Computer Networks 106 (2016) 196–208

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Presto: Towards efficient online virtual network embedding in virtualized cloud data centers

Ting Wang[a,1,*], Mounir Hamdi[b,c,2]

[a] Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong
[b] College of Science, Engineering and Technology in Hamad Bin Khalifa University, Qatar
[c] Hong Kong University of Science and Technology, Hong Kong

ARTICLE INFO

Article history:
Received 17 September 2015
Revised 3 June 2016
Accepted 27 June 2016
Available online 1 July 2016

Keywords:
Data center network
Virtual network embedding
Network virtualization
Resource allocation

ABSTRACT

As an efficient solution to diversify the future Internet for resource sharing in data centers, the network virtualization enables seamless integration of network experiments, services and architectures with different features by allowing multiple heterogeneous virtual networks (VNs) to simultaneously coexist on a shared substrate infrastructure. Embedding multiple virtual networks onto a shared substrate by allocating substrate resources to virtual nodes and virtual links of VN requests under a collection of constrains is known to be an NP-hard problem even for the offline VN embedding. To deal with this issue, this paper formulates the VN embedding problem as a new multiple objective linear programming optimization program, and solves it in a preemptive strategy by decomposing the problem into node mapping and link mapping phases. Furthermore, based on an Artificial Intelligence resource abstraction model, named Blocking Island (BI), we propose an efficient online heuristic VN embedding algorithm called *Presto*. *Presto* operates with quite low computation complexity and greatly reduces the search space, which far outperforms other candidates. The goal of *Presto* is to maximize the economic revenue of infrastructure providers while minimizing the embedding cost. The extensive simulation results further prove the feasibility and good performance of *Presto* in revenue, VN request acceptance ratio, computation efficiency and resource utilization.

1. Introduction

As highly multiplexed shared environments, cloud data centers are equipped with a large number of physical servers and virtual machines (VMs) hosted in servers to simultaneously offer multiple tenants with on-demand use of computing resources in a pay-as-you-go manner [1–5]. How to efficiently share the physical network resources among multiple tenants that have diversified network topologies with different network characteristics is a key concern. With respect to this issue, network virtualization has emerged as an efficient technology for resource sharing, where multiple heterogeneous network architectures are allowed to coexist on a shared substrate [6–8]. Upon on the virtualized shared data centers, the infrastructure providers then make best effort to utilize the substrate resources to serve the users that request customized services with required resources (such as CPU capacities, network bandwidth, etc.) running over different user self-defined network topologies, which are also known as *virtual networks* (VNs). Each virtual network consists of a set of virtual nodes interconnected through a set of virtual links with required capacities. The allocation of substrate resources to the virtual networks is called *virtual network embedding*[3] (VNE). Each virtual node is mapped onto a substrate node, while each virtual link is mapped onto a substrate path connecting the corresponding substrate nodes under a series of pre-defined constraints. Fig. 1 illustrates an example of virtual network embedding, where two embedded virtual networks share the same substrate network.

The main objective of solving VNE problem is to make efficient use of substrate resources through dynamic and effective VN mapping algorithms. Although embedding diversified virtual networks of different users onto the underlying physical network can maximize the benefits gained from existing hardware of the infrastructure, VNE has been presented as a very challenging resource allocation problem [9–11] that has been addressed in many research

* Corresponding author.
E-mail addresses: twangah@connect.ust.hk (T. Wang), hamdi@cse.ust.hk (M. Hamdi).
[1] Student Member, IEEE
[2] Fellow, IEEE

[3] In this paper, “embedding” and “mapping” are used interchangeably.

http://dx.doi.org/10.1016/j.comnet.2016.06.036
1389-1286/

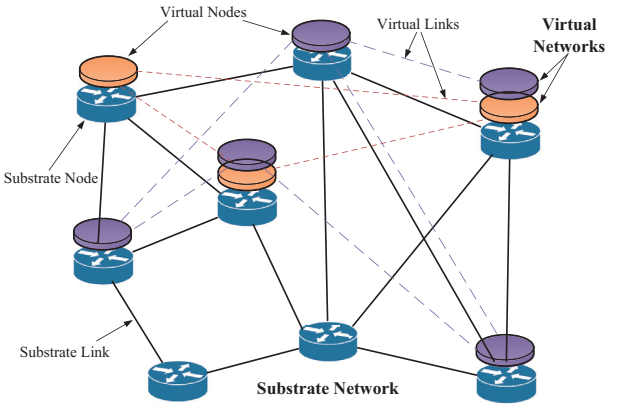

Fig. 1. Virtual network embedding on a shared substrate network.

studies [12–17]. In fact, the VNE problem is NP-hard [13,15,16,18], even in the offline case. Even when all the virtual nodes are embedded, to embed the virtual links is still NP-hard [15,19,20]. Naturally, the online case of VNE problem would be more intractable.

In response to this issue, in this paper we propose an efficient online heuristic VNE algorithm, named *Presto*, based on an Artificial Intelligence resource abstraction model called Blocking Island. *Presto* decomposes the VNE problem into two separate phases: virtual node mapping and virtual link mapping. In each phase, with the help of BI model, *Presto* ranks and embeds the virtual nodes or virtual links in a most advantageous order aiming to maximize the acceptance ratio. All the VN requests are also sorted according to some specific metrics, which targets at maximizing the economic revenues and minimizing the embedding cost. In addition, the proposed sliding window based batch processing approach enables *Presto* with the ability of lookahead and forward checking when processing the dynamically arriving VN requests. Moreover, with the benefit of BI paradigm, the search space is significantly decreased, and accordingly, the computation efficiency is greatly improved.

The primary contributions of this paper can be summarized as below:

1. Two heuristic variable ordering algorithms HVNO and HVLO are designed.
2. A node mapping algorithm HNM and a link mapping algorithm HLM are put forward.
3. To the best of our knowledge, we are the first to apply BI paradigm to solve the VNE problem.
4. Extensive simulations are conducted to evaluate the performance of *Presto*.

The rest of the paper is organized as follows. First we briefly review the related research literature in Section 2. Then Section 3 demonstrates the VN mapping model and problem formulation. In Section 4, the formulated multiple objective linear programming problem is presented. Afterwards, Section 5 introduces the BI paradigm. The *Presto* framework is designed in Section 6 followed by evaluations in Section 7. Section 8 concludes this paper.

2. Related work and motivation

A considerable number of research and investigations have been conducted on the virtual network embedding problem in recent years. In order to deal with the computationally intractable VNE problem, most of the proposals resort to heuristic algorithms aiming to find some feasible solutions other than optimal solutions. Generally, the existing works can be classified into two categories: offline algorithms [14,21,22] and online algorithms [12,13,16,18,23].

The offline algorithms process based on the assumption that all VN requests are defined and known in advance. The authors in [21] proposed a Max-Min Ant Colony metaheuristic based approach which uses parallel artificial ants to iteratively explore the search space for feasible candidates and select the candidates based on some specific objectives. The work of [14] also studied the offline problem, but they only considered the single VN embedding in specific backbone-star topologies and assumed the substrate resources are unlimited with only bandwidth constraints. Comparatively, I.Houidi [22] proposed a distributed VN embedding approach also by assuming that the substrate resources are unlimited and all the VN requests are known in advance. However, this distributed approach needs a number of signalling messages exchanged between the substrate nodes asynchronously so as to organize the distributed algorithm iterations, which increases time delay and signalling network overload. Correspondingly, The work [24,25] introduced a distributed and parallel VNE framework DPVNE, which can be used in combination with various cost-reducing embedding algorithms. According to their evaluation results, DPVNE achieves smaller message overhead and suggests a tradeoff between message overhead and the parallelism level. In these offline methods, all the VN requests can be considered together for embedding, which may achieve a better result than the online methods which do not have any global view and knowledge about the future allocations. However, to know all the VN requests in advance is not practical in real world, where dynamic online algorithms are more suitable and preferred.

The online VNE algorithms study the on-demand VN assignment problem, where algorithms dynamically compute a feasible set of substrate nodes and links to embed the virtual nodes and links upon the arrival of a VN request. Yu et al. [16] advocated an effective online approach, where they simplified the VNE problem by applying two specific strategies, path splitting and path migration, with admission control. In this work, a virtual link is allowed to be mapped to multiple substrate paths for splittable flows and path migration is used to periodically re-optimize the utilization of substrate resources. Comparatively, the paper [12] solved the VNE problem efficiently by applying the PageRank algorithm, where both substrate nodes and virtual nodes are ranked based on some metrics and then the virtual network is embedded based on these ranks. Their goal is to increase the acceptance ratio of online VN requests and as well as the overall revenue. The work [13] formulated the VNE problem as a mixed integer programming problem through substrate network augmentation by adding a set of node constraints like geographical locations. Then the formulated MIP problem is relaxed to a linear program and solved using deterministic and randomized rounding techniques. Another online embedding algorithm [18] is designed by reducing the VNE problem to the well known Subgraph Isomorphism Detection problem, where it maps nodes and links during the same stage.

Besides of considering the embedding cost and revenues, the computation efficiency is also a very important factor, especially in data centers. However, known as an NP-hard problem, the VNE suffers from intractable computation complexity with huge search domain. Though researchers proposed many heuristic algorithms to improve computation efficiency at the cost of sacrificing the optimality of results, the computation is still not efficient enough and the search space is still too huge. In regard to this issue, we propose a heuristic algorithm based on the powerful BI model which greatly increases the computation efficiency and significantly decreases the search domain. Moreover, it also saves much time in determining the acceptance of a VN request.

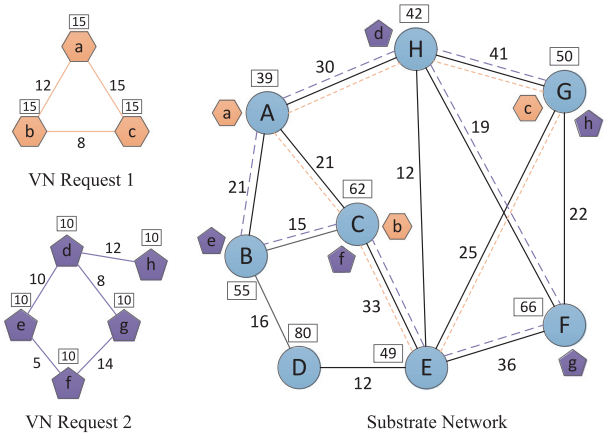

Fig. 2. An example of mapping VN requests onto a shared substrate network.

3. VN mapping model and problem formulation

3.1. Substrate network model

The underlying substrate network can be modelled as a weighted undirected graph $G_s = (N_s, E_s)$, where N_s is the set of substrate nodes of the network and E_s is the set of substrate links between nodes of N_s. Each substrate node $n_s \in N_s$ has an associated capacity weight value $C(n_s)$ to denote the available CPU capacity of the physical node n_s. Each substrate link $e_s(i, j) \in E_s$ between two substrate nodes i and j is associated with a bandwidth capacity $B(e_s)$, which denotes the amount of available bandwidth. The set of all substrate paths is denoted by P_s and the available bandwidth of a substrate path $p \in P_s$ between two substrate nodes is represented by $B(p)$. Fig. 2 illustrates an example of substrate network graph, where the numbers associated with the links indicate the available bandwidth and the values in rectangles denote the available CPU resources of the nodes.

3.2. Virtual network model

The virtual network topologies are set up on demand over the shared substrate network based on the received VN requests from users. Likewise, the virtual network also can be represented by a weighted undirected graph $G_v = (N_v, E_v)$, where N_v indicates the set of virtual nodes and E_v is the set of virtual links between nodes of N_v. Each virtual node $n_v \in N_v$ and virtual link $e_v \in E_v$ are associated with a minimum required CPU capacity $C(n_v)$ and a minimum required bandwidth capacity $B(e_v)$, respectively.

We denote the virtual network (VN) request by a quadruple $\mathbb{R}_v = (Rid, G_v, C_v, B_v)$, where $G_v = (N_v^{Rid}, E_v^{Rid})$, Rid is the unique identifier of a VN request, $C_v = [C(n_v^i)]$ indicates a vector of minimum required CPU capacities for virtual nodes n_v^i where $1 \le i \le |N_v^{Rid}|$, and $B_v = [B(e_v(i, j))]$ represents a matrix of minimum required bandwidth capacity for links $e_v(i, j) \in E_v^{Rid}$ between virtual nodes n_v^i and n_v^j where $1 \le i, j \le |N_v^{Rid}|$. Fig. 2 gives an example of two VN requests with node and link constraints.

3.3. Measurement of network resources

The decision of VN request allocation should be made upon the usage status of substrate network resources so as to avoid exceeding the node capacity. In this work, we use actual amount of allocated and available CPU and bandwidth resources to measure the substrate network resources other than the number of virtual nodes and links.

3.3.1. CPU resources of nodes

Let $\Psi_N(n_s)$ be the total amount of CPU resources allocated to the virtual nodes embedded on the substrate node $n_s \in N_s$, then we have

$$\Psi_N(n_s) = \sum_{n_v^i} C(n_v^i), \qquad 0 \le i \le \Omega_{n_s} \tag{1}$$

where Ω_{n_s} is the number of different virtual nodes hosted on the substrate node n_s.

Correspondingly, if we denote the residual or available CPU resources of a substrate node n_s by $\Lambda_N(n_s)$, then it can be derived that

$$\Lambda_N(n_s) = C(n_s) - \Psi_N(n_s) \tag{2}$$

3.3.2. Bandwidth resources of links

By analogy, let $\Psi_E(e_s)$ be the total amount of bandwidth allocated for the virtual links whose mapped substrate path $p \in P_s$ passes through the substrate link $e_s \in E_s$, then we have

$$\Psi_E(e_s) = \sum_{e_v^i} B(e_v^i), \qquad 0 \le i \le \Omega_{e_s} \tag{3}$$

where Ω_{e_s} is the number of different virtual links mapped onto the substrate link e_s.

The available bandwidth $\Lambda_E(e_s)$ of substrate link e_s then can be computed as

$$\Lambda_E(e_s) = B(e_s) - \Psi_E(e_s) \tag{4}$$

3.3.3. Bandwidth availability of substrate paths

The available bandwidth capacity $B(p)$ of a substrate path $p \in P_s$ between two substrate nodes can be measured as the minimal residual bandwidth of the links along the substrate path:

$$\Lambda_E(p) = \min_{e_s(i,j) \in p} \Lambda_E(e_s(i, j)) \tag{5}$$

3.3.4. Cost measurement of VN embedding

During the process of embedding one VN request $\mathbb{R}_v = (Rid, G_v, C_v, B_v)$, the total amount of reserved substrate nodes' CPU capacities $Cost(N_v^{Rid})$ for its virtual nodes N_v^{Rid} is deterministic, which equals to the sum of their required CPU capacities, i.e.,

$$Cost(N_v^{Rid}) = \sum_{n_v^i \in N_v^{Rid}} C(n_v^i), \quad 1 \le i \le |N_v^{Rid}| \tag{6}$$

However, the total amount of allocated link bandwidth capacities $Cost(E_v^{Rid})$ is indeterminate, and it depends on the length of substrate paths onto which the virtual links mapped. The link mapping cost can be computed as

$$Cost(E_v^{Rid}) = \sum_{e_v(i,j) \in E_v^{Rid}} B(e_v(i, j)) * |p_s(n_v^i, n_v^j)| \tag{7}$$

where virtual link $e_v(i, j)$ is mapped onto substrate path p_s and $|p_s(n_v^i, n_v^j)|$ denotes the number of links on path p_s.

Take Fig. 2 as an example, given that the virtual nodes of VN request 1 are mapped as {a → A, b → C, c → G} and the virtual links mapped to the substrate paths as {(a,b) → (A,C), (b,c) → (C,E,G), (c,a) → (G,H,A)}, then the total reserved link bandwidth for VN request 1 will be $Cost(E_v^1) = 21 * 1 + 8 * 2 + 15 * 2 = 67$. If we keep the way of node mapping the same, but remap virtual links to substrate paths as {(a,b) → (A,B,C), (b,c) → (C,E,F,G), (c,a) → (G,H,A)} then the link mapping cost will increase to be $Cost(E_v^1) = 21 * 2 + 8 * 3 + 15 * 2 = 96$. Thus, different way of link mapping induces different substrate link bandwidth cost.

3.4. Virtual network embedding problem description

The Virtual Network Embedding for a VN request $\mathbb{R}_v = (Rid, G_v, C_v, B_v)$ refers to a mapping $\mathcal{M}$ from the virtual network G_v^{Rid} onto a subset of substrate network G_s with respect to certain constraints.

$$\mathcal{M} : G_v \rightarrow (N_s^{'}, P^{'}, R_N, R_E) \tag{8}$$

where $N_s^{'} \subset N_s$, $P^{'} \subset P_s$, and R_N and R_E denote the substrate node and link resources allocated to VN request $\mathbb{R}_v$.

Different from the offline VN embedding algorithm which supposes all VN requests are defined and known in advance, the online VN embedding algorithm processes the incoming VN requests online once the requests arrive without any foresights about future VN requests, and the allocated resources are released once the VN expires. The challenge is to find the best mapping between current virtual network G_v and the substrate graph G_s so as to achieve a better resource utilization while not hinder future VN allocations and increase the acceptance ratio.

Since the VN embedding involves of the resource allocation both in nodes and links, thus it can be naturally decomposed into two sub-problems: *Virtual Node Mapping* $\mathcal{M}_\mathcal{N}$ and *Virtual Link Mapping* $\mathcal{M}_E$.

3.4.1. Virtual node mapping

Each virtual node of the same VN request $\mathbb{R}_v$ is mapped to a single distinct substrate node by a one-to-one mapping

$$\mathcal{M}_\mathcal{N} : (N_v^{Rid}, C_v) \rightarrow (N_s^{'Rid}, R_N) \tag{9}$$

where $N_s^{'Rid} \subset N_s$. The mapping should satisfy the following primary constraints, for $\forall\ n_v, m_v \in N_v$:

$$\mathcal{M}_\mathcal{N}(n_v) = \mathcal{M}_\mathcal{N}(m_v) \quad \text{iff} \quad n_v = m_v \tag{10}$$

$$C(n_v) \leq \Lambda_N(\mathcal{M}_\mathcal{N}(n_v)) \tag{11}$$

3.4.2. Virtual link mapping

Each virtual link of a VN request $\mathbb{R}_v$ is mapped to a single unsplittable substrate path (single path routing) or a set of multiple splittable substrate paths (multiple path routing) between the substrate nodes on which the virtual nodes of that virtual link are embedded.

$$\mathcal{M}_E : (E_v^{Rid}, B_v) \rightarrow (P^{'Rid}, R_E) \tag{12}$$

where $P^{'Rid} \subset P_s$ and for $\forall\ p \in \mathcal{M}_E(e_v)$ it meets

$$B(e_v) \leq \sum_{p \in \mathcal{M}_E(e_v)} \Lambda_E(p) \tag{13}$$

Fig. 2 illustrates a feasible VN embedding solution for two VN requests.

3.5. Objectives

In this work, the virtual network embedding algorithm is implemented by satisfying multiple objectives under a set of constraints. The ultimate objective is to compute a VN mapping solution that maximizes the *revenue* while reducing the *cost*. Besides, the VN embedding algorithm in this paper is proposed for the online problem, where VN requests arrive and expire/depart over time.

3.5.1. Maximize revenue

This objective is to maximize the economic benefit of accepting VN requests. In order to achieve this goal, the infrastructure providers prefer to allocate resources for more VN requests with a higher VN request acceptance ratio. Similar to the previous works [12,13,15,16], we define the revenue $\mathcal{R}ev$ of a VN request $\mathbb{R}_v = (Rid, G_v, C_v, B_v)$ as the sum of its requested resources:

$$\mathcal{R}ev(\mathbb{R}_v^{Rid}) = \sum_{n_v \in N_v} C(n_v) + \alpha \sum_{e_v \in E_v} B(e_v) \tag{14}$$

where α is a tuning parameter for substrate providers to balance the revenues between the two substrate resources.

3.5.2. Minimize embedding cost

In order to increase the VN request acceptance ratio and thus in turn increase the revenue, the algorithm should minimize the resources (especially the link bandwidth) spent on embedding a VN request, and save more resources to accept more VN requests. Since the cost of node embedding is fixed and deterministic, thus in our two-phase heuristic algorithm we just need to focus on minimizing the cost of link embedding $Cost(E_v^{Rid})$ in the second phase of the algorithm.

4. Multiple objectives linear programming optimization model

In this section, the virtual network embedding problem is formulated as a Multiple Objectives Linear Programming (MOLP) optimization problem. The goal is to find some feasible solutions which aim to optimize the two given objectives while satisfying a series of constraints including capacity constraints, flow constraints, domain constraints and some binary constraints.

4.1. MOLP model

In order to better present the MOLP model, we first introduce several useful variables as below which will be involved in the MOLP problem formulation.

- $f_{uv}^{e_v}$: A flow (i.e., bandwidth) variable denoting the amount of flow from u to v on the substrate path for the virtual link e_v.
- $x_{n_s}^{n_v}$: A binary variable denoting whether or not the virtual node n_v is embedded on the substrate node n_s.
- y^{Rid} : A binary variable denoting whether or not the request with the identifier Rid is accepted.

Then the MOLP optimization problem can be modelled in the following form:

Objectives:

$$\textit{Maximize} \qquad \sum_{Rid} y^{Rid} * \mathcal{R}ev(\mathbb{R}_v^{Rid})$$

$$\textit{Minimize} \qquad \sum_{Rid} \sum_{e_v \in E_v^{Rid}} \sum_{(u,v) \in E_s} (f_{uv}^{e_v} + f_{vu}^{e_v})$$

Constraints:

- Capacity Constraints:

$$\sum_{Rid} \sum_{e_v \in E_v^{Rid}} (f_{uv}^{e_v} + f_{vu}^{e_v}) \leq \Lambda_E(u,v) \quad \forall (u,v) \in E_s \tag{15}$$

$$\sum_{n_v^{Rid} \in N_v} x_{n_s}^{n_v^{Rid}} * C(n_v^{Rid}) \leq \Lambda_N(n_s) \quad \forall n_s \in N_s \tag{16}$$

- Flow Constraints:

$$\sum_{w \in N_s} f_{uw}^{e_v^{Rid}} - f_{wu}^{e_v^{Rid}} = x_u^{s_v^{Rid}} B(e_v^{Rid}) \quad \forall e_v^{Rid} = (s_v^{Rid}, t_v^{Rid}) \tag{17}$$

$$\sum_{w \in N_s} f_{uw}^{e_v^{Rid}} - f_{wu}^{e_v^{Rid}} = -x_u^{t_v^{Rid}} B(e_v^{Rid}) \quad \forall e_v^{Rid} = (s_v^{Rid}, t_v^{Rid}) \tag{18}$$

- Binary Constraints:

$$\frac{1}{M}\sum_{(u,v)\in E_s}\sum_{e_v\in E_v^{Rid}}(f_{uv}^{e_v}+f_{vu}^{e_v})\le y^{Rid}\quad \forall Rid \tag{19}$$

$$y^{Rid}\le\sum_{n_s\in N_s}x_{n_s}^{n_v}\quad \forall Rid,\ \forall n_v\in N_v^{Rid} \tag{20}$$

$$\sum_{n_v\in N_v^{Rid}}x_{n_s}^{n_v}\le 1\quad \forall Rid,\ \forall n_s\in N_s \tag{21}$$

- Domain Constraints:

$$f_{uv}^{e_v^{Rid}}\ge 0\quad \forall Rid,\ \forall (u,v)\in E_s \tag{22}$$

$$x_{n_s}^{n_v^{Rid}}\in\{0,1\}\quad \forall n_s\in N_s,\ \forall n_v^{Rid} \tag{23}$$

$$y^{Rid}\in\{0,1\}\quad \forall Rid \tag{24}$$

where the notations and statements are explained as below:

- The objectives are to maximize the revenue of accepting VN requests, i.e., the summation of revenue over all VN requests with $y^{Rid}=1$, and also to minimize the cost of embedding VN requests, i.e., the summation of bandwidth over all edges embedded with VN requests (note that it is a slight modification of (7))
- Capacity constraint (15) requires all embedded bandwidth on any substrate edge (u, v) not exceed the available bandwidth $\Lambda_E(u, v)$, while (16) requires all allocated CPU resources of any substrate node n_s not exceed the available CPU resources $\Lambda_N(n_s)$
- If $x_u^{s_v^{Rid}}=0$ (or $x_u^{t_v^{Rid}}=0$), meaning that the virtual node s_u^{Rid} (or t_u^{Rid}) is not embedded on the substrate node u, then the flow constraint (17) (or (18)) refers to the flow conservation. On the other hand, if $x_u^{s_v^{Rid}}=1$ (or $x_u^{t_v^{Rid}}=1$), meaning that the virtual node s_u^{Rid} (or t_u^{Rid}) is embedded on the substrate node u, then the flow constraint (17) (or (18)) refers to the completion of the bandwidth allocation for the VN request e_v^{Rid}.
- In the binary constraint (19), M is a sufficient large parameter such that any number divided by M is less than 1. Thus, constraint (19) means whenever any virtual edge of a VN request is embedded on any substrate path, the VN request is accepted and then $y^{Rid}=1$. Constraint (20) means whenever a VN request is accepted, each of its virtual nodes should be embedded on certain substrate node. Constraint (21) means each virtual node from the same VN request is assigned to a different substrate node ensuring the one-to-one node mapping.
- The domain constraint (22) indicate the permitted value range of a valid flow variable. Constraints (23) and (24) are the most basic form of integrity constraints, both of which are binary variables.

4.2. Strategies to solve MOLP problem

Generally, there are two common solution strategies, which are preemptive way and weighted sum method, for the optimization problem with multiple objectives [26–32]. The key strategy of preemptive optimization is to perform the optimization by considering one objective (based on priorities) at a time, and obtain an optimal objective value as a bound. Then use this optimized bound value as a new constraint to optimize the second objective function. This procedure is repeated until all objectives are processed. The final obtained objective value will be an efficient solution to the original multiple objective model. Comparatively, the weighted sum method assigns several appropriate weights to different objectives and converts the multiple objectives into one single objective by summing up all objective functions. The final optimal solution to the converted single objective optimization problem will be a feasible solution to the original multiple-objective problem.

The already existing VNE works involving multiple objectives mostly adopt the weighted sum method [13,17,33,34] by assigning each objective with different weight parameters. On the contrary, the preemptive strategy is applied in our approach. We first consider the objective targeting at maximizing the revenue at node mapping phase, then we optimize the second objective aiming to decrease the cost at link mapping phase based upon the already mapped virtual nodes.

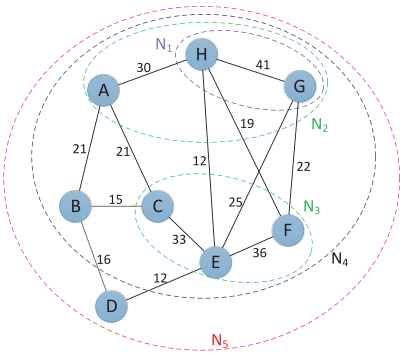

Fig. 3. An example of Blocking Island Graph based on link capacities, in which N_1 is 40-BI, N_2 and N_3 are 30-BIs, N_4 is 20-BI, and N_5 is 10-BI.

4.3. Discussion

Both of the two LP problems are known to be NP-hard, either offline or online. Thus, it is not reasonable to compute an optimal solution in polynomial time, even for offline case. Even the theoretical upper or lower bounds for this general problem do not exist. Nevertheless, we can obtain some near-optimal solutions to specific objectives by adopting efficient heuristic algorithms.

5. Blocking island paradigm

As a resource abstraction model derived from Artificial Intelligence, Blocking Island (BI) was firstly proposed by Christian Frei, et al. in [35] to represent the availability of network link bandwidth. BI is defined as: *A β-Blocking Island (β-BI) for a node x is the set of all nodes of the network that can be reached from x using links with at least β available resources, including x*. In addition to link bandwidth abstraction, in this paper we extend the BI model to abstract the available CPU capacities of nodes as well. The key idea of BI is to abstract the original network graph into a hierarchy tree containing available network resource information at different levels of abstraction. Figs. 3 and 4 illustrate some BIs, for example, N_1 in Fig. 3 represents a 40-BI.

BI holds several fundamental properties which are very useful in resource allocation decidability. Here some of the most important ones are listed without proof (The proof of these are given in [35]).

- **Unicity:** Each node has one unique β-BI. If S is the β-BI for node x, then S is the β-BI for all the nodes in S.
- **Solution existence:** An unallocated virtual link $e_v(src, dest, \beta_e)$ can be satisfied with at least one substrate route if and only if both the endpoints *src* and *dest* are in the same substrate-link-bandwidth-based β_e-BI. Likewise, for an allocated virtual node n_v with required β_n, it can be satisfied with at least one

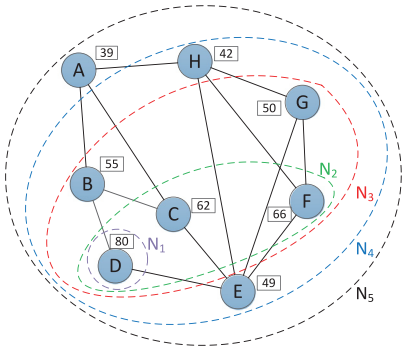

Fig. 4. An example of Blocking Island Graph based on node capacities, in which N_1 is 80-BI, N_2 is 60-BI, N_3 is 50-BI, N_4 is 40-BI, and N_5 is 30-BI.

substrate node iff the substrate-node-CPU-based β_n-BI is not empty.

- **Route location:** The links of a substrate route with β available bandwidth are all in the β-BI of its endpoints.
- **Inclusion:** If β_i is larger than β_j, then the β_i-BI for a node is a subset of β_j-BI for the same node.

The unique β-BI for a given node x can be obtained by a simple greedy algorithm. Algorithm 1 depicts the construction of a link capacity based BI whose complexity is linear in $O(L)$, where L is the number of links. The construction of node capacity based BI is almost the same, with complexity of $O(N)$, where N denotes the number of nodes. The obtained BIs can be used to construct the Blocking Island Graph (BIG), which is a graph abstraction of the entire available network resources. Figs. 3 and 4 exhibit examples of a link-based BIG and a node-based BIG, respectively. A recursive decomposition of BIGs in decreasing order of demands (βs) can be further constructed, and the lowest level has the largest β. This layered BIG structure is named as Blocking Island Hierarchy (BIH). It can be used to identify all bottlenecks, i.e. critical links (inter-links between different BIs), of the network. The BIH can also be viewed as an abstraction tree when taking the father-child relation into consideration. Fig. 5 shows two BIH tree examples for the BIGs of Figs. 3 and 4. The leaves of the BIH tree are the network nodes, and the other vertices denote the abstract BIs. This abstraction tree can reflect the real-time state of the available network bandwidth.

Algorithm 1 Construct β-BI.

```
1: function ConstructBI(G = {V, E}, β)
2:     L ← {∅}                                   ▷ L: Result β-BI list
3:     for all v in V do
4:         if not visited(v) then
5:             I ← ConstructBIFromNode(G, β, v)
6:             L ← L.Add(I)
7:         end if
8:     end for
9:     return L
10: end function

11: function ConstructBIFromNode(G = {V, E}, β, x)
12:     I ← {x}                                   ▷ I: Result β-BI
13:     S ← {links incident to x and weight ≥ β}  ▷ S: stack
14:     while S ≠ ∅ do
15:         l ← pop(S)
16:         e ← another endpoint of l
17:         if e ∉ I and weight(l) ≥ β then
18:             I ← I ∪ {e}
19:             S ← S ∪ {links incident to e and weight ≥ β}
20:         end if
21:     end while
22:     return I
23: end function
```

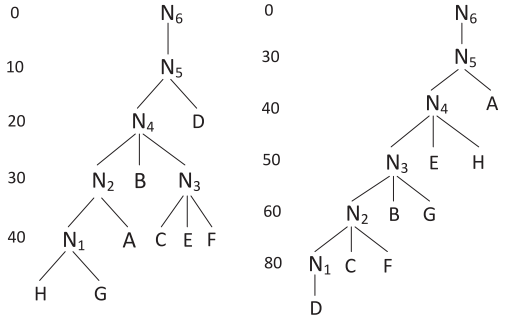

Fig. 5. BIH trees for BIGs of Figs. 3 and 4, and N_0 indicates 0-BI.

The BI paradigm provides an extremely efficient way to represent the availability of network resources, and greatly reduces the search space and computation complexity. For example, to embed a node requiring 50 CPU capacity, the satisfaction decision can be made just by checking if the 50-BI is empty with complexity of $O(1)$ without computing and checking all the nodes' status. For another example, when embedding a VN request we need to check whether its virtual links with required bandwidth can be satisfied, the traditional way is to firstly spend lots of time in searching the entire network space and computing the feasible routes for every virtual link with very high exponential computation complexity then to decide if this VN request can be satisfied. Comparatively, the BI-based approach only needs to check if the endpoints of every virtual link are in the same corresponding β-BI with computation complexity $O(1)$ since only two hashing operations are needed. One may concern about the time cost in the maintenance of BIs. However, we only need to update the BIs which are involved in allocations or deallocations by means of splitting or merging operations with complexity of $O(n)$ or $O(l)$, where n and l denote the number of involved nodes and links, respectively. Besides, according to the experimental results the overall computation efficiency is still greatly improved though the maintenance of BI may take some time. Actually, traditional VNE algorithms also should maintain the global knowledge of node/link states up to date.

6. *Presto*: BI-based online heuristic virtual network embedding framework

As aforementioned, the virtual network embedding (VNE) is an NP-hard problem and the traditional approaches suffer from a high complexity due to the huge searching space consisting of exponential number of nodes and routes. In response to this issue, with the benefit of BI model, we propose an efficient BI based heuristic framework, named *Presto*, to solve the online virtual network embedding problem with much lower and more manageable complexity.

6.1. Overview on Presto framework

In practice, the arrival interval of VN requests behaves randomly. The VN request may arrive one after another in a certain interval, but it also may happen that multiple VN requests arrive almost at the same time. Thus, designed as an online virtual network embedding algorithm, in addition to the ability of processing VN requests one by one, *Presto* adopts sliding window-based approach to batch process the multiple arriving VN requests online. This sliding window-based technique was initially used in the

Table 1
Summary of algorithms.

Algorithm	Description
HNM	Heuristic node mapping algorithm
HLM	Heuristic link mapping algorithm
HVNO	Heuristic virtual node ordering algorithm
HVLO	Heuristic virtual link ordering algorithm
DHRO	Dynamic heuristic VN request ordering algorithm

work [13]. It evolves from the packet-level sliding window based data transmission protocols such as Transmission Control Protocol (TCP). The difference is that the window size in *Presto* is measured in time units other than buffer size. Each arriving VN request $\mathbb{R}_v$ is assigned with a unique identifier *Rid* to be denoted as $\mathbb{R}_v^{Rid}$. *Presto* queues all the arrived VN requests in a sliding window with certain size, and then optimize resource allocation by processing them together based on a priority metric (i.e. revenue, cost). The online batch processing manner not only provides the capability of looking ahead and ordering the requests in the most beneficial way, but also enables the forward checking ability offered by BI which helps increase the acceptance ratio.

In order to solve the formulated MOLP problem (Section 4) more efficiently and better optimize the allocation of VN requests, *Presto* decomposes the VN embedding problem into two phases:

1. Firstly, *Presto* applies a heuristic node mapping algorithm (HNM) to map the virtual nodes of a VN request to a set of most advantageous substrate nodes targeting at increasing the acceptance ratio and maximizing revue. Besides, a virtual node ordering algorithm (HVNO) is designed before node mapping.
2. Secondly, *Presto* uses a heuristic link mapping algorithm (HLM) to compute the most beneficial routes between the allocated substrate nodes to allocate every virtual links with required bandwidth aiming to minimize the embedding cost. Moreover, a virtual link ordering algorithm (HVLO) is designed before link mapping.
3. Notably, before embedding, a dynamic heuristic VN request ordering algorithm (DHRO) should be executed to determine the VN embedding order.

All the involved algorithms for virtual network embedding are summarized in Table 1.

6.2. Variable ordering

As a common issue for the constraint satisfaction problem (CSP), it cannot be guaranteed to find an assignment to satisfy all the variables (virtual nodes, virtual links, VN requests) all the time. How to select next variable from the set of unallocated variables to allocate has a great impact on the overall allocation success ratio, and also impacts the computation and search efficiency on the whole. This issue exists in every step of VN embedding procedure, including node mapping phase (the order of virtual node selection from a certain VN request), link mapping phase (i.e. which virtual link should be allocated first), and VN request selection (how to sort the VN requests queued in the same window). As shown in [36], the common way for this kind of CSP is to use fail-first principle based technique which tries those tests in the given set of tests that are most likely to fail: "*To succeed, try first where you are most likely to fail*". There are also some static techniques, such as to first select the demand with the greatest value. However, they are not suitable for the online VN embedding problem in data center networks which require more efficient mechanism in a dynamic way. Based on this observation, a set of variable ordering mechanisms are carefully designed for virtual node ordering, virtual link ordering and VN request ordering. The basic intuition is to decrease the domain of search tree and to prune the tree branches that cannot lead to a feasible solution as early as possible when choosing a variable.

6.3. Node mapping algorithm

The heuristic node mapping algorithm HNM is designed to map each virtual node of a VN request to the substrate nodes with sufficient CPU capacities. For each virtual node with required β CPU capacity, before node mapping HNM firstly checks whether the virtual node is in a β-BI or higher level BIs (referring to BI's Inclusion property) according to the solution existence property. If there is any virtual node of the VN request that cannot be satisfied, then *Presto* rejects the request for this time and postpones it to the next window for processing to see if any resources are released because of some VN expires. If all virtual nodes can be satisfied with their required capacities based on BI checking, then HNM manages the resource allocation by mapping virtual nodes to capacitated substrate nodes.

6.3.1. Virtual node ordering

Before embedding a certain individual virtual network request, its virtual nodes are needed to be sorted firstly, through a heuristic ordering algorithm named HVNO, and then the requests are sequentially allocated. HVNO sorts the VN requests as below.

(i) *Presto* prefers to choose the virtual node with β required CPU capacities, where the β-level substrate node-based BI contains fewest substrate nodes. It is derived from the perspective of limited network resources, and follows the first-fail principle since the fewer nodes in the domain the fewer the feasible solutions.
(ii) If there are multiple such virtual nodes, the one with larger required CPU capacities is given a higher priority for selection. This also abides by the fail-first law.
(iii) Finally, *Presto* randomly sorts the virtual nodes that still have the same order.

For each virtual network $G_v = (N_v^{Rid}, E_v^{Rid})$, *Presto* recursively selects one virtual node n_v from N_v to allocate in the above order until all the virtual nodes are allocated. Through sorting the order of allocating virtual nodes, the search tree is largely pruned and the search space is decreased, which greatly improves the computation and search efficiency.

6.3.2. Node mapping

In order to increase the acceptance ratio while decreasing computation time, HNM also selects substrate nodes with priorities for the virtual nodes to embed. The basic principle of HNM is to preferentially choose the substrate node that causes minimum BI splittings after allocation if the splitting cannot be avoidable. For each virtual node, HNM selects the qualified substrate node to embed as follows:

(i) Choose the substrate node, within the β-BI, that causes no splitting. This intends to avoid BI splitting where BI splitting may result in more future request allocation failures and higher computation cost of updating BIs.
(ii) If the splitting is unavoidable then choose the substrate node that causes fewest BI splittings.
(iii) If there are multiple such substrate nodes then select the substrate node n_s which has the highest available resource $\Lambda_N(n_s)$. The intuition behind this rule is that HNM is prone to use less critical nodes and increase the success ratio of future resource allocations.

HNM finishes the node selection process as long as the output of any of the above 3 procedures is unique, and embeds the virtual node to the selected substrate node. Then HNM recursively chooses the next unallocated virtual node to embed until all the virtual nodes of the VN request are allocated successfully.

HNM performs as the first step towards an efficient virtual network embedding. The key advantages of HNM are mainly revealed in the following aspects.

(i) Prior to VN embedding, with the benefit of BI, HNM makes faster decisions on the acceptance of a VN request than traditional approaches.
(ii) HNM is devoted to increasing the acceptance ratio and takes the future allocation into consideration.
(iii) Higher acceptance ratio induces higher revenue. This is the first step to maximize the revenue.
(iv) Following the minimum-splitting principle, HNM reduces much time cost in updating the involved BIs and further increases computation efficiency.

6.4. Link mapping algorithm

After mapping the virtual nodes of a VN request, the heuristic link mapping algorithm HLM is then applied to map each virtual link of the VN request to the substrate path with sufficient bandwidth. Similarly, for each virtual link of the VN request, before mapping virtual links HLM also needs to check if there are any satisfied substrate paths between the corresponding substrate nodes which host the end virtual nodes of the virtual link. The traditional approaches usually firstly must search the entire network space and compute the possible routes with an exponential complexity $O(|N_s|^x)$, and then to make a decision if the virtual link can be embedded or not. Comparatively, by applying BI model HLM just needs to check if the two end embedded substrate nodes are in the same β-BI, where β indicates the required bandwidth of that virtual link, through a simple hashing operation in $O(1)$ time, which greatly improves the computation efficiency.

6.4.1. Virtual link ordering

Similarly, in order to increase the overall acceptance ratio, we also propose a heuristic ordering algorithm for virtual link selections, named HVLO, which works as below.

(i) *Presto* intends to first choose the virtual link where the Lowest Common Father of its corresponding embedded end substrate nodes (LCF_{sn}) in BIH tree is highest. The intuition behind this rule is to first allocate the virtual link whose mapped substrate path will pass through more critical substrate links. The higher the LCF_{sn} the more constrained the virtual link, which complies with the fail-first principle. For example, as shown in Figs. 3 and 5 (left), suppose there are two virtual links l_1 and l_2 need to be allocated, the end virtual nodes of l_1 are embedded on substrate nodes *A* and *H* while the end virtual notes of l_2 are mapped on substrates nodes *G* and *F*, then the virtual link l_2 will be preferred to be firstly allocated because the LCF_{sn} of (*A, H*) and (*G, F*) are N_2 and N_4, respectively, and N_4 is the highest one.
(ii) If there are multiple virtual links with the same value for the first step, then *Presto* gives a higher priority to the virtual link whose LCF_{sn} contains fewer nodes. This also follows the fail-first principle.
(iii) In case there are still more than one virtual links with the same order, then *Presto* prioritizes the one with higher required link capacities.
(iv) Finally, *Presto* randomly sort the virtual links that still have the same order.

For each virtual network $G_v = (N_v^{Rid}, E_v^{Rid})$, after embedding the virtual nodes *Presto* then recursively maps virtual links e_v from E_v to a substrate path in the above order until all the virtual links are embedded. Likewise, the resulted pruned search tree after sorting greatly decreases the search domain and increases the computation efficiency.

6.4.2. Link mapping

If all the virtual links can be satisfied after BI-based acceptance checking, then HLM performs the link mapping to assign a best route for each virtual link in the prearranged order. However, the domain is too large and the route set is too time-consuming to be computed. In order to further improve the search efficiency and select the best route as fast as possible, we need to generate the routes in the most advantageous order as well. Thus, for each virtual link, several rules are carefully regulated for route selection:

(i) As few critical links (inter-links between different BIs) as possible should be involved in the substrate route. This not only aims to reduce the failure ratio of future allocations, but also to decrease the computation cost of updating BIs.
(ii) The route prefer to choose the shortest path which targets at minimizing the link mapping cost $Cost(E_v^{Rid})$.
(iii) The allocation for current virtual link should impact the future allocation as little as possible.
(iv) The computation cost should be as low as possible.

In accordance with these directive guidelines, HLM strictly follows the procedures as below, for each virtual link:

(i) Firstly, search the lowest level (with the largest β) BI in which both the two corresponding end substrate nodes (which host the end virtual nodes of the virtual link) are clustered, and generate a set of candidate substrate routes. For example, in the link-based BIG or BIH as shown in Figs. 3 and 5, the lowest level BI for end nodes *G* and *A* is N_2, and for *A* and *C* it is N_4. This fashion is prone to use as less critical bottleneck links (inter-BI links) as possible thus increasing the success ratio of future allocations. Eventually, this can help achieve a tighter network with better link utilization. This step follows the first and the third rules.
(ii) Secondly, choose the shortest substrate path from the candidate routes. This step aims to decrease the reserved link bandwidth cost, which follows the second rule.
(iii) If there are still multiple such substrate paths, choose the one that causes minimum BI splitting after resource allocation. This step is in line with fourth rule.
(iv) Finally, if there are still more than one such paths, we randomly choose one from the candidate routes.

HLM achieves almost the same benefits as demonstrated in HNM, such as faster decisions, better acceptance ratio, and higher computation efficiency. Moreover, HLM regards the embedding cost as its primary optimization objective, and prefers to choose a substrate path with lowest cost.

6.5. Sliding window-based batch processing

If the arrival interval of VN requests is short or even multiple VN requests arrive simultaneously, then the fashion of batch processing will be more favorable and beneficial. Therefore, *Presto* applies a sliding window based method to batch process several requests together at the end of the window time. In this way, when mapping VN requests *Presto* can lookahead within the sliding window and order the requests in a most beneficial way for further resource allocation, which can achieve a higher acceptance ratio. Furthermore, *Presto* can also do forward checking with the help of

BI model when allocating one VN request. Each VN request is associated with an additional field named as *maximum standing time* T_s, which indicates the maximum time a VN request can wait in the queue for processing.

6.5.1. VN request ordering

Before embedding, the coming VN requests within the same window period need to be ordered in a proper way. Thus, we propose a dynamic heuristic approach, denoted as DHRO, to determine the order of selecting VN requests taking the optimization objectives into consideration. The principle of DHRO to sort the VN requests within a sliding window is: (i) to give higher priorities to the VN requests gaining higher revenues and (ii) to increase the acceptance ratio as much as possible, both of which can contribute to increasing the overall revenue.

Based on these careful observations, the criterion of VN request ordering in the priority queue is proposed as below:

(i) Firstly, the VN request $\mathbb{R}_v^{Rid}$ with the highest revenue $\mathcal{R}ev(\mathbb{R}_v^{Rid})$ is given the highest priority and will be firstly considered. This aims to maximize the economic revenue.
(ii) Secondly, if there are multiple VN requests with the same revenue, *Presto* assigns a higher priority to the one that requests higher link bandwidth resources $\sum_{e_v \in E_v} B(e_v)$ since the network bandwidth is a more constrained resource [37] than CPU resources in data centers. This follows the first-fail principle.
(iii) Thirdly, if there are still more than one requests with the same order, then the VN request, whose maximum standing time T_s is minimum, is preferentially selected. Consequently, this rule favors more urgent requests and decreases the overall allocation failure ratio.
(iv) Otherwise, keep the current sequence unchanged.

The dynamic heuristic approach DHRO sorts the VN requests in the most beneficial order, which can help make better embedding decisions. DHRO not only can increase the overall revenue, but also can increase the acceptance ratio.

6.5.2. Batch processing with lookahead

Different from individual VN embedding at its arrival, *Presto* segments time into consecutive window-based time units, and all the coming VN requests within the same window time will be processed together at the end of the window. In this way, *Presto* is enabled with lookahead abilities which can take the future requests within the same window into consideration while making embedding decisions for current request, which can help make a better decision increasing the overall acceptance ratio. The working procedure of *Presto* is as below:

(i) At the end of each window period, DHRO algorithm is applied to sort the arrived VN requests within the window.
(ii) After sorting, *Presto* processes the requests in order. For each VN request whose maximum standing time T_s does not expire, it is recursively processed as below:
  – Firstly, HVNO algorithm is applied to sort the virtual nodes of the virtual network.
  – Then, HNM algorithm is used to embed the sorted virtual nodes onto substrate nodes in order.
  – Afterwards, HVLO algorithm is adopted to sort the virtual links of the virtual network.
  – Finally, HLM algorithm is utilized to map the virtual links onto substrate paths in the prearranged order.
(iii) Place the unsuccessful embedded VN requests to the next window, and repeat the procedure from step (i).

The last step means that if in case any VN request cannot be satisfied with required resources currently, then it will be deferred to be processed in the next window period expecting that some already allocated VNs depart and release enough resources if it is not expired yet. The whole working procedure of *Presto* is presented in Algorithm 2.

Algorithm 2 The online VN embedding algorithm *Presto*.

```
1: procedure PRESTO ALGORITHM
2:     T_win = Window_size
3:     Wnd_next ← {}
4:     loop
5:         T_0 ← Current time
6:         Wnd_cur ← Wnd_next
7:         Wnd_next ← {}
8:         repeat
9:             Wnd_cur ← New arriving R_v = (Rid, G_v, C_v, B_v)
10:            T_cur ← Current time
11:        until T_cur − T_0 ≥ T_win          ▷ The current window expires
12:        Apply DHRO algorithm to sort Wnd_cur
13:        for all R_v^Rid ∈ Wnd_cur do
14:            if T_s(R_v^Rid) expires before T_cur then
15:                Reject R_v^Rid
16:            else
17:                Sort virtual nodes using HVNO algorithm
18:                HNM embeds virtual nodes with forward checking
19:                if HNM fails to embed R_v^Rid then
20:                    Add R_v^Rid to Wnd_next
21:                end if
22:                Sort virtual links using HVLO algorithm
23:                HLM embeds virtual links with forward checking
24:                if HLM fails to embed R_v^Rid then
25:                    Add R_v^Rid to Wnd_next
26:                end if
27:            end if
28:        end for
29:    end loop
30: end procedure
```

6.5.3. Forward checking

In addition to the lookahead ability, with the benefit of BI paradigm *Presto* is also enabled with the capacity of forward checking in both node mapping phase and link mapping phase. Taking advantage of BI's solution existence property, we know at any point in the search if the substrate network is still possible to allocate a demand (virtual node or virtual link) just by checking if the BI at the level of its required capacity in $O(1)$ time with one hashing operation, without having to compute solutions. Therefore, after allocating a virtual node or virtual link, *Presto* executes forward checking to examine the satisfaction of all unallocated virtual nodes or virtual links of the current virtual network. If there exists any one demand (virtual node or virtual link) that cannot be satisfied, another allocation solution must be tried for the current virtual node or virtual link. The capacity of forward checking ensures *Presto* with a better embedding decision and higher acceptance ratio.

7. Evaluation

In order to evaluate the performance of *Presto*, we implemented a *Presto* prototype by extending our DCNSim simulator [38] with some accompanying Python scripts. We firstly give an overview on the simulation environment. Then we demonstrate the evaluation results of *Presto* with respect to some performance metrics including acceptance ratio, gained revenue, embedding cost and computation efficiency.

7.1. Evaluation settings

7.1.1. Substrate network

In the experiment, we use SprintNet [39,40] topology as the substrate network topology. The substrate resources (CPU and link bandwidth) were uniformly distributed between 50 and 150.

7.1.2. Virtual network

The virtual network topologies were randomly generated following similar setups in previous work [12,13,15,16], where the number of virtual nodes follows a uniform distribution between 2 and 10, and each pair of virtual nodes are randomly connected with probability 0.5. The requested CPU capacity was uniformly distributed between 0 and 30, and the bandwidth is between 0 and 50; The arrival rate of VN requests (the number of coming VN requests per window period) was determined by the Poisson process varying from 4 to 10 VN requests per window. The maximum standing time T_s of a VN request conforms to an exponential distribution between 1 and 5 window time on average, while the duration of a VN request conforms to an exponential distribution with 10 time windows on average. One window time was primarily defined as 100 time units.

7.1.3. Compared algorithm

It is difficult to compare our algorithm with other proposed algorithms due to different problem formulation and different objectives with different strategies (e.g. online/offline, admission control, one/two stages). For the sake of fairness, the compared algorithms are two-staged and uncoordinated algorithms. Consequently, we modified four existing algorithms fitting in our model to facilitate comparisons. The first algorithm we compared with is the baseline algorithm of the work [16], denoted as VNE-Yu, which uses greedy node mapping algorithm with k-shortest path link mapping algorithm only considering the unsplittable flow problem. Another compared algorithm is [17], denoted as VNE-SP, which employs greedy node mapping algorithm with shortest path link mapping algorithm without reconfiguration. The third compared algorithm is named RW-MaxMatch proposed in [12]. This is also a two-staged algorithm, which firstly maps the virtual nodes and then embeds the virtual links by finding shortest paths with unsplittable paths. Moreover, we also implemented a modified version of [18], denoted as vnmFlib, to facilitate comparisons, where it allows multiple VN requests with dynamic arriving rates, and disables path splitting, with β=90, ϵ=10.

7.2. Evaluation results

7.2.1. Average revenue over time

The gained revenue varies under different conditions (e.g. VN request arrival rate, the value of tuning parameter α defined in Eq. 14). In this simulation, we use the average gained revenue over time (i.e. $\lim_{T\to\infty} \frac{\sum_{t=0}^{T} \mathcal{R}ev(\mathbb{R}_v(t))}{T}$) to evaluate the overall average revenue.

Fig. 6 shows the gained average revenue over time under different Poisson distribution means $E(rate)$ of VN arrival rates. The tuning α is set to be 0.5 as default. The evaluation result reveals that higher VN arrival rate yields higher revenue in the long run. Besides, *Presto* achieves more revenues on average than VNE-Yu, RW-MaxMatch and VNE-SP algorithms where the increment ranges from 6.21% to 19.64%. Comparatively, the evaluation results shown in Fig. 7 present the gained average revenues with different weights α to the revenue induced by bandwidth $\Sigma B(e_v)$. The mean of arrival rate is fixed to be six requests per time window as default. It can be seen that higher α results in higher revenue on the whole, where from another respective the substrate bandwidth resource is more scarce than CPU resources during the VN embeddings. As well *Presto* still achieves a better performance than other two comparison algorithms for different values of α.

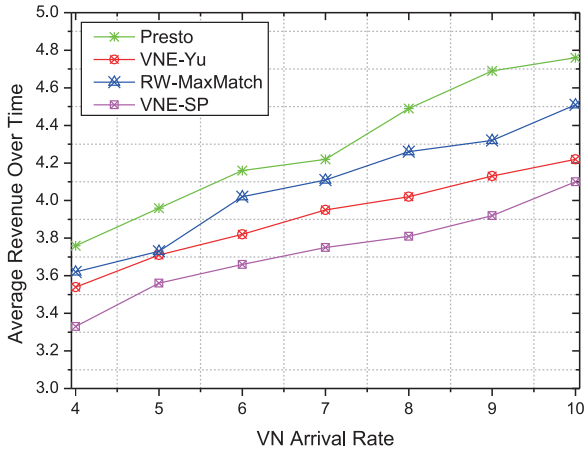

Fig. 6. The average revenue under different VN arrival rates.

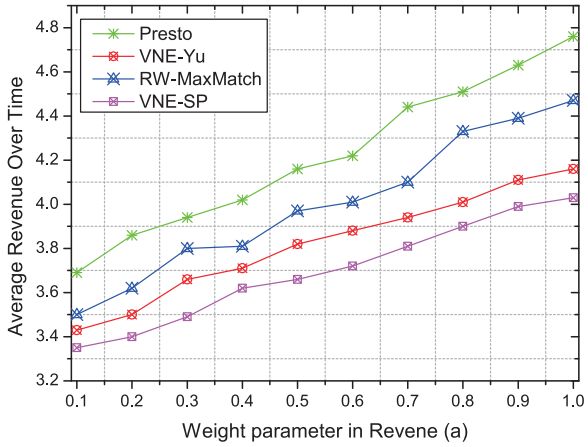

Fig. 7. The average revenue with different values of tuning parameter α.

7.2.2. Average embedding cost

As aforementioned in Section 3, the CPU cost is deterministic and remains the same after VN embeddings, while the bandwidth cost stays uncertain and depends on the way of VN embedding. Therefore, in this experiment we use the average bandwidth cost to evaluate the embedding cost. Fig. 8 exhibits the induced bandwidth costs of VN embeddings under different VN arrival rates with a fixed α=0.5. The result shows that higher VN arrival rate can help reduce the embedding cost, where a higher VN arrival rate means more VN requests can be processed together which can help make a better embedding decision leading to less embedding cost and higher revenue. Benefit from virtual link/node ordering/selection strategies proposed in HNM and HLM, *Presto* causes less embedding cost than the compared algorithms as shown in Fig. 8, which in turn increases the acceptance ratio (Fig. 9) and overall revenue (Figs. 6 and 7).

7.2.3. Acceptance ratio

From a certain perspective, the acceptance ratio, gained revenue and embedding cost are interrelated with each other. A lower embedding cost may lead to a higher acceptance ratio, which in turn increases the overall revenue. The evaluation results shown in Fig. 9 further prove the derivation. As can be seen, *Presto* achieves 1% ∼ 7%, 10% ∼ 15% and 20% ∼ 25% higher acceptance ratios than

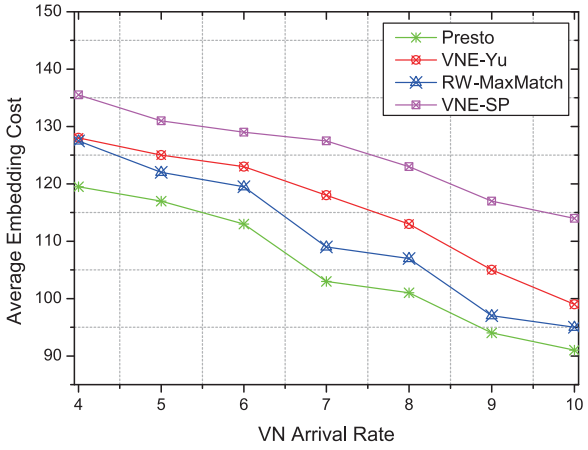

Fig. 8. The average bandwidth cost.

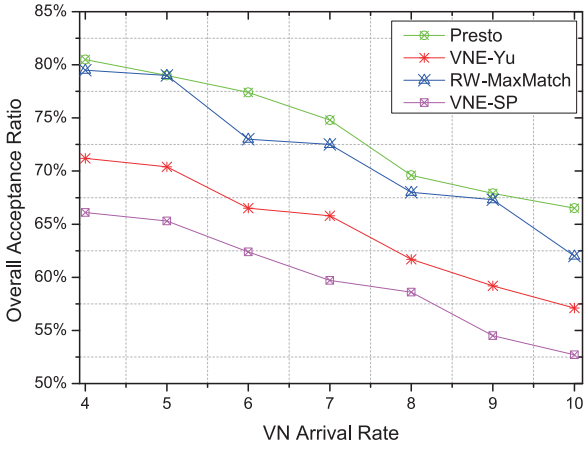

Fig. 9. The overall acceptance ratio.

RW-MaxMatch, VNE-Yu and VNE-SP on the whole, respectively, reckoning in the failed VN requests due to expiration.

7.2.4. Computation efficiency

Another big advantage of our framework *Presto* lies in its computation efficiency. Generally, the traditional algorithms have a bad exponential time complexity due to the huge searching space resulting in high computation complexity. Comparatively, in *Presto*, with the advantage of BI model, the heuristic node/link selection algorithms together with the variable ordering mechanism are purposefully designed to guide the resource allocation and search, and greatly improves the computation efficiency by reducing the search space.

In order to better demonstrate the computation efficiency of *Presto* and compare with other algorithms namely VNE-Yu, vnmFlib, and VNE-SP, we offline run 500 instances of VN embeddings and calculate the computation time spent on completing all allocations. We assume the substrate resources (i.e. CPU and bandwidth) are unlimited so as to guarantee the 100% allocation. Fig. 10 gives the simulation results, which reveal that *Presto* is around 1.5 times faster than vnmFlib and two times faster than both VNE-Yu and VNE-SP on average. Approximately 45% allocations can be completed within one second and 100% in 2.47 s using *Presto*, while vnmFlib, VNE-Yu and VNE-SP only complete 33.1%, 14.8% and 12.7% within one second, respectively, and cost 3.6 s, 5.2 s and 5.7 s to finish 100% allocations, respectively. This definitely proves the high computation efficiency of *Presto*.

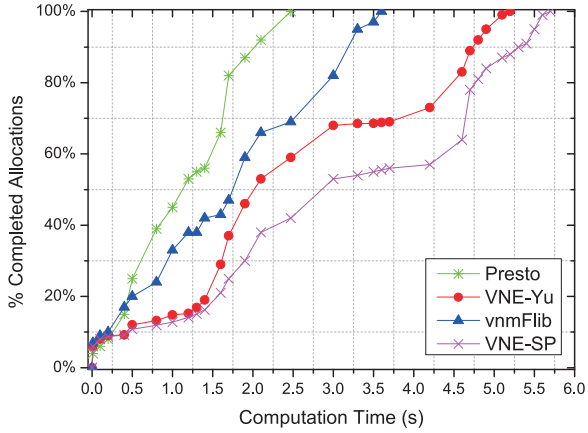

Fig. 10. The performance of computation efficiency.

8. Conclusion and future work

This paper aims to achieve an efficient online virtual network embedding algorithm in virtualized cloud data centers. In order to deal with the computationally intractable VNE problem, which is known as NP-hard, we formulated it as an MOLP problem with multiple practical objectives and designed an efficient VNE framework *Presto* consisting of a series of heuristic algorithms such as DHRO, HVNO, HVLO, HNM and HLM. With the benefit of Blocking Island paradigm derived from an AI model, *Presto* achieves a good performance in various aspects including revenue, embedding cost, acceptance ratio and computation efficiency. To the best of our knowledge, we are the first to apply BI model to deal with VNE problem. The extensive simulation results have witnessed the effectiveness of *Presto* adopting BI model. After all, there are still some open issues. For example, what the performance of *Presto* will be if path splitting and migration is allowed? How will the window size affect the acceptance ratio, overall revenue and embedding cost? How's Presto's performance if coordinated node and link mapping approach is applied? These issues are left for our future work. Besides, in future work more sophisticated recently proposed VNE algorithms will be implemented to conduct more comprehensive analysis and comparisons with our algorithm.

References

[1] T. Wang, Z. Su, Y. Xia, M. Hamdi, Rethinking the data center networking: Architecture, network protocols, and resource sharing, Acc. IEEE 2 (2014) 1481–1496.
[2] Q. Yan, R. Yu, Q. Gong, J. Li, Software-defined networking (sdn) and distributed denial of service (ddos) attacks in cloud computing environments: A survey, some research issues, and challenges, IEEE Commun. Surv. Tut. (2015). 1–1
[3] T. Wang, M. Hamdi, Jielin: A scalable and cost-effective server-centric data center network architecture, in: 2015 IEEE Global Communications Conference (GLOBECOM), IEEE, 2015, pp. 1–6.
[4] T. Wang, Y. Xia, D. Lin, M. Hamdi, Improving the efficiency of server-centric data center network architectures, in: Communications (ICC), 2014 IEEE International Conference on, IEEE, 2014, pp. 3088–3093.
[5] T. Wang, Z. Su, Y. Xia, B. Qin, M. Hamdi, Novacube: A low latency torus-based network architecture for data centers, in: Global Communications Conference (GLOBECOM), 2014 IEEE, IEEE, 2014, pp. 2252–2257.
[6] T. Anderson, L. Peterson, S. Shenker, J. Turner, Overcoming the internet impasse through virtualization, Computer 38 (4) (2005) 34–41.
[7] M.F. Bari, R. Boutaba, R. Esteves, L.Z. Granville, M. Podlesny, M.G. Rabbani, Q. Zhang, M.F. Zhani, Data center network virtualization: A survey, Commun. Surv. Tut. IEEE 15 (2) (2013) 909–928.
[8] A. Fischer, J.F. Botero, M. Till Beck, H. De Meer, X. Hesselbach, Virtual network embedding: A survey, Commun. Surv. Tut. IEEE 15 (4) (2013) 1888–1906.
[9] G. Su, B. Chen, X. Lin, H. Wang, L. Li, Qos constrained optimization of cell association and resource allocation for load balancing in downlink heterogeneous cellular networks, Ksii Trans. Internet Inf. Syst. 9 (5) (2015) 1569–1586.
[10] J. Li, Z. Ming, M. Qiu, G. Quan, X. Qin, T. Chen, Resource allocation robustness in multi-core embedded systems with inaccurate information, J. Syst. Arch. 57 (9) (2011) 840–849.

[11] L. Wang, K. Wu, J. Xiao, M. Hamdi, Harnessing frequency domain for cooperative sensing and multi-channel contention in crahns, Wireless Commun. IEEE Trans. 13 (1) (2014) 440–449.
[12] X. Cheng, S. Su, Z. Zhang, H. Wang, F. Yang, Y. Luo, J. Wang, Virtual network embedding through topology awareness and optimization, Comput. Netw. 51 (6) (2012) 1797–1813.
[13] N.M.K. Chowdhury, M.R. Rahman, R. Boutaba, Vineyard: Virtual network embedding algorithms with coordinated node and link mapping, IEEE/ACM Trans. Netw. (TON) 20 (1) (2012) 206–219.
[14] J. Lu, J. Turner, Efficient mapping of virtual networks onto a shared substrate, Washington University in St. Louis, Tech. Rep (2006).
[15] M.R. Rahman, I. Aib, R. Boutaba, Survivable virtual network embedding, in: NETWORKING 2010, Springer, 2010, pp. 40–52.
[16] M. Yu, Y. Yi, J. Rexford, M. Chiang, Rethinking virtual network embedding: substrate support for path splitting and migration, ACM SIGCOMM Comput. Commun. Rev. 38 (2) (2008) 17–29.
[17] Y. Zhu, M.H. Ammar, Algorithms for assigning substrate network resources to virtual network components., in: INFOCOM, 2006, pp. 1–12.
[18] J. Lischka, H. Karl, A virtual network mapping algorithm based on subgraph isomorphism detection, in: Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures, ACM, 2009, pp. 81–88.
[19] J.M. Kleinberg, Approximation algorithms for disjoint paths problems, Citeseer, 1996 Ph.D. thesis.
[20] S.G. Kolliopoulos, C. Stein, Improved approximation algorithms for unsplittable flow problems, in: Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on, IEEE, 1997, pp. 426–436.
[21] I. Fajjari, N. Aitsaadi, G. Pujolle, H. Zimmermann, Vne-ac: Virtual network embedding algorithm based on ant colony metaheuristic, in: Communications (ICC), 2011 IEEE International Conference on, IEEE, 2011, pp. 1–6.
[22] I. Houidi, W. Louati, D. Zeghlache, A distributed virtual network mapping algorithm, in: Communications, 2008. ICC'08. IEEE International Conference on, IEEE, 2008, pp. 5634–5640.
[23] T. Wang, B. Qin, M. Hamdi, An efficient framework for online virtual network embedding in virtualized cloud data centers, in: 4th IEEE International Conference on Cloud Networking (CloudNet), IEEE, 2015.
[24] M. Till Beck, A. Fischer, H. de Meer, J.F. Botero, X. Hesselbach, A distributed, parallel, and generic virtual network embedding framework, in: Communications (ICC), 2013 IEEE International Conference on, IEEE, 2013, pp. 3471–3475.
[25] M. Till Beck, A. Fischer, J.F. Botero, C. Linnhoff-Popien, H. de Meer, Distributed and scalable embedding of virtual networks, J. Netw. Comput. Appl. 56 (2015) 124–136.
[26] J.R. Chromy, Design optimization with multiple objectives, in: Proceedings of the section on survey research methods, 1987, pp. 194–199.
[27] F. Lootsma, Optimization with multiple objectives(1988).
[28] R.T. Marler, J.S. Arora, Survey of multi-objective optimization methods for engineering, Struct. Multidisciplin. Optim. 26 (6) (2004) 369–395.
[29] Y. Wilamowsky, S. Epstein, B. Dickman, Optimization in multiple-objective linear programming problems with pre-emptive priorities, J. Oper. Res. Soc. (1990) 351–356.
[30] G.Q. Zeng, J. Chen, Y.X. Dai, L.M. Li, C.W. Zheng, M.R. Chen, Design of fractional order pid controller for automatic regulator voltage system based on multi-objective extremal optimization, Neurocomputing 160 (C) (2015) 173–184.
[31] Z. Liang, R. Song, Q. Lin, Z. Du, J. Chen, Z. Ming, J. Yu, A double-module immune algorithm for multi-objective optimization problems, Appl. Soft Comput. 35 (C) (2015) 161–174.
[32] J. Li, M. Qiu, Z. Ming, G. Quan, X. Qin, Z. Gu, Online optimization for scheduling preemptable tasks on iaas cloud systems, J. Parallel Distrib. Comput. 72 (5) (2012) 666–677.
[33] C. Papagianni, A. Leivadeas, S. Papavassiliou, V. Maglaris, C. Cervelló-Pastor, A. Monje, On the optimal allocation of virtual resources in cloud computing networks, Comput. IEEE Trans. 62 (6) (2013) 1060–1071.
[34] Y. Xin, I. Baldine, A. Mandal, C. Heermann, J. Chase, A. Yumerefendi, Embedding virtual topologies in networked clouds, in: Proceedings of the 6th International Conference on Future Internet Technologies, ACM, 2011, pp. 26–29.
[35] C. Frei, B. Faltings, Simplifying network management using blocking island abstractions, Internal Note from the IMMuNe Project, April (1997).
[36] R.M. Haralick, G.L. Elliott, Increasing tree search efficiency for constraint satisfaction problems, Artif. Intel. 14 (3) (1980) 263–313.
[37] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113.
[38] Y. Liu, J. Muppala, Dcnsim: A data center network simulator, The 3rd international workshop on Data Center Performance (DCPerf), Philadelphia, USA, IEEE, 2013.
[39] T. Wang, Z. Su, Y. Xia, Y. Liu, J. Muppala, M. Hamdi, Sprintnet: A high performance server-centric network architecture for data centers, in: Communications (ICC), 2014 IEEE International Conference on, IEEE, 2014, pp. 4005–4010.
[40] T. Wang, Z. Su, Y. Xia, J. K. Muppala, M. Hamdi, Designing efficient high performance server-centric data center network architecture, Comput. Netw. 79 (2015) 283–296.

Ting Wang received the Ph.D. degree in Computer Science and Engineering from Hong Kong University of Science and Technology, Hong Kong SAR China, in 2015, the Master Eng. degree from Warsaw University of Technology, Poland, in 2011, and the Bachelor Sci. degree from University of Science and Technology Beijing, China, in 2008. From 02.2012 to 08.2012 he visited as a research assistant in the Institute of Computing Technology, Chinese Academy of Sciences, China. He is currently a research scientist in Alcatel Lucent Bell Labs, Shanghai, China. His research interests include data center networks, cloud computing, green computing, network function virtualization, software defined network and next generation networks.

Mounir Hamdi received the B.S. degree in Electrical Engineering – Computer Engineering minor (with distinction) from the University of Louisiana in 1985, and the MS and the PhD degrees in Electrical Engineering from the University of Pitts-burgh in 1987 and 1991, respectively. He is a Chair Professor at the Hong Kong University of Science and Technology, and was the head of department of computer science and engineering. Now he is the Dean of the College of Science, Engineering and Technology at the Hamad Bin Khalifa University, Qatar. He is an IEEE Fellow for contributions to design and analysis of high-speed packet switching. He is/was on the Editorial Board of various prestigious journals and magazines including IEEE Transactions on Communications, IEEE Communication Magazine, Computer Networks, Wireless Communications and Mobile Computing, and Parallel Computing as well as a guest editor of IEEE Communications Magazine, guest editor-in-chief of two special issues of IEEE Journal on Selected Areas of Communications, and a guest editor of Optical Networks Magazine. He has chaired more than 20 international conferences and workshops including The IEEE International High Performance Switching and Routing Conference, the IEEE GLOBECOM/ICC Optical networking workshop, the IEEE ICC High-speed Access Workshop, and the IEEE IPPS HiNets Workshop, and has been on the program committees of more than 200 international conferences and workshops. He was the Chair of IEEE Communications Society Technical Committee on Transmissions, Access and Optical Systems, and Vice-Chair of the Optical Networking Technical Committee, as well as member of the ComSoc technical activities council. He received the best paper award at the IEEE International Conference on Communications in 2009 and the IEEE International Conference on Information and Networking in 1998. He also supervised the best PhD paper award among all universities in Hong Kong.