ViNEYard: Virtual Network Embedding Algorithms With Coordinated Node and Link Mapping

Mosharaf Chowdhury, *Student Member, IEEE*, Muntasir Raihan Rahman, *Member, IEEE*, and Raouf Boutaba, *Senior Member, IEEE*

***Abstract*—Network virtualization allows multiple heterogeneous virtual networks (VNs) to coexist on a shared infrastructure. Efficient mapping of virtual nodes and virtual links of a VN request onto substrate network resources, also known as the VN embedding problem, is the first step toward enabling such multiplicity. Since this problem is known to be $\mathcal{NP}$-hard, previous research focused on designing heuristic-based algorithms that had clear separation between the node mapping and the link mapping phases. In this paper, we present ViNEYard—a collection of VN embedding algorithms that leverage better coordination between the two phases. We formulate the VN embedding problem as a mixed integer program through substrate network augmentation. We then relax the integer constraints to obtain a linear program and devise two online VN embedding algorithms D-ViNE and R-ViNE using deterministic and randomized rounding techniques, respectively. We also present a generalized window-based VN embedding algorithm (WiNE) to evaluate the effect of lookahead on VN embedding. Our simulation experiments on a large mix of VN requests show that the proposed algorithms increase the acceptance ratio and the revenue while decreasing the cost incurred by the substrate network in the long run.**

***Index Terms*—Heuristics, network virtualization, online algorithms, optimization, resource allocation, virtual network embedding.**

I. Introduction

DUE TO the existence of multiple stakeholders with conflicting goals and policies, even the necessary alterations to the present architecture of the Internet are now hard to realize. To mitigate this impasse, network virtualization has been propounded as a fundamental diversifying attribute of the future internetworking paradigm that will allow multiple heterogeneous network architectures to coexist on a shared substrate [2], [3]. In a network virtualization environment, multiple *service providers* (SPs) will be able to create heterogeneous *virtual networks* (VNs) to offer customized end-to-end services to the end-users by leasing shared resources from one or more *infrastructure providers* (InPs) without significant investment in physical infrastructure [3]–[5]. Similarly, network virtualization will enable researchers to design and evaluate new networking protocols on heterogeneous experimental architectures [6], [7].

Each VN in a network virtualization environment is a collection of virtual nodes connected together by a set of virtual links. A virtual node is hosted on a particular substrate node, whereas a virtual link spans over a path in the substrate network. The *VN embedding problem*[1] deals with the mapping of a VN request, with *constraints on virtual nodes and links*, onto specific physical nodes and paths in the substrate network that has *finite resources*. Since multiple VNs share the underlying physical resources, efficient and effective embedding of *online* VN requests is of utmost importance in order to increase the utilization of substrate network resources and InP revenue.

However, the VN embedding problem is known to be $\mathcal{NP}$-hard even in the offline case. With constraints on virtual nodes and links, the offline VN embedding problem can be reduced to the $\mathcal{NP}$-hard *multiway separator problem* [8]. Even when all the virtual nodes have already been mapped, embedding the virtual links with bandwidth constraints onto substrate paths is still $\mathcal{NP}$-hard in the *unsplittable flow* scenario [9], [10]. As a result, a number of heuristic-based algorithms have appeared in the relevant literature [11]–[14]. Most of these proposals focused primarily on link mapping (using shortest path, k-shortest paths, and multicommodity flow algorithms) after employing greedy methods to preselect the node mappings. However, preselecting node mappings without considering its relation to the link mapping phase restricts the solution space and can result in poor performance.

In this paper, we introduce better correlation between the node mapping and the link mapping phases by presenting the ViNEYard algorithms, which include Deterministic VN Embedding (D-ViNE), Randomized VN Embedding (R-ViNE), and their extensions. We map virtual nodes onto substrate nodes in a way that facilitates the mapping of virtual links to physical paths in the subsequent phase.

To this end, we extend the substrate network graph by introducing meta nodes for each virtual node and connect the meta

Manuscript received May 24, 2010; revised November 15, 2010 and February 22, 2011; accepted May 16, 2011; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor C. Westphal. Date of publication July 14, 2011; date of current version February 15, 2012. This work was supported in part by the Natural Science and Engineering Council of Canada (NSERC) under its Discovery Program and by the World Class University (WCU) Program under the Korea Science and Engineering Foundation funded by the Ministry of Education, Science and Technology (Project No. R31-2008-000-10100-0). An earlier version of this paper appeared in the Proceedings of the IEEE Conference on Computer Communications (INFOCOM), Rio de Janeiro, Brazil, April 19–25, 2009.

M. Chowdhury is with the Computer Science Division, University of California, Berkeley, CA 94720 USA (e-mail: mosharaf@cs.berkeley.edu).

M. R. Rahman is with the Department of Computer Science, University of Illinois, Urbana–Champaign, IL 61801 USA (e-mail: mrahman2@illinois.edu).

R. Boutaba is with the David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada, and also with the Division of IT Convergence Engineering, Pohang University of Science and Technology (POSTECH), Pohang 790-784, Korea (e-mail: rboutaba@cs.uwaterloo.ca).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2011.2159308

[1]The words "embedding," "mapping," and "assignment" are used interchangeably throughout this paper.

nodes to a selected subset of physical nodes. We then treat each virtual link with bandwidth constraints as a commodity consisting of a pair of meta nodes. As a result, finding an optimal flow for the commodity is equivalent to mapping the virtual link in an optimal way. By introducing additional binary constraints that force only one meta edge to be selected for each meta node, we can effectively select exactly one substrate node for each meta node corresponding to a particular virtual node. We use mixed integer programming (MIP) formulation [15] to solve the embedding problem with binary constraints on the meta edges and linear constraints on actual substrate network links. Since solving an MIP is known to be $\mathcal{NP}$-hard [15], finding an optimal VN embedding using MIP is computationally intractable. Consequently, we relax the integer constraints to obtain a linear programming formulation that can be solved in polynomial time. We then use deterministic (in D-ViNE) and randomized (in R-ViNE) rounding techniques on the solution of the linear program to approximate the values of the binary variables in the original MIP. Once all the virtual nodes have been mapped, we use the multicommodity flow algorithm to map the virtual links onto substrate network paths between the mapped virtual nodes [14], [16]. This can also be solved in polynomial time since we assume that path splitting is supported by the substrate network [14].

Our findings from an extensive simulation study on a large mix of VN requests and substrate network topologies indicate that the proposed algorithms outperform their counterparts in terms of acceptance ratio, revenue, cost, and resource utilization.

In addition to the core ViNEYard algorithms (D-ViNE and R-ViNE), we present a generalized window-based VN embedding mechanism (WiNE), which allows batch processing of VN requests. We show that the proposed algorithms maintain their dominance in terms of the performance metrics in this generalized setting.

Major contributions of this paper are summarized in the following

- D-ViNE and R-ViNE are two rounding-based VN embedding algorithms that increase the VN request acceptance ratio and InP revenue without incurring additional cost by leveraging coordinated node and link mapping.
- D-ViNE-LB and R-ViNE-LB are extensions to D-ViNE and R-ViNE that focus on balancing load across substrate resources to improve the acceptance ratio, often causing inflated link utilization across the substrate network.
- WiNE is a generalized window-based VN embedding mechanism for equipping any existing online VN embedding algorithm with lookahead capabilities.
- The ViNEYard model presents a flexible and extensible mathematical programming formulation of the VN embedding problem.

The rest of this paper is organized as follows. Section II formalizes the network model and the VN embedding problem. In Section III, we provide the optimal MIP formulation for the VN embedding problem using substrate network augmentation. Section IV relaxes the MIP formulation to obtain a linear program and presents D-ViNE and R-ViNE using deterministic and randomized rounding techniques. In Section V, we present WiNE—a generalized window-based VN embedding mechanism. Section VI presents simulation results that evaluate the proposed algorithms. We discuss limitations and possible workarounds in Section VII, summarize related work in Section VIII, and conclude in Section IX.

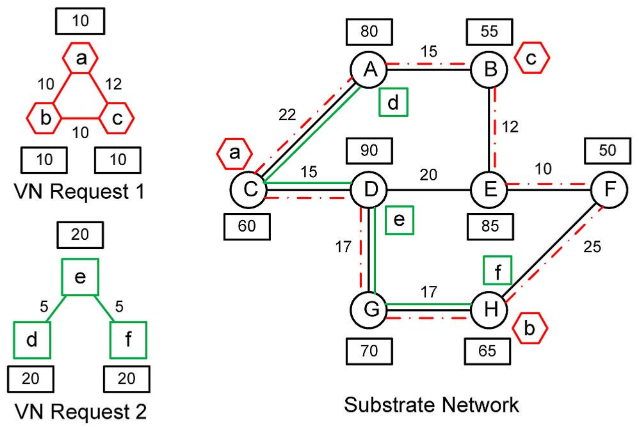


Fig. 1. Mapping of VN requests onto a shared substrate network.

II. Network Model and Problem Description

A. *Substrate Network*

We model the substrate network as a weighted undirected graph and denote it by $G^{\mathrm{S}} = (N^{\mathrm{S}}, E^{\mathrm{S}})$, where N^{S} is the set of substrate nodes and E^{S} is the set of substrate links. Each substrate node $n^{\mathrm{S}} \in N^{\mathrm{S}}$ is associated with the CPU capacity weight value $c(n^{\mathrm{S}})$ and its location $\mathrm{loc}(n^{\mathrm{S}})$ on a globally understood coordinate system. Each substrate link $e^{\mathrm{S}}(i,j) \in E^{\mathrm{S}}$ between two substrate nodes i and j is associated with the bandwidth capacity weight value $b(e^{\mathrm{S}})$ denoting the total amount of bandwidth.

We denote the set of all substrate paths by $\mathcal{P}^{\mathrm{S}}$ and the set of substrate paths from the source node s to the destination node t by $\mathcal{P}^{\mathrm{S}}(s,t)$.

Fig. 1 shows a substrate network, where the numbers over the links represent available bandwidths and the numbers in rectangles represent available CPU resources.

B. *VN Request*

Similar to the substrate network, we model VN requests as weighted undirected graphs and denote a VN request by $G^{\mathrm{V}} = (N^{\mathrm{V}}, E^{\mathrm{V}})$. We express the requirements on virtual nodes and virtual links in terms of the attributes of the nodes and links of the substrate network. Each VN request has an associated nonnegative value D^{V} expressing how far a virtual node $n^{\mathrm{V}} \in N^{\mathrm{V}}$ can be embedded from its preferred location $\mathrm{loc}(n^{\mathrm{V}})$. D^{V} can be expressed naturally in terms of link delay or round-trip time (RTT) from $\mathrm{loc}(n^{\mathrm{V}})$. We assume a landmark-based approach for requesting virtual nodes in our model so that InPs will not have to expose their network topologies[2]. Fig. 1 shows two VN requests with node and link constraints.

[2]Given the secrecy of ISPs in the existing Internet, it is unlikely that InPs in a network virtualization environment will be any more willing to expose their network topologies.

C. Measurement of Substrate Network Resources

In order to quantify the resource usage of the substrate network, we use the notion of *stress*. The substrate node stress $S_N(n^{\mathrm{S}})$ is defined as the total amount of CPU capacity allocated to different virtual nodes hosted on the substrate node $n^{\mathrm{S}} \in N^{\mathrm{S}}$

$$S_N(n^{\mathrm{S}}) = \sum_{n^{\mathrm{V}} \rightarrow n^{\mathrm{S}}} c(n^{\mathrm{V}}) \tag{1}$$

where $x \rightarrow y$ denotes that the virtual node x is hosted on the substrate node y.

Similarly, the substrate link stress $S_E(e^{\mathrm{S}})$ is defined as the total amount of bandwidth reserved for the virtual links whose substrate paths pass through the substrate link $e^{\mathrm{S}} \in E^{\mathrm{S}}$

$$S_E(e^{\mathrm{S}}) = \sum_{e^{\mathrm{V}} \rightarrow e^{\mathrm{S}}} b(e^{\mathrm{V}}) \tag{2}$$

where $x \rightarrow y$ denotes that the substrate path of the virtual link x passes through the substrate link y.

The definitions of *node stress* and *link stress* are similar to that in [13] with the difference that we use the actual amounts of CPU and bandwidth resources used by the embedded VNs to measure substrate resource stresses, not just the number of virtual nodes and links.

The *residual* or the available capacity of a substrate node $R_N(n^{\mathrm{S}})$ is defined as the available CPU capacity of the substrate node $n^{\mathrm{S}} \in N^{\mathrm{S}}$

$$R_N(n^{\mathrm{S}}) = c(n^{\mathrm{S}}) - S_N(n^{\mathrm{S}}).$$

Similarly, the residual capacity of a substrate link $R_E(e^{\mathrm{S}})$ is defined as the total amount of bandwidth available on the substrate link $e^{\mathrm{S}} \in E^{\mathrm{S}}$

$$R_E(e^{\mathrm{S}}) = b(e^{\mathrm{S}}) - S_E(e^{\mathrm{S}}).$$

The available bandwidth capacity of a substrate path $P \in \mathcal{P}^{\mathrm{S}}$ is given by

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

D. VN Assignment

When a VN request arrives, the substrate network has to determine whether to accept the request or not. If the request is accepted, the substrate network then determines a suitable assignment for the VN and allocates network resources on the substrate nodes and paths selected by that assignment. The allocated resources are released once the VN expires.

The assignment of a VN request G^{V} onto the substrate network can be decomposed into two major components.

1) Node Assignment: Each virtual node from the same VN request[3] is assigned to a *different* substrate node by a mapping $\mathcal{M}_N : N^{\mathrm{V}} \rightarrow N^{\mathrm{S}}$ from virtual nodes to substrate nodes such that for all $n^{\mathrm{V}}, m^{\mathrm{V}} \in N^{\mathrm{V}}$

$$\begin{aligned} &\mathcal{M}_N(n^{\mathrm{V}}) \in N^{\mathrm{S}} \\ &\mathcal{M}_N(m^{\mathrm{V}}) = \mathcal{M}_N(n^{\mathrm{V}}), \qquad \text{iff}\, m^{\mathrm{V}} = n^{\mathrm{V}} \end{aligned}$$

subject to

$$c(n^{\mathrm{V}}) \le R_N\left(\mathcal{M}_N(n^{\mathrm{V}})\right) \tag{3a}$$
$$\mathrm{dis}\left(\mathrm{loc}(n^{\mathrm{V}}), \mathrm{loc}\left(\mathcal{M}_N(n^{\mathrm{V}})\right)\right) \le D^{\mathrm{V}} \tag{3b}$$

where $\mathrm{dis}(i,j)$ measures the distance between the locations of two substrate nodes i and j.

In Fig. 1, the first VN request has the node mapping $\{a \rightarrow C, b \rightarrow H, c \rightarrow B\}$, and the second VN request has $\{d \rightarrow A, e \rightarrow D, f \rightarrow H\}$. Note that two virtual nodes b and f from different VN requests are mapped onto the same substrate node H.

2) Link Assignment: Each virtual link is mapped to a substrate path (unsplittable flow) or a set of substrate paths (splittable flow) between the corresponding substrate nodes that host the end virtual nodes of that virtual link. It is defined by a mapping $\mathcal{M}_E : E^{\mathrm{V}} \rightarrow \mathcal{P}^{\mathrm{S}}$ from virtual links to substrate paths such that for all $e^{\mathrm{V}} = (m^{\mathrm{V}}, n^{\mathrm{V}}) \in E^{\mathrm{V}}$

$$\mathcal{M}_E(m^{\mathrm{V}}, n^{\mathrm{V}}) \subseteq \mathcal{P}^{\mathrm{S}}\left(\mathcal{M}_N(m^{\mathrm{V}}), \mathcal{M}_N(n^{\mathrm{V}})\right)$$

subject to

$$\sum_{P \in \mathcal{M}_E(e^{\mathrm{V}})} R_E(P) \ge b(e^{\mathrm{V}}). \tag{4}$$

The first VN request in Fig. 1 has been assigned the link mapping $\{(a,b) \rightarrow \{(C,D),(D,G),(G,H)\}, (a,c) \rightarrow \{(C,A),(A,B)\}, (b,c) \rightarrow \{(H,F),(F,E),(E,B)\}\}$, and the second VN request has the link mapping $\{(d,e) \rightarrow \{(A,C),(C,D)\}, (e,f) \rightarrow \{(D,G),(G,H)\}\}$.

E. Objectives

Our main interest in this paper is to propose online VN embedding algorithms that map multiple VN requests with node and link constraints. We also want to increase revenue and decrease cost of the InP in the long run, in addition to balancing load across the substrate network resources.

Similar to the previous work in [13] and [14], we define the revenue of a VN request as

$$\mathbb{R}(G^{\mathrm{V}}) = \sum_{e^{\mathrm{V}} \in E^{\mathrm{V}}} b(e^{\mathrm{V}}) + \sum_{n^{\mathrm{V}} \in N^{\mathrm{V}}} c(n^{\mathrm{V}}). \tag{5}$$

While revenue gives an insight into how much an InP will gain by accepting a VN request, it is not very useful without knowing the cost the InP will incur for embedding that request. We define the cost of embedding a VN request as the sum of total substrate resources allocated to that VN.

$$\mathbb{C}(G^{\mathrm{V}}) = \sum_{e^{\mathrm{V}} \in E^{\mathrm{V}}} \sum_{e^{\mathrm{S}} \in E^{\mathrm{S}}} \alpha_{e^{\mathrm{S}}} f_{e^{\mathrm{S}}}^{e^{\mathrm{V}}} + \sum_{n^{\mathrm{V}} \in N^{\mathrm{V}}} \beta_{n^{\mathrm{S}}} c(n^{\mathrm{V}}) \tag{6}$$

[3]Even though multiple virtual nodes from different VN requests can be mapped to the same substrate node.

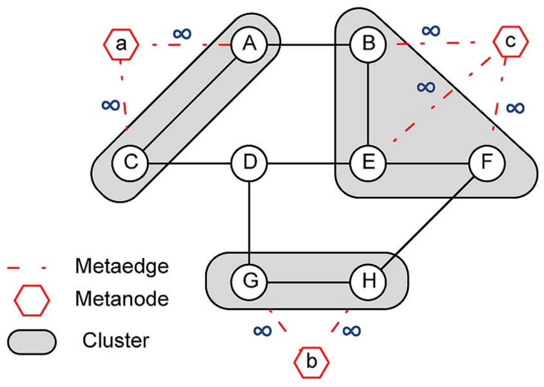


Fig. 2. Construction of an augmented substrate graph with meta nodes and meta edges for a VN request.

where $n^{\mathrm{V}} \to n^{\mathrm{S}}$ and $f_{e^{\mathrm{S}}}^{e^{\mathrm{V}}}$ denotes the total amount of bandwidth allocated on the substrate link e^{S} for virtual link e^{V}. $\alpha_{e^{\mathrm{S}}}$ and $\beta_{n^{\mathrm{S}}}$ are tuning parameters to set relative costs of substrate resources.

III. Mixed Integer Programming Formulation for Optimal VN Embedding

A. Augmented Substrate Graph Construction

In order to coordinate the node mapping and the link mapping phases, the base substrate network must be extended to create an *augmented substrate graph* using the location requirements of virtual nodes as the basis for the extension. Since each $n^{\mathrm{V}} \in N^{\mathrm{V}}$ has an associated constraint $\mathrm{loc}(n^{\mathrm{V}})$ on its possible placement, we can create one *cluster* for each virtual node ($|N^{\mathrm{V}}|$ in total) in the substrate network—each with radius D^{V}. We denote such a cluster by $\Omega(n^{\mathrm{V}})$ and call it the Ω set of the virtual node n^{V}

$$\Omega(n^{\mathrm{V}}) = \left\{ n^{\mathrm{S}} \in N^{\mathrm{S}} \,|\, \mathrm{dis}\left(\mathrm{loc}(n^{\mathrm{V}}), \mathrm{loc}(n^{\mathrm{S}})\right) \le D^{\mathrm{V}} \right\}.$$

In Fig. 2, substrate nodes B, E, and F are within D^{V} distance of the virtual node c, hence $\Omega(c) = \{B, E, F\}$.

For each $n^{\mathrm{V}} \in N^{\mathrm{V}}$, we create a corresponding *meta node* $\mu(n^{\mathrm{V}})$ and connect $\mu(n^{\mathrm{V}})$ to all the substrate nodes belonging to $\Omega(n^{\mathrm{V}})$ using *meta edges* with infinite bandwidth. We will sometimes write the Ω set as $\Omega(m)$ instead of $\Omega(n^{\mathrm{V}})$, where $m = \mu(n^{\mathrm{V}})$. We combine all the meta nodes and meta edges with the substrate graph to create the augmented substrate graph $G^{\mathrm{S}'} = (N^{\mathrm{S}'}, E^{\mathrm{S}'})$, where

$$N^{\mathrm{S}'} = N^{\mathrm{S}} \cup \{\mu(n^{\mathrm{V}}) \,|\, n^{\mathrm{V}} \in N^{\mathrm{V}}\}$$
$$E^{\mathrm{S}'} = E^{\mathrm{S}} \cup \{(\mu(n^{\mathrm{V}}), n^{\mathrm{S}}) \,|\, n^{\mathrm{V}} \in N^{\mathrm{V}}, n^{\mathrm{S}} \in \Omega(n^{\mathrm{V}})\}.$$

An example of augmented graph construction is shown in Fig. 2.

B. MIP Formulation

The VN embedding problem can now be formulated as a mixed integer $|E^{\mathrm{V}}|$-commodity flow problem. We consider each virtual link e_i^{V} $(1 \le i \le |E^{\mathrm{V}}|)$ as a commodity with source and destination nodes s_i and t_i, respectively $(\forall i,\ s_i, t_i \in N^{\mathrm{S}'} \setminus N^{\mathrm{S}})$. In this setting, each flow starts from a meta node and ends in another meta node. By introducing restrictions on the meta edges, each meta node $\mu(n^{\mathrm{V}})$ can be forced to choose only one meta edge to connect itself to an actual substrate node in $\Omega(n^{\mathrm{V}})$. This effectively selects a substrate node for each meta node, i.e., maps the virtual node corresponding to that meta node to a substrate node. At the same time, all the virtual links (i.e., flows) are also mapped efficiently inside the substrate network with the help of path splitting. We present the MIP formulation in the following.

VNE_MIP

Variables:

- f_{uv}^i: A flow variable denoting the total amount of flow from u to v on the substrate edge (u, v) for the ith virtual edge.
- x_{uv}: A binary variable, which has the value "1" if $\sum_i (f_{uv}^i + f_{vu}^i) > 0$; otherwise, it is set to "0."

Objective:

$$\text{minimize} \sum_{uv \in E^{\mathrm{S}}} \frac{\alpha_{uv}}{R_E(u,v) + \delta} \sum_i f_{uv}^i + \sum_{w \in N^{\mathrm{S}}} \frac{\beta_w}{R_N(w) + \delta} \sum_{m \in N^{\mathrm{S}'} \setminus N^{\mathrm{S}}} x_{mw}\, c(m). \quad (7)$$

Constraints:

— Capacity Constraints:

$$\sum_i \left(f_{uv}^i + f_{vu}^i\right) \le R_E(u,v) x_{uv} \qquad \forall u, v \in N^{\mathrm{S}'} \quad (8)$$

$$R_N(w) \ge x_{mw} c(m) \qquad \forall m \in N^{\mathrm{S}'} \setminus N^{\mathrm{S}}, \forall w \in N^{\mathrm{S}}. \quad (9)$$

— Flow-Related Constraints:

$$\sum_{w \in N^{\mathrm{S}'}} f_{uw}^i - \sum_{w \in N^{\mathrm{S}'}} f_{wu}^i = 0 \qquad \forall i, \forall u \in N^{\mathrm{S}'} \setminus \{s_i, t_i\} \quad (10)$$

$$\sum_{w \in N^{\mathrm{S}'}} f_{s_i w}^i - \sum_{w \in N^{\mathrm{S}'}} f_{w s_i}^i = b\left(e_i^{\mathrm{V}}\right) \qquad \forall i \quad (11)$$

$$\sum_{w \in N^{\mathrm{S}'}} f_{t_i w}^i - \sum_{w \in N^{\mathrm{S}'}} f_{w t_i}^i = -b\left(e_i^{\mathrm{V}}\right) \qquad \forall i. \quad (12)$$

— Meta and Binary Constraints:

$$\sum_{w \in \Omega(m)} x_{mw} = 1 \qquad \forall m \in N^{\mathrm{S}'} \setminus N^{\mathrm{S}} \quad (13)$$

$$\sum_{m \in N^{\mathrm{S}'} \setminus N^{\mathrm{S}}} x_{mw} \le 1 \qquad \forall w \in N^{\mathrm{S}} \quad (14)$$

$$x_{uv} = x_{vu} \qquad \forall u, v \in N^{\mathrm{S}'}. \quad (15)$$

— Domain Constraints:

$$f_{uv}^i \ge 0 \qquad \forall u, v \in N^{\mathrm{S}'} \quad (16)$$

$$x_{uv} \in \{0, 1\} \qquad \forall u, v \in N^{\mathrm{S}'}. \quad (17)$$

Remarks:

- The objective function of the MIP (7) tries to minimize the cost of embedding the VN request as well as balance the load. By dividing the cost with the residual capacity, it is ensured that the resources with more residual capacities are preferred over the resources with less residual capacities. In this paper, we use $\alpha_{uv} \in \{1, R_E(u,v)\}$ and $\beta_{uv} \in \{1, R_N(u,v)\}$ to control the importance of load balancing while embedding a request. $\delta \to 0$ is a small positive constant to avoid dividing by zero in computing the objective function.

- Constraint sets (8) and (9) enforce the capacity bounds of substrate nodes and edges. Summing up f^i_{uv} and f^i_{vu} in (8) ensures that the summation of flows on both directions of the undirected edge (u,v) remains within its available bandwidth.
- Constraint sets (10)–(12) refer to the flow conservation conditions, which denote that the net flow to a node must be zero except for source nodes s_i and sink nodes t_i.
- Constraint sets (13) and (14) are related to the augmented portion of the substrate graph. Constraint set (13) makes sure that only one substrate node is selected for each meta node, whereas constraint set (14) ensures that no more than one meta node is placed on a substrate node.
- Constraint set (15) together with (4) ensures that x_{uv} is set whenever there is any flow in either direction of the substrate edge (u,v).
- Finally, constraint sets (16) and (17) denote the real and binary domain constraints on the variables f^i_{uv} and x_{uv}, respectively.

IV. LP Relaxation and Rounding-Based Algorithms

Since solving an MIP is known to be computationally intractable [15], simultaneous node and link embedding using **VNE_MIP** is practically infeasible. Hence, we relax the integer constraints (17) of the MIP and obtain the following linear program (**VNE_LP_RELAX**). Once we have the LP solution, we use deterministic and randomized rounding techniques to obtain integer values for x and embed VN requests.

VNE_LP_RELAX

Objective:

$$\text{minimize} \sum_{uv\in E^{\mathrm{S}}} \frac{\alpha_{uv}}{R_E(u,v)+\delta} \sum_i f^i_{uv} + \sum_{w\in N^{\mathrm{S}}} \frac{\beta_w}{R_N(w)+\delta} \sum_{m\in N^{\mathrm{S}'}\setminus N^{\mathrm{S}}} x_{mw}\, c(m). \quad (18)$$

Constraints:

— Domain Constraints:

$$0 \le x_{uv} \le 1 \qquad \forall u,v \in N^{\mathrm{S}'}. \quad (19)$$

Remarks:

- The domain constraint set (19) on the x_{uv} variables has been relaxed.
- The rest of the constraints are the same as in **VNE_MIP**.

A. Deterministic Rounding-Based Virtual Network Embedding Algorithm (D-ViNE)

D-ViNE (Fig. 3) takes online VN requests as inputs and maps them onto the substrate network one at a time. Since the integer domain constraints (17) have already been relaxed, we no longer get integer values for x from the solution of **VNE_LP_RELAX**. Instead, we employ deterministic rounding technique. We introduce $\varphi : N^S \to \{0,1\}$, which is initially set to zero for all $n^{\mathrm{S}} \in N^{\mathrm{S}}$, signifying that all the substrate nodes are initially unused. Whenever a virtual node is mapped to a particular physical

```
1: procedure D-ViNE($G^V = (N^V, E^V)$)
2:     Create augmented substrate graph $G^{S'} = \left(N^{S'}, E^{S'}\right)$
3:     Solve VNE_LP_RELAX
4:     for all $n^S \in N^S$ do
5:         $\varphi(n^S) \leftarrow 0$
6:     end for
7:     for all $n \in N^V$ do
8:         if $\Omega(n) \setminus \{n^S \in N^S | \varphi(n^S) = 1\} = \emptyset$ then
9:             Reject $G^V$
10:            return
11:        end if
12:        for all $z \in \Omega(n)$ do
13:            $p_z \leftarrow (\sum_i f^i_{\mu(n)z} + f^i_{z\mu(n)}) x_{\mu(n)z}$
14:        end for
15:        Let $z_{max} = \arg\max_{z\in\Omega(n)}\{p_z | \varphi(z) = 0\}$
16:        set $\mathcal{M}_N(n) \leftarrow z_{max}$
17:        $\varphi(z_{max}) \leftarrow 1$
18:    end for
19:    Solve MCF to map virtual links
20:    if MCF succeeded then
21:        Update residual capacities of substrate resources
22:    else
23:        Reject $G^V$
24:    end if
25: end procedure
```

Fig. 3. D-ViNE: Deterministic rounding-based virtual network embedding algorithm.

node n^{S}, we set $\varphi(n^{\mathrm{S}})$ to 1 to ensure that no substrate node is used twice for the same VN request.

1) Description and Discussion: The procedure begins by creating an augmented substrate graph $G^{\mathrm{S}'} = (N^{\mathrm{S}'}, E^{\mathrm{S}'})$ for the VN request $G^{\mathrm{V}} = (N^{\mathrm{V}}, E^{\mathrm{V}})$ using the augmentation method described in Section III-A. Next, it solves **VNE_LP_RELAX** to get a fractional solution that is at least as good as the integer solution of **VNE_MIP**. For each virtual node, D-ViNE first checks whether there are any unmapped substrate nodes within its Ω set. If any of the Ω sets is empty, D-ViNE stops the embedding process immediately and rejects the VN request. Otherwise, the deterministic rounding procedure is initiated in Line 12.

For each virtual node n, D-ViNE calculates a value p_z for each substrate node $z \in \Omega(n)$ in its cluster. p_z is calculated as the product of the value $x_{\mu(n)z}$ and the total flow passing through the meta edge $\mu(n)z$ in both directions. The intuition behind using the product instead of just $x_{\mu(n)z}$ is as follows.

In the MIP solution, x_{uv} is set to binary values based on the presence of flows in either direction in the edge (u,v). When the binary constraint x is relaxed, one might expect that the fractional values of x_{uv} would also be proportional to the total flow in the edge (u,v). However, during the LP relaxation process, the correlation between the flow variable f and the binary variable x is lost. It is because a linear program tries to optimize the objective function without violating the constraints; it does not care about the values as long as they are within their permitted domains. As a result, in the relaxed linear program, it is possible that the f values are very high and the corresponding

x values are very low, or vice versa. Multiplying the f and x values thwarts the possibility of selecting a substrate node based on high x value but very low f value on its corresponding meta edge, and vice versa. The ones that have better values for both the variables f and x are more likely to be in the solution of the MIP than others. D-ViNE maps the virtual node n onto the unmapped substrate node z (i.e., $\varphi(z) = 0$) with the highest p_z value, breaking ties arbitrarily.

Once all the virtual nodes have been mapped to different substrate nodes, D-ViNE applies the multicommodity flow (MCF) algorithm to map the virtual links in E^{V} onto substrate paths. One can also use k-shortest path algorithms [17] when path splitting is not supported by the substrate network. Finally, D-ViNE updates the residual capacities of substrate nodes and links to prepare for the next request.

2) Time Complexity: An important aspect of D-ViNE is that the multicommodity flow algorithm is executed twice: first, during the node mapping phase (since **VNE_LP_RELAX** is a linear programming relaxation of the original mixed integer multicommodity flow problem), and second, during the link mapping phase. Since we can solve linear programs in polynomial time using either the ellipsoid algorithm or Karmarkar's interior point algorithm for linear programming [15], D-ViNE is a polynomial-time algorithm.

We can now formally express the time complexity of D-ViNE. We can create the augmented substrate graph in Line 2 in time proportional to $\mathcal{O}(|N^{\mathrm{V}}|)$. **VNE_LP_RELAX** in Line 3 can be solved in time $\mathcal{O}((|E^{\mathrm{S}'}|(1 + |E^{\mathrm{V}}|))^{3.5}L^2 \ln L \ln\ln L)$, where L denotes the desired input precision in terms of the number of bits required to specify inputs to the linear program **VNE_LP_RELAX** [18]. The for loop in Line 7 runs in time $\mathcal{O}(|N^{\mathrm{V}}|)$. Finally, the MCF (Line 19) runs in time $\mathcal{O}((|E^{\mathrm{S}}||E^{\mathrm{V}}|)^{3.5}L^2 \ln L \ln\ln L)$. The dominating factor in the running time of D-ViNE is the time to solve **VNE_LP_RELAX**, which is the overall time complexity of D-ViNE.

B. Randomized Rounding-Based Virtual Network Embedding Algorithm (R-ViNE)

R-ViNE (Fig. 4) is quite similar to D-ViNE except that it uses randomized rounding instead of deterministic rounding. Once the p_z values are calculated as in D-ViNE, R-ViNE normalizes those values to restrict them within the 0 to 1 range. The normalized values for each $z \in \Omega(n)$ correspond to the probabilities of n being mapped to z by the optimal MIP. R-ViNE selects a substrate node $z \in \Omega(n)$ to map a virtual node n with probability p_z. The remainder of this algorithm is similar to its deterministic counterpart, and it also runs in polynomial time.

```
1:  procedure R-ViNE($G^V = (N^V, E^V)$)
2:      Create augmented substrate graph $G^{S'} = \left(N^{S'}, E^{S'}\right)$
3:      Solve VNE_LP_RELAX
4:      for all $n^S \in N^S$ do
5:          $\varphi(n^S) \leftarrow 0$
6:      end for
7:      for all $n \in N^V$ do
8:          if $\Omega(n) \setminus \{n^S \in N^S | \varphi(n^S) = 1\} = \emptyset$ then
9:              Reject $G^V$
10:             return
11:         end if
12:         for all $z \in \Omega(n)$ do
13:             $p_z \leftarrow (\sum_i f^i_{\mu(n)z} + f^i_{z\mu(n)}) x_{\mu(n)z}$
14:         end for
15:         $p_{sum} \leftarrow \sum_{z \in \Omega(n)} p_z$
16:         for all $z \in \Omega(n)$ do
17:             $p_z \leftarrow p_z / p_{sum}$
18:         end for
19:         set $\mathcal{M}_N(n) \leftarrow z$ with probability $p_z$
20:         $\varphi(z) \leftarrow 1$ with probability $p_z$
21:     end for
22:     Solve MCF to map virtual links
23:     if MCF succeeded then
24:         Update residual capacities of substrate resources
25:     else
26:         Reject $G^V$
27:     end if
28: end procedure
```

Fig. 4. R-ViNE: Randomized rounding-based virtual network embedding algorithm.

V. VN Embedding With Lookahead

In a realistic network virtualization scenario, VN requests may not always arrive one after another in regular intervals. This can occur, for example, if multiple SPs request VNs almost at the same time. In that case, the InP can queue all the requests and then optimize resource allocation by processing them together. In this section, we study the effect of lookahead for VN embedding algorithms. The pertinent issue is to analyze performance gains that can be achieved when the VN embedding algorithms not only know the current VN request to be served, but also a finite number of future requests.

The ViNEYard algorithms have been developed for the pure online intradomain VN embedding problem. However, VN requests with longer lifetimes can be expected to allow certain waiting periods before they must be processed. Such a model will necessitate designing of algorithms with *lookahead* capabilities. We can extend the ViNEYard algorithms with similar capabilities using window-based batch processing of VN requests. Our approach will be to store incoming VN requests for a certain period of time depending on their permissible waiting periods and then process them in batches based on some priority metric like revenue.

WiNE is a generalized mechanism for extending a pure online VN embedding algorithm with lookahead capabilities. It discretizes time into consecutive windows, and instead of making individual embedding decisions for each VN arrival, it collects the requests to batch process them at the end of a window period. In this setting, each VN request comes with an additional piece of information known as the *maximum waiting period* $t_{\mathrm{w}}(\cdot)$, which sets a deadline on how long WiNE can defer making an embedding decision for that VN request. The basic workflow of WiNE is presented using pseudocode in Fig. 5 and described in the following.

WiNE collects the arriving VN requests within the current window period in $\mathcal{W}_{\mathrm{cur}}$. At the end of the window, all the re-

```
1:  procedure WiNE(Online VN embedding algorithm $\mathcal{A}$)
2:      $\mathcal{W}_{old} \leftarrow \{\}$
3:      loop
4:          $\mathcal{W}_{cur} \leftarrow \mathcal{W}_{old}$
5:          $\mathcal{W}_{old} \leftarrow \{\}$
6:          repeat
7:              Add new VN $G^V = (N^V, E^V)$ to $\mathcal{W}_{cur}$
8:          until Current window expires
9:          Sort $\mathcal{W}_{cur}$ according to revenue
10:         $T_{cur} \leftarrow$ Current time
11:         for all $G^V \in \mathcal{W}_{cur}$ do
12:             if $t_w(G^V)$ expired before $T_{cur}$ then
13:                 Reject $G^V$
14:             else
15:                 Embed $G^V$ using $\mathcal{A}$
16:                 if $\mathcal{A}$ failed to embed $G^V$ then
17:                     Add $G^V$ to $\mathcal{W}_{old}$
18:                 end if
19:             end if
20:         end for
21:     end loop
22: end procedure
```

Fig. 5. WiNE: A generalized window-based mechanism for virtual network embedding.

quests in $\mathcal{W}_{\text{cur}}$ are sorted based on their revenues, so that the VN with the maximum revenue is considered first for embedding. Then, for each VN in $\mathcal{W}_{\text{cur}}$, WiNE first checks if its maximum waiting period has already expired or not. If yes, WiNE rejects the VN request outright. Otherwise, it attempts to embed the request using any VN embedding algorithm under consideration (e.g., D-ViNE or R-ViNE). If the embedding is not successful, the VN request is postponed and added to $\mathcal{W}_{\text{old}}$ to be considered in the next window period ($\mathcal{W}_{\text{cur}}$ is set to $\mathcal{W}_{\text{old}}$ at the start of each window period, and $\mathcal{W}_{\text{old}}$ is initialized to be an empty collection).

VI. Performance Evaluation

In this section, we describe the simulation environment followed by the main evaluation results. Our evaluation focused primarily on quantifying the advantage of coordinating node and link mapping phases in terms of acceptance ratio, revenue, and cost. We examined the competitive advantages of the proposed algorithms against selected existing ones by varying the VN request arrival rate on different network topologies. We also evaluated the effect of lookahead by varying window sizes and maximum waiting periods of VN requests.

A. Simulation Environment

We have implemented a discrete event simulator to evaluate the proposed algorithms and used the open source linear programming toolkit `glpk` [19] to solve **VNE_LP_RELAX** and MCFs.

Since network virtualization is an emerging field, the actual characteristics of substrate networks and VN requests are still not well understood. Therefore, we used synthetic network topologies to evaluate the proposed algorithms.

Substrate network topologies in our experiments were randomly generated with 50 nodes using the GT-ITM tool [20] in (25×25) grids.[4] Each pair of substrate nodes were randomly connected with probability 0.5. The CPU and bandwidth resources of the substrate nodes and links were real numbers uniformly distributed between 50 and 100.

We assumed that VN requests arrived in a Poisson process and evaluated the algorithms by varying arrival rates from four VNs per 100 time units to eight VNs per 100 time units. Each VN request had an exponentially distributed lifetime with an average of $\mu = 1000$ time units. In each VN request, the number of virtual nodes was randomly determined by a uniform distribution between 2 and 10, following similar setups in the existing work [13], [14]. The average VN connectivity was fixed at 50%. The CPU requirements of the virtual nodes were uniformly distributed between 0 and 20, and the bandwidth requirements of the virtual links were uniformly distributed between 0 and 50. Virtual nodes were also located on (25×25) grids.

We ran each simulation for 50 000 time units and used truncation for initial transient removal [21] to present steady-state performance throughout the rest of this section.

B. Performance Metrics

We used the following five metrics in our evaluations to measure the performance of our algorithms against the existing ones.

1) *Acceptance ratio:* The acceptance ratio of an algorithm measures the percentage of total VN requests accepted by that algorithm over a given period. While it gives a sense of how well an algorithm is performing, it cannot completely capture the performance when the ultimate goal of an InP is to increase its revenue. An algorithm can accept many smaller or less profitable VN requests to increase this ratio without actually maximizing the overall revenue and leaving the resources underutilized.
2) *Generated revenue ($\mathbb{R}$):* We also measured the generated revenue [defined in (5)] of an embedding algorithm over time. An algorithm can be considered to be performing better than its counterparts when it generates more revenue in addition to a higher acceptance ratio.
3) *Provisioning cost ($\mathbb{C}$):* We measured the cost [defined in (6)] that an algorithm incurs in order to embed a particular VN request. This is particularly useful for calculating the cost–revenue ratio of an embedding, which can later be used for admission control purposes.
4) *Average node utilization:* The average node utilization of the substrate network is measured by averaging the stress [defined in (1)] of all the substrate nodes.
5) *Average link utilization:* Similarly, the average link utilization is the average of the link stresses [defined in (2)] of all the substrate links.

C. Compared Algorithms

We compared six algorithms that combine different node mapping and link mapping strategies including our contributions and algorithms from previous research [13], [14] modified to fit into our model (i.e., no reassignment). Notations for

[4]For any substrate node n^S, $\text{loc}(n^S)$ is a point on the xy plane with a coordinate $(x, y) \in \{\{0, 1, \dots, 24\} \times \{0, 1, \dots, 24\}\}$.

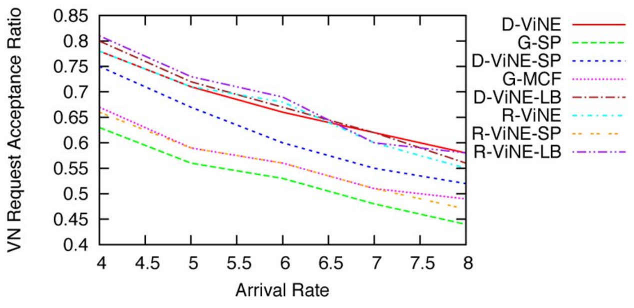


Fig. 6. VN request acceptance ratio.

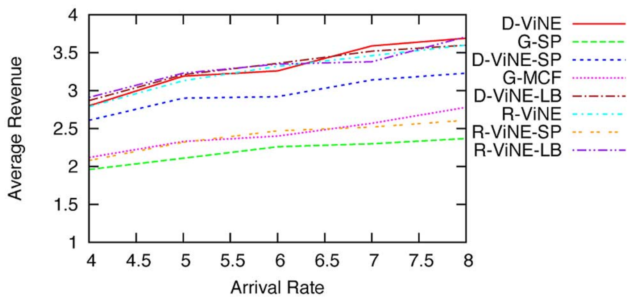


Fig. 7. Time average of generated revenue.

TABLE I
COMPARED ALGORITHMS

Notation	Algorithm Description
D-ViNE	Deterministic node mapping with MCF link mapping
R-ViNE	Randomized node mapping with MCF link mapping
G-SP [13]	Greedy node mapping with shortest path link mapping
G-MCF [14]	Greedy node mapping with MCF link mapping
ViNE-LB	Deterministic or randomized node mapping with MCF link mapping, where $\alpha_{uv} = \beta_w = 1, \forall u, v, w \in N^S$
ViNE-SP	Deterministic or randomized node mapping with shortest path link mapping

different algorithms are enumerated in Table I along with short descriptions of their characteristics. We have included ViNE-SP algorithms for completeness.

D. Comparative Performance

We evaluated and compared the algorithms in Table I by changing the VN request arrival rates from four requests per 100 time units to eight requests per 100 time units while keeping the average VN lifetime fixed. Our key observations for any fixed arrival rate are summarized in the following.

1) Coordinated node and link mapping leads to a higher acceptance ratio and larger revenue. Figs. 6 and 7 depict that the proposed algorithms led to better acceptance ratio as well as higher revenue than the existing algorithms (G-SP and G-MCF) through coordinated node and link mapping. Higher revenue along with better acceptance ratio imply that the proposed algorithms actually embedded VN requests that generate more revenue, instead of embedding smaller VN requests just to increase the acceptance ratio.

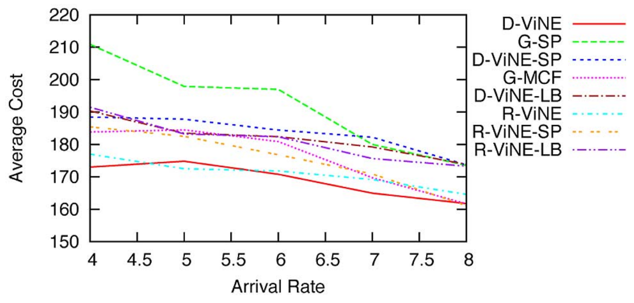


Fig. 8. Average cost of accepting VN requests.

2) Load balancing further increases the acceptance ratio and the revenue. From Figs. 6 and 7, we observe that both D-ViNE-LB and R-ViNE-LB generated more revenue and accepted more VN requests than D-ViNE and R-ViNE, respectively. In D-ViNE-LB and R-ViNE-LB, the value of the objective function (18) of **VNE_LP_RELAX** depends on the residual capacity of the network resources in addition to the provisioning cost (α and β values are set to 1 here). The lower the residual capacity of a particular node or link, the higher the value of the objective function. As a result, they tried to avoid highly utilized nodes and links as much as possible, leaving those critical resources available for future VN requests.

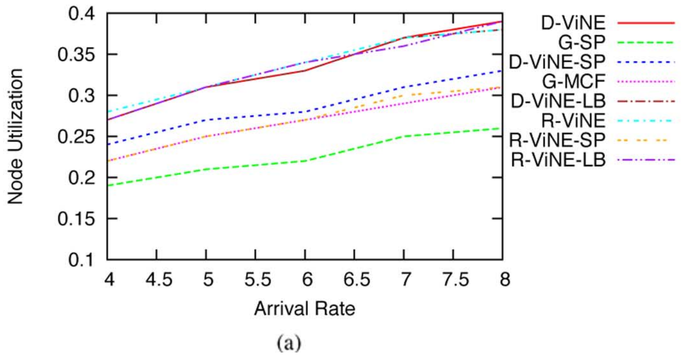
3) Randomization is often as effective as a deterministic solution. It is well established in the algorithm design literature that randomization allows efficient solutions to many intractable problems in polynomial time with low probability of error. Our experiments showed that the randomized version of the proposed VN embedding algorithms performed similar to, and often better than, their deterministic counterparts in terms of acceptance ratio and revenue generation (Figs. 6 and 7).

Additionally, for networks with large numbers of nodes, randomization has been shown to be effective for load balancing [22]. This phenomena can also be observed in our experiments since R-ViNE often performed similarly to D-ViNE-LB, while R-ViNE-LB further improved the performance.

4) Load balancing slightly increases the average provisioning cost. Although load balancing increases revenue and acceptance ratio by avoiding highly utilized resources, it runs the risk of increasing the average provisioning cost as shown in Fig. 8. While trying to avoid highly utilized resources, D-ViNE-LB and R-ViNE-LB might embed virtual links on longer substrate paths resulting in slightly higher average provisioning cost in the long run.

5) Coordination increases resource utilization. Fig. 9(a) and (b) depicts the average utilization of substrate nodes and substrate links for different VN embedding algorithms. Since ViNE-LB algorithms had the highest acceptance ratios, they also demonstrated the highest node and link utilizations.

However, ViNE-LB algorithms had a relatively higher increase in link utilization over their counterparts than in node utilization. We believe that the reason behind this is their distributive nature. In order to avoid links with lower residual capacities, D-ViNE-LB and R-ViNE-LB used longer paths containing more substrate links with higher residual capacities to embed virtual links.



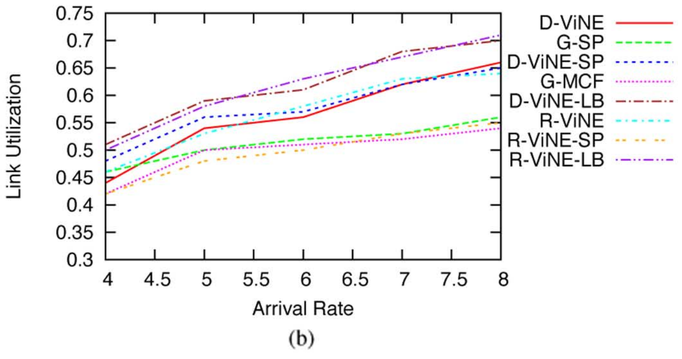


Fig. 9. Average resource utilization in the substrate network for varying VN request arrival rates. (a) Average node utilization. (b) Average link utilization.

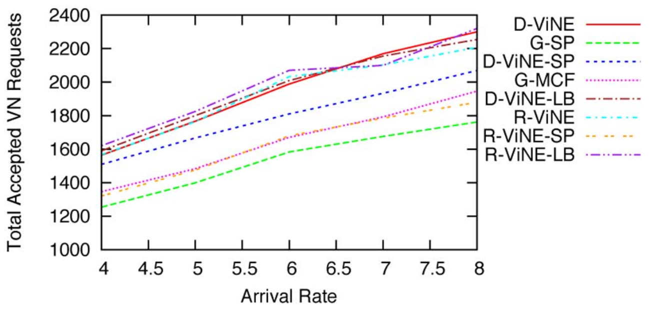


Fig. 10. Total number of accepted VN requests.

TABLE II
COMPARATIVE PERFORMANCE ON HUB-AND-SPOKE TOPOLOGIES

	Acceptance Ratio	Revenue	Cost	Node Util.	Link Util.
D-ViNE	0.756	2.672	167.084	0.284	0.449
G-SP	0.613	1.917	201.611	0.195	0.451
G-MCF	0.725	2.496	183.658	0.239	0.482
D-ViNE-LB	0.772	2.702	175.493	0.283	0.495
R-ViNE	0.745	2.622	169.057	0.284	0.463
R-ViNE-LB	0.796	2.748	182.673	0.271	0.499

E. Effect of Increasing Load on VN Embedding Algorithms

We summarize the behavior of the proposed algorithms under increasing load (expressed in terms of arrival rate) in the following.

1) Dominance of the proposed algorithms is not diminished by load. It is evident from Figs. 6–9 that the proposed algorithms maintained their relative superiority in terms of acceptance ratio, revenue, cost, and average resource utilization with increasing load. As before, D-ViNE and R-ViNE maintained a close resemblance, while D-ViNE-LB and R-ViNE-LB accrued more revenue without any significant changes with load.

2) Increasing load leads to slightly better cost–revenue ratio. With increasing load, all the algorithms achieved better cost–revenue ratios (Figs. 7 and 8). However, the proposed algorithms had steeper decreases in their average costs with similar increases in their revenues, resulting in higher cost–revenue improvements than the existing algorithms.

3) Relative link utilizations of the ViNE-LB algorithms worsen with load. As the arrival rate increased from four to eight VN requests per 100 time units, average link utilizations of D-ViNE-LB and R-ViNE-LB increased more quickly than the total number of VN requests accepted by them in comparison to D-ViNE and R-ViNE.

For example, in terms of the total number of accepted VN requests, D-ViNE-LB's advantage over D-ViNE diminished when the arrival rate was increased from four to eight requests in 100 time units (Fig. 10), while the extra link resource usage increased [Fig. 9(b)]. This negative correlation suggests that D-ViNE-LB is susceptible to running out of resources faster than D-ViNE when the load (i.e., the arrival rate) is too high.

F. Performance on Specific VN Topologies

So far, we have focused only on random VN request topologies. However, some classes of topologies can be expected to be more prevalent than others due to their use in popular applications. For example, hub-and-spoke topologies are commonly used to connect distributed sites to a centralized server (e.g., in content distribution networks).

In this section, we evaluate the performance of the proposed algorithms on two specific classes of topologies: hub-and-spoke and full mesh. We used the same simulation settings described in Section VI-A for this set of experiments. None of the algorithms included any topology-specific modifications.

1) Hub-and-Spoke Topologies: We ensured that all the VN requests had hub-and-spoke topologies instead of random graphs. Table II summarizes the results of the compared algorithms for an arrival rate of four VNs per 100 time units. As seen in Table II, relative performance of the compared algorithms were unchanged for hub-and-spoke topologies. Note that relevant observations for VN requests with random topologies also held true in this case.

2) Mesh Topologies: In this case, we made sure that all the VN requests form full mesh topologies. Simulation results for the unmodified algorithms in steady states are summarized in Table III for an arrival rate of four VNs per 100 time units. The most noticeable change in this case was the overall performance degradation across the board. This was expected because the natural dense formation of mesh topologies often called for more resources than the substrate network could provide. However, relative performance of the compared algorithms were mostly unchanged.

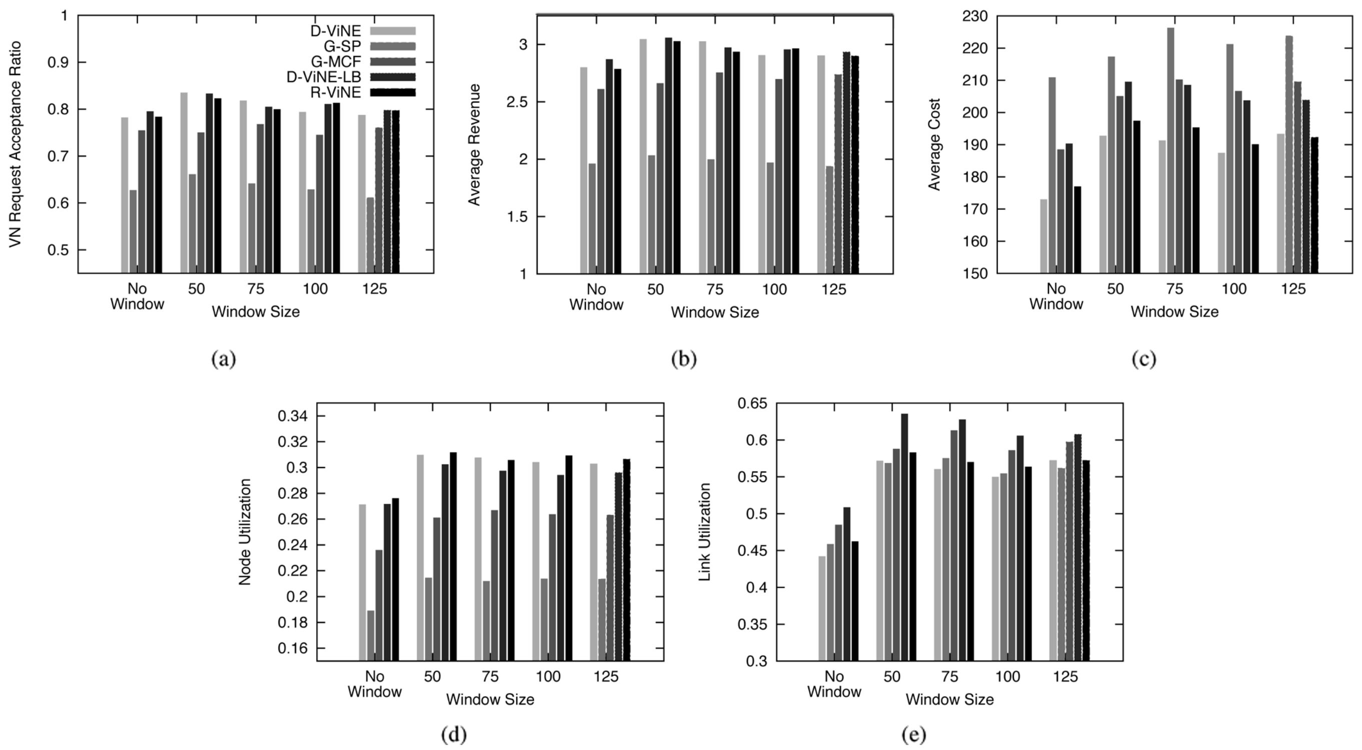


Fig. 11. Effect of lookahead extent on compared VN embedding algorithms in terms of (a) VN request acceptance ratio, (b) revenue, (c) provisioning cost, (d) average node utilization, and (e) average link utilization. Maximum waiting period of each VN request is fixed at 1/2 its lifetime.

TABLE III
COMPARATIVE PERFORMANCE ON MESH TOPOLOGIES

	Acceptance Ratio	**Revenue**	**Cost**	**Node Util.**	**Link Util.**
D-ViNE	0.592	2.381	207.976	0.202	0.488
G-SP	0.483	1.633	235.847	0.135	0.444
G-MCF	0.517	2.061	227.605	0.154	0.467
D-ViNE-LB	0.601	2.481	224.087	0.203	0.539
R-ViNE	0.548	2.178	205.749	0.189	0.463
R-ViNE-LB	0.604	2.471	225.373	0.198	0.531

G. *Effect of Lookahead on VN Embedding*

This section evaluates the performance of the algorithms in the generalized window-based VN embedding setting introduced in Section V. We evaluated the proposed algorithms by varying the window size and the maximum waiting period.

In Fig. 11, we present the performance metrics for window sizes 50, 75, 100, and 125 time units, while keeping the maximum waiting period of a VN request fixed at 1/2 of its lifetime. In Fig. 12, we kept the window size fixed at 50 time units and varied the maximum waiting period of a VN request among 1/2, 1/4, 1/5, 1/8, and 1/10 times of its intended lifetime. In both Figs. 11 and 12, we present the results from experiments without any lookahead for reference. Key observations from our simulations are summarized in the following.

1) Lookahead does not change the ranking of the algorithms. The ViNEYard algorithms maintained their relative superiority in terms of all the performance metrics. This was expected because WiNE does not affect the order of arrival of the VN requests, and a VN request that would have been accepted in the online version will always be accepted in WiNE unless there is a more profitable alternative.

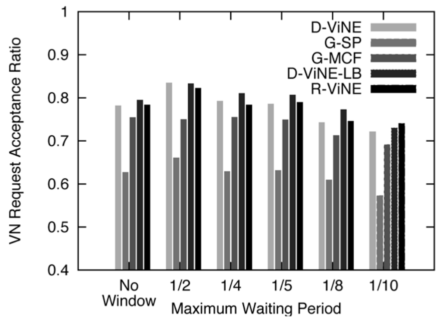
2) Lookahead increases the acceptance ratio. We also noticed that lookahead resulted in increase of the acceptance ratios and the revenues of all the algorithms. This was also expected because a window-based algorithm allows an initially rejected VN request with sufficient waiting period to be embedded in subsequent window periods.

3) Finding an optimal window size is nontrivial. We observed that finding the optimal window size (the one that will maximize the performance metrics) depended on the maximum waiting period. For the simulation results presented in Figs. 11 and 12, the optimal window size for the maximum waiting period of 1/2 of VN lifetimes was 50 time units, and vice versa. However, it can vary for different maximum waiting periods.

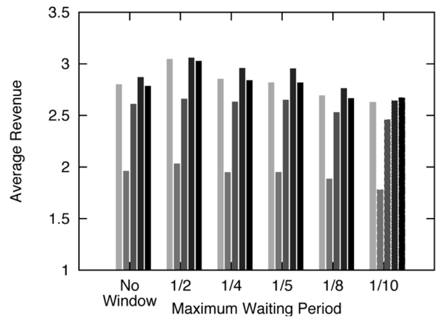
4) Selecting an appropriate maximum waiting period is a tradeoff. We observe in Fig. 12 that for a window period of 50 time units, the highest acceptance ratio was achieved when all the VN requests intended to wait as long as half their lifetimes. If VN requests do not wait long enough, the overall acceptance ratio suffers because many VN requests leave the system even before they are processed. In a practical scenario, however, waiting 1/2 of a VN's lifetime would be too long for most VN requests. Instead, a tradeoff must be made between a longer waiting period and the increased risk of not getting embedded. For Fig. 12, this tradeoff point would be 1/4 or at least 1/5 of lifetime.

VII. DISCUSSION

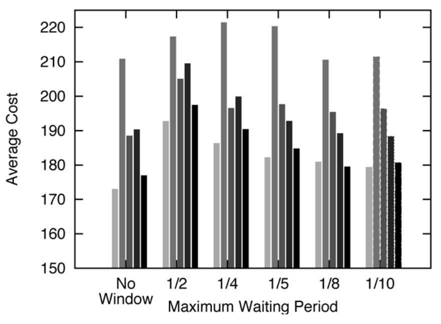
This section discusses some of the major challenges toward theoretical analysis and efficient implementation of the ViNEYard algorithms and considers possible solutions.



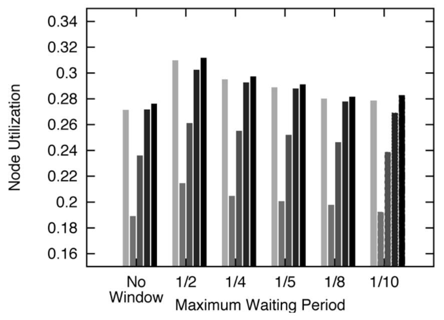
(a)



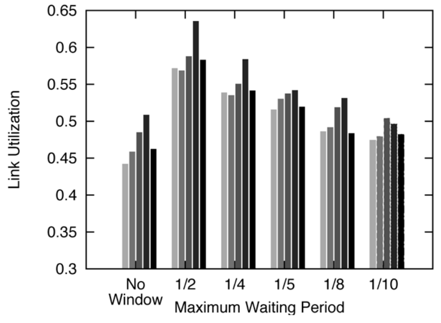
(b)



(c)



(d)



(e)

Fig. 12. Effect of maximum waiting period on compared VN embedding algorithms in terms of (a) VN request acceptance ratio, (b) revenue, (c) provisioning cost, (d) average node utilization, and (e) average link utilization. Window size is fixed at 50 time units.

A. Execution Time

While the ViNEYard algorithms outperform their counterparts across a large mix of VN requests, they do fall short in terms of their running times. This is expected because they need to solve two linear programs to embed a VN request, whereas G-MCF requires solving only one during embedding links, and G-SP requires none. On average, to embed each VN request across experiments, G-SP took 35 ms, G-MCF 417 ms, and D-ViNE, R-ViNE, ViNE-LB, and ViNE-SP algorithms took 1.413, 1.483, 1.656, and 1.014 s, respectively. Note that the execution times reported here depend on the linear program solver (`glpk`) and the machine used for the experiments (Ubuntu 8.04 VM on top of a Windows Vista host).

We also observed that solving the first linear program on the augmented graph took at least twice as much time as solving the second on the actual VN request graph. This is because our technique inflates the substrate graph by introducing meta nodes and meta edges, which in turn increases the number of variables and constraints in the linear program.

B. Challenges in Theoretical Analysis

1) Approximation Ratio Analysis: Both online and offline versions of the VN embedding problem is known to be $\mathcal{NP}$-hard and, therefore, we do not expect to be able to compute an optimal solution efficiently. To the best of our knowledge, for the general version of the problem, theoretical upper or lower bounds do not exist.

Since our algorithms are based on rounding techniques, a standard approximation ratio analysis to bound the performance of the proposed algorithms seems reasonable. The goal is to compare the cost of the optimal MIP and the cost of DViNE. Recall that we have location constraint sets for the virtual nodes in our problem formulation. If we assume that these location sets are disjoint and bounded, then the approximation ratio of D-ViNE can be expressed as a function of the $\Omega(\cdot)$ set given that the $f_{e^{\mathrm{S}}}^{e^{\mathrm{V}}}$ variables remain unchanged during relaxation and rounding (Appendix). However, as explained in Section IV-A, $f_{e^{\mathrm{S}}}^{e^{\mathrm{V}}}$ variables do not retain their original values from the MIP solution after relaxation. If we can modify the algorithms so that the $f_{e^{\mathrm{S}}}^{e^{\mathrm{V}}}$ variables remain unchanged through relaxation and rounding, we can obtain the desired bound.[5]

2) Analysis of Window-Based VN Embedding: It seems logical that a window-based VN embedding algorithm should have better performance compared to a pure online algorithm. However, this can be contradicted with a simple worst-case counterexample [23]. Allowing an online algorithm to see the next k VNs would not yield any advantage in the worst case since any sequence of n VN requests can be replaced by a new nk-size sequence of VN requests, where each VN is replicated k times. However, this example is only a worst-case scenario, and traditional competitive ratio analysis does not deal with average case analysis, which is required in our case to approximate the best window size.

Online algorithms with lookahead have been investigated extensively for paging and bin packing [23], [24]. A similar result for online VN embedding could help us to approximate the optimal window size in the worst case. Competitive analysis with lookahead assumes a finite number of instantaneous online requests, so it is not directly applicable for potentially infinite number of VN requests with waiting periods.

[5]It should be noted that the approximation ratio is only related to the cost metric and not to other metrics like acceptance ratio or node and link utilization. The reason for this is that our MIP formulation only considers cost explicitly in the objective function, whereas acceptance ratio and node and link utilization are computed statistically during our simulation experiments.

3) Stochastic Analysis: A simple M/M/1 queue would not suffice to model the VN embedding problem using queueing theory. Although we have assumed memoryless Poisson arrivals and exponential lifetime for VNs, a simple queue of VNs is not adequate to accurately model the system. The system churn (in terms of arrival and departure of VNs over time) affects each substrate node and link. As a result, an interdependent system of queues, with separate queues for each substrate node and link, would be required to model the system. The complexity of this multidimensional system model prohibits queueing theoretic performance analysis [25]. Similarly, a Markov model of this problem becomes infeasible due to state explosion.

C. Pricing Model

In the absence of a real-world marketplace, it is nontrivial to come up with a pricing model that can capture the interactions between buyers, sellers, and brokers in a network virtualization environment. We have used a simple pricing model based on the existing literature [13], [14], where revenue and cost are linear functions of requested and allocated resources. In practice, there could be many different pricing models, and VN embedding algorithms need to be optimized for each one individually. The ViNEYard algorithms can also incorporate new pricing models by updating their cost and revenue functions. For example, embedding algorithms for a pricing model that prioritizes some critical resources over others must include resource prioritization while making an embedding decision. We provide limited support for prioritization through the parameters α and β and tune them to devise D-ViNE-LB.

Lifetime of a VN request should ideally be an important factor in pricing it. However, in our current model, the revenue and cost functions are independent of VN lifetime. If we include lifetime in the form of multiplying the allocated resources by it, the decision of selecting a VN request from a mixed workload of requests consisting of short-lived expensive VNs and long-lived cheap VNs becomes nontrivial and policy dependent. Similarly, if a VN request has a really long lease time and the market price of resources allocated to that request keep fluctuating over the lease period, inclusion of time in the revenue function requires modeling a full-fledged economic model for the network virtualization environment.

VIII. Related Work

Existing research on assigning virtual private networks (VPNs) in a shared provider topology [26], [27] is similar to the VN embedding problem. However, a typical VPN request consists only of bandwidth requirements that are specified in terms of a traffic matrix without putting constraints on its nodes. Consequently, most VPN design algorithms boil down to finding paths for source/destination pairs. In addition, VPNs usually have standard topologies like full mesh and hub-and-spoke [28]. We have demonstrated through simulation that ViNEYard algorithms can provide improved embedding for standard as well as arbitrary VN request topologies.

VN embedding is also related to the network test-bed mapping problem [29]. The *Assign* algorithm used in the Emulab test bed [29] considers bandwidth constraints alongside constraints on exclusive use of nodes (i.e., different VNs cannot share a substrate node). However, sharing of substrate nodes and links by multiple VNs is one of the core principles of network virtualization [5], and VN embedding algorithms must support these objectives. Emulab itself is aligning its resource mapping policies with that of network virtualization [30].

In order to reduce the hardness of the VN embedding problem and to enable efficient heuristics, existing research has been restricting the problem space in different dimensions, which include the following:

1) considering offline version of the problem (i.e., all VN requests are known in advance) [12], [13];
2) ignoring either node or link requirements [11], [12];
3) assuming infinite capacity of substrate nodes and links to obviate admission control [11]–[13];
4) focusing on specific VN topologies [12].

Yu *et al.* [14] considered all these issues, except for the location constraints on virtual nodes, by envisioning support from the substrate network through node and link migration as well as multipath routing. We do not restrict the problem space by assuming infinite capacity of the substrate network resources nor do we assume any specialized VN topologies.

Contrary to the algorithms proposed in this paper, all the existing algorithms can be separated into two basic phases:

1) assigning virtual nodes using some greedy heuristics (e.g., assign virtual nodes with higher requirements to substrate nodes with more available resources [13], [14]);
2) embedding virtual links onto substrate paths using shortest path algorithms [13] in case of unsplittable flows, or using multicommodity flow algorithms in case of splittable flows [14], [16].

Lischka *et al.* [31] proposed a backtracking-based VN embedding algorithm using subgraph isomorphism detection that extensively searches the solution space in a single stage. While completely combining the two phases is $\mathcal{NP}$-hard, coordinating them does improve embedding performance as evinced by the proposed algorithms through extensive simulation over diverse workloads. Houidi *et al.* [32] presented a distributed algorithm that simultaneously maps virtual nodes and virtual links without any centralized controller.

Throughout this paper, we assumed that the substrate network is always operational. In practice, however, not only can components in the substrate network fail, a single failure of a physical resource can disrupt operations of multiple VNs. Rahman *et al.* [33] extended the ViNEYard model with a fast rerouting strategy that utilizes preallocated backup quota on each physical link to ensure survivability from individual substrate link failures. Yeow *et al.* [34] enabled recovery from both substrate node and link failures while minimizing backup resources through pooling.

Over time, as VN requests arrive and leave the system, substrate network resources become fragmented and cause underutilization of resources. In addition, at different points in time, different substrate nodes and links can become more important than others (e.g., due to bottleneck shifts or price fluctuation). Butt *et al.* [35] extended the ViNEYard model with support for resource prioritization and dynamic reoptimization and reembedding to make it more responsive.

The integer and mixed integer programming approaches had been applied to a number of resource allocation and optimization problems in the networking area. Kuman *et al.* [36] proposed an integer programming model to solve the VPN tree computation problem for bandwidth provisioning in VPNs. Techniques of randomized rounding for linear programming relaxations to obtain approximation algorithms was first introduced in [37]. In this paper, we take a formal approach to solve the online VN embedding problem using a mixed integer programming formulation. To the best of our knowledge, this is the first attempt to create a mathematical programming formulation for this problem.

The effect of lookahead on online algorithms have been thoroughly investigated in the algorithm design literature [23], [38]. Yu *et al.* [14] used batch processing to improve the performance of VN embedding. We propose a generalized window-based VN embedding mechanism that is based on a fixed window size and prespecified maximum waiting periods for VN requests and can extend any existing purely online VN embedding algorithm.

IX. Conclusion

We argued in this paper that coordinating node and link mapping phases during VN embedding significantly increases the solution space and improves the quality of embedding heuristics. ViNEYard algorithms proposed in this paper outperform their counterparts in terms of acceptance ratio, revenue, and provisioning cost. We also developed a window-based extension for VN embedding that can incorporate existing VN embedding algorithms and demonstrated the dominance of the proposed algorithms in this generalized setting.

A number of issues still remain unresolved, however. We are specially interested in extending intradomain ViNEYard algorithms to support VN embedding across multiple administrative domains [39]. Designing advanced economic models to replace the simple revenue model used in the existing literature for VN pricing also requires further attention. Available approaches to directly solve integer and mixed integer programs (e.g., using column generation) can be employed to develop efficient algorithms to obtain optimal or near-optimal solutions for the original mixed integer formulation without any relaxation. Finally, finding the optimal window size for window-based VN embedding is an open problem. A possible approach to find the optimal window size would be to express our performance metrics as functions of the window size and the maximum waiting period, which would require explicit expressions for the performance metrics. One way to address this would be to find the best-fitting curves from the performance graphs and use them as approximate functions.

Appendix
Approximation Ratio for D-ViNE With Independent f Assumption

We assume that f^i_{uv} variables do not change during rounding and relaxation (RR) and ignore them in the approximation ratio calculation[6]. Consequently, the objective function in (7) can be reduced to the following:

$$\text{minimize} \quad \mathcal{F} = \sum_{m,w} a(m,w)x_{mw}$$

where $a(m,w)$ is a function of all the coefficients of x_{mw}.

Let, x^*_{mw} denote the relaxed value for x_{mw} and $x^*_{\max}$ be the maximum among the relaxed values. Consequently, we have $x^*_{\max} \geq \frac{1}{|\Omega(m)|}$. Let $\mathcal{F}^*$ be the approximate value of the objective function. Therefore

$$\begin{aligned}\mathcal{F}^* &= \sum_{m,w} a(m,w)x^*_{mw} \\ &\geq \frac{1}{|\Omega(m)|}\sum_{m,w} a(m,w) \\ &\geq \frac{1}{|\Omega(m)|}\sum_{m,w} a(m,w)x_{mw} \\ &= \frac{1}{|\Omega(m)|}\mathcal{F}.\end{aligned}$$

[6]This analysis only holds for VN Embedding without window support.

Acknowledgment

The authors are grateful to the anonymous reviewers for their insightful comments and to the associate editor for handling the review process. The authors would also like to thank M. Young and Q. Zhang for their suggestions.

References

[1] N. M. M. K. Chowdhury, M. R. Rahman, and R. Boutaba, “Virtual network embedding with coordinated node and link mapping,” in *Proc. IEEE INFOCOM*, Apr. 2009, pp. 783–791.
[2] T. Anderson, L. Peterson, S. Shenker, and J. Turner, “Overcoming the internet impasse through virtualization,” *Computer*, vol. 38, no. 4, pp. 34–41, 2005.
[3] J. Turner and D. Taylor, “Diversifying the internet,” in *Proc. IEEE GLOBECOM*, 2005, vol. 2, pp. 755–760.
[4] N. Feamster, L. Gao, and J. Rexford, “How to lease the internet in your spare time,” *Comput. Commun. Rev.*, vol. 37, no. 1, pp. 61–64, 2007.
[5] N. M. M. K. Chowdhury and R. Boutaba, “A survey of network virtualization,” *Comput. Netw.*, vol. 54, no. 5, pp. 862–876, 2010.
[6] “GENI: Global environment for network innovations,” [Online]. Available: http://www.geni.net/
[7] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, “In VINI veritas: Realistic and controlled network experimentation,” in *Proc. ACM SIGCOMM*, 2006, pp. 3–14.
[8] D. Andersen, “Theoretical approaches to node assignment,” 2002 [Online]. Available: http://www.cs.cmu.edu/~dga/papers/andersen-assign.ps
[9] J. Kleinberg, “Approximation algorithms for disjoint paths problems,” Ph.D. dissertation, MIT, Cambridge, MA, 1996.
[10] S. Kolliopoulos and C. Stein, “Improved approximation algorithms for unsplittable flow problems,” in *Proc. IEEE FOCS*, 1997, pp. 426–435.
[11] J. Fan and M. Ammar, “Dynamic topology configuration in service overlay networks—A study of reconfiguration policies,” in *Proc. IEEE INFOCOM*, 2006, pp. 1–12.
[12] J. Lu and J. Turner, “Efficient mapping of virtual networks onto a shared substrate,” Washington University, Seattle, WA, Tech. Rep. WUCSE-2006-35, 2006.
[13] Y. Zhu and M. Ammar, “Algorithms for assigning substrate network resources to virtual network components,” in *Proc. IEEE INFOCOM*, 2006, pp. 1–12.
[14] M. Yu, Y. Yi, J. Rexford, and M. Chiang, “Rethinking virtual network embedding: Substrate support for path splitting and migration,” *Comput. Commun. Rev.*, vol. 38, no. 2, pp. 17–29, Apr. 2008.
[15] A. Schrijver, *Theory of Linear and Integer Programming*. New York: Wiley, 1986.

[16] W. Szeto, Y. Iraqi, and R. Boutaba, "A multi-commodity flow based approach to virtual network resource allocation," in *Proc. IEEE GLOBECOM*, 2003, pp. 3004–3008.
[17] D. Eppstein, "Finding the k shortest paths," *SIAM J. Comput.*, vol. 28, no. 2, pp. 652–673, 1998.
[18] N. Karmarkar, "A new polynomial-time algorithm for linear programming," in *Proc. ACM STOC*, 1984, pp. 302–311.
[19] "GNU linear programming kit," Free Software Foundation, Boston, MA, 2008 [Online]. Available: http://www.gnu.org/software/glpk/
[20] E. Zegura, K. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *Proc. IEEE INFOCOM*, 1996, pp. 594–602.
[21] R. Jain, *The Art of Computer Systems Performance Analysis*. New York: Wiley, 1991.
[22] M. Mitzenmacher, A. W. Richa, and R. Sitaraman, "The power of two random choices: A survey of techniques and results," in *Handbook of Randomized Computing*. Norwell, MA: Kluwer, 2001, pp. 255–312.
[23] S. Albers, "On the influence of lookahead in competitive paging algorithms," *Algorithmica*, vol. 18, no. 3, pp. 283–305, 1997.
[24] E. F. Grove, "Online bin packing with lookahead," in *Proc. ACM-SIAM SODA*, 1995, pp. 430–436.
[25] J.-Y. Le Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Berlin, Germany: Springer-Verlag, 2001.
[26] A. Gupta, J. M. Kleinberg, A. Kumar, R. Rastogi, and B. Yener, "Provisioning a virtual private network: A network design problem for multicommodity flow," in *Proc. ACM STOC*, 2001, pp. 389–398.
[27] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan, and J. E. van der Merwe, "Resource management with hoses: Point-to-cloud services for virtual private networks," *IEEE/ACM Trans. Netw.*, vol. 10, no. 5, pp. 679–692, Oct. 2002.
[28] S. Raghunath, K. K. Ramakrishnan, S. Kalyanaraman, and C. Chase, "Measurement based characterization and provisioning of IP VPNs," in *Proc. ACM IMC*, 2004, pp. 342–355.
[29] R. Ricci, C. Alfeld, and J. Lepreau, "A solver for the network testbed mapping problem," *Comput. Commun. Rev.*, vol. 33, no. 2, pp. 65–81, Apr. 2003.
[30] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau, "Large-scale virtualization in the Emulab network testbed," in *Proc. USENIX ATC*, 2008, pp. 113–128.
[31] J. Lischka and H. Karl, "A virtual network mapping algorithm based on subgraph isomorphism detection," in *Proc. ACM VISA*, 2009, pp. 81–88.
[32] I. Houidi, W. Louati, and D. Zeghlache, "A distributed virtual network mapping algorithm," in *Proc. IEEE ICC*, 2008, pp. 5634–5640.
[33] M. R. Rahman, I. Aib, and R. Boutaba, "Survivable virtual network embedding," in *Proc. IFIP Netw.*, 2010, pp. 40–52.
[34] W.-L. Yeow, C. Westphal, and U. Kozat, "Designing and embedding reliable virtual infrastructures," in *Proc. ACM VISA*, 2010, pp. 33–40.
[35] N. F. Butt, M. Chowdhury, and R. Boutaba, "Topology-awareness and reoptimization mechanism for virtual network embedding," in *Proc. IFIP Netw.*, 2010, pp. 27–39.
[36] A. Kumar, R. Rastogi, A. Silberschatz, and B. Yener, "Algorithms for provisioning virtual private networks in the hose model," *IEEE/ACM Trans. Netw.*, vol. 10, no. 4, pp. 565–578, Aug. 2002.
[37] P. Raghavan and C. D. Tompson, "Randomized rounding: A technique for provably good algorithms and algorithmic proofs," *Combinatorica*, vol. 7, no. 4, pp. 365–374, 1987.
[38] S. Albers, "A competitive analysis of the list update problem with lookahead," *Theor. Comput. Sci.*, vol. 197, no. 1–2, pp. 95–109, 1998.
[39] M. Chowdhury, F. Samuel, and R. Boutaba, "PolyViNE: Policy-based virtual network embedding across multiple domains," in *Proc. ACM VISA*, 2010, pp. 49–56.

Mosharaf Chowdhury (S'09) received the B.Sc. degree in computer science and engineering from Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, in 2006, and the M.Math. degree in computer science from the University of Waterloo, Waterloo, ON, Canada, in 2009, and is currently pursuing the Ph.D. degree in computer science at the University of California, Berkeley.

His research interests are primarily in large-scale data-parallel systems, data center networking, and clean-slate designs for the future Internet architecture.

Muntasir Raihan Rahman (M'10) received the B.Sc. degree in computer science and engineering from Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, in 2006, and the M.Math. degree in computer science from the University of Waterloo, Waterloo, ON, Canada, in 2010, and is currently pursuing the Ph.D. degree in computer science at the University of Illinois at Urbana–Champaign.

His research interests include algorithmic and experimental aspects of distributed systems, cloud computing, and algorithmic game theory.

Raouf Boutaba (M'93–SM'01) received the M.Sc. and Ph.D. degrees in computer science from the University Pierre & Marie Curie, Paris, France, in 1990 and 1994, respectively.

He is currently a Professor of computer science with the University of Waterloo, Waterloo, ON, Canada. His research interests include network, resource, and service management in wired and wireless networks.

Prof. Boutaba is the founding Editor-in-Chief of the IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, serving from 2007 to 2010, and has served on the Editorial Boards of other journals. He has received several best paper awards and other recognitions such as the Premier's Research Excellence Award, the IEEE Hal Sobol Award in 2007, the Fred W. Ellersick Prize in 2008, and the Joe LociCero and the Dan Stokesbury awards in 2009.