Global Resource Capacity Algorithm with Path Splitting for Virtual Network Embedding

Soroush Haeri, Qingye Ding, Zhida Li, and Ljiljana Trajković
Simon Fraser University
Vancouver, British Columbia, Canada
Email: {shaeri, qingyed, zhidal, ljilja}@sfu.ca

***Abstract*—Network virtualization enables support and deployment of new services and applications that the current Internet architecture is unable to support. Virtual Network Embedding (VNE) problem that addresses efficient mapping of virtual network elements onto a physical infrastructure (substrate network) is one of the main challenges in network virtualization. The Global Resource Capacity (GRC) is a VNE algorithm that utilizes for virtual link mapping a modified version of Dijkstra's shortest path algorithm. In this paper, we propose the GRC-M algorithm that utilizes the Multicommodity Flow (MCF) algorithm. MCF enables path splitting and yields to higher substrate resource utilizations. Simulation results show that MCF significantly enhances performance of the GRC algorithm.**

I. Introduction

Rapid development of Internet applications has led to increasing demands on the current Internet infrastructure. Network virtualization [1], [2] plays an important role in efficient deployment of these applications. It allows multiple virtual networks to operate on a common substrate network and enables running applications economically using virtual components. Infrastructure Providers (InPs) and Service Providers (SPs) are the main components of the virtualized network architecture [3]. InPs own and operate substrate networks while SPs assign computing resources from multiple InPs to virtual networks (VNs). Virtual Private Networks (VPNs), Virtual Local Area Networks (VLANs), and Virtual Machines (VMs) benefit from enabling virtualization.

Substrate Networks (SNs) consist of physical nodes that are interconnected via physical links. Virtual networks consist of virtual nodes that are interconnected via virtual links. Virtual Network Embedding (VNE) problem deals with embedding virtual nodes and links into substrate nodes and paths, respectively. Limited substrate resources make finding profitable virtual network embeddings challenging. Furthermore, since VNRs need to be served as they arrive, in the absence of an accurate model for the arrival processes of VN Requests (VNRs), finding a VNE solution is even more challenging [4].

VNE problem may be reduced to the multiway separator problem, which is NP-hard [5], [6]. It may be divided into Virtual Node Mapping (VNoM) and Virtual Link Mapping (VLiM) subproblems. The VNE algorithms may be categorized into three classes depending on the method employed to solve these two subproblems: uncoordinated, coordinated two-stage, and coordinated one-stage algorithms [7]. In the coordinated two-stage approach, VNoM is first solved while considering the virtual link constrains. The VLiM is then solved by using the VNoM solution as the input. Even though the one-stage algorithms often find better solutions, they are computationally more complex and may only be used for solving small VNEs. Solving a VNE in two stages makes the problem more tractable and enables considering larger networks. Coordination between VNoM and VLiM was proposed to enhance the solutions identified by two-stage algorithms [4].

The Global Resource Capacity (GRC) [8], D-Vine [4], and R-Vine [4] are examples of coordinated two-stage algorithms. The GRC algorithm relies on a node-ranking approach to solve the VNoM subproblem. The rank of a node is calculated by considering the CPU capacity of the node and the bandwidth of the incident links [8]. After calculating the ranks for substrate and virtual nodes, the GRC algorithm employs a *large-to-large and small-to-small* [9] mapping scheme to embed virtual nodes onto substrate nodes. After completing the VNoM assignments, a modified Dijkstra's algorithm is used to calculate the shortest path between mapped nodes in order to complete VLiM. D-Vine and R-Vine algorithms formulate the VNoM problem as a Mixed Integer Program (MIP). They employ a rounding-based approach to solve the Linear Programming (LP) relaxation of the proposed MIP. The Multicommodity Flow (MCF) [5] algorithm is then employed to solve the VLiM subproblem. MCF treats nodes and links of VNRs as commodities and flows, respectively. When solving VLiM, MCF may use path splitting [6]. It has been shown that utilizing MCF results in better substrate resource utilizations [4].

In this paper, we propose the GRC-M algorithm, a coordinated two-stage algorithm that combines GRC and MCF for solving VNE problems. We aim to increase an InP's profit by maximizing the acceptance ratio of VNRs while minimizing the VNR embedding costs. The results show that GRC-M improves the VNR acceptance ratio by up to 25% and 10% compared to GRC and both Vine algorithms, respectively.

The remainder of this paper is organized as follows. In Section II, we introduce the VNE problem and its objectives and describe metrics for measuring VNE performance. GRC and MCF formulation for solving VNoM and VLiM are presented in Section III. In Section IV, we compare performance of the GRC-M with GRC, R-Vine, and D-Vine algorithms. We conclude with Section V.

II. Description of Virtual Network Embedding

Let $G^s(N^s, E^s)$ denote an SN graph, where N^s represents the set of substrate nodes $\{n_1^s, n_2^s, ..., n_k^s\}$ and E^s is the set of substrate links $\{e_1^s, e_2^s, ..., e_k^s\}$. Nodes and links possess multiple resources: nodes possess CPU capacity and links

possess bandwidth. We denote by $\mathcal{C}(n^s)$ the CPU capacity of a substrate node n^s. The bandwidth of a substrate link is denoted by $\mathcal{B}(e^s)$. The ith VNR is denoted by $\Psi_i(G^{\Psi_i}, \omega^{\Psi_i}, \xi^{\Psi_i})$, where G^{Ψ_i} is the VN graph, ω^{Ψ_i} is the arrival time of the VNR, and ξ^{Ψ_i} is the VNR's life-time.

Let $G^{\Psi_i}(N^{\Psi_i}, E^{\Psi_i})$ denote the graph of the ith virtual network request, where $N^{\Psi_i} = \{n_1^{\Psi_i}, n_2^{\Psi_i}, ..., n_k^{\Psi_i}\}$ and $E^{\Psi_i} = \{e_1^{\Psi_i}, e_2^{\Psi_i}, ..., e_k^{\Psi_i}\}$ are the sets of virtual nodes and links, respectively. We assume that $\mathcal{C}(n^{\Psi_i})$ and $\mathcal{L}(n^{\Psi_i})$ denote the CPU requirement and the location preference of a virtual node n^{Ψ_i}, respectively, while $\mathcal{B}(e^{\Psi_i})$ denotes the bandwidth requirement of a virtual link e^{Ψ_i}.

A substrate node n^s may be used for embedding a virtual node n^{Ψ_i} if:

$$\mathcal{C}(n^s) \geq \mathcal{C}(n^{\Psi_i}) \tag{1}$$

and

$$d\left(\mathcal{L}(n^s), \mathcal{L}(n^{\Psi_i})\right) \leq \delta^{\Psi_i}, \tag{2}$$

where $d(.,.)$ is the Euclidean distance and δ^{Ψ_i} is its upper limit.

The objective of a virtual network embedding algorithm is to maximize an InP's revenue while minimizing the cost of embeddings. The average revenue is directly proportional to the acceptance ratio. Furthermore, increasing the node and link utilizations simultaneously may generate additional profit. Note that high node utilization is always desirable because it is a result of high acceptance ratio while the link utilization should be maintained as low as possible.

Revenue: The revenue $R(G^{\Psi_i})$ is the weighted sum of the VNR's CPU and bandwidth requirements defined as:

$$\mathbf{R}(G^{\Psi_i}) = \omega_c \sum_{n^{\Psi_i} \in N^{\Psi_i}} \mathcal{C}(n^{\Psi_i}) + \omega_b \sum_{e^{\Psi_i} \in E^{\Psi_i}} \mathcal{B}(e^{\Psi_i}), \tag{3}$$

where ω_c and ω_b are unit prices of the requested CPU and bandwidth, respectively. InPs generate revenue only if the VNR is accepted. Note that this revenue only depends on the VNR's resource requirements.

Cost: The cost of a VNE $\mathcal{C}(G^{\Psi_i})$ depends on the allocated substrate resources. It is calculated as:

$$\mathbf{C}(G^{\Psi_i}) = \sum_{n^{\Psi_i} \in N^{\Psi_i}} \mathcal{C}(n^{\Psi_i}) + \sum_{e^{\Psi_i} \in E^{\Psi_i}} \sum_{e^s \in E^s} f_{e^s}^{e^{\Psi_i}}, \tag{4}$$

where $f_{e^s}^{e^{\Psi_i}}$ is the bandwidth of the substrate link e^s that is allocated for the virtual link e^{Ψ_i}.

Acceptance Ratio: The acceptance ratio is calculated as:

$$p_a^{\tau} = \frac{|\Psi^a(\tau)|}{|\Psi(\tau)|}, \tag{5}$$

where $|\Psi^a(\tau)|$ and $|\Psi(\tau)|$ denote the number of accepted VNRs and the number of VNRs that arrived in the time interval τ, respectively.

Node and Link Utilizations: Node utilization $\mathcal{U}(N^s)$ is calculated as:

$$\mathcal{U}(N^s) = 1 - \frac{\sum_{n^s \in N^s} \mathcal{C}(n^s)}{\sum_{n^s \in N^s} \mathcal{C}_{max}(n^s)}, \tag{6}$$

where $\mathcal{C}(n^s)$ is the available CPU resource of a substrate node n^s and $\mathcal{C}_{max}(n^s)$ is the maximum CPU resource of the node. Similarly, link utilization $\mathcal{U}(E^s)$ is calculated as:

$$\mathcal{U}(E^s) = 1 - \frac{\sum_{e^s \in E^s} \mathcal{B}(e^s)}{\sum_{e^s \in e^s} \mathcal{B}_{max}(e^s)}, \tag{7}$$

where $\mathcal{B}(e^s)$ is the available bandwidth of a substrate link e^s and $\mathcal{B}_{max}(e^s)$ is the maximum bandwidth of the link.

III. GRC-M Algorithm

GRC-M employs the GRC algorithm [8] for solving VNoM while employing the MCF [5] algorithm for solving VLiM. GRC algorithm is effective in calculating the embedding potential of substrate nodes. The MCF algorithm enables path splitting, which improves resources utilization.

A. GRC Algorithm for Node Mapping

Let $Adj(n_i^s)$ denote the set of substrate nodes adjacent to n_i^s and $e^s(n_i^s, n_j^s)$ denote the substrate link that connects the substrate nodes n_i^s and n_j^s. The GRC algorithm first calculates the embedding capacity $r(n_i^s)$ of a substrate node n_i^s as:

$$r(n_i^s) = (1-d)\hat{\mathcal{C}}(n_i^s) + d \sum_{n_j^s \in Adj(n_i^s)} \frac{\mathcal{B}\left(e^s(n_i^s, n_j^s)\right) \times r(n_j^s)}{\sum_{n_k^s \in Adj(n_j^s)} \mathcal{B}\left(e^s(n_j^s, n_k^s)\right)}, \tag{8}$$

where $0 < d < 1$ is a constant damping factor and $\hat{\mathcal{C}}(n_i^s)$ is the normalized CPU resources of n_i^s:

$$\hat{\mathcal{C}}(n_i^s) = \frac{\mathcal{C}(n_i^s)}{\sum_{n^s \in N^s} \mathcal{C}(n^s)}. \tag{9}$$

We assume that an SN consists of p nodes. The vector form of the embedding capacity of substrate nodes is:

$$\mathbf{r} = (1-d)\hat{\mathbf{c}} + d\mathbf{M}\mathbf{r}, \tag{10}$$

where $\hat{\mathbf{c}} = \left(\hat{\mathcal{C}}(n_1^s), \ldots, \hat{\mathcal{C}}(n_p^s)\right)^T$, $\mathbf{r} = \left(r(n_1^s), \ldots, r(n_p^s)\right)^T$, and $\mathbf{M}$ is a $p \times p$ square matrix. The m_{ij} element of $\mathbf{M}$ is calculated as:

$$m_{ij} = \begin{cases} \dfrac{\mathcal{B}\left(e^s(n_i^s, n_j^s)\right)}{\sum_{n_k^s \in Adj(n_j^s)} \mathcal{B}\left(e^s(n_j^s, n_k^s)\right)} & \forall\, e^s(n_i^s, n_j^s) \in E^s \\ 0 & \text{otherwise} \end{cases}. \tag{11}$$

The GRC algorithm iteratively calculates the embedding capacity vector $\mathbf{r}$ by initially setting $\mathbf{r}_0 = \hat{\mathbf{c}}$ and calculating $\mathbf{r}_{k+1}$ as:

$$\mathbf{r}_{k+1} = (1-d)\mathbf{c} + d\mathbf{M}\mathbf{r}_k, \tag{12}$$

where $\mathbf{r}_0$ is the result of $\mathbf{r}$ after k iterations. The iterative process is terminated when:

$$|\mathbf{r}_{k+1} - \mathbf{r}_k| < \sigma, \tag{13}$$

where $\sigma << 1$ is a predefined stopping threshold. Similar procedure is used to calculate embedding capacity of virtual nodes.

When the embedding capacity vectors $\mathbf{r}$ for substrate and virtual nodes are calculated, the nodes are ranked based on their capacity. The GRC algorithm then performs the embedding by matching the substrate and virtual nodes with the highest ranks, provided that the substrate node satisfies the virtual node CPU requirement. Otherwise, the substrate node with the second highest rank is selected. The VNR is denied if no substrate node satisfies the virtual node CPU requirement.

B. MCF Algorithm for Link Mapping

The output of the GRC algorithm is a node mapping that is used as the input to the MCF algorithm [5] to find the virtual link mapping. A flow is a commodity $k_i = (s_i, t_i, d_i)$, where s_i, t_i, and d_i are the flow source, destination, and demand, respectively. Let us assume that the virtual nodes $n_a^{\Psi_i}$ and $n_b^{\Psi_i}$ are mapped by GRC onto substrate nodes n_a^s and n_b^s, respectively. A virtual link $e^{\Psi_i}(n_a^{\Psi_i}, n_b^{\Psi_i})$ between the virtual nodes $n_a^{\Psi_i}$ and $n_b^{\Psi_i}$ may be viewed as a flow between the substrate nodes n_a^s and n_b^s. It may be denoted as a commodity $k_{e^{\Psi_i}}\big(s_{e^{\Psi_i}} = n_a^s, t_{e^{\Psi_i}} = n_b^s, d_{e^{\Psi_i}} = \mathcal{B}(e^{\Psi_i})\big)$. MCF may be formulated as the linear program with the following objective and constraints:

Objective:

$$\text{minimize} \quad \sum_{e^s \in E^s} \frac{1}{\mathcal{B}(e^s) + \epsilon} \sum_{e^{\Psi_i} \in E^{\Psi_i}} f_{e^s}^{e^{\Psi_i}}, \tag{14}$$

where $\epsilon << 1$ and $f_{e^s}^{e^{\Psi_i}}$ is the bandwidth of the substrate link e^s that is allocated for the virtual link e^{Ψ_i}.

Capacity constraint:

$$\sum_{e^{\Psi_i} \in E^{\Psi_i}} \left[f_{e^s(n_j^s, n_k^s)}^{e^{\Psi_i}} + f_{e^s(n_k^s, n_j^s)}^{e^{\Psi_i}} \right] \leq \mathcal{B}(e^s(n_j^s, n_k^s)) \quad \forall n_j^s, n_k^s \in N^s. \tag{15}$$

Flow conservation constraint:

$$\sum_{n_j^s \in N^s} \left[f_{e^s(n_j^s, n_k^s)}^{e^{\Psi_i}} - f_{e^s(n_k^s, n_j^s)}^{e^{\Psi_i}} \right] = 0 \quad \forall e^{\Psi_i} \in E^{\Psi_i}, \ \forall n_k^s \in N^s \setminus \{s_{e^{\Psi_i}}, t_{e^{\Psi_i}}\}. \tag{16}$$

Demand satisfaction constraints:

$$\sum_{n_j^s \in N^s} \left[f_{e^s(s_{e^{\Psi_i}}, n_j^s)}^{e^{\Psi_i}} - f_{e^s(n_j^s, s_{e^{\Psi_i}})}^{e^{\Psi_i}} \right] = d_{e^{\Psi_i}} \quad \forall e^{\Psi_i} \in E^{\Psi_i}, \tag{17}$$

$$\sum_{n_j^s \in N^s} \left[f_{e^s(t_{e^{\Psi_i}}, n_j^s)}^{e^{\Psi_i}} - f_{e^s(n_j^s, t_{e^{\Psi_i}})}^{e^{\Psi_i}} \right] = -d_{e^{\Psi_i}} \quad \forall e^{\Psi_i} \in E^{\Psi_i}. \tag{18}$$

This MCF formulation is used in order to compare performance of the proposed GRC-M algorithm with R-Vine and D-vine algorithms that use similar formulation.

IV. Performance Evaluation

We compare performance of the GRC-M algorithm with GRC [8], R-Vine [4], and D-Vine [4] algorithms that have been proposed in the literature. We compare performance of these algorithms by considering acceptance ratio, revenue to cost ratio, and average node and link utilizations.

A. Simulation Environment

Simulations were performed using a Dell Optiplex-790 with 16 GB memory and the Intel Core i7 2600 processor. We have developed *VNE-Sim*, a discrete-event VNE simulator written in C++. It is based on the discrete event system specification (DEVS) framework [10] and employs the *Adevs* library [11]. Discrete-event systems may be modeled, designed, analyzed, and simulated using the DEVS framework. We use the GNU Scientific Library [12] to generate random numbers and the required probability distributions. Substrate and virtual network topologies and resources as well as parameters of the GRC and Vine algorithms are adopted from the literature [4], [8], [9].

The Boston University Representative Topology Generator (BRITE) [13] has been used to generate the substrate and VNR graphs. The substrate graph is composed of 50 nodes that are randomly placed on a 25×25 grid. Each node is connected to a maximum of 5 other nodes, resulting in a substrate network graph with 221 edges. Node connections are generated based on the Waxman algorithm [14] with the parameter $\alpha = 0.5$ and the exponential parameter $\beta = 0.2$ [13], [15]. The VNR graphs are generated using a similar process. The number of nodes in each VNR graph is uniformly distributed between 3 and 10 [4]. Each virtual node is connected to a maximum of 3 other nodes. The CPU capacity of substrate nodes and the bandwidth of substrate links are uniformly distributed between 50 and 100 units. The CPU requirements of the virtual nodes are uniformly distributed between 2 and 20 units while the bandwidth requirements of the virtual links are uniformly distributed between 0 and 50 units [4]. The maximum allowable distance δ for embedding VNR nodes is uniformly distributed between 15 and 25 distance units. We assume that the VNRs arrive according to Poisson distribution with a mean arrival rate of λ requests per unit time. Their life-times are exponentially distributed with a mean $\frac{1}{\mu} = 1,000$ generating VNR traffic of λ/μ Erlangs. Each simulation lasted 50,000 time units [8], [9]. Performance of the four algorithms was compared using VNR arrival rates between 1 and 8 requests per 100 time units generating traffic loads of 10, 20, 30, 40, 50, 60, 70, and 80 Erlangs [8].

B. Simulation Results

Simulation results are shown in Fig. 1. GRC-M achieves the highest acceptance ratio in all scenarios while its revenue to cost ratio decreases quickly. The substrate network resources become scarce at higher VNR traffic loads. Therefore, there is a higher likelihood of embedding virtual nodes onto substrate

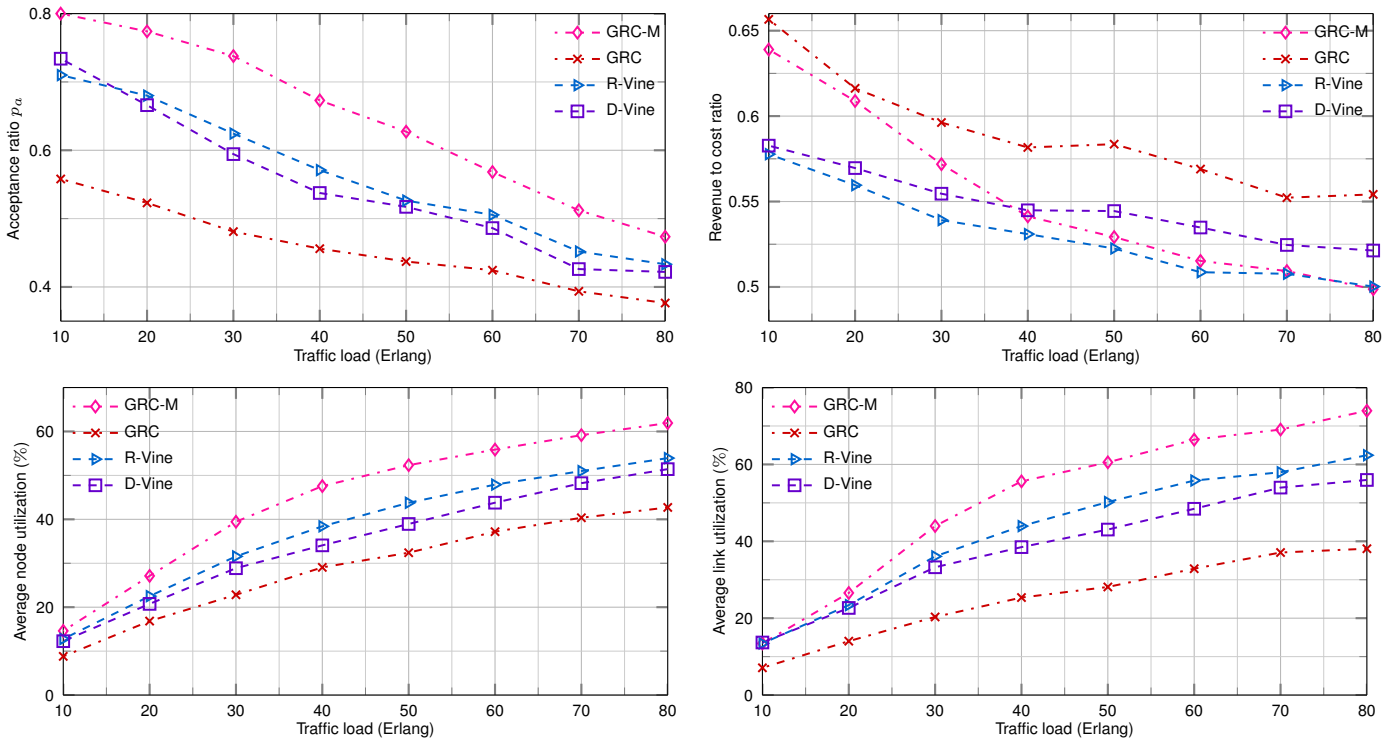

Fig. 1. Performance of the algorithms with various VNR traffic loads. Shown are performance metrics as functions of the VNR traffic load.

nodes that are further apart, which results in more costly embeddings and, consequently, smaller revenue to cost ratios [8]. Although high acceptance and revenue to cost ratios are desirable for VNE algorithms, achieving both simultaneously is not feasible. Therefore, the two ratios should be considered at the same time. For example, consider the R-Vine and D-Vine algorithms at 80 Erlangs. Their acceptance ratios are similar while D-Vine has a 5% higher revenue to cost ratio. Therefore, in this case D-Vine performs better than R-Vine. GRC-M has the highest substrate node and link utilizations. Its average node utilization is high as a result of the high acceptance ratio. The high link utilization of GRC-M contributes to the fast decreasing trend of its revenue to cost ratio.

V. Conclusion

In this paper, we proposed an improved virtual network embedding algorithm named GRC-M that employs the Global Resource Capacity for virtual node mapping while using the Multicommodity Flow algorithm for link mappings. We compare GRC-M with GRC, R-Vine, and D-Vine algorithms in terms of acceptance ratio, revenue to cost ratio, and average node and link utilizations. Simulation results show that while GRC-M achieves revenue to cost ratios comparable to the Vine algorithms, it enables embedding additional virtual network requests onto a substrate network.

References

[1] T. Anderson, L. Peterson, S. Shenker, and J. S. Turner, "Overcoming the Internet impasse through virtualization," *Computer*, vol. 38, no. 4, pp. 34–41, Apr. 2005.

[2] 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, Jan. 2007.

[3] N. M. M. K. Chowdhury and R. Boutaba, "Network virtualization: state of the art and research challenges," *IEEE Commun. Mag.*, vol. 47, no. 7, pp. 20–26, July 2009.

[4] M. Chowdhury, M. R. Rahman, and R. Boutaba, "ViNEYard: Virtual network embedding algorithms with coordinated node and link mapping," *IEEE/ACM Trans. Netw.*, vol. 20, no. 1, pp. 206–219, Feb. 2012.

[5] D. G. Andersen, "Theoretical approaches to node assignment," Dec. 2002, Unpublished Manuscript. [Online]. Available: http://repository.cmu.edu/compsci/86/.

[6] 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. 19–29, Mar. 2008.

[7] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: a survey," *IEEE Commun. Surveys & Tutorials*, vol. 15, no. 4, pp. 1888–1906, Feb. 2013.

[8] L. Gong, Y. Wen, Z. Zhu, and T. Lee, "Toward profit-seeking virtual network embedding algorithm via global resource capacity," in *Proc. IEEE INFOCOM*, Toronto, ON, Canada, Apr. 2014, pp. 1–9.

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

[10] A. M. Uhrmacher, "Dynamic structures in modeling and simulation: a reflective approach," *ACM Trans. on Modeling and Comput. Simulation*, vol. 11, no. 2, pp. 206–232, Apr. 2001.

[11] J. J. Nutaro, *Building Software for Simulation: Theory and Algorithms, with Applications in C++*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2010.

[12] (2015, July) GLS-GNU Scientific Library. [Online]. Available: http://www.gnu.org/software/gsl/.

[13] (2015, July) Boston University Representative Internet Topology Generator. [Online]. Available: http://www.cs.bu.edu/brite/.

[14] B. M. Waxman, "Routing of multipoint connections," *IEEE J.Sel. Areas Commun.*, vol. 6, no. 9, pp. 1617–1622, Dec. 1988.

[15] E. W. Zegura, K. L. Calvert, and M. J. Donnahoo, "A quantitative comparison of graph-based models for Internet topology," *IEEE/ACM Trans. Netw.*, vol. 5, no. 6, pp. 770–783, Dec. 1997.