Progressive Virtual Topology Embedding in OpenFlow Networks

Roberto Riggio, Francesco De Pellegrini, Elio Salvadori, Matteo Gerola, and Roberto Doriguzzi Corin
CREATE-NET, Via Alla Cascata 56/C, 38123 Trento, Italy
Emails: {rriggio,fdepellegrini,esalvadori,mgerola,rdoriguzzi}@create-net.org

***Abstract*—Future internet would provide a flexible and simpler architectural design by combining novel low level clean–slate techniques such as OpenFlow with high level design principles such as network virtualization. However, previous research outlined that at the core of network virtualization stands a new set of challenges for network resources allocation. In this work we focus on one such challenge, namely the problem of virtual topology embedding. In this context users need to leverage the infrastructure substrate by accomodating logical topologies with high degree of flexibility. The network provider, on the other hand, aims at maximizing its revenue in term of size and number of topologies accepted while minimizing costs accounting for the substrate network resources used. To this aim, we present *VT-Planner* a novel virtual network embedding technique with reduced computational cost and very efficient over substrate topologies encountered in practice. Extensive numerical simulations are provided comparing this technique with state–of–the–art solutions: our results show that *VT-Planner* is able to achieve a good balance in terms of complexity and performance.**

***Index Terms*—Resource allocation, Virtual Topology Embedding, OpenFlow, Algorithms**

I. Introduction

Network virtualization is expected to play an important role in the networking domain by renewing the way resources sharing is performed throughout networking and computational facilities. In particular, network virtualization allows multiple heterogeneous virtual networks (VN) to coexists *at the same time* over a shared physical infrastructure, namely, the *substrate* network. This approach is set to favor both research and operative activities. In fact, researchers will design new protocols and algorithms while developing and testing them over the production network, but, without affecting the users who are not involved/interested in the experimentation. Practitioners, conversely, will take advantage of new empowered virtualization items; for instance, using network virtualization they will be able shift their business paradigm and start offering to their customers ad hoc usage of their infrastructure, i.e., what is referred to as Infrastructure as a Service (IaaS).

However, in order to achieve these goals, the topology of the VN must be completely decoupled from the underlying physical topology of the substrate network. It should allow experimenters to construct the VN according to the needs of his/her research agenda, and the infrastructure provider to tune how the details of the physical network are exposed to its customers. The latter requirement is particularly important in order to accommodate users who want direct control of the network they run and users that instead are interested in providing added–value services on top of their VN. Finding an efficient mapping between VNs requested by the end–users and the substrate network resources is the first step in achieving such level of flexibility.

The *VN embedding problem* refers precisely to the mapping of a VN requested by the user into the substrate network. The VN consists of a topology where links and nodes are both weighted to express precise constraints on both the computational resources that should be made available at each virtual node and the performance of the virtual links between such nodes to be granted on a substrate network. Performance figures can be represented by any isomorphic performance metric, e.g. delay, throughput, jitter. The problem is known to be NP-hard [1] and consequently there exists a significant amount of literature on designing heuristics to find suitable mappings between virtual nodes and virtual links on the substrate network. In particular, we will focus on the case of unsplittable paths [2]: in fact, our solution is meant to permit network virtualization over OpenFlow enabled substrate networks which do not encompass multipath routing. OpenFlow [3] is an emerging networking technology that allows virtualization and control of the network environment through secure and standardized interfaces.

Furthermore, our target reference substrate network is well represented by the infrastructure deployed in the OFELIA project [4], a facility which exploits Software Defined Networking (SDN) concepts and is based on OpenFlow. The testbed facility, i.e., the substrate network, is composed of several sub–networks, called *islands*; each island is composed of a variable number of switches and links including a virtualized server infrastructure. Islands are geographically separated and are interconnected using the public Internet. Moreover, within each island only a subset of the switches is actually connected to the virtualized server infrastructure.

Users of the infrastructure can leverage the substrate topology by issuing requests for arbitrary VNs, together with some computing resources in the form of virtual machine instances, i.e., virtual hosts for simplicity's sake. In this context, we are compelled to ignore relocation techniques: once VNs are allocated, mappings between VNs currently supported can not be recomputed and re-issued in order not to impair ongoing running experiments. For more information about the network virtualization architecture and prototypical implementation the reader can refer to [5]

978-3-901882-50-0

This paper has two main contributions. First, we compare the performances of state–of–the–art VN embedding heuristics using both random *and* regular topologies. In particular we are interested in investigating the challenges in embedding canonical topologies such as clique and star topologies onto the substrate networks. Besides random substrate networks, meant to represent general topologies, we are interested also in specific topologies such those typically found in data centers, e.g. fat–trees. Second, starting from the aforementioned performance comparison, we propose an algorithm, named *VT-Planner*, solving the VN embedding problem.

The rest of the paper is structured as follows. Related work are discussed in Sec. II. In Sec. III we formalize the network model and the virtual topology embedding problem. Section IV introduces the heuristics exploited by the *VT-Planner* in order to perform the virtual topology embedding. Section V introduces the evaluation methodology and the performance metrics. The results of our numerical simulations campaign are reported in Sec. VI. Finally, Sec. VII draws the conclusions and future research directions.

II. Related work

The problem of VN embedding has been explored in a few reference works in literature. The work [6] describes the problem and characterizes the core tradeoff for such resource allocation problem. In particular, the authors introduce the notion of *node stress* and of *link stress*. Those metrics describe the concurrent effect of bandwidth requests and node capacity requests. Assigning virtual nodes to substrate nodes, affects the node stress, but also determines the substrate links that are traversed by virtual links, which in turns affects the link stress. The control parameter that we use to balance edge demands and node loads resembles notion of stress in [6].

Chowdhury et al. [7] formulate the network embedding algorithms as a mixed-integer program that jointly address the nodes and the links mapping phases. Simulations carried out using a large number of requests (i.e. virtual network topologies) show that the proposed algorithms can effectively improve the acceptance rate while minimizing a cost function which takes into account the load generated on the underlying network. The model that we follow in the rest of this paper adopts the same formalism and the notation that is developed in [7]. The work developed in [8] extends the virtual topology embedding problem to the case where the substrate to be covered spans multiple domains, and no centralized resources broker is available; the focus is on protocol operations.

The paper [9] provides a primal-dual algorithm that is able to optimize the VT embedding problem under convexity assumptions and in the case when only the virtual link capacity is accounted for. The authors of [10] adopt a graph isomorphism approach to perform VT embedding based on a revenue-cost mapping similar to what proposed in [7]; with such an approach, both nodes and links are embedded simultaneously. Another technique is proposed in the paper [2] where ant-colony methaheuristics are adopted in order to search the solution space. The paper [11] demonstrate the potential of multiple path splitting as an effective mean to allocate substrate link bandwidth to virtual links; such an approach is also encompassed in the solution proposed by [7]. As mentioned before, we intend to consider only unsplittable paths because of technological constraints.

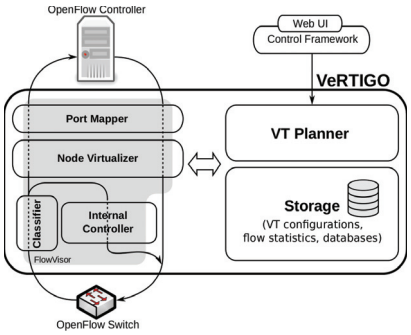
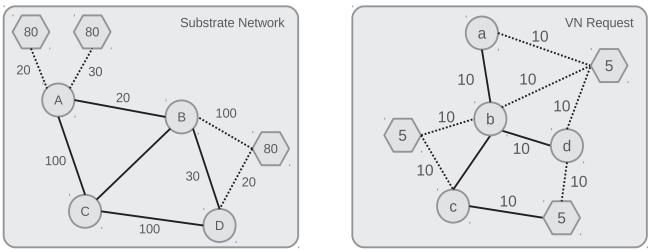


Fig. 1: Main building blocks of VeRTIGO.



(a) **(b)**

Fig. 2: Substrate network (a) and VN Request (b). Round circles denote switches whereas hexagons denote servers. Dashed lines denote links between servers and switches, while solid lines denote links between switches.

III. Problem definition

In the VN embedding problem the input data are the VN requests performed by the users, whereas the substrate network provides the physical constraints in terms of bandwidth and capacity. From an architectural standpoint the VN embedding algorithm is the engine of the resource allocation component in a virtualized networking infrastructure. In the specific case of the OFELIA facility, the network virtualization enabler is named VeRTIGO and its high level architecture is sketched in Fig. 1. VeRTIGO is implemented around the latest release of FlowVisor, the OpenFlow native network slicing layer, and adds a set of additional modules aiming at providing enhanced flexibility in the setup of network slices. More precisely, VeRTIGO allows to expose to the OpenFlow controller arbitrary network topologies. For a detailed description of the VeRTIGO architecture and implementation the reader can refer to [5].

Before introducing the proposed solution we need to detail specific notation for the substrate and the virtual topologies.

A. Physical Network model

Let $G^S = (N^S, E^S)$ be an undirected graph modeling the physical network, where $N = N_1^S \cup N_2^S$ is the set of $n_1 = |N_1^S|$ switches and $n_2 = |N_2^S|$ servers and E^S is

the set of edges or links. An edge $e^S_{nm} \in E^S$ if and only if a point–to–point connection exists between $n \in N^S$ and $m \in N^S$. With respect to the physical network, links are actual wiring media, e.g., an Ethernet cable interconnecting the two switches[1]. Servers can only connect to switches, but not to other servers; to this respect, we denote $E^S_1 \subseteq N^S_1 \times N^S_1$ the set of links between switches and with $E^S_2 \subseteq N^S_1 \times N^S_2$ the set of links between switches and servers, where $E^S = E^S_1 \cup E^S_2$.

A sample substrate network is sketched in Fig. 2a: as seen there, the physical network is composed by a set of $n_1 = 4$ switches interconnected together, plus a set of $n_2 = 3$ servers connected to one or more switches. Round circles denote switches whereas hexagons denote servers or computational nodes. In the picture dashed lines denote links between servers and switches, while solid lines denote links between switches.

A weight $c(n^S)$ assigned to each node $n \in N^S_2 : c(n^S) \in \mathbb{N}^+$ represents the amount of computational resources available on that node; for the sake of simplicity we assume that computational resources can be expressed as an integer number. Another weight $b(e^S)$ assigned to each link $e^S \in E^S_1 : b(e^S) \in \mathbb{N}^+$ represents the capacity of the link connecting two switches. Observe that it will be possible that a server is connected to two or more different switches, i.e., we account for servers multi-homing. Also, we assume that servers will implement onboard traffic shaping functionality, such in a way that the capacity of links towards switches is never exceeded. Finally, let P^S be the set of all substrate paths and $P^S(s,t)$ the set of all substrate paths between nodes s and t.

The resources required to embed a VN request G^V onto a substrate network G^S can be quantified using the concepts of substrate node S^N and link S^E stress defined as follows:

$$S_N(n^S) = \sum_{n^V \to n^S} c(n^V)$$
$$S_E(e^S) = \sum_{e^V \to e^S} b(n^V) \qquad (1)$$

We define residual node capacity as the available capacity of the substrate node $n^S \in N^S$

$$R_N(n^S) = c(n^S) - S_N(n^S)$$

The residual capacity of a substrate link is defined as the total amount of bandwidth available on the substrate link $e^S \in E^S$.

$$R_E(e^S) = b(e^S) - S_E(e^S)$$

Finally, we can define the available bandwidth of a substrate path $P \in P^S$ as the residual capacity of the bottleneck link

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

B. VN Requests

Users are allowed to request instances of arbitrary topologies $G^V = (N^V, E^V)$ on top of the physical network. Again, N^V_1 denotes the set of switches and N^V_2 the set of computational nodes G^V has two types of links: $E^V_1 \subseteq N^V_1 \times N^V_1$ is the set of links between switches, and $E^V_2 \subseteq N^V_1 \times N^V_2$ is set of links between switches and servers, where $E^V = E^V_1 \cup E^V_2$. A sample slice composed by 3 virtual machines, 4 switches, and several (virtual) links is sketched in Fig. 2b.

[1]In this work we consider undirected links for simplicity

C. Decision

In the event of a VN Request the *VT-Planner* has to decide whether it can be accepted or if it must be refused. If a request is accepted then the *VT-Planner* is in charge of mapping the request onto the substrate network, i.e., network resources must be allocated on both the substrate nodes and the substrate links. The embedding of a VN request G^V onto the substrate network is subject to the following constraints:

1) Node assignment. Each node in the VN request is mapped to a *different* substrate node with sufficient computational capacity. Notice that with node we refer to both switches and virtual hosts. The mapping function $M_N : N^V \to N^S$ from virtual nodes to substrate nodes is such that $\forall\ n^V, n^S \in N^V$,

$$M_N(n^V) \in N^S$$
$$M_N(m^V) = M_N(n^V), \quad \text{iff } m^V = n^V \qquad (2)$$
$$\text{subject to:} \quad c(n^V) \leq R_N(M_N(n^V)).$$

2) Link assignment. Each virtual link is mapped to a *single* substrate path between the substrate nodes on top of which the two endpoints of the virtual link have been mapped. Only substrate paths with sufficient capacity on their bottleneck links are considered. Link assignment is defined by a mapping function $M_E : E^V \to P^S$ from virtual links to substrate paths such that $\forall\ e^V = (m^V, n^V) \in E^V$,

$$M_E(m^V, n^V) \subset P^S(M_N(m^V), M_N(n^V))$$
$$\text{subject to:} \quad \sum_{P \in M_E(e^V)} R_E(P) \geq b(e^V).$$

Notice that the single flow constraints derives from the fact that due to the OpenFlow protocol definition, it is not possible to split a flow across multiple paths.

IV. *VT-Planner*

The objective of our algorithm is to embed multiple VN requests consisting of a set of nodes and links each with its own constraints on top of a given substrate network. We assume that substrate network topologies can be either tree–like topologies, e.g. *fat–tree*, commonly found in datacenters, and *random* topologies which aim at modeling core networks. VN requests coming from users can consist of *random* topologies as well as more regular topologies such as *clique* and *star*.

VT-Planner is a recursive algorithm implementing a Breadth–first traversal of both the substrate network and the VN request. At each step the algorithm tries to map a virtual node to the substrate node that minimizes a virtual edge stress metric. Such metric has been designed in such a way to control how VN request are embedded onto the substrate network. More specifically the metric aims at balancing the load on both the computational nodes and the substrate network resources. Finally, the visit on both the substrate network graph and on the VN request graph begins from the two highest capacity

Algorithm 1 *VT-Planner*

```
1: procedure Embed_Graph(G^V, G^S)
2:     n^V ← Select_Node(G^V)                ▷ pick VN Request node
3:     n^S ← Select_Node(G^S)                ▷ pick substrate node
4:     M(n^V) ← n^S                          ▷ embed node
5:     φ(n^S) ← 1                            ▷ mark node as used
6:     if not Embed_Node(n^V) then
7:         Reject(G^V)
8:     end if
9: end procedure
```

Algorithm 2 Node embedding (Recursive)

```
1: procedure Embed_Node(n^V)
2:     ψ(n^V) ← 1                            ▷ mark node as visited
3:     for all {m^V ∈ neighbors(n^V)} do
4:         if ψ(m^V) == 0 then               ▷ Virtual node not visited
5:             Θ^S = G^S \ {n^S ∈ N^S | φ(n^S) == 0}
6:             if Θ^S == ∅ then              ▷ No substrate node available
7:                 return False
8:             end if
9:             m^S = argmin_{m^S ∈ Θ^S} [W^S(e^V, m^S)]
10:            if m^S == ∅ then
11:                return False
12:            end if
13:            M(m^V) ← m^S                  ▷ embed node
14:            φ(n^S) ← 1                    ▷ mark node as used
15:        end if
16:        Allocate path between M(m^V) and M(n^V)
17:    end for
18:    for all {m^V ∈ neighbors(n^V) | ψ(m^V) == 0} do
19:        if not Embed_Node(m^V, M(m^V)) then
20:            return False
21:        end if
22:    end for
23:    return True
24: end procedure
```

nodes. Notice that a node capacity is computed as the sum of the weights of all links centered on that node. For the ease of description we introduce $\phi : N^S \rightarrow \{0,1\}$ and $\psi : N^V \rightarrow \{0,1\}$ to track, respectively, the substrate nodes and the virtual nodes that have been visited.

The algorithm (see Alg. 1) begins by picking two nodes $n^V \in N^V$ and $n^S \in N^S$. Then, it maps $M(n^V) \leftarrow n^S$ and starts a breadth–first traversal (see Alg. 2) on the virtual topology starting at virtual node n^V: the visit corresponds to exploring a tree rooted on the substrate node n^S. At each step the visited node is mapped to the substrate node with the minimum virtual edge stress. We define the virtual edge stress $W^S : E^V \times N^S \rightarrow \mathbb{R}$ between a virtual edge $e^V = (n^V, m^V) \in E^V$ and a substrate node $m^S \in N^S$ as follows:

$$W^S(e^V, m^S) = (1-\alpha) \min_{e^S} \big[R_E(e^S) - b(e^V)\big] + \alpha\big[R_N(m^S) - c(m^V)\big] \quad (3)$$

where $e^S \in P^S(n^S, m^S)$. This distance is the convex combination of the residual capacity of the substrate node m^S after the mapping the virtual node m^V and the residual bandwidth on the bottleneck substrate link e^S after mapping the virtual link e^V. The parameter $0 \leq \alpha \leq 1$ can be used to give priority to the residual computational capacity or to the residual bandwidth. For example, if $\alpha = 1$ the algorithm will embed the virtual node m^V to the substrate node with the highest residual computational capacity, instead if $\alpha = 0$ the algorithm will try to embed the virtual node m^V to the substrate node whose path from $M(n^V)$ has the smallest residual bottleneck link capacity. In our experimental setup we used the value $\alpha = 1/2$ in order to weight equally between computational capacity and bandwidth.

After mapping the node $M(m^V) \leftarrow m^S$ the algorithm maps the virtual edge e^V to the shortest path between $M(n^V) = n^S$ and $M(m^V) = m^S$. The procedure stops when all the virtual nodes have been visited or if the substrate network cannot accommodate the VN Request. The latter case can happen if there are not enough nodes in the substrate topology ($\Theta^S == \emptyset$) or if either the computational capacity or the link bandwidth has been exhausted.

We observe that the *VT-Planner* has complexity $O((N^S)^2 \log N^S)$ in the number of substrate nodes, since it visits all N^V nodes of the input VN, which are N^S in the worst case, and for each node it then builds a spanning tree rooted at visited substrate node, at a cost $O(N^S \log N^S)$; in the case of a bounded size N^V for the input VT, the complexity indeed becomes $O(N^S \log N^S)$.

TABLE I: Simulation scenarios.

Substrate	VN Request	Nodes (Substrate)	Nodes (VN Request)
Random	Random	25	[2, 5]
Fat–tree	Clique	36	[2, 10]
Fat–tree	Star	36	[2, 6]

V. Performance Evaluation

In this section we shall first describe the simulation environment and then the performance metrics. The goal of this study is to compare the *relative* performance of different node and link mapping strategies using synthetic random topologies *and* canonical topologies.

A. Simulation Environment

Simulations are carried out in a discrete event simulator implemented in Matlab® . In our simulations we assume that VN request arrive according to a Poisson process and the various algorithms are evaluated for increasing arrival rates, staring with 4 VN requests and up to 8 VN requests every 100 time units. Each VN request has an exponentially distributed lifetime with an average $\mu = 1000$ time units after which the resources occupied on the substrate network are freed.

The substrate network's size is kept constant at 25 nodes for the random topologies and at 36 nodes for the fat–tree topologies, while the number of nodes in each VN request's is uniformly distributed between 2 and 5. The computational and

the bandwidth resources on the substrate nodes are uniformly distributed between 50 and 100, while the computational and bandwidth requirements for VN requests are uniformly distributed between, respectively, 0 and 20, and 0 and 50. A sample of each class of topology used in this study is reported in Fig 3. Nodes in the random topologies are deployed over a 25×25 grid. Each pair of node (both substrate and virtual) are connected with probability 0.5. Table I summarizes the scenarios and the simulations parameters used in this study. Notice that for the star–shaped topologies a single node acts as switch (no computational capacity) while the rest of the nodes are acting as hosts (computational capacity > 0) while for the clique–shaped topologies each switch is connected to an host.

B. Evaluation Metrics

The metrics used in this study are standard ones adopted in several other related work (see, e.g., [7], [11], [6]).

1) *Acceptance ratio*. Measures the percentage of VN Request accepted by an algorithm.
2) *Generated Revenue* and *Provisioning Cost*. Denoting with $f_{e^S}^{e^V}$ the total amount of bandwidth allocated on the substrate link e^S for the virtual link e^V, we define the revenue $\mathcal{R}$ and the cost $\mathcal{C}$ of a VN request as follows:

$$\mathcal{R}(G^V) = \sum_{e^V \in E^V} b(e^V) + \sum_{n^V \in N^V} c(n^V)$$

/home/rriggio

$$\mathcal{C}(G^V) = \sum_{e \in E^V} \sum_{e^S \in E^S} f_{e^S}^{e^V} + \sum_{n^V \in N^V} c(n^V)$$

3) *Average node and link utilization*. The average node and link utilization of the substrate network computed as the averages of, respectively, the node stress and the link stress (1).

VI. Results

In this section we shall report on the results of our numerical simulation study. We compared 4 different embedding strategies: *Greedy*, *D-Vine* and *R-Vine* [7], and *VT-Planner*. ViNe algorithms are considered in their Shortest Path (SP) version and with no constraint on the placement of virtual nodes [7]; indeed, the OpenFlow protocol which we are targeting as our network virtualization enabler is not able to split flows across multiple paths. Finally, notice that our comparison is limited to the ViNe algorithms due to both space constraints and due to the fact that they proved [7] to be significantly more efficient than the algorithms presented in [6], [11].

The *Greedy* algorithm is a reference simple solution that works as follows: it iteratively maps nodes in the VN request to a random substrate with enough residual computational capacity and substrate nodes are then interconnected via their shortest path. The Greedy algorithm has same complexity as *VT-Planner*. Results are reported for arrival rates increasing from 4 up to 8 new VN requests every 100 time units.

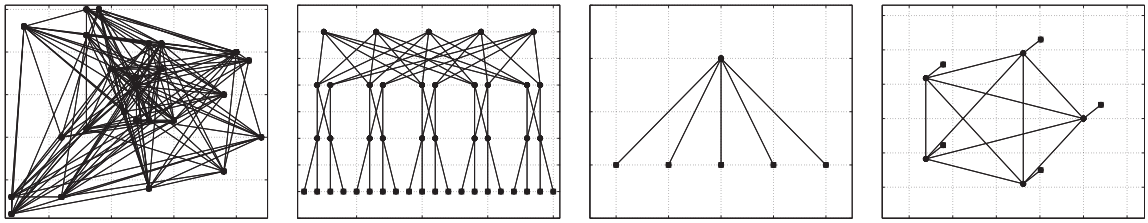
First, we observe that all algorithms saturate both nodes and links capacities in the case of random substrate network topologies, i.e., node and link utilization for all algorithms ≈ 1. However, in the case of fat–tree shaped substrate networks they tend to saturate links capacities before saturating nodes capacity: this result is expected due to the reduced bandwidth available on fat–tree topologies compared to the random topologies used in this study.

Figure 4 reports on the results for the Random on Random scenario, i.e, both the VN Requests and the substrate network are random topologies. As seen there, the relative performance of each algorithm does not depend on the load. In particular, the *VT-Planner* algorithm is capable of accepting the highest fraction of VN requests while at the same time delivering the highest embedding revenue at the lowest embedding cost. Moreover, increasing the load does not seem to impact the average embedding cost that remains constant while it does result in a decrease in the average revenue. We ascribe this behavior to the fact that, as the load increases, only small topologies, whose revenue is lower, can be successfully mapped onto the substrate network while large topologies, whose revenue is proportionally higher, must be rejected due to lack of available resources on the substrate network.

Figure 5 and Fig. 6 describe the results for, respectively, the Clique on Fat-Tree and the Star on Fat–Tree scenarios. As it can be seen the *VT-Planner* is again capable of accepting the highest fraction of VN requests consisting of star and clique shaped topologies. However, only in the case of star topologies the *VT-Planner* delivers the highest average embedding revenue at the lowest average embedding cost, while in the case of clique topologies the *VT-Planner* does show the lowest embedding cost but at the price of a lower embedding revenue compared to all the other algorithms. This behavior results in the fact that *VT-Planner* tends to accept small topologies characterized by a low revenue rejecting larger VN requests (this effect can be tuned by reducing by increasing α; optimization of α is part of future work.). Finally, we observe that the significantly lower performances delivered by the ViNE algorithms is due to the fact that such family of algorithms rely on a geographical vicinity localization of the virtual node w.r.t. the substrate nodes. As mentioned before, such localization assumption is not valid anymore in our context: performance gain of *VT-Planner* is larger in the case of highly structured topologies such as fat–trees and lower in the case when the substrate is random. In the latter case in fact, the impact of node placement is less severe, due to the lower diameter of such topologies, i.e., bad placement of nodes is compensated by the higher number of disjoint paths.

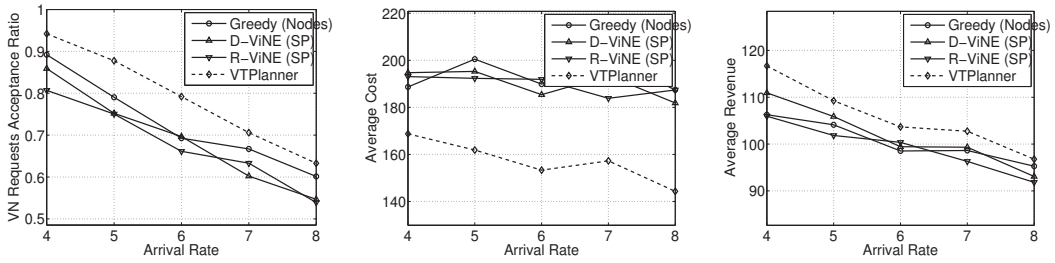
VII. Conclusions

In this paper we presented the *VT-Planner* algorithm, solving the joint nodes and links mapping in the VN embedding problem. *VT-Planner* is a greedy algorithm implementing a joint Breadth–first traversal of both the substrate network and the VN request. Being the visit of the substrate based on spanning trees, It has low requirements in terms of computational



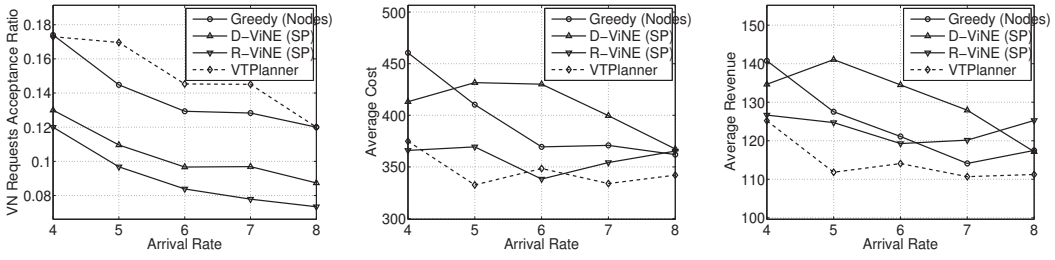
(a) Random (25 nodes). **(b)** Fat–tree (45 nodes). **(c)** Star (6 nodes). **(d)** Clique (10 nodes).

Fig. 3: Reference topologies used for our numerical evaluation. Notice that squares indicate node with computational capacity > 0 (hosts), while circles indicate nodes with computational capacity $= 0$ (switches).



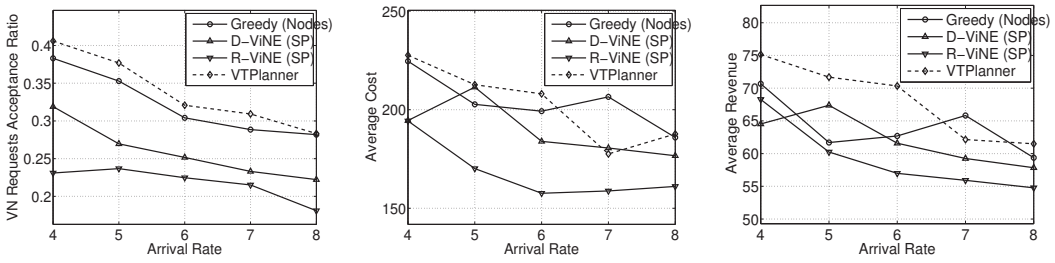
(a) Acceptance Rate. **(b)** Cost. **(c)** Revenue.

Fig. 4: Random on Random.



(a) Acceptance Rate. **(b)** Cost. **(c)** Revenue.

Fig. 5: Clique on Fat-Tree.



(a) Acceptance Rate. **(b)** Cost. **(c)** Revenue.

Fig. 6: Star on Fat-Tree.

and memory capacity. However, *VT-Planner* significantly outperforms state–of–the–art solutions both for number of VN requests accepted and for embedding cost/revenue.

Future work will extend the comparison to other scenarios including different random substrate networks. Moreover, testing over larger substrate networks will permit to verify scalability properties of *VT-Planner*. Several other optimizations are possible: one of such tuning is how to select the root node from which the Breadth–first traversal is started. We expect this choices to have different impact of the algorithm performance on different substrates.

Finally, tuning the control parameter α may provide additional degree of freedom to better fit the resource allocation performed by *VT-Planner* to specific network substrates. In future work we plan to provide a set of guidelines the will help the network administrator in properly configuring this parameters according to both the substrate network properties and the expected workload.

REFERENCES

[1] A. Chakrabarti, C. Chekuri, A. Gupta, and A. Kumar, "Approximation algorithms for the unsplittable flow problem," in *Proc. of APPROX*, Rome, Italy, 2002.

[2] I. Fajjari, N. Aitsaadi, G. Pujolle, and H. Zimmermann, "VNE-AC: Virtual network embedding algorithm based on ant colony metaheuristic," in *Proc. of IEEE ICC*, 2011.

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[4] "Ofelia." [Online]. Available: http://www.fp7-ofelia.eu/

[5] R. Doriguzzi Corin, M. Gerola, R. Riggio, F. De Pellegrini, and E. Salvadori, "VeRTIGO: Network Virtualization and Beyond," in *Proc. of EWSDN*, 2012.

[6] Y. Zhu. and M. Ammar, "Algorithms for Assigning Substrate Network Resources to Virtual Network Components," in *Proc. of IEEE INFOCOM*, Barcelona, Spain, April 23-29 2006.

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

[8] M. Chowdhury, F. Samuel, and R. Boutaba, "Polyvine: policy-based virtual network embedding across multiple domains," in *Proc. of ACM VISA*, 2010.

[9] J. He, R. Zhang-Shen, Y. Li, C.-Y. Lee, J. Rexford, and M. Chiang, "Davinci: dynamically adaptive virtual networks for a customized internet," in *Proc. of ACM CoNEXT*, 2008.

[10] J. Lischka and H. Karl, "A virtual network mapping algorithm based on subgraph isomorphism detection," in *Proc. of ACM VISA*, 2009.

[11] Y. Minlan, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: substrate support for path splitting and migration," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 17–29, Mar. 2008.