Rethinking Virtual Network Embedding in Reconfigurable Networks

Max Curran, Md. Shaifur Rahman, Himanshu Gupta and Vyas Sekar[†]
Department of Computer Science, Stony Brook University, NY, USA.
[†]Department of ECE, Carnegie Mellon University, PA, USA.

***Abstract*—The virtual network embedding (VNE) problem of mapping virtual network (VN) requests to a substrate network is a key component of network virtualization in datacenters. In a bid to improve datacenter network's performance and cost, there has been recent interest in "reconfigurable" network architectures, wherein the network topology can be changed at runtime to better handle current traffic patterns. Such reconfigurable networks seem naturally well-suited for efficient network virtualization—as networks can be "tailored" to accommodate the incoming VN requests. Motivated by the above, in this paper, we address the problem of virtual network embedding in reconfigurable networks; to the best of our knowledge, this has not been addressed before. In particular, we address the VNE problem in reconfigurable networks under two different models of VN link demands: fixed-bandwidth and stochastic-bandwidth demands. The former is the traditional model, while we propose the the latter to improve network utilization and leverage the runtime reconfiguration capability of reconfigurable networks. For the stochastic demand model, we employ a novel concept of embedding with "runtime-binding," wherein the embedding of a VN link is "configured" at runtime (via network reconfiguration) depending on the prevailing network state and traffic. We evaluate the efficiency of our proposed models and techniques via simulation using real VN requests and traffic statistics from large datacenters, and show that our proposed models and techniques offer significant performance advantages (up to 30-40%) over traditional models.**

I. Introduction

Network virtualization has recently gained intensive attention within both the research community as well as industry [1]. It enables sharing of virtualized resources which is a key enabler for cloud computing. In a network virtualization environment, multiple tenants can request computing resources in the form of virtual networks (VNs) to offer services to their end users, and infrastructure providers serve the requests efficiently to maximize their monetary gain. Tenants benefit by not having to make significant investment in physical infrastructure.

A key component of the above paradigm is efficient allocation of shared resources to the VN requests, i.e., the VN embedding (VNE) problem. Typically, a VN is represented as a graph with CPU and bandwidth demands associated with nodes and links respectively, and the VNE problem is to reject or embed the arriving VNs onto the substrate network, such that the total revenue from accepted VNs is maximized. The VNE problem is well-known to be NP-hard [2]. In fact, we show (§II) that the online VNE problem is inapproximable even for very special cases.

In this work, we revisit the VNE problem in light of recently proposed *reconfigurable* network architectures, which have the ability to change the network topology at runtime based on the network state (e.g., prevailing network traffic). Examples of reconfigurable networks include c-Through [3], Flyways [4], [5], Firefly [6], [7], ProjecToR [8], etc.; they use steerable wireless (free space optics or RF) devices or optical switches to reconfigure the network. We stipulate that reconfigurable networks are particularly well-suited for efficient allocation of VNs, as they can tailor the network topology based on the specific VN requests. However, the VNE problem in reconfigurable networks has not been addressed before, to the best of our knowledge. The key challenge in handling VNE problem in reconfigurable networks is to take advantage of the flexibility in configuring the network topology—in particular, when and how to reconfigure the network in response to arriving VNs. We show that failing to reconfigure the network appropriately can affect performance. To best leverage the reconfiguration capabilities, we propose the use of stochastic-bandwidth demand model which introduces additional challenges to develop an efficient embedding strategy, e.g., runtime binding of link embeddings.

We consider two different models to represent bandwidth demands of a VN link: (i) *fixed-bandwidth* demand model, wherein each VN link demand is expressed as a fixed bandwidth (as in the traditional VN models), and (ii) *stochastic-bandwidth* demand, wherein each VN link demand is represented as stochastic traffic parameters. The stochastic-bandwidth demand model is motivated by various factors in the context of reconfigurable networks: (a) to improve overall network utilization, (b) to better leverage the runtime reconfiguration ability of reconfigurable networks via our newly proposed embedding model of *runtime-binding*, (c) to relax otherwise overly exacting user constraints, a major source of VN allocation failures [9]. The "burstable performance instances" supported by Amazon Web Service [10] uses a similar "elastic" model wherein a baseline demand is allocated with an allowance to burst over the baseline, and the revenue computed accordingly. This model also facilitates a way to trade performance for revenue in a controlled manner, by varying the "slack-factor" parameter (see §IV).

Contributions. In the above context, we make the following specific contributions:

1) We show that the traditional VNE problem in standard

networks is inapproximable (§II); these results also imply that VNE problems considered in this paper for reconfigurable networks are inapproximable.

2) We formulate the VNE problem in reconfigurable networks, and explore the design space with appropriate (link-demand and embedding) models. In particular, we consider the VNE problem in reconfigurable networks with (i) fixed-bandwidth model in §III, and (ii) stochastic-bandwidth demand in §IV.
3) We demonstrate significant performance advantage of our proposed models and techniques via evaluations over real datacenter requests and traffic traces (§V).

II. Background and Intractability Results

In this section, we describe the traditional VNE problem and related work, followed by some intractability results. We start with a few definitions.

Substrate and Virtual Networks; Virtual Network Embedding (VNE). Typically, both the substrate network and virtual networks are modeled as graphs of nodes and edges connecting them. In the substrate network, the nodes are servers with given computing capacity, and the edges between the nodes are associated with a given bandwidth. Each VN is also represented as a graph with computing and bandwidth *demands* for each of its nodes and links respectively. Informally, embedding a VN onto a substrate network entails locating the VN nodes onto the substrate nodes, and mapping the VN links onto a *path* connecting the corresponding substrate network nodes such that the CPU usage and bandwidth constraints are satisfied. We now formalize the above.

Definition 1: (Virtual Networks; VN Embedding.) Consider a substrate network represented by a graph $G'(V', E')$, where each vertex $v' \in V'$ has $c_{v'}$ units of *available* (as described below) CPU, and each edge $e' \in E'$ has $b_{e'}$ units of *available* bandwidth. Also, consider a VN $G(V, E)$, where each vertex $v \in V$ requires c_v units of CPU and each edge $e \in E$ requires b_e units of bandwidth demand (i.e., a fixed-bandwidth demand of b_e).

An embedding of a VN $G(V, E)$ onto the substrate network $G'(V', E')$ are mappings $M_1 : V \mapsto V'$ and $M_2 : E \mapsto P'$ where P' is the set of all paths in G', such that: (i) M_1 is a one-to-one function (i.e, each node v of V is mapped to a different node v' in V'; (ii) if $M_1(v) = v'$, then the CPU demand of v is less than the available CPU in v'; and (iii) if $M_2(e) = p'$, where $e = (u, v)$ and $p' = (u', w_1, w_2, \ldots, w_k, v')$, then $M_1(u) = u'$ and $M_1(v) = v'$; and (iv) $\sum_{e \in S(e')}(b_e) \leq b_{e'}$, where $S(e')$ is the set of all links e of G that are mapped (via M_2) to a path containing e'. □

Definition 2: (VNE Problem.) Given a substrate network and a stream of arriving VNs with a duration associated with each, the (online) VNE problem is to immediately (on VN's arrival)[1] reject or accept-and-embed each VN onto the substrate network, so as to maximize the total number of accepted VNs or total revenue from accepted VNs. The *revenue* for an accepted VN is typically defined as its total demand (CPU plus bandwidth, summed up across all nodes and links) multiplied by the duration; the total revenue is the sum of revenues of the accepted VNs. □

Notes: (i) Each accepted VN runs in the substrate network only for the duration associated with it; (ii) At any instant, the *available* CPU at a substrate node v' is equal to the original CPU capacity minus the CPU demands of the *running* VN nodes mapped onto v'; (iii) Similarly, at any instant, the *available* bandwidth at a substrate edge e' is equal to the bandwidth capacity of e' minus the bandwidth demands of the running VN links that are mapped onto a path p' containing e'; (iv) By default, once embedded, the embedding of a VN is not changed (say, to accommodate a newly arrived VN). However, in other variants [11]–[13], such re-embedding are permitted.

Related Work. The above described VNE problem is NP-hard [2] and even inapproximable (see §II-A), and hence, most works have focused on developing efficient heuristics. See [14] for a detailed taxonomy of different approaches to solve the above VNE problem and its variations. These heuristics can be categorized into two broad categories: (i) Two phase, and (ii) merged phase. The two phase approach involves first mapping the VN nodes onto the substrate network nodes, followed by embedding the VN links onto the substrate network paths [12], [15]–[18]. In contrast, the merged-phase approaches [19] map the VN nodes and links simultaneously onto the substrate network, i.e., the node mapping is done contingent on the feasibility of embedding the corresponding VN links. In both approaches, backtracking with branch and bound is sometimes used to explore the space of embeddings [16], [17], [19]. Some recent approaches [16] have developed techniques to rank/sort the nodes, based on residual topological resources around them, and used this ranking to drive the node mapping. In addition, stochastic algorithms like simulated annealing [20], particle swarm optimization [11], tabu search, or genetic algorithms, have also been used to solve the VNE problem. The main drawback of stochastic algorithms, besides their relatively long execution times, is their high likelihood of getting stuck in local optima.

The works that are closest to the techniques explored in our paper are: (i) [15] considers VNs with probabilistic "sub-workloads" and addresses the resulting resource allocation problem at each time slot, (ii) [12] considers embedding each VN link onto multiple paths rather than a single path (but the overall embedding is still fixed at VN's arrival), (iii) [21] considers single-node VNs with stochastic traffic demands (no CPU requirement) and addresses the resulting stochastic "bin-packing" problem, and (iv) [22] addresses the bandwidth allocation problem in a hybrid optical-electric datacenter; they do not consider CPU demands (so, inapplicable in our context).

[1] It would be interesting to consider a variant, wherein a VN may be accepted after a (bounded) delay.

A. Intractability Results

To the best of our knowledge, the only theoretical result known about the above VNE problem is that it's NP hard [14]. Here, we show that even the simplest versions (e.g., restricted to single-node VNs) are inapproximable, i.e., no online algorithm can have a non-zero *competitive ratio*.[2] Since the above VNE problem is a special case of the VNE problems in reconfigurable networks addressed in our paper, the below hardness results also apply to our problems. Below, we observe that, even for the special case of single-node VNs, the above VNE problem is inapproximable for either number of VNs accepted or revenue as a maximization objective. However, when restricted to "accommodating sequences" and allowing re-embeddings, the revenue-maximization VNE problem for single-node VNs can actually be solved optimally in exponential time. In addition, the revenue-maximization VNE problem has a simple 2-competitive online algorithm when restricted to permanent (infinite duration) single-node VNs. We omit the proofs of Corollary 1 and 3 (they follow from arguments in [25]).

Corollary 1: For the objective of maximizing the number of VNs accepted, the VNE problem is inapproximable (i.e., no online algorithm can have a non-zero competitive ratio) even for the special case of single-node VNs. □

Corollary 2: For the objective of maximizing revenue from accepted VNs, the VNE problem is inapproximable even for the special case of single-node VNs. This hardness result holds even if we permit re-embedding or restrict to *accommodating sequences* of VNs (i.e., arriving VNs that can be all accepted by the optimal algorithm).

PROOF: Consider single-node VNs and a 2-node substrate. Let two VNs L_1 and L_2 arrive, each with a unit CPU demand and small duration. If they are accepted, then subsequent two VNs, each of a unit CPU demands and infinite duration would be rejected. If L_1 and L_2 are rejected, then no other VN may arrive. Here, allowing re-embeddings wouldn't help. Similar sequences can be constructed for accommodating sequences (we omit details). ■

Corollary 3: For the special case of single-node permanent (infinite duration) VNs, the VNE problem has a simple 2-competitive online algorithm. □

Corollary 4: Consider the VNE problem which allows re-embeddings but is restricted to accommodating sequences of single-node VNs. This problem can be solved optimally in exponential time.

PROOF: At every VN arrival, do optimal "bin packing" of the active VNs into the servers. Since the sequence is accommodating, all the active VNs must fit into the servers—and this packing can be easily determined in exponential time. ■

Corollaries 3 and 4 suggest that restricting the VNE problem to VNs with infinite duration, accommodating sequences, and/or allowing re-embeddings may make the problem more tractable. On the other hand, all the above results consider only single-node VNs; multi-node VNs bring in the embedding bandwidth constraint, which adds considerable complexity to the problem.

III. Fixed-Bandwidth Link Demands

In this section, we formally define reconfigurable networks, and address the VNE problem in them with the traditional VN model (i.e., with fixed-bandwidth demands on links as per Def. 1).

Reconfigurable (Datacenter) Networks. We model a datacenter as a set of racks, each consisting of a set of servers. Each server is connected to the top of rack (ToR) switch on its rack. The ToR ports are connected via *candidate links* to form a *candidate graph* among them; each node in the candidate graph is a port in a ToR.[3] See Figure 1. Any candidate link can be "activated" at runtime (with some minimal reconfiguration delay), with the following two *reconfiguration constraints*: (i) Each node can have at most one active candidate link incident on it, at any instant; (ii) A set of candidate links can be active simultaneously at any instant only if they do not cause any wireless interference among them. The second constraint can be modeled by enumerating all possible sets of candidate links that can be active simultaneously. In optical or FSO-based reconfigurable networks, the second constraint does not exist due to lack of wireless interference.

Network Reconfiguration entails activation and/or deactivation of one or more candidate links, without violating the above constraints; Link activation latency depends on the steering mechanism, and can vary from a few microseconds [8] to a few milliseconds [7]. Techniques from [7] can be used to maintain network consistency during flux and to ensure traffic flow during VN embedding (at its arrival). Runtime-binding of link embeddings (§IV) is done for each traffic flow, and hence is viable only for large flows; short flows are handled as suggested in §IV. Note that network reconfiguration, VN embedding, and runtime bindings are all done seamlessly without involving the VN applications or clients.

Partially-Reconfigurable Networks. The above definition can generalized to *augmented* or *partially-reconfigurable* networks such as c-Through [3], Flyways [4], wherein the overall network essentially consists of two subnetworks: (i) a fixed or static subnetwork, and (ii) a reconfigurable subnetwork. Here, the non-server ports of the ToR switches are categorized into *static* and *reconfigurable* ports, and the static ports form the static subnetwork (via possibly additional core switches) and the reconfigurable ports form a reconfigurable subnetwork as defined above.

Motivating Example. We now give an example that illustrates how reconfigurable networks can accommodate more VN requests. See Figure 2. Here, the reconfigurable substrate network has 6 nodes. Each node has a unit CPU, and each link has a unit bandwidth capacity. Assume that the ToR ports

[2] *Competitive ratio* [23], [24] of an online algorithm is simply the worst-case ratio of the online algorithm's performance with that of the optimal *offline* algorithm that has access to the entire sequence of inputs.

[3] The nodes corresponding to ports of the same ToR are implicitly connected by high-bandwidth ToR-switch links.

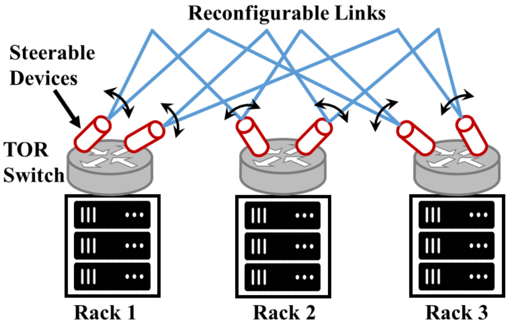

Fig. 1: Reconfigurable Datacenter Networks.

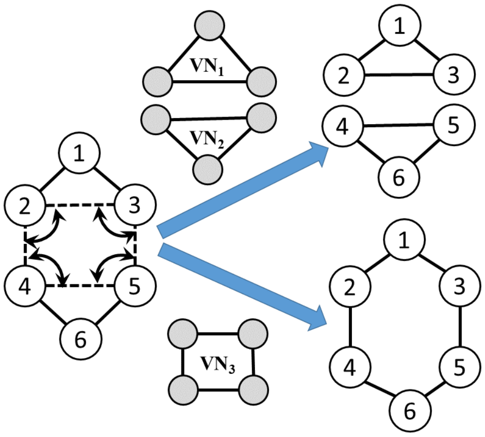

Fig. 2: VNE in a Reconfigurable Network with Fixed-Bandwidth Demands.

are configured in a way so that: (i) solid edges are/can be always active, and (ii) nodes 2 to 5 can have at most one active dashed edge incident on them. Consider the shown VNs VN_1, VN_2, VN_3, where each VN node requires a unit CPU and each VN link requires a unit bandwidth. As shown, the given reconfigurable substrate network can embed both $\{VN_1, VN_2\}$ as well as VN_3, with an appropriate reconfiguration. However, any static substrate network with 2 links per node (i.e., any single static configuration of the given network) can only accept either $\{VN_1, VN_2\}$ or VN_3, but not both.

VNE Problem with Fixed Demand Model (FXD). Given a reconfigurable network and a sequence of arriving VNs (with links having fixed-bandwidth demands as in Def. 1), the FXD problem is to, at arrival of a VN L, reject L or accept and embed L over the graph of active candidate links (after possible network reconfiguration). The objective is to maximize the total revenue derived from the accepted VNs. For now, we disallow re-embeddings, i.e., we assume that once a VN is embedded, its embedding is not changed (we consider the variant with re-embeddings, later). Thus, active candidate links that are being used by a *running* (i.e., accepted and not yet finished) VN can't be deactivated for the VN's duration.

Observation 1: In the FXD problem, there is no advantage in reconfiguring the substrate network or re-embedding (link or nodes) previously-accepted VNs, *between VN arrivals.* □

The above observation holds since embedding of a VN with fixed demands essentially "reserves" the requisite demand—thus, there is no advantage (for VNE purposes) in changing the network topology or embedding until a new VN arrives.

FXD Algorithm. At a high-level, the algorithm first ranks the substrate network nodes, wherein the rank signifies the amount of free resources (CPU and bandwidth) available in the node's neighborhood (including itself). Similarly, the VN nodes are also ranked, based on the resource demands. Then, these rankings are used to find matching between the VN nodes and the substrate nodes; in particular, we try to map the highest-ranked VN nodes with the highest-ranked substrate nodes, while, as a merged-phase approach [19], also ensuring that VN links incident on mapped VN nodes also find a feasible embedding. To maximize the number of links that are mapped simultaneously with the nodes, we traverse the VN nodes in a pre-order manner (i.e., over a breadth-first tree).

Formal Description. First, we rank the substrate and VN nodes, based on the formula described in the below paragraph. Let T be some breadth-first tree of the VN graph, rooted at the highest-ranked VN node. We traverse T in a pre-order manner, while traversing the nodes at any particular level in decreasing order of their ranks. For each traversed VN node u, we do the following:

1) Map the virtual node u to the highest-ranked substrate node that has more CPU available than u's CPU demand.
2) Embed/map the VN links connecting u to already-mapped VN nodes. Embedding a VN link in a reconfigurable network may involve careful reconfiguration of the network, as discussed below.
3) If any of the above steps fail, we backtrack to the first step above and map u to the *next* available substrate node. After sufficient backtracking for node u, we remove u's embedding and backtrack to the previously-mapped VN node w and try the *next* mapping for w, and so on.
4) If all VN nodes and links have been mapped successfully, then we return `accept`, else we `reject` the given VN.

Our algorithm is an extension of the algorithm in [17] for the traditional VNE problem in non-reconfigurable networks. The key differences of our algorithm from the one in [17] are: (i) To rank the nodes, we need to define the "effective" bandwidth of a candidate link to incorporate the fact that the candidate link may or may not be activated eventually; (ii) Embedding of VN links requires a careful reconfiguration of the substrate network. We discuss each of these below.

Node Ranking. Let the graph $G = (V, E)$ denote the new VN request. We rank each node $u \in V_v$ by the following formula [17]:

$$r(u) = (1-\mu)\frac{c_u}{\sum_{v \in V_v} c_v} + \mu \sum_{v \in N(u)} \frac{\hat{b}_{(u,v)} \times r(v)}{\sum_{w \in N(v)} \hat{b}_{(w,v)}} \quad (1)$$

where $\hat{b}_e$ is the *effective* (as defined below) bandwidth of an edge e, $N(u)$ is the set of neighbors of u and μ is a factor that determines the relative weights between normalized node capacity and normalized link capacity. The above equation is recursive, since a node's rank depends on the ranks of its neighboring nodes and vice versa. We can denote the formula in matrix notation as follows:

$$\begin{aligned} &\mathbf{r} = (1-\mu)\mathbf{c} + \mu\mathbf{M}\mathbf{r} \\ &\implies \mathbf{r} = (1-\mu)(\mathbf{I} - \mu\mathbf{M})^{-1}\mathbf{c} \end{aligned} \tag{2}$$

Here the matrices $\mathbf{r} = [r(1), r(2), \ldots, r(n)]^T$ and $\mathbf{c} = [\hat{c_1}, \hat{c_2}, \ldots, \hat{c_n}]^T$, $n = |V_v|$, the relative capacity $\hat{c_u} = \frac{c_u}{\sum_{v \in V_v} c_v}$, and the entry in the u-th row and v-th column in matrix $\mathbf{M}$ is $m(u,v) = \frac{\hat{b}_{uv}}{\sum_{w \in N(v)} \hat{b}_{(u,w)}}$. By solving the matrix equation, the rank of each node can be determined.

We define the *effective* bandwidth of VN links as their bandwidth demands, but the effective bandwidth of the candidate links of the substrate network is defined as follows:

$$\hat{b}_{(m,n)} = \begin{cases} \textit{available} \text{ bandwidth in } (m,n) & \text{if } (m,n) \text{ is active} \\ \frac{b_{(m,n)}}{Q} & \text{otherwise} \end{cases} \tag{3}$$

where $b_{m,n}$ above is the original capacity of a substrate link, and Q is the number of candidate links that can't be activated (due to reconfiguration constraints) in conjunction with (m,n).

Link Embedding in a Reconfigurable Network. Embedding a specific VN link, say $e = (u,v)$, entails finding the shortest path p from $u' = M_1(u)$ and $v' = M_1(v)$ in the "residual" (defined below) candidate graph, such that activation of links on p doesn't cause violation of the reconfiguration constraints. When such a path p is determined, the links on the path are activated.

Residual Candidate Graph. In the residual candidate graph (for a given VN link e that is to be embedded), the available bandwidth of the active candidate links is reduced by the bandwidth used by embeddings of running VNs. In addition, we deactivate any unused active candidate links, and remove: (i) active candidate links whose available bandwidth is less than the b_e, the bandwidth demand of e, and (ii) inactive candidate links that can't be activated (due to the reconfiguration constraints)[4] in conjunction with other active candidate links used by running VNs.

FXD Problem with Re-embeddings. We now consider the FXD problem which permits re-embeddings of previously accepted VNs to accommodate new VNs; however, we never "remove" a previously accepted VN till its completion. By Observation 1, a re-embedding can be useful only at the time of a new VN arrival, when the newly arrived VN is otherwise being rejected. At a high-level, the algorithm with re-embeddings for a VN L works as follows. For a node u (of L) that cannot be mapped, rather than backtracking to an already-mapped node v of L, we instead look for a node w (of an already-embedded VN L') whose successful re-embedding would accommodate u. Similar strategy is used for a VN link that can't be mapped. In addition, if neither of the above strategies of single node/link re-embedding succeed, then we look to re-embed an entire already-embedded VN L' to accommodate L. If no such L' exists, we reject L. Re-embedding cost is optimized in the above strategies by considering the re-embedding units (nodes or links or VNs) in order of their re-embedding (e.g., total resource usage) cost.

IV. Stochastic-Bandwidth Link Demands

In this section, we consider the VNE problem in reconfigurable networks, with a stochastic-bandwidth link demand model that improves network utilization and best leverages runtime reconfiguration capability of the substrate network.

A. Stochastic Demands and Runtime-Binding Embedding

In the traditional VN model (Def. 1 and §III), the VN links have a fixed bandwidth requirement which entails reserving a fixed amount of bandwidth in the substrate network's path to which it is mapped. This results in inefficient utilization of network resources, since fixed bandwidth demand typically represents peak usage and in practice, at most times, the bandwidth used is likely to be much less. To alleviate this problem, we consider the stochastic demand model wherein we represent the VN link bandwidth demands in terms of stochastic traffic parameters rather than a fixed value. In this context, we also consider embedding of VN links with runtime-binding in the substrate network, wherein a VN link is mapped to a set of paths at arrival time, and at runtime, one of these mapped paths is chosen (e.g., on a per-flow basis). The above two concepts leverage the key advantage of reconfigurable networks, namely, runtime reconfiguration, while improving network utilization compared to the traditional VNE model. We start with a motivating example.

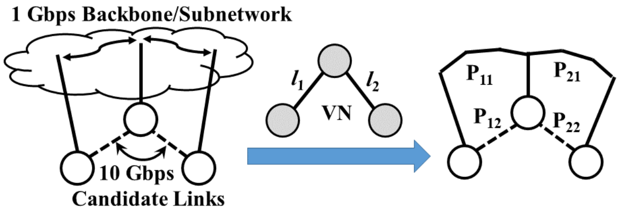

Fig. 3: VN Embedding with Runtime-binding

Illustrating/Motivating Example. For this example, we use the following stochastic traffic model of VN link demand: each VN link requires 10 Gbps bandwidth 10% of the time, and the remaining 90% of the time, it requires only minimal (1Gbps) bandwidth. Consider a VN with three nodes and two links, as shown in Figure 3. The figure also shows a substrate network of three nodes (let's assume that other nodes are already used up), each of which has two ports—one of which is connected to a backbone/subnetwork of 1Gbps bisection bandwidth, while the other port is connected to 10Gbps candidate link(s) as shown. Let's assume that only one of the two candidate links can be active at any time. The figure also shows a proposed embedding with runtime-binding, wherein each VN link (e.g., l_1) is mapped to two paths—one (P_{11}) over the static 1Gbps backbone and the other (P_{12}) over the 10Gbps

[4] Here, for simplicity, we implicitly assume reconfiguration networks with no wireless interference (e.g., FSO-based networks). Incorporating wireless interference would require a slightly more involved strategy; we skip the details here.

candidate link. The candidate links are activated/deactivated as needed to satisfy the bandwidth needs of the VN links.

Now observe that the proposed embedding will be able to satisfy the VN link requirements 99% of the time, since only 1% of the time *both* the VN links will require 10Gbps bandwidth (in which case, one of the VN links (say l_1) will activate and use the 10Gbps candidate link P_{12}, while l_2 will use the sub-optimal 1Gbps P_{21} path. The above simple example illustrates and motivates the concept of embedding with runtime-binding; in practice, each VN link may be mapped to many more candidate paths in the substrate network, thereby reducing the probability of sub-optimal performance to a minimal value (which could be a user parameter).

VNs with Stochastic-Bandwidth Demands (ST-VN). The VN with stochastic-bandwidth demand (denoted by ST-VN)[5] is different from the traditional VN model in how the link bandwidth demand is represented. In ST-VN, the bandwidth requirement of a VN link is represented by two parameters: (i) flow's (mean) arrival rate and its distribution function (e.g., Poisson, uniform, or normal), and (ii) probability distribution of the flow sizes.

ST-VN's Embedding with Runtime-Binding. The traditional way of embedding a VN onto a substrate network is essentially an embedding with *arrival-time binding*, wherein each VN link L is mapped onto a path (or multiple paths [12]) p in the substrate network and L's traffic is then always routed along p. However, when embedding a ST-VN onto a reconfigurable network, we propose *embedding with runtime-binding*—wherein each link L of the ST-VN is mapped onto a *set* P of potential (candidate) paths in the reconfigurable network. Only at runtime, e.g. on a per-flow basis, *one* of the paths in P is chosen (and activated), based on the network state, to route the flow on L. As only one path is chosen for each flow, there are no TCP related issues. Each path $p \in P$ is given a fractional weight, such that aggregate weight across all paths is one. The weight essentially corresponds to the probability of the path being used at runtime; for simplicity, we use equal weights for all paths in P since the probability of a path being used depends on the runtime state whose distribution is difficult to estimate. The motivation for embedding with runtime-binding in the context of ST-VNs is to best leverage the stochastic nature of VN traffic and the runtime reconfiguration capability of reconfigurable networks. In our evaluations, we will show that embedding with runtime-binding offers a significant performance advantage over the traditional (arrival-time binding) embedding model; however, the embedding constraint is more involved as described below.

Link Constraint for Embedding with Runtime-Binding. When embedding a ST-VN link onto a substrate network, we must ensure that none of the substrate network's resources are "over utilized." We do this by ensuring that the aggregate arrival rate of VN links mapped to any "unit-link" (defined below) of the substrate network is at most some precomputed *maximum arrival rate* λ_{max}. Mathematically, if the flow size (=d) is uniform and the arrivals follow a Poisson distribution, then the arrival rate λ_{max} that results in the probability of "collision" being less than σ_c in a link of bandwidth b is given by (we omit the derivation here):

$$\lambda_{max} = \frac{b}{d}\sqrt{\sigma_c} \tag{4}$$

We refer to σ_c as the *slack factor*, a parameter that can be used by the infrastructure provider to control the revenue-performance trade-off. If the flow sizes have a non-uniform known distribution, then we estimate the λ_{max} by using the same above equation, but use d as the *average* flow size; our simulations have shown that this estimation approach is quite accurate. Based on this, we enforce the constraint that the aggregate arrival rate over any "unit-link" of the substrate network is at most λ_{max}. We define a *unit-link* of a substrate network as any *set* C of candidate links wherein every pair of links in C conflict with each other (i.e., only one link in C can be active at any time). E.g., in the case of FSO-based networks [7], [8], any set of candidate links sharing an FSO device is a conflicting set of links. More formally, for a set of conflicting links C, we enforce the constraint:

$$\lambda_{max} \geq \sum_{e' \in C} \sum_{e \in S(e')} w(e, e') \cdot a_e \tag{5}$$

where $S(e')$ is the set of VN links that are mapped to a path containing e', a_e is the arrival rate of VN link e, and $w(e, e')$ is the aggregate weight of all the paths (of e's embedding) that contain e'.

Handling Short Flows. Network reconfiguration incurs a delay which can be on the order of few microseconds [8] to a few milliseconds [7]. Thus, reconfiguring the network to efficiently route a short flow (e.g., $<$ 10MB flows) may not be effective. There are two possible approaches to effectively handle short flows in reconfigurable networks: (i) Use appropriate network management techniques to ensure network connectivity at all times [7], and then reconfigure the network only for large flows (short flows will be automatically routed via *some* route in the connected graph); (ii) Reserve a static backbone/subnetwork of sufficient bisection bandwidth exclusively for short flows. In the latter approach, embedding with runtime-binding of VN links should be done only with respect to the large flows, i.e., in the above Eqn. 5, the value d used is the average size of large flows. Also, each VN link will have an implicit (without any explicit reservation) embedding in the static backbone to route the short flows.

B. ST−RB *Problem and Algorithm*

We now formulate ST−RB, the VNE problem of embedding ST-VNs with runtime-binding in reconfigurable networks.

ST−RB Problem Formulation. Given a reconfigurable substrate network with ST-VNs arriving, the (online) ST−RB problem is to either reject, or accept and embed the arriving ST-VN with runtime-binding, so as to maximize the total revenue from accepted ST-VNs. The ST−RB problem

[5] We sometimes just refer to them as VNs, when obvious from the context.

is essentially the same as the FXD problem except for two differences: (i) The arriving VNs are actually ST-VNs which have a different model for link demands, and (ii) the VN embedding is runtime-binding, as defined above. In the ST-RB problem, the revenue of an accepted ST-VN is defined as the sum of CPU demands plus the aggregate expected-traffic (= average flow size times arrival rate) of all links.

ST-RB Algorithm. At a high-level, the ST-RB Algorithm is similar to the FXD Algorithm of previous section, except for two key differences: (i) the node ranking formula is adjusted slightly to account for the stochastic traffic model, and (ii) the link embedding in ST-RB is more involved, as it entails embedding with runtime-binding of a ST-VN link. We discuss each of the these in more detail below.

ST-RB Node Ranking. The ranking of nodes in ST-RB is similar to the one for FXD, except that we need to replace the effective bandwidth $\hat{b}$ in Eqn. 1 with effective arrival-rates as defined below. First, note that in the ST-RB problem, candidate links are *not* activated during the embedding process, and thus, the reconfiguration constraints do not play a role. We define the *effective arrival-rate* as follows: (i) For a ST-VN link l, it is just the arrival-rate demand of l; (ii) For a candidate link l' of substrate network, it is the *available arrival-rate* of l' at that point, which is the maximum arrival rate λ_{max} minus the used up arrival rate by previous running embeddings.

Embedding with Runtime-binding of a ST-VN Link. Recall that embedding of an ST-VN with runtime-binding involves mapping each link onto a set of paths in the substrate network with appropriate weights. Runtime-binding embedding of a single ST-VN link $e = (u, v)$ thus entails searching for a set of paths between substrate nodes u' and v' (to which the VN nodes u and v are mapped to) and assigning them weights. In particular, the mapping of a ST-VN link $e = (u, v)$ onto a set of paths $p_1, p_2, \ldots, p_k$ between u' and v' require the following considerations: (i) The number of chosen paths (k) should not be very small so as to provide sufficient "flexibility" to the runtime-binding; in our evaluations, we choose about 50 paths. (ii) The path lengths should be minimal, to minimize the usage of substrate network resources. (iii) Since each path is assigned a weight of $1/k$, the links on the paths should have at least a_e/k available arrival-rate (assuming edge-disjoint paths) where a_e is the arrival rate demand of e. The combination of the above considerations makes the problem of finding such paths for runtime-binding embedding challenging. We use the following heuristic.

Let m be the shortest path length between u' and v', and m' be some given threshold on the largest path length to consider for embedding. Then, for each d between m and m' in increasing order: Find (as described later) a set P_d of d-length paths (called *valid*) such that if each path in P_d is assigned a weight of $a_e/|P_d|$, there is no substrate link e' that gets an aggregate weight of more than the available arrival-rate of e'. We pick the smallest d that yields a valid P_d. Now, for a given d, we find a valid set of paths P_d as follows. For a path p, let a_p be $\min_{e' \in p} a_{e'}$ where $a_{e'}$ is the available arrival-rate on e'. Now, first, find a sufficiently large set of d-length paths, and sort them in decreasing order of their a_p's to yield the sorted list $p_1, p_2, \ldots, p_m$. Let's first assume these paths to be pairwise edge-disjoint. Then, it's easy to see that a sublist $p_1, p_2, \ldots, p_n$ is a valid set of paths P_d iff $a_{p_n} \geq a_e/n$. If the paths $p_1, p_2, \ldots, p_m$ are not pairwise disjoint, then we greedily pick one path at a time, in the above order, to create P_d while ensuring that the P_d remains valid at each stage.

V. Evaluation Results

We now discuss the performance evaluation of our models and techniques. The objective of our evaluation study is to demonstrate the following: (i) Our developed algorithms perform well (by comparison with ILP-based algorithm for small networks); (ii) Even the traditional VN model with reconfigurable substrate network offers a performance advantage; (iii) Our proposed models, namely, the ST-VN as well as embedding with runtime-binding, offer significant performance advantage compared to the traditional models. To evaluate our techniques, we use a custom-built flow-level simulator over an FSO-based datacenter network architecture, and use traffic and VN request traces from production datacenters. We start with describing our evaluation setting in more detail.

Substrate Network. For our evaluation, we consider the FireFly [7] or ProjectTor [8] like free-space optics (FSO) based reconfigurable network. In such FSO-based DC networks, each rack is equipped with a ToR and a number of FSO devices (equal to the number of servers on the rack), and each FSO device can be steered at runtime to target another FSO device on another rack. Our default network is a 512 server network, organized in 64 racks of 8 servers each. Each rack is equipped with a ToR switch connected to 8 FSO devices. Each FSO device is connected to 8 candidate links—only one of which is active at a time. Thus, each rack has 64 candidate links, and hence, we create a candidate link between every pair of racks, with the grouping of candidate links to FSO devices done randomly. Each link has a 10Gbps bidirectional bandwidth.

Virtual Network Requests. There are two components to generating VN requests: VN topology, and CPU and bandwidth demands on nodes and links respectively. We pick VN topologies directly from the DeterLab [9] dataset which contains anonymized virtual network requests made by actual clients. The CPU demand is randomly picked between 0 and 0.25 units, where 1 represents the CPU capacity of one server. We choose the arrival-rate demands of ST-VNs randomly between 0 and λ_{max} (as defined in Eqn 5), based on a slack factor of 10%. This λ_{max} corresponds to 3.3Gbps expected traffic; thus, we choose the fixed bandwidth demands in traditional VN model randomly between 0 and 3.3Gbps.

Runtime Simulation. For scalability, we use a custom-built flow-level simulation using a fluid model, which does not model any packet-level or TCP effects as they are not important in our context. Following prior works [5], [7], we use synthetic traffic models based on DC measurements [26]. In particular, we treat each VN link independently, and generate

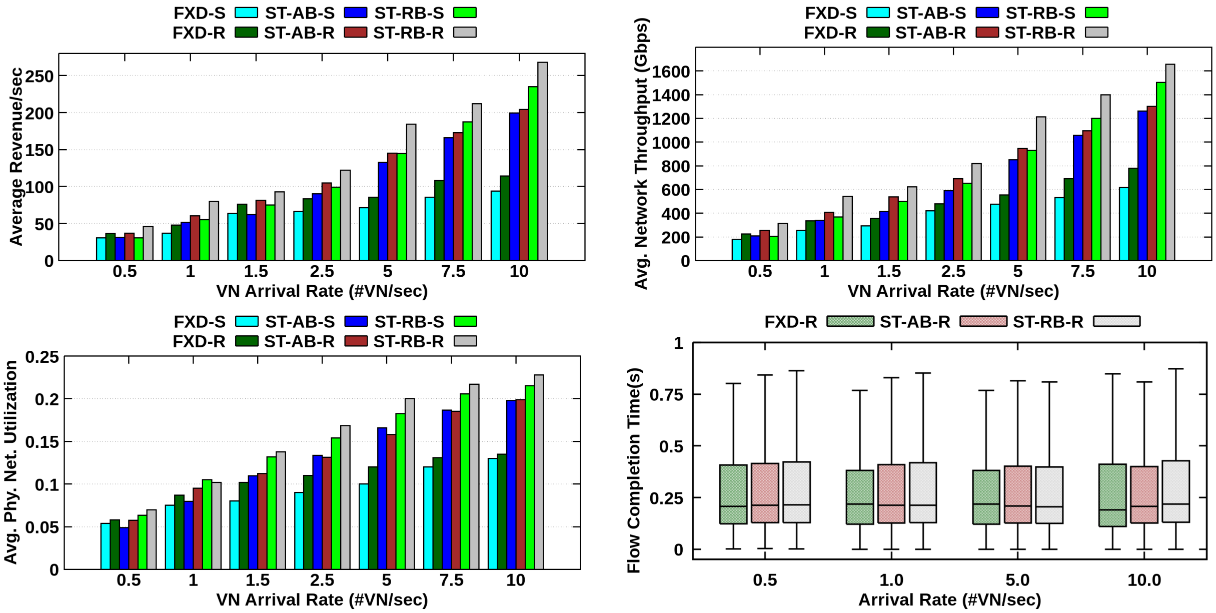

Fig. 4: (a) Revenue, (b) Throughput, (c) Network Utilization, (d) Flow completion times, of various algorithms, for a 512 server (64-rack, with 8 servers each) network, for varying arrival rates of VNs.

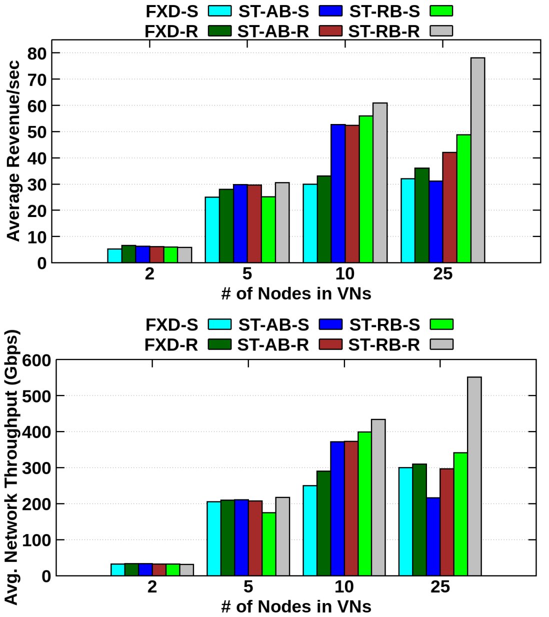

Fig. 5: Revenue and throughput comparison for synthetic VNs with varying sizes.

flows with an empirically-derived flow size distribution. We focus only on large flows, assuming small flows are handled as discussed in previous section, and their impact is negligible on the large flows' performance if they are handled in the same subnetwork. As in prior works [5], [7], we use a Poisson distribution to model flow arrival times. For fixed demand links, the arrival rate is chosen appropriately to match with the bandwidth demand. We run the simulation for 5000 seconds. For embedding with runtime-binding schemes, we route a flow of a VN link on the least congested "realizable" path from the set of paths the link was mapped to. A path p is realizable if the candidate links on the path can be activated without needing to deactivate any already-used active links. If there are no realizable paths among the mapped paths, then we delay the flow until a realizable path becomes available.

Algorithms. To address our evaluation objectives stated above, we consider the following six algorithms. In essence, we consider three useful combinations[6] of demand (fixed or stochastic) and embedding (runtime or arrival-time binding; see §IV) models. For each of these three combinations, we consider both types of networks, i.e., reconfigurable and static (non-reconfigurable), yielding six algorithms. We label these six algorithms as FXD-R, FXD-S, ST-RB-R, ST-RB-S, ST-AB-R, and ST-AB-S, where R or S suffix refers to reconfigurable or static network. Note that, here, FXD-⋆ algorithms correspond to the fixed demand and arrival-time binding combination, and the ST-AB-⋆ algorithms refer to the stochastic demands and arrival-time binding combination.[7] Finally, we also consider two ILP-based algorithms, labeled as ILP-FXD-R and ILP-ST-RB-R. The ILP-based algorithms use, for each arriving VN L, an ILP formulation to embed L plus all the already-embedded VNs with minimum resource (CPU capacity plus network bandwidth) usage if possible (if not, L is rejected).

Performance Results. We evaluate the above algorithms for the following performance metrics: (i) Revenue (as defined below) derived from accepted VNs, (ii) Total network throughput averaged over time, (iii) Flow Completion Time (FCT) statistics across all flows (for clarity, we only show the FCT of reconfigurable network schemes), and (iv) Network utilization (total bandwidth usage over all network links) averaged over time. Above, we define the *revenue* derived from an accepted VN as the sum of (i) total (fractional) CPU demands of the

[6] By Observation 1, the fourth combination (of fixed-bandwidth demand with runtime-binding) is not useful and hence, not considered.

[7] For the ST-AB-⋆ algorithms, we reuse the FXD algorithm of §III with the following modifications: (i) map the substrate network's link bandwidths to maximum arrival rate λ_{max}, as per the above Eqn 5, and (ii) use arrival rates instead of bandwidths in the embedding constraint.

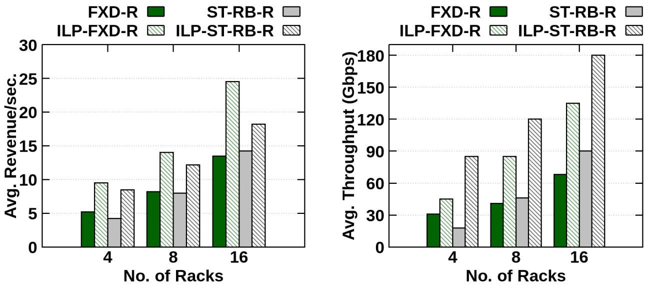

Fig. 6: Revenue and throughput comparison with ILP-based approaches, for small networks.

VN nodes, and (ii) total bandwidth (or expected traffic for ST-VNs) demands of the VN links normalized by the bandwidth capacity of a substrate network link. Note that the **revenue metric is a more generalized and meaningful metric** than fraction of VNs accepted, especially for VNs of varying sizes and topologies. Figure 4(a-d) show the performance of various algorithms for each of the above metrics, in the default setting of 64 racks with 8 FSOs on each rack and each FSO with 8 candidate link (single link for the fixed networks). We make two key observations: (i) ST-RB-R, the algorithm, which incorporates the stochastic traffic VNs as well as embedding with runtime-binding model, significantly outperforms the other algorithms in terms of revenue, the key performance metric. Not surprisingly, it also results in best network throughput, while exhibiting similar FCT performance to other schemes. (ii) There is a significant revenue advantage (up to 30-40%) due to reconfigurable networks, as evidenced by the performance difference between the corresponding $\star$-R and $\star$-S algorithms. Similarly, we note that embedding with runtime-binding model by itself (see ST-RB-$\star$ vs. ST-AB-$\star$) also offers significant revenue advantage (up to 30-40%). The above observations validate the effectiveness of our proposed models and techniques. We do not directly compare the VN-$\star$ vs. ST-$\star$ approaches, as they have slightly different input and revenue models.

Synthetic VNs of Varying Sizes. The above results are for the DeterLab [9] dataset containing VNs with a *mix* of sizes, ranging from 2 to 1000 (average 20) nodes. To corroborate our observations for more typical (smaller) VN sizes, we plot performance comparison (see Figure 5) for synthetically created VNs of certain sizes up to 25; here, we use topologies with linear number of links. We observe that ST-RB-R easily outperforms the other algorithms, with the performance gap, as expected, increasing with increase in the size of VNs.

Comparison with ILP for Small Networks. To demonstrate the absolute performance of our techniques, we compare the revenue and throughput performance of our algorithms with the ILP-based algorithms, for small network sizes (simulating ILP-based approaches for larger networks was infeasible). See Figure 6; here, for clarity, we only show the two main algorithms, FXD-R and ST-RB-R, and their ILP versions. We observe that the revenue as well as throughput of our algorithms is within a factor of 0.5 to 0.75 of the ILP approaches, which gives some confirmation of absolute performance efficiency of our algorithms.

VI. Conclusions

In this work, we have explored the design space of VN and embedding models to give effective formulations of the VNE problem in reconfigurable networks. We designed algorithms for the formulated problems, and evaluated our models and techniques over real data traces to demonstrate their significant performance advantage over traditional models. Ours is the first work in this space, and many improvements of our techniques and open questions remain. E.g., can we generalize the technique of runtime-binding to VN *nodes* too? I.e., embed a VN node to multiple substrate nodes at arrival time, and then pick one of them at runtime. This would require efficient migration techniques, in addition to efficiently solving the corresponding VNE problem. Moreover, could *delaying* the acceptance or rejection of a VN be an effective way of circumventing the intractability of VNE problems?

References

[1] T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the internet impasse through virtualization," *Computer*, 2005.

[2] D. G. Andersen, "Theoretical approaches to node assignment," 2002, unpublished Manuscript.

[3] G. Wang *et al.*, "c-Through: Part-time optics in data centers," in *ACM SIGCOMM*, 2010.

[4] D. Halperin *et al.*, "Augmenting data center networks with multi-gigabit wireless links," in *ACM SIGCOMM*, 2011.

[5] X. Zhou *et al.*, "Mirror mirror on the ceiling: Flexible wireless links for data centers," in *ACM SIGCOMM*, 2012.

[6] N. Hamedazimi, H. Gupta, V. Sekar, and S. R. Das, "Patch panels in the sky: A case for free-space optics in data centers," in *HotNets*, 2013.

[7] N. Hamedazimi *et al.*, "FireFly: A reconfigurable wireless data center fabric using free-space optics," in *ACM SIGCOMM*, 2014.

[8] M. Ghobadi *et al.*, "ProjecToR: Agile Reconfigurable Data Center Interconnect," in *ACM SIGCOMM*, 2016.

[9] J. Mirkovic, H. Shi, and A. Hussain, "Reducing allocation errors in network testbeds," in *IMC*, 2012.

[10] "Amazon Web Services EC2 Instance Types."

[11] Y. Zhu and M. H. Ammar, "Algorithms for assigning substrate network resources to virtual network components," in *IEEE INFOCOM*, 2006.

[12] M. Yu *et al.*, "Rethinking virtual network embedding: substrate support for path splitting and migration," *Comp. Comm. Review*, 2008.

[13] N. F. Butt *et al.*, "Topology-awareness and reoptimization mechanism for virtual network embedding," in *IFIP TCNC*, 2010, pp. 27–39.

[14] A. Fischer *et al.*, "Virtual network embedding: A survey," *IEEE Communications Surveys and Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.

[15] S. Zhang *et al.*, "An opportunistic resource sharing and topology-aware mapping framework for virtual networks," in *IEEE INFOCOM*, 2012.

[16] X. Cheng *et al.*, "Virtual network embedding through topology-aware node ranking," *Computer Communication Review*, vol. 41, no. 2, 2011.

[17] L. Gong *et al.*, "Toward profit-seeking virtual network embedding algorithm via global resource capacity," in *IEEE INFOCOM*, 2014.

[18] F. Bianchi and F. L. Presti, "A markov reward model based greedy heuristic for the VNE problem," in *IEEE MASCOTS*, 2016.

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

[20] R. Ricci, C. Alfeld, and J. Lepreau, "A solver for the network testbed mapping problem," *SIGCOMM Comput. Commun. Rev.*, 2003.

[21] M. Wang *et al.*, "Consolidating virtual machines with dynamic bandwidth demand in data centers," in *IEEE INFOCOM*, 2011, pp. 71–75.

[22] S. Hegde *et al.*, "Dynamic embedding of virtual networks in hybrid optical-electrical datacenters," in *23rd, ICCCN*, 2014, pp. 1–8.

[23] Wikipedia, "Comptetitive analysis (online algorithm)."

[24] X. Zhu, B. Tang, and H. Gupta, "Delay efficient data gathering in sensor networks," in *Intl. Conf. on Mobile Ad-Hoc and Sensor Networks*, 2005.

[25] M. Cygan, Ł. Jeż, and J. Sgall, "Online knapsack revisited," *Theory of Computing Systems*, 2016.

[26] A. Greenberg *et al.*, "VL2: A scalable and flexible data center network," in *ACM SIGCOMM*, 2009.