provider wants to apply. In particular, we use the embedding cost to revenue ratio (CRR) in order to express the profit that a provider generates from the embedding of a VN request. In this respect, an InP can set an upper bound to CRR, adjusting the trade-off between revenue generation rate and resource efficiency. This CRR threshold can be adjusted dynamically based on the substrate network resource utilization and VN request arrival rate. Relying on CRR bounds for expressing VNE policies, we present a VNE algorithm that embeds the most profitable subset of a VN request according to the InP's policy. Our simulation results show that our algorithm can increase the generated revenue by a large margin depending on the policy adjustment. Our algorithm also enables an InP to trade short-term revenue gains for higher revenue in the long term and cope better with evolving demands. The proposed algorithm can comprise an essential component of any multi-provider VNE architecture that relies on VN request relaying across providers or employs auction mechanisms.

The remainder of the paper is organized as follows. In Section II, we define our network models and VNE metrics. In Section III, we introduce a policy dimension to VNE. Section IV exemplifies our algorithm for policy-based VNE. In Section V, we present our simulation results and discuss the efficiency of our algorithm. Section VI discusses related work. Finally, in Section VII, we highlight our conclusions.

II. Network Model and VNE Metrics

In this section, we introduce the substrate and virtual network models, and define metrics for VNE.

A. Network Model

Substrate Network Model. The substrate network is represented as a weighted directed graph $G_s = ((N_s \cup P), L_s)$, where N_s is the set of the substrate nodes and L_s is the set of substrate links between these nodes. The set P comprises all the nodes of the substrate network where peering has been established with other network providers. Each substrate node $u \in N_s$ is associated with the residual capacity $r_u \in R_N$. Each substrate link $(u,v) \in L_s$ between two substrate nodes u and v is associated with the residual capacity denoted by $r_{uv} \in R_L$.

VN Request Model. A VN request consists of a combined set of the virtual nodes N_v and the used peering nodes $P' \subset P$, and a set of bandwidth demands $d^{ij} \in D_L$ between any pair of virtual nodes $i, j \in N_v$ and between any virtual node $i \in N_v$ and peering node $j \in P'$. We use a traffic matrix to express all bandwidth demands. Compared to topology-based VN requests that are typically used by the majority of VNE algorithms, traffic-matrix based VN requests allow more flexibility in VNE and a higher level of abstraction which can simplify the specification of network service requirements [4], [12]. We include the peering nodes P' in order to accumulate the bandwidth demands between virtual nodes assigned to different InPs, as shown in Fig. 1. Furthermore, each virtual node $i \in N_v$ is associated with the requested capacity denoted by $g^i \in D_N$.

The mapping of the virtual nodes to the substrate nodes is expressed as $M_{Nv} : N_v \rightarrow N_s$.

B. VNE Metrics

We initially define the *revenue* of a VN request as:

$$\mathbb{R} = \sum_{i \in N_v} g^i + \alpha \sum_{\substack{i,j \in N_v \\ (i \neq j)}} d^{ij} + \left\{ \frac{\alpha}{2} \sum_{i \in N_v, j \in P'} (d^{ij} + d^{ji}) \right\} \quad (1)$$

Revenue essentially accumulates all the node and link capacity units of the VN request. The revenue generated from peering links is split between the two InPs. Similar to [13], the parameter α is used as a weight in the sum of the computing and the bandwidth revenues. For the purpose of revenue-based node reordering, as required by our algorithm, we further define the revenue of the virtual node $i \in N_v$, as follows:

$$\mathbb{R}(i) = g^i + \frac{\alpha}{2} \sum_{\substack{j \in N_v \cup P' \\ (i \neq j)}} d^{ij} + d^{ji} \quad (2)$$

The *VN embedding cost* is defined as follows:

$$\mathbb{C} = \sum_{i \in N_v} g^i + \alpha \sum_{\substack{i,j \in N_v \cup P' \\ (i \neq j)}} d^{ij} \cdot dist(u_i, u_j) \quad (3)$$

$$u_i, u_j \in M_{Nv}(N_v) \cup P'$$

The VN embedding cost accumulates all node and link costs.

An InP will seek to increase his revenue while maintaining a low embedding cost. As such, we use the *Cost-to-Revenue ratio (CRR)* = $\mathbb{C}/\mathbb{R}$ to express VNE efficiency and eventually, how much profit an InP gains by embedding a VN request. Based on the preceding definitions, a CRR of 1.0 yields the highest profit.

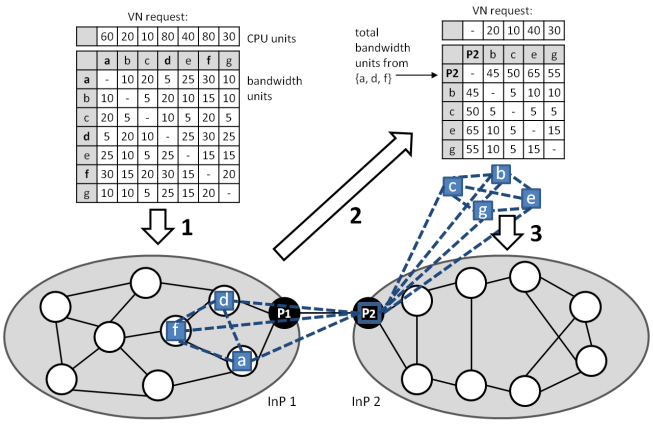


Fig. 1. VN request example.

III. Problem Description

VNE consists in mapping VN topologies onto a shared substrate network. Existing VNE algorithms optimize VN assignment based on objectives, such as maximizing VN acceptance rate, minimizing VN embedding cost, maximining revenue for the InP or achieving load balancing across the substrate network [3], [5], [9], [13], [15]. A common feature among these VNE algorithms is that they embed complete VNs, exactly as requested, as long as there are sufficient substrate network resources. Complete VN request embedding is more appropriate for small VNs that can be mapped onto a single substrate provider with a relatively low embedding cost. Larger VN requests that possibly exceed the geographic footprint of a substrate provider have to be partitioned across multiple InPs.

One approach to multi-provider VN embedding is to allow the InPs to embed the subset of a VN request that generates more profit. More precisely, in auction-based VNE environments the InP will seek to place a higher bid for the most profitable subset of a VN request [14], while in a distributed VNE framework the InP will seek to identify and embed the profitable VN subset and subsequently relay the rest of the VN request to a neighboring InP, as exemplified by PolyVine [2]. Both VNE frameworks raise the requirement for embedding only the profitable subset of a VN request, which is not fulfilled by any of the existing VNE algorithms, according to our knowledge. We note that the profit gained by embedding a particular VN request subset may vary among different InPs, and consequently, unprofitable VN subsets relayed from a certain InP can be efficiently embedded onto other neighboring InPs.

Fig. 2 illustrates the evolution of CRR during the iterative embedding of VN requests with 10, 15, and 20 nodes. Irrespective of the VN size, CRR increases as a larger subset of the VN is being mapped. We observe a more severe increase in the CRR for larger VNs (i.e., 20 nodes). As such, embedding complete VN requests of large size incurs a penalty in terms of resource efficiency. This penalty stems from the increasing

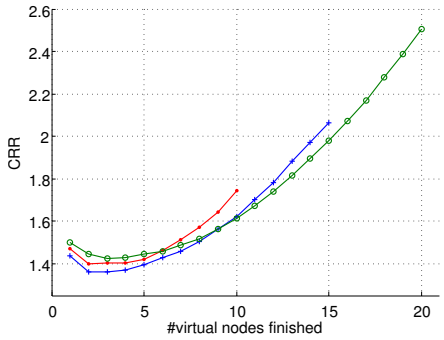


Fig. 2. CRR evolution.

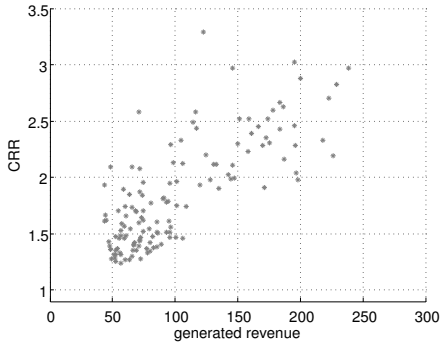


Fig. 3. CRR vs. generated revenue.

hop count of virtual links, which counterbalances the extra revenue generated by embedding additional virtual nodes and links. Fig. 3 also corroborates this observation, uncovering a correlation between the CRR and the VN revenue.

Based on these observations, we rely on CRR to identify the most profitable subset of a VN request. As such, we seek the embedding of the largest subset of a VN request that does not violate a predefined CRR threshold. In principle, this threshold should be adjusted by the InP according to its policy. The CRR threshold represents a trade-off between revenue and resource efficiency. More precisely, a low CRR threshold is expected to yield higher resource efficiency but will generate revenue at lower rate, since only small subsets of VN requests will be embedded. Conversely, a high CRR threshold will generate revenue faster but may impair resource efficiency. To this end, in the following section we present a new algorithm that enables the embedding of subsets of VN requests according to an adjustable CRR threshold. The impact of CRR threshold adjustment on resource efficiency and generated revenue is further discussed in Section V-B.

IV. Policy-Compliant VNE Algorithm

In this section, we describe the proposed policy-compliant VNE algorithm. In the following, we give an overview of the algorithm (Section IV-A), present an exemplary VN embedding (Section IV-B), and provide further details about the algorithm (Section IV-C).

A. Algorithm Overview

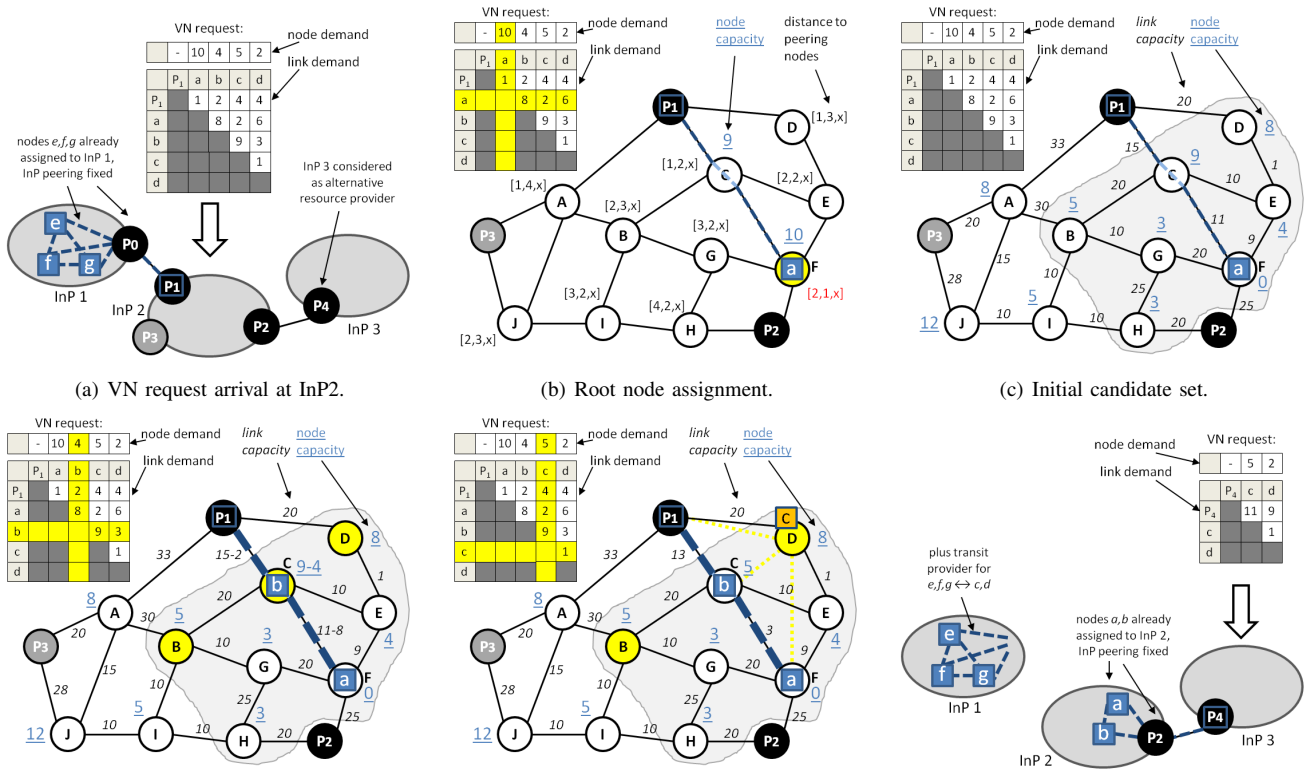
The main objective of this algorithm is to identify and embed the subset of a VN request that generates the highest revenue without violating a predefined CRR threshold. To this end, the algorithm seeks feasible solutions by iteratively embedding virtual nodes and their adjacent links. In order to maximize the generated revenue, the algorithm uses the virtual node revenue, as defined in Eq. 2, to sort the nodes of the VN request in terms of revenue in decreasing order. As such, the embedding starts with the virtual node that generates the highest revenue (called *root node*). Since virtual links spanning multiple hops increase the CRR, the assignment of each virtual node aims at minimizing the hop count of the adjacent virtual links. Therefore, for the assignment of the root node, among the substrate nodes with sufficient capacity the algorithm selects the one with the minimum total distance from the peering nodes (e.g., P_1 and P_2 for InP 2 in Fig. 4(a)). Similarly, the remaining virtual nodes are assigned iteratively. Each iteration comprises an evaluation of candidates for virtual node mapping by computing the shortest paths for each virtual link connecting the candidate node with every assigned virtual and peering node. The candidate with the minimum additional embedding cost is selected and its respective node and link mappings are taken into account when the remaining virtual nodes are mapped onto the substrate network. We reduce the solution space by defining a substrate network region that contains all nodes with a maximum distance from the substrate node on which the root node was mapped.

The algorithm keeps track of the evolution of CRR, after each virtual node has been assigned, and checks the feasibility of the current solution. In the case that the complete VN request can be embedded without violating the CRR threshold, the algorithm returns this solution and terminates its execution. Otherwise, the algorithm computes and stores the largest subset of the VN request that can be embedded without exceeding the threshold for all possible assignments of the root node. To speed up the algorithm execution, we do not consider solutions whose root node mapping is very distant from the peering nodes, since this incurs a significant penalty in terms of CRR. Finally, among all feasible solutions, the algorithm returns the VN subset that generates the highest revenue.

B. An Example

Hereby, we illustrate an exemplary VN embedding with our algorithm. Fig. 4(a) shows three InPs and a VN request being submitted to InP 2. Assume that 3 virtual nodes (i.e., e, f, g) previously belonging to this VN request have been already assigned to InP 1. The VN request consists of virtual node demands (i.e., expressed in terms of compute units) and virtual link demands (i.e., expressed in terms of bandwidth units) formulated as a symmetric traffic matrix[1] (Fig. 4(a)). Besides the bandwidth demands between each pair of virtual nodes, the traffic matrix also includes the bandwidth demand between

[1]A symmetric traffic matrix is shown for simplicity. Our algorithm can process and embed VN requests expressed with asymmetric traffic matrices.



(a) VN request arrival at InP2.

(b) Root node assignment.

(c) Initial candidate set.

(d) Second virtual node assignment.

(e) Third virtual node assignment – CRR threshold violation.

(f) VN subset embedding.

Fig. 4. Exemplary policy-compliant VN embedding.

each virtual node and the peering node P_1. This essentially expresses the sum of bandwidth demands between the virtual nodes assigned to InP 1 (i.e., e, f, g) and the nodes of this request (i.e., a, b, c, d). Note that InP 2 does not have any information about the substrate topology and the virtual node mappings on InP 1. In this example, we assume that any subset of the VN request that cannot be embedded (because its mapping is either infeasible or unprofitable) will be relayed to InP 3. Since the peering link $P_2 - P_4$ can be potentially used, the set of peering nodes within InP 2 consists of P_1 and P_2.

Based on this example, Figs. 4(b) to 4(e) illustrate the steps followed by our algorithm for the embedding of this request onto InP 2. Table I shows VN embedding statistics, including the CRR value, after the completion of each step. In this example, we consider that InP 2 has adjusted the CRR threshold to 1.3. The algorithm starts with the assignment of the root node, which is virtual node a as it generates the highest revenue (Fig. 4(b)). Two candidate substrate nodes (i.e., C, F) are being identified for a, based on the total distance from the peering nodes. Since only F satisfies the capacity requirement of a, the root node is assigned to F. Subsequently, the virtual link between P_1 and a is mapped to the substrate network.

Since the current CRR value (i.e., 1.14) does not violate the threshold, the algorithm proceeds with the mapping of the virtual node with the 2^{nd} largest revenue, i.e., virtual node b. To this end, the algorithm defines a region containing all substrate nodes with a maximum distance from the root node (set to 2 in our example). This designates the following candidate set $S := \{B, C, D, E, G, H\}$, from which all the previously assigned substrate nodes (to the same VN request) are removed (Fig. 4(c)). In addition, the following nodes are eliminated from this set: G and H due to insufficient node capacity, and E due to insufficient link capacity, as the total residual capacity of all links attached to E (i.e., 20 bandwidth units) cannot fulfill the total bandwidth demand of 21 units for b. The algorithm eventually selects the substrate node C, since it incurs the lowest embedding cost.

TABLE I
VNE STATISTICS FOR THE EXAMPLE IN FIG. 4

Fig.	$N_v \rightarrow N_s$	$\mathbb{R}$	$\mathbb{C}$	CRR	$CRR \leq CRR_{max}$
4(b)	a→F	10.5	12.0	1.14	true
4(d)	a→F, b→C	23.5	26.0	1.11	true
4(e)	a→F, b→C, c→D	43.5	59.0	1.36	false

Since there is no CRR threshold violation, the algorithm examines the assignment of the virtual node c, as depicted in Fig. 4(e). The candidate node set is $S = \{B, D\}$. Assigning c to either B or D violates the CRR threshold, and since there is no other candidate node, the VN subset comprising nodes a and b is identified as the most profitable. Thereby, the algorithm returns the mapping computed for this VN subset.

The remaining VN request (i.e., consisting of nodes c and d) is relayed to InP 3, as shown in Fig. 4(f).

C. Algorithm Details

Algorithms 1, 2, and 3 illustrate the pseudocode of the policy-compliant VNE algorithm. Algorithm 1 performs the VNE computation, while Algorithms 2 and 3 include auxiliary functions to identify candidate nodes with sufficient residual capacity and low link embedding cost.

Hereby, we explain Algorithm 1 in detail. In the lines 1–8, an initialization phase takes place. All virtual nodes are sorted in terms of revenue and subsequently, the set of candidate substrate nodes for the root node is being identified using the function *Candidate Substrate Node Preselection* in Algorithm 2. Any candidate node exceeding a maximum distance from all the peering nodes (computed based on the minimum distance and a relative threshold θ, which is adjusted to 0.1 by default) is removed from the candidate set and is no longer considered for the placement of the root node.

In the lines 10–15, the algorithm checks whether the root node can be assigned to a substrate node within the candidate set based on the availability of substrate network paths (between this node and the peering nodes) satisfying the bandwidth demands. If this is feasible, the algorithm proceeds with the assignment of the following nodes of the VN requests as long as the CRR threshold is not violated (i.e., lines 16–37); otherwise, the assignment of the root node to the next candidate node is being checked (i.e., lines 9–12). The first step for the assignment of each additional virtual node is to identify the substrate node with the minimum embedding cost of all adjacent links, including the links attached to the peering nodes, (i.e., line 17) using the function *Candidate Substrate Node Selection* in Algorithm 3. The assignment is restricted to the substrate nodes whose distance (i.e., number of hops) from the root node does not exceed a predefined threshold denoted by ζ. This essentially eliminates all inefficient solutions, i.e., substrate nodes that would significantly increase the CRR. We discuss the adjustment of the threshold ζ in Section V-B. Furthermore, we take into account the link embedding cost by computing the corresponding capacity-constrained shortest path. Eventually, the most cost-efficient candidate node is being selected.

Upon the assignment of each virtual node j, the algorithm updates the CRR value and checks for any CRR threshold violations. Instead of making decisions based only on the current CRR value (i.e., CRR(j)), we further take into account the CRR value of the previous virtual node (i.e., CRR(j-1)) in order to capture any trends in the CRR evolution. Fig. 5 illustrates the decision-making process with an example of a CRR curve, similar to Fig. 2. In particular, although $j = 1$ exceeds the CRR threshold, there is a decreasing trend in CRR, and therefore, the algorithm permits the assignment of this virtual node. A case of CRR threshold violation is represented by $j = 6$, where the current CRR value exceeds the threshold with an increasing trend. The conditions for the CRR threshold violation are shown in the line 19.

Algorithm 1 Policy-Compliant VNE

Input: $N_s, P, L_s, R_N, R_L, N_v, P', D_N, D_L, dist$

1: $j_{max} \leftarrow -1, \quad M_{Nv\_sol} \leftarrow \emptyset$
2: $CRR = \{c_0, ..., c_{|Nv|-1} | c_i = \infty\}, \quad CRR_{sol} \leftarrow CRR$
3: $R_{N,rollback} \leftarrow R_N, \quad R_{L,rollback} \leftarrow R_L$
4: Sort the nodes $i \in N_v$ by descending revenue $\mathbb{R}(i)$
5: $C \leftarrow$ **Candidate Substrate Node Preselection** (0)
6: $dist_{P,min} \leftarrow min_{u \in C}(\sum_{p \in P} dist_{p,u})$
7: $dist_{P,max} \leftarrow dist_{P,min} \cdot (1 + \theta)$
8: $C \leftarrow C \setminus \{u \in C | \sum_{p \in P} dist_{p,u} > dist_{P,max}\}$
9: **while** $C \neq \emptyset$ **and** $j_{max} < |N_v| - 1$ **do**
10: $M_{Nv} \leftarrow \emptyset$
11: $u_{root} \leftarrow arg\ min_{u \in C}(\sum_{p \in P} dist_{p,u})$
12: **if** Shortest paths with sufficient capacity exist between u_{root} and all $u \in P'$ **then**
13: $M_{Nv,0} \leftarrow u_{root}$
14: Update residual capacity for node u_{root} and for all links used between u_{root} and all $u \in P'$
15: Compute $CRR(0)$
16: **for** $j := 1..|N_v| - 1$ **do**
17: $u_{cand} \leftarrow$ **Candidate Substrate Node Selection** (j)
18: Compute $CRR(j)$
19: **if** $u_{cand} \neq \emptyset$ **and** $(CRR(j) \leq CRR_{max}$ **or** $CRR(j-1) > CRR_{max})$ **then**
20: $M_{Nv,j} \leftarrow u_{cand}$
21: Update residual capacity for all links between u_{cand} and all nodes $u \in \{M_{Nv}, P'\}$
22: **if** $j = |N_v| - 1$ **then**
23: **if** $CRR(j) < CRR_{sol}(j)$ **then**
24: $j_{max} \leftarrow j$
25: $M_{Nv\_sol} \leftarrow M_{Nv}$
26: $CRR_{sol} \leftarrow CRR$
27: **end if**
28: **end if**
29: **else**
30: **if** $CRR(j-1) \leq CRR_{max}$ **and** $(j - 1 > j_{max}$ **or** $CRR(j-1) < CRR_{sol}(j-1))$ **then**
31: $j_{max} \leftarrow j - 1$
32: $M_{Nv\_sol} \leftarrow M_{Nv}$
33: $CRR_{sol} \leftarrow CRR$
34: **end if**
35: **break**: exit the for loop
36: **end if**
37: **end for**
38: **end if**
39: $C \leftarrow C \setminus u_{root}$
40: $R_N \leftarrow R_{N,rollback}, \quad R_L \leftarrow R_{L,rollback}$
41: **end while**
42: **return** M_{Nv_sol}

Algorithm 2 Candidate Substrate Node Preselection

Input: i global: $N_s, R_N, R_L, N_v, D_N, D_L, M_{Nv}$

Find for the mapping of virtual node i all nodes out of N_s with sufficient node capacity. Exclude all nodes definitively having not sufficient link capacity and nodes that have already been used for mapping.

1: $N_{cand} \leftarrow \emptyset$
2: $d_{from} \leftarrow 0, \quad d_{to} \leftarrow 0$
3: **for all** $j \in N_v$ **do**
4: **if** $M_{Nv,j} = \emptyset$ **then**
5: $d_{from} \leftarrow d_{from} + d^{ij}$
6: $d_{to} \leftarrow d_{to} + d^{ji}$
7: **end if**
8: **end for**
9: **for all** $u' \in N_s$ **do**
10: **if** $u' \notin M_{Nv}$ **then**
11: **if** $g^i \leq r'_u$ **then**
12: $r_{from} = \sum_{\{u,v\} \in L_s, u=u'} r_{uv}$
13: $r_{to} = \sum_{\{u,v\} \in L_s, v=u'} r_{uv}$
14: **if** $d_{from} \leq r_{from}$ **and** $d_{to} \leq r_{to}$ **then**
15: $N_{cand} \leftarrow N_{cand} \cup u$
16: **end if**
17: **end if**
18: **end if**
19: **end for**
20: **return** N_{cand}

Algorithm 3 Candidate Substrate Node Selection

Input: j global: $N_s, R_L, N_v, P', D_L, M_{Nv}, u_{root}$

Find for the mapping of virtual node j the substrate node out of N_s which requires minimum additional link embedding cost.

1: $u_{cand} \leftarrow \emptyset$
2: $S \leftarrow$ **Candidate Substrate Node Preselection** (j)
3: **for all** $u \in S$ **do**
4: **if** $dist_{u_{root},u} \leq \zeta$ **then**
5: $cost_u \leftarrow 0$
6: **for all** $i \in \{N_v, P'\}$ **do**
7: **if** $i \in P'$ **then**
8: $v \leftarrow i$
9: **else**
10: $v \leftarrow M_{Nv,i}$
11: **end if**
12: **if** $v \neq \emptyset$ **then**
13: **if** Shortest path with sufficient capacity exists between u and v **then**
14: $cost_u \leftarrow cost_u + dist_{u,v} \cdot d^{ij}$
15: $cost_u \leftarrow cost_u + dist_{v,u} \cdot d^{ji}$
16: **else**
17: $cost_u \leftarrow \infty$
18: **break**: exit the for loop
19: **end if**
20: **end if**
21: **end for**
22: **end if**
23: **end for**
24: **if** $min_{u \in S}(cost_u) \neq \infty$ **then**
25: $u_{cand} \leftarrow arg\ min_{u \in S}(cost_u)$
26: **end if**
27: **return** u_{cand}

If the last node of the VN request is reached and the corresponding mapping of this node is successful then the current mapping will replace any previous solutions with higher CRR value (i.e., lines 22–28). The *else* branch (i.e., lines 29–36) is reached after the VN embedding becomes either unprofitable or infeasible with virtual node j. In this case, the VN subset of iteration $j-1$ can be considered as as a solution. The current mapping is considered as a better solution only if it is associated with a larger VN subset or if it yields a lower CRR compared to the previous solution. The final VNE solution is returned in M_{sol_Nv}.

V. Evaluation

In this section, we evaluate the efficiency of the proposed VNE algorithm against a variant of our algorithm that embeds only full VN requests. We further shed light into VNE policy configuration by investigating the impact of diverse CRR threshold adjustments on revenue and VN request acceptance rate. In Section V-A, we present our simulation environment and in Section V-B we discuss our simulation results.

A. Evaluation Environment

We have implemented a C/C++ based simulation environment for VN embedding. Our implementation comprises

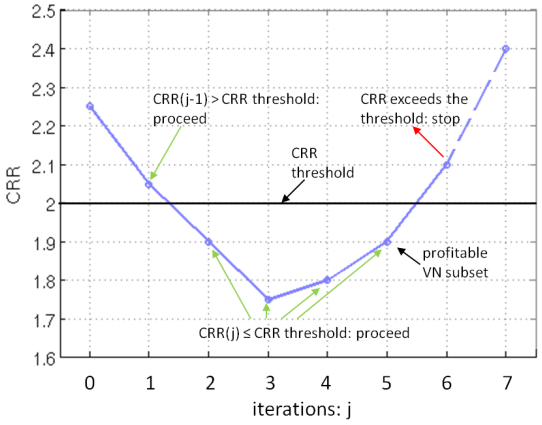


Fig. 5. Decisions for CRR threshold violation.

TABLE II
NOTATION FOR THE PSEUDOCODE

Symbol	Description
C	set of candidate substrate nodes for root node assignment
$cost$	set of virtual link embedding costs for all candidate substrate nodes
CRR	set of CRR values computed after the assignment of each virtual node
CRR_{max}	CRR threshold
CRR_{sol}	set of CRR values for the best solution for each VN subset
d_{from}, d_{to}	inbound / outbound bandwidth demand at a substrate node
$dist$	hop-count of shortest-paths between each pair of substrate nodes
$dist_{P,min}$	minimum distance (number of hops) to all peering nodes
$dist_{P,max}$	maximum allowable distance (number of hops) to all peering nodes
D_L	bandwidth demands for each pair of virtual nodes
D_N	set of CPU demands for each virtual node
L_s	set of substrate links
M_{Nv}	temporary VN mappings
M_{Nv_sol}	final VN mapping
N_{cand}	set of candidate substrate nodes
N_s	set of substrate nodes
N_v	set of virtual nodes
P	set of peering nodes
P'	set of peering nodes specified in a VN request
r_{from}, r_{to}	available inbound / outbound bandwidth at a substrate node
R_L	residual link capacity (bandwidth units)
$R_{L,rollback}$	link capacity stored for rolling back incomplete mappings
R_N	residual node capacity / CPU units
$R_{N,rollback}$	node capacity stored for rolling back incomplete mappings
S	set of candidate substrate nodes
u_{cand}	candidate for virtual node mapping
u_{root}	candidate for root node mapping
θ	distance tolerance to all peering nodes in relation to $dist_{P,min}$
ζ	maximum number of hops from the root node

a set of modules for VN request generation, VN request processing, VNE logging, and substrate network configuration and management. We conducted our VNE evaluation on a server with two Intel Xeon quad-core CPUs at 2.53 GHz and 12 GB of main memory.

Substrate network. We used *IGen* [7] to generate synthetic substrate network topologies for our simulations. We particularly ran our tests on a substrate network with 200 nodes and 400 links which are distributed based on the two-trees method [8]. We also designated 8 substrate nodes for peering with other substrate providers. Initially, all substrate network resources are unutilized. The residual capacity of substrate nodes and links is updated after the embedding of a VN request.

VN request. A VN request consists of the CPU requirements for each virtual node and the bandwidth demands between all pairs of virtual nodes, represented as a traffic matrix (TM). The TM further contains the bandwidth requirements between each virtual node and each peering node (i.e., 3 peering nodes are randomly assigned among the 8 available nodes designated for peering). As such, we take into account the embedding cost of virtual links spanning multiple substrate providers. The number of virtual nodes per VN request is randomly sampled from a uniform distribution, between 10 and 30. In each simulation run, we generate and process a sequence of 1000 non-expiring VN requests, which gradually utilize most of the substrate network resources and allow to assess VNE efficiency under various network utilization levels. Each of the following evaluation results is based on 50 simulation runs.

B. Evaluation Results

Initially, we discuss the efficiency of our VNE algorithm. Fig. 6 illustrates the CRR versus the generated revenue with three CRR threshold adjustments (i.e., 1.8, 2.0, and 2.2). This scatter plot validates the operation of our algorithm, as in each case the CRR does not exceed the predefined threshold. We observe a correlation between the CRR and the generated revenue, i.e., embedding larger VNs incurs a penalty in terms of resource efficiency. More precisely, setting the CRR to 1.8 allows the embedding of a VN subset with revenue up to 180. Adjusting the CRR to 2.0 or 2.2 permits the embedding of VNs with larger revenue (i.e., up to 220). Furthermore, Fig. 7 depicts the proportion of the embedded VN size over the VN request size with these CRR threshold adjustments. The results are classified into 5 different groups of VN request sizes. In many cases, small VN requests (i.e., 10-13 nodes) are fully embedded, especially for a CRR threshold of 1.8. For larger VN requests, only a subset is usually being embedded, and the relative subset size decreases as the VN request size becomes larger. As shown in the figure, lower CRR threshold adjustments result in embedding smaller VN subsets.

Fig. 8 illustrates the generated revenue with a wide range of adjustments for the maximum distance of the root node from the peering nodes (i.e., parameter θ). As shown in the plot, adjusting θ to a value greater than 10% does not lead to notable revenue gains, while it increases the solver runtime. As such, we use 10% as the default value for θ. We further identified based on simulations that the adjustment $\zeta = 5$ generates the highest revenue.

Next, we evaluate the efficiency of embedding subsets of VN requests. Fig. 9 depicts the cumulative revenue generated by embedding full requests and the most profitable subset of a VN request according to the CRR threshold adjustment. Our VNE algorithm generates much higher revenue compared to full VN embedding, since it embeds the VN request subsets that exhibits higher efficiency. In contrast, full VN embedding results in low revenue due to inefficient resource utilization, as shown in Fig. 10. According to this plot, for a given level of resource utilization, subset VN embedding generates more revenue, although it may require to process a larger number of requests compared to full VN embedding. Among the various CRR threshold adjustments, 1.8 achieves higher revenue, since it exhibits less tolerance to resource inefficiencies.

Furthermore, Fig. 11 shows the VN request acceptance rate with full and subset VN embedding. The acceptance rate of full VN requests drops quickly, due to the inefficient resource utilization. Depending on the CRR threshold adjustment, our algorithm rejects the VN requests that are not profitable, even if the substrate network is not saturated. However, in the long run, embedding VN subsets leads to a higher acceptance rate.

We also present the revenue generation rate in Fig. 12. Full VN embedding initially generates revenue at higher rate, but

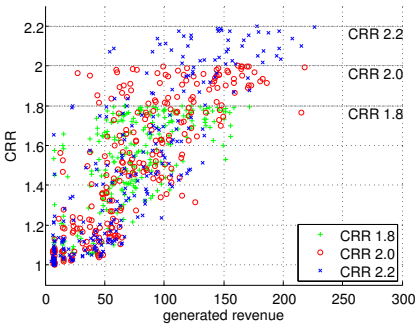


Fig. 6. CRR vs. generated revenue.

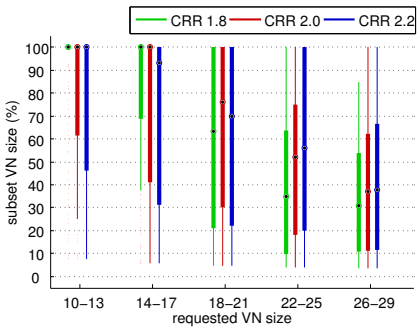


Fig. 7. Relative VN subset size classified into 5 groups of VN request sizes (number of virtual nodes).

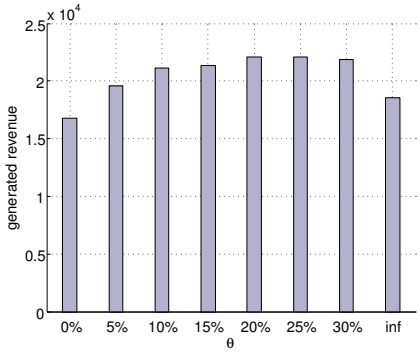


Fig. 8. Impact of θ on generated revenue.

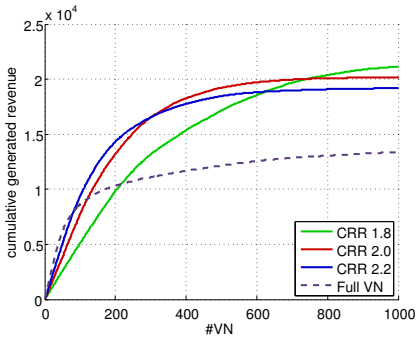


Fig. 9. Cumulative generated revenue.

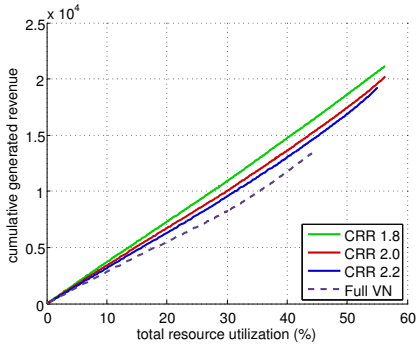


Fig. 10. Generated revenue vs. resource utilization.

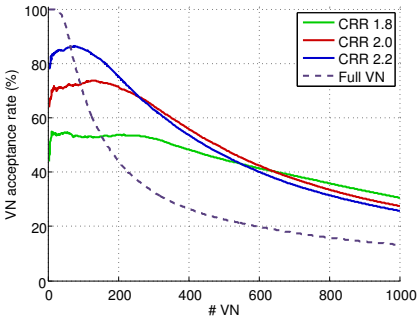


Fig. 11. VN request acceptance rate.

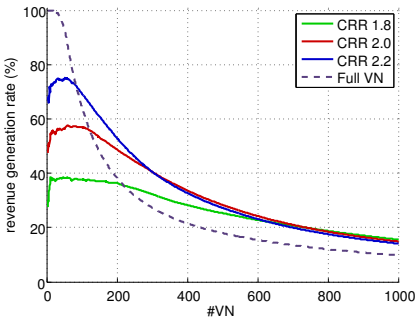


Fig. 12. Revenue generation rate with 200 substrate nodes.

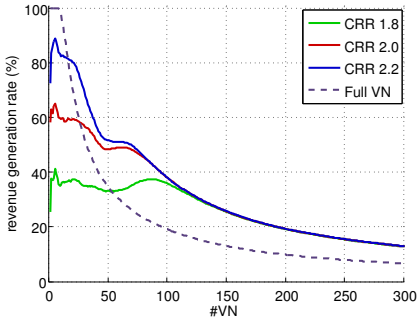


Fig. 13. Revenue generation rate with 50 substrate nodes.

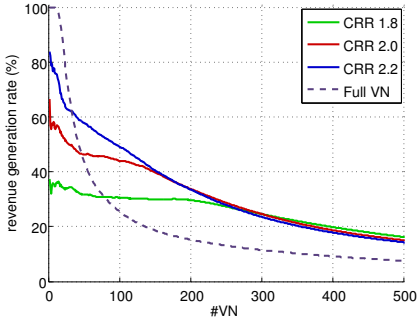


Fig. 14. Revenue generation rate with 100 substrate nodes.

after the first 150 VN requests VN subset embedding with high CRR threshold values (i.e., 2.0 and 2.2) yields a higher revenue generation rate. Comparing among the CRR threshold values, low threshold adjustments restrict the revenue generation rates when the substrate network is underutilized, but achieve slightly higher rates for medium and high utilization levels and eventually generate more revenue for the provider in the long run (Fig. 9). Essentially, a high CRR threshold value results in a greedier behavior, generating revenue faster which may be suited to providers that do not anticipate a large number of VN requests. On the other hand, a low CRR threshold value is deemed more beneficial for smaller substrate networks, in which the computing resources can be saturated with a smaller number of VN requests.

In this respect, Figs. 13 and 14 illustrate the revenue generation rate with a substrate network of 50 and 100 nodes, respectively. In the smaller substrate network (i.e., 50 nodes), the revenue generation rates with the different CRR thresholds converge after 100 VN requests, i.e., much sooner compared to the 200-node substrate network. Although in our simulations we used static CRR threshold values, providers are envisaged to apply policies that require the dynamic adjustment of the CRR threshold depending on the network utilization and resource demand. The insights gained from our simulation results can be useful for the specification of such VNE policies.

VI. Related Work

We briefly discuss related work on VNE with single and multiple substrate providers.

Single-Provider VNE. There is a large body of work on embedding VN requests onto a substrate network [3], [5], [9], [13], [15]. Existing VNE solutions mainly rely on heuristic algorithms (e.g., [15], [13]) or linear programs (e.g., [3]), while attention has been also given to path splitting [13] and VNE distribution [5]. These VNE techniques aim at optimizing the mapping of VN topologies and always embed full VN requests when this is feasible. As opposed to these techniques that ignore the policies of substrate providers, we take a different approach by tailoring VNE to the provider's policy. In particular, our VNE algorithm restricts the solution space according to the policy.

Multiple-Provider VNE. VNE across multiple substrate providers is more challenging, due to limited information disclosure from the providers [4]. The VNE architectures in [6], [4] rely on a centralized coordinator for VN request partitioning among the substrate providers. In contrast, PolyVine [2] carries out VN embedding in a distributed manner, where each provider embeds the subset of the VN request which yields higher profit and subsequently relays the remaining part to one of its peers. V-Mart [14] uses a two-stage Vickrey auction model to enable resource trading between clients and providers for VN embedding. Both PolyVine and V-Mart can benefit from our work. In PolyVine, a substrate provider can use our VNE algorithm to identify and embed the most profitable subset of a VN request. Similarly, in auction-based VNE environments such as V-Mart, our algorithm can assist a provider in adjusting his bid for the embedding of VN requests.

VII. Conclusions and Future Work

In this paper, we tackled the VNE problem from a different angle, tailoring VNE to the InP's policy. In this respect, we developed a new policy-compliant VNE algorithm that seeks to maximize the revenue of an InP without violating his policy. We express this policy as an upper bound to CRR. An InP can adjust the CRR threshold performing a balancing act between short-term revenue gains (i.e., using a high threshold value) and more revenue in the long term (i.e., using a low threshold value), taking into account the network utilization and the anticipated resource demand. Since virtualized network infrastructures constitute highly dynamic environments, the CRR threshold can be dynamically adjusted. For example, an InP can perform a downward adjustment to the threshold value when network utilization increases and resource efficiency becomes critical. Certainly, VNE policies are not limited to CRR upper bounds, but they may represent various restrictions that an InP wants to apply depending on the substrate network size, level of utilization, infrastructure (e.g., OpenFlow-enabled network devices [10]), or particular topology abstractions exposed to tenants.

Our simulation results show that embedding the most profitable subsets of VN requests generates much higher revenue compared to full VN request embedding. Besides the revenue gains, our algorithm can be used for bidding decisions in auction-based VNE environments. In future work, we plan to integrate our algorithm into distributed VNE architectures (e.g., PolyVine) and auction-based environments, and investigate potential gains that our algorithm could bring.

VIII. Acknowledgments

This work was partially supported by the L3S ProCloud project. We are thankful to Georgios Katsenos for discussions on earlier versions of this paper and his reflections on VNE policies.

References

[1] T. Anderson, L. Peterson, S. Shenker, and J. Turner, Overcoming the Internet Impasse through Virtualization, IEEE Computer, 38(4), 2005, pp. 34–41.

[2] M. Chowdhury, F. Samuel, and R. Boutaba, PolyViNE: Policy-based Virtual Network Embedding Across Multiple Domains, Proc. ACM SIGCOMM VISA, New Delhi, India, September 2010.

[3] N. Chowdhury, M. Rahman, and R. Boutaba, Virtual Network Embedding with Coordinated Node and Link Mapping, Proc. IEEE Infocom 2009, Rio de Janeiro, Brazil, April 2009.

[4] D. Dietrich, A. Rizk, and P. Papadimitriou, Multi-Domain Virtual Network Embedding with Limited Information Disclosure, Proc. IFIP Networking, New York, USA, May 2013.

[5] I. Houidi, W. Louati, and D. Zeghlache, A Distributed Virtual Network Mapping Algorithm, Proc. IEEE ICC 2008, Beijing, China, May 2008.

[6] I. Houidi, W. Louati, W. Bean-Ameur, and D. Zeghlache, Virtual Network Provisioning Across Multiple Substrate Networks, Computer Networks, 55(4), March 2011.

[7] IGen Network Topology Generator, http://informatique.umons.ac.be/networks/igen.

[8] B. Quoitin, V. Van den Schrieck, P. Franois, O. Bonaventure, IGen: Generation of Router-level Internet Topologies through Network Design Heuristics, Proc. 21st International Teletraffic Congress, Paris, September 2009.

[9] J. Lischka and H. Karl, A Virtual Network Mapping Algorithm based on Subgraph Isomorphism Detection, Proc. ACM SIGCOMM VISA, Barcelona, Spain, August 2009.

[10] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, OpenFlow: Enabling Innovation in Campus Networks, ACM SIGCOMM CCR, 38(2), 2008.

[11] G. Schaffrath, C. Werle, P. Papadimitriou, A. Feldmann, R. Bless, A. Greenhalgh, A. Wundsam, M. Kind, O. Maennel, and L. Mathy, Network Virtualization Architecture: Proposal and Initial Prototype, Proc. ACM SIGCOMM VISA, Barcelona, Spain, August 2009.

[12] C. Wang and T. Wolf, Virtual Network Mapping with Traffic Matrices, Proc. ACM/IEEE ANCS 2011, New York, USA, October 2011.

[13] M. Yu, Y. Yi, J. Rexford, and M. Chiang: Rethinking Virtual Network Embedding: Substrate Support for Path Splitting and Migration, ACM SIGCOMM Computer Communications Review, 38(2), April 2008, pp. 17–29.

[14] F. Zaheer, J. Xiao, and R. Boutaba, Multi-provider Service Negotiation and Contracting in Network Virtualization, Proc. IFIP NOMS 2010, Osaka, Japan, April 2010.

[15] Y. Zhu and M. Ammar: Algorithms for Assigning Substrate Network Resources to Virtual Network Components, Proc. IEEE Infocom 2006, Barcelona, Spain, April 2006.

[16] Y. Zu, R. Zhang-Shen, S.Rangarajan, and J. Rexford, Cabernet: Connectivity Architecture for Better Network Services, Proc. ACM ReArch '08, Madrid, Spain, December 2008.