An Efficient Framework for Online Virtual Network Embedding in Virtualized Cloud Data Centers

Ting Wang*, Bo Qin*, Mounir Hamdi*‡
*Hong Kong University of Science and Technology, ‡Hamad Bin Khalifa University
{twangah, bqin, hamdi}@cse.ust.hk

***Abstract*—Embedding multiple virtual networks (VNs) 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 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 AI model, named Blocking Island, we propose an efficient online heuristic VN embedding framework 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*.**

I. 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]. 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 [2][3][4]. 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 virtual networks is called *virtual network embedding* (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 constraints.

The main objective of solving VNE problem is to make efficient use of substrate resources through dynamic and effective VN mapping algorithms. VNE has been presented as a very challenging resource allocation problem that has been addressed in many research studies [5][6][7][8][9][10]. In fact, the VNE problem is NP-hard [6][11][8][9], even in the offline case. Even when all the virtual nodes are embedded, to embed the virtual links is still NP-hard [8]. Naturally, the online case of VNE problem would be more intractable.

In response to this issue, this paper proposes an efficient online 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 rest of the paper is organized as follows. First we briefly review the related works in Section II. Then Section III demonstrates the VN mapping model and problem formulation. The *Presto* framework is designed in Section IV followed by evaluations in Section V. Section VI concludes this paper.

II. Related Work and Motivation

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 ones. Generally, the existing works can be classified into two categories: offline algorithms and online algorithms.

The offline algorithms process based on the assumption that all VN requests are defined and known in advance. The work of [7] 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 [12] proposed a distributed VNE approach by assuming the substrate resources are unlimited. However, this distributed approach needs a number of signalling messages exchanged between the substrate nodes asynchronously, which increases time delay and signalling network overload. In these offline methods, all VN requests can be processed together, which may achieve better results than online methods which do not have any global view and knowledge about the future allocations. However, to know all VN requests in advance is not practical in real world.

The online algorithms study the on-demand VNE 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 [9] 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

978-1-4673-9501-4/15/$31.00 ©2015 IEEE

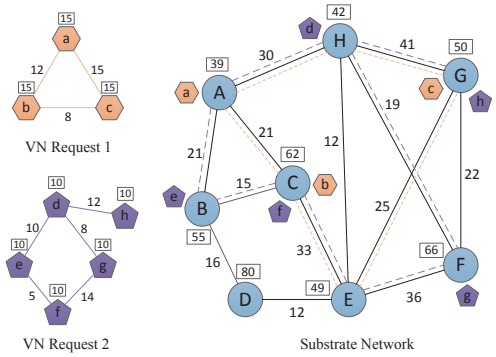

Fig. 1. An example of mapping VN requests onto a shared substrate network.

splittable flows and path migration is used to periodically re-optimize the utilization of substrate resources. Comparatively, the paper [5] solved the VNE problem 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 [6] 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 MIX problem is relaxed to a linear program and solved using deterministic and randomized rounding techniques. Another online embedding algorithm [11] 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 the embedding cost and revenues, the computation efficiency is also a very important factor, especially in data centers. However, known as a NP-hard problem, the VNE suffers from intractable computation complexity with huge search domain. Although 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. To deal with this, we propose a BI-based heuristic which greatly increases the computation efficiency and decreases the search domain. Moreover, it also saves much time in determining the acceptance of a VN request.

III. VN Mapping Model and Problem Formulation

A. Substrate Network Model

The substrate network can be modelled as a weighted undirected graph $G_s = (N_s, E_s)$, where N_s and E_s is the set of substrate nodes and links, respectively. For each substrate node $n_s \in N_s$, $C(n_s)$ denotes the available CPU capacity of 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 $p \in P_s$ is represented by $B(p)$. Fig. 1 illustrates an example of substrate network graph.

B. Virtual Network Model

Likewise, the virtual network also can be represented by a weighted undirected graph G_v=(N_v, E_v), where N_v and E_v indicate the set of virtual nodes and E_v virtual links. Each $n_v \in N_v$ and $e_v \in E_v$ are associated with a minimum required CPU capacity $C(n_v)$ and a minimum required bandwidth capacity $B(e_v)$, respectively. The VN request is denoted 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 $1 \leq i \leq |N_v^{Rid}|$, 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 n_v^i and n_v^j where $1 \leq i,j \leq |N_v^{Rid}|$.

C. Virtual Network Embedding Problem Description

The embedding of 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 certain constraints.

$$\mathcal{M} : G_v \rightarrow (N_s^{'}, P^{'}, R_N, R_E) \tag{1}$$

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 $\mathbb{R}_v$. The VNE problem is then decomposed into two sub-problems: *Virtual Node Mapping* $\mathcal{M}_\mathcal{N}$ and *Virtual Link Mapping* $\mathcal{M}_E$.

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

$$\mathcal{M}_\mathcal{N} : (N_v^{Rid}, C_v) \rightarrow (N_s^{'Rid}, R_N) \tag{2}$$

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

$$\mathcal{M}_\mathcal{N}(n_v) = \mathcal{M}_\mathcal{N}(m_v) \ \text{ iff } \ n_v = m_v \tag{3}$$

$$C(n_v) \leq \Lambda_N(\mathcal{M}_\mathcal{N}(n_v)) \tag{4}$$

where $\Lambda_N(\mathcal{M}_\mathcal{N}(n_v))$ denotes the residual or available CPU resources of substrate node $\mathcal{M}_\mathcal{N}(n_v)$.

2) Virutal Link Mapping: Each virtual link of a VN request $\mathbb{R}_v$ is mapped to a single unsplittable substrate path or a set of multiple splittable substrate paths.

$$\mathcal{M}_E : (E_v^{Rid}, B_v) \rightarrow (P^{'Rid}, R_E) \tag{5}$$

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

$$B(e_v) \leq \sum_{p \in \mathcal{M}_E(e_v)} \Lambda_E(p) \tag{6}$$

where $\Lambda_E(p) = \min_{e_s(i,j) \in p} \Lambda_E(e_s(i,j))$ and $\Lambda_E(e_s(i,j))$ is the residual bandwidth of the link $e_s(i,j)$. Fig. 1 illustrates a feasible VN embedding solution for two VN requests.

D. Objectives

In this work, the VNE problem is formulated as a Multi-Objective Linear Programming (MOLP) optimization problem. The ultimate objective is to compute a VN mapping solution that maximizes the *revenue* while reducing the *cost*.

1) Maximize Revenue: In order to maximize the economic benefit, the providers aim to allocate resources for more VN requests with a higher VN request acceptance ratio. Similar to the previous works [5][6][8][9], we define the revenue $\mathcal{R}ev$ of a VN request $\mathbb{R}_v$ as the sum of its requested resources:

$$\mathcal{R}ev(\mathbb{R}_v^{Rid}) = \sum_{n_v \in N_v} C(n_v) + \alpha \sum_{e_v \in E_v} B(e_v) \tag{7}$$

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

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 spent on embedding a VN request, and save more resources to accept more VN requests. Since the node embedding cost is fixed and deterministic, thus we only need to minimize the cost of link embedding cost $Cost(E_v^{Rid})$ in the second phase of the algorithm. The $Cost(E_v^{Rid})$ is computed as

$$Cost(E_v^{Rid}) = \sum_{e_v(i,j) \in E_v^{Rid}} B(e_v(i,j)) * |p_s(n_v^i, n_v^j)| \quad (8)$$

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.

IV. *Presto*: BI-based Online Heuristic Virtual Network Embedding Framework

As aforementioned, the VNE is an NP-hard problem and traditional approaches suffer from high complexities due to the huge search space. In response to this issue, we propose an efficient BI based heuristic framework, named *Presto*, to solve the online VNE problem with much lower and more manageable complexity.

A. Blocking Island Paradigm

Derived from artificial intelligence, Blocking Island (BI) provides an efficient way to represent the availability of network resources. *A β-BI for a node x is the set of all nodes that can be reached from x using links with at least β available resources, including x* [13]. Each node has one unique β-BI. A virtual link request $e_v(src, dest, \beta_e)$ can be satisfied with at least one substrate route if and only if both the endpoints src and $dest$ are in the same β_e-BI. Likewise, for a virtual node n_v with required β_n, it can be satisfied with at least one substrate node iff the substrate-node-CPU-based β_n-BI is not empty. Fig.2 and Fig.3 exhibit examples of a link-based BI and a node-based BI, respectively. BIs can further to be constructed into a tree, named Blocking Island Hierarchy (BIH) tree. Fig.4 shows two BIH tree examples for the BIs of Fig.2 and Fig.3. This abstraction tree can reflect the real-time state of the available network bandwidth.

The BI paradigm greatly reduces the search space and computation complexity. For 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 plenty 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.

B. Overview on Presto Framework

Designed as an online VNE algorithm, in addition to the ability of processing VN requests one by one, *Presto* also can batch process multiple arriving VN requests online. 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 window with certain size, and then process them together based on a priority metric (i.e. revenue, cost). The online batch processing manner not only provides the capability of looking ahead, but also enables the forward checking ability offered by BI which helps increase the acceptance ratio.

Presto decomposes the VNE problem into two phases: Firstly, *Presto* applies the node mapping algorithm to map the virtual nodes of a VN request to a set of substrate nodes targeting at increasing the acceptance ratio and maximizing revenue. Secondly, *Presto* uses the link mapping algorithm to compute the most beneficial routes to allocate every virtual link aiming to minimize the embedding cost.

C. Variable Ordering

As a common issue for the constraint satisfaction problem, it cannot be guaranteed to find an assignment to satisfy all the requests all the time. How to select next variable from the set to allocate has a great impact on the overall allocation success ratio. The most common approach is the fail-first principle based technique which tries those tests in the given set of tests that are most likely to fail [14]. Based on this observation, a set of variable (virtual link/node, VN request) ordering mechanisms are carefully designed. 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.

D. Node Mapping Algorithm

The heuristic node mapping algorithm, named HNM, is designed to map each virtual node to substrate nodes.

1) Virtual Node Ordering: Before embedding a VN request, its virtual nodes are sorted 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 required β CPU capacities, where the β-BI contains fewest substrate nodes. This 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.

By doing this, the search tree is largely pruned and the search space is decreased, which greatly improves the computation and search efficiency.

2) Node Mapping: In order to increase the acceptance ratio and decrease computation time, the targeted substrate nodes are also selected with priorities. 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) Finally, select the substrate node n_s with 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 recursively chooses the next unallocated virtual node to embed in the above order until all virtual nodes of the VN request are allocated successfully. The key advantages of HNM are mainly revealed in the following aspects.

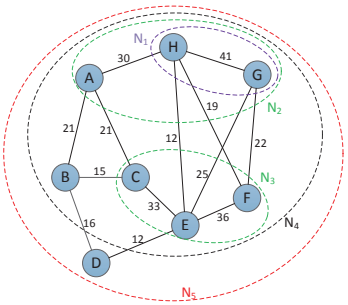

Fig. 2. A BI based on link capacities, in which N_1 is 40-BI, N_2 and N_3 are 30-BIs, N_4 is 20-BI, and N_5 is 10-BI.

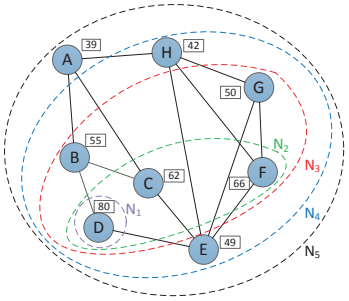

Fig. 3. A BI based on node capacities, in which N_1 is 80-BI, N_2 is 60-BI, N_3 is 50-BI, N_4 is 40-BI, and N_5 is 30-BI.

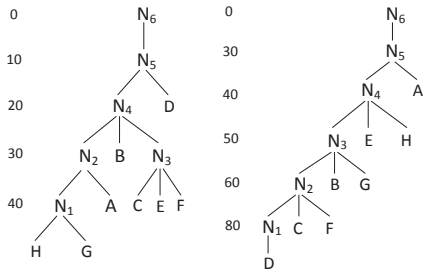

Fig. 4. BIH trees for BIGs of Fig.2 and Fig.3, and N_0 indicates 0-BI.

(i) Using 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.

E. Link Mapping Algorithm

After virtual node mapping, the heuristic link mapping algorithm HLM is then applied to map each virtual link to a substrate path.

1) Virtual Link Ordering: Similarly, aiming to increase the overall acceptance ratio, a heuristic virtual link ordering algorithm is proposed, named HVLO, which works as below.

(i) *Presto* intends to first choose the virtual link where the Lowest Common Father (LCF_{sn}) of its corresponding embedded end substrate nodes in BIH tree is highest. For example, in Fig.2 and Fig.4 (left), suppose two virtual links l_1 and l_2 need to be allocated, the end nodes of l_1 and l_2 are mapped to (A, H) and (G, F), respectively. Then l_2 will 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. 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.
(ii) If there are multiple choices for the first step, then *Presto* prefers to choose the virtual link whose LCF_{sn} contains fewer nodes. This also follows the fail-first principle.
(iii) If there are still multiple such virtual links, then *Presto* prioritizes the one with higher required link capacities.
(iv) Finally, *Presto* randomly sort the virtual links that still have the same order.

After sorting, the resulted pruned search tree greatly decreases search domain and increases computation efficiency.

2) Link Mapping: If all the virtual links can be satisfied after BI-based acceptance checking, then HLM performs 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, some rules are designed for route selection: (i) Fewest critical links (inter-links between different BIs) are involved in the 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 are clustered, and generate a set of candidate substrate routes. For example, in the link-based BIs or BIH as shown in Fig.2 and Fig.4, 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 less critical bottleneck links (inter-BI links) thus increasing the success ratio of future allocations. 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.

Besides the advantages as demonstrated in HNM, HLM also regards the embedding cost as its optimization objective, and prefers to choose a substrate path with lowest cost.

F. Sliding Window-based Batch Processing

Presto applies a sliding window based method to batch process several requests together at the end of a 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 help achieve a higher acceptance ratio. Furthermore, *Presto* can also do forward checking with the help of BI when allocating one VN request. Each VN request is associated with a field named as *maximum standing time* T_s, which indicates the maximum time a VN request can wait in the queue for processing.

1) VN Request Ordering: A dynamic heuristic approach, denoted as DHRO, is proposed to determine the order of selecting VN requests within the same window period taking the optimization objectives into consideration. The principle of DHRO to sort the VN requests 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 such VN requests, *Presto* assigns a higher priority to the one that requests higher link bandwidth resources since the bandwidth is a more constrained resource [15] 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 favours more urgent requests.
(iv) Otherwise, keep the current sequence unchanged.

The dynamic heuristic DHRO not only can increase the overall revenue, but also can increase the acceptance ratio.

2) Batch Processing with Lookahead: Different from individual VN embedding at its arrival, *Presto* segments time into consecutive 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. 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) Then *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 firstly 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 currently, then it will be deferred to be processed in the next window period expecting some already allocated VNs depart and release enough resources.

3) Forward Checking: In addition to the lookahead ability, BI paradigm also enables *Presto* with the forward checking capacity. With the help of BI paradigm, we know at any point in the search whether 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/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 cannot be satisfied, another allocation solution must be tried for the current virtual node/link. The capacity of forward checking ensures *Presto* with a better embedding decision and higher acceptance ratio.

V. Evaluation

In order to evaluate the performance of *Presto*, we implemented a *Presto* prototype in Java language by extending our DCNSim simulator [16] with some additional 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.

A. Evaluation Settings

1) Substrate Network: In the experiment, we use SprintNet [17] [18] topology as the substrate network topology. The substrate resources (CPU and link bandwidth) were uniformly distributed between 50 and 150.

2) Virtual Network: The virtual network topologies were randomly generated following similar setups in previous work [5][6][8][9], 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. One window time was primarily defined as 100 time units.

3) Compared Algorithm: It is difficult to compare *Presto* with other algorithms due to different problem formulations and different objectives with different strategies (e.g. online/offline, admission control, one/two stages). Nevertheless, we modified two existing algorithms fitting in our model to facilitate comparisons. The first algorithm to compare with is the baseline algorithm of the work [9], 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 [10], denoted as VNE-SP, which employs greedy node mapping algorithm with shortest path link mapping algorithm without reconfiguration.

B. Evaluation Results

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 Equation 7). 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.5 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 both VNE-Yu and VNE-SP algorithms where the increment ranges from 6.21% to 19.64%. Comparatively, the evaluation results shown in Fig.6 present the gained average revenues with different weights α to the revenue induced by bandwidth $\sum B(e_v)$. 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 scare 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.

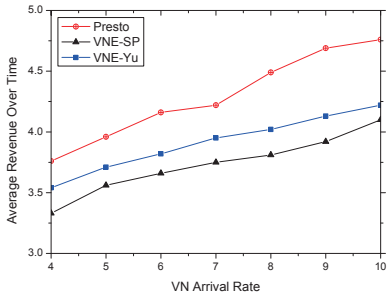

Fig. 5. The average revenue under different VN arrival rates.

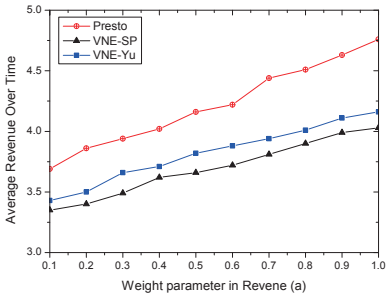

Fig. 6. The average revenue with different values of tuning parameter α.

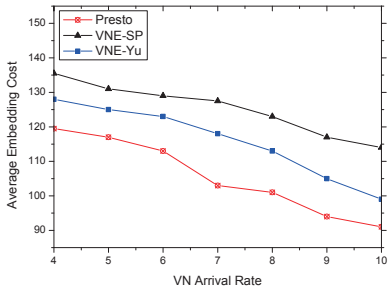

Fig. 7. The average bandwidth cost.

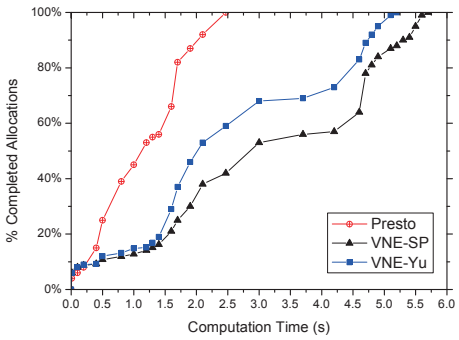

Fig. 8. The performance of computation efficiency.

2) Average Embedding Cost: As aforementioned in Section III, 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.7 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.7, which in turn increases the acceptance ratio and overall revenue.

3) Computation Efficiency: Another big advantage of *Presto* lies in its computation efficiency. Generally, the traditional algorithms have a bad exponential time complexity due to the huge search space resulting in high computation complexity. Comparatively, in *Presto*, with the advantage of BI model, the heuristic request selection algorithms together with 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 two algorithms, we offline runs 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.8 gives the simulation results, which reveal that *Presto* is around 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 seconds using *Presto*, while VNE-Yu and VNE-SP only complete 14.8% and 12.7% within one second, respectively, and cost 5.2 seconds and 5.7 seconds to finish 100% allocations, respectively. This well proves the high computation efficiency of *Presto*.

VI. Conclusion

This paper aims to achieve an efficient online VNE algorithm in virtualized cloud data centers. In order to deal with this computationally intractable problem, 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. With the benefit of BI paradigm, *Presto* achieves a good performance in various aspects including revenue, embedding cost, acceptance ratio and computation efficiency. The extensive simulation results have witnessed the effectiveness of *Presto* framework.

VII. Acknowledgement

This paper is supported by RGC grants 612912 and 613113.

References

[1] T. Wang, et al. Rethinking the data center networking: Architecture, network protocols, and resource sharing. *IEEE Access*, 2014.
[2] Thomas Anderson, et al. Overcoming the internet impasse through virtualization. *Computer*, 38(4):34–41, 2005.
[3] Md Faizul Bari, et al. Data center network virtualization: A survey. *Communications Surveys & Tutorials, IEEE*, 15(2):909–928, 2013.
[4] Andreas Fischer, et al. Virtual network embedding: A survey. *Communications Surveys & Tutorials, IEEE*, 15(4):1888–1906, 2013.
[5] Xiang Cheng, et al. Virtual network embedding through topology-aware node ranking. *ACM SIGCOMM*, 41(2):38–47, 2011.
[6] NM M. K. Chowdhury, et al. Virtual network embedding with coordinated node and link mapping. In *INFOCOM*. IEEE, 2009.
[7] Jing Lu and Jonathan Turner. Efficient mapping of virtual networks onto a shared substrate. *Washington University in St. Louis, Tech. Rep*, 2006.
[8] Muntasir Raihan Rahman, et al. Survivable virtual network embedding. In *NETWORKING 2010*, pages 40–52. Springer, 2010.
[9] Minlan Yu, et al. Rethinking virtual network embedding: substrate support for path splitting and migration. *ACM SIGCOMM*, 2008.
[10] Yong Zhu, et al. Algorithms for assigning substrate network resources to virtual network components. In *INFOCOM*, 2006.
[11] J. L., et al. A virtual network mapping algorithm based on subgraph isomorphism detection. In *the 1st ACM workshop on VISA*. ACM, 2009.
[12] Ines Houidi, et al. A distributed virtual network mapping algorithm. In *Communications, IEEE International Conference on*. IEEE, 2008.
[13] Christian Frei, et al. Simplifying network management using blocking island abstractions. *Internal Note from the IMMuNe Project, April*, 1997.
[14] Robert M Haralick, et al. Increasing tree search efficiency for constraint satisfaction problems. *Artificial intelligence*, 14(3):263–313, 1980.
[15] Jeffrey Dean, et al. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
[16] Liu Yang et al. Dcnsim: A data center network simulator. In *The DCPerf, Philadelphia, USA*. IEEE, 2013.
[17] T. Wang, et al. Sprintnet: A high performance server-centric network architecture for data centers. In *IEEE ICC*, pages 4005–4010, 2014.
[18] T. Wang, et al. Designing efficient high performance server-centric data center network architecture. *Computer Networks*, 79:283–296, 2015.