HAL
open science

On Tackling Virtual Data Center Embedding Problem

Md Golam Rabbani, Rafael Esteves, Maxim Podlesny, Gwendal Simon,
Lisandro Zambenedetti Granville, Raouf Boutaba

To cite this version:

Md Golam Rabbani, Rafael Esteves, Maxim Podlesny, Gwendal Simon, Lisandro Zambenedetti Granville, et al.. On Tackling Virtual Data Center Embedding Problem. IM 2013 : IFIP/IEEE International Symposium on Integrated Network Management, May 2013, Ghent, Belgium. hal-00846051

HAL Id: hal-00846051

https://hal.science/hal-00846051v1

Submitted on 18 Jul 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

On Tackling Virtual Data Center Embedding Problem

Md Golam Rabbani*, Rafael Esteves*†, Maxim Podlesny*, Gwendal Simon‡
Lisandro Zambenedetti Granville†, Raouf Boutaba*
* D.R. Cheriton School of Computer Science, University of Waterloo, Canada
† Institute of Informatics, Federal University of Rio Grande do Sul, Brazil
‡ Telecom Bretagne, Institut Mines Telecom, France

***Abstract*—Virtualizing data center networks has been considered a feasible alternative to satisfy the requirements of advanced cloud services. Proper mapping of virtual data center (VDC) resources to their physical counterparts, also known as virtual data center embedding, can impact the revenue of cloud providers. Similar to virtual networks, the problem of mapping virtual requests to physical infrastructures is known to be NP-hard. Although some proposals have come up with heuristics to cope with the complexity of the embedding process focusing on virtual machine placement, these solutions ignore the correlation among other data resources, such as switches and storage. In this paper, we propose a new embedding solution for data centers that, in addition to virtual machine placement, explicitly considers the relation between switches and links, allows multiple resources of the same request to be mapped to a single physical resource, and reduces resource fragmentation in terms of CPU. Simulations show that our solution results in high acceptance ratio of VDC requests, improves utilization of the physical substrate, and generates increased revenue for infrastructure providers.**

I. Introduction

With the increasing popularity of cloud computing, data centers have been widely deployed by companies such as Amazon, Google, Facebook, and Yahoo! to support large-scale applications and to store large volumes of data. However, the deployment of large-scale data centers usually comes with high provisioning and maintenance costs, making data centers prohibitive for smaller enterprises and research labs. To allow multiple users to share data centers and, consequently, reduce the operational costs of such infrastructures, data center architectures proposed over the recent years [1]–[3] rely on *server virtualization* technologies (*e.g.*, Xen and VMWare). Server virtualization partitions a physical server into multiple isolated slices called *virtual machines* (VMs) and makes them work independently from each other. A data center relying on server virtualization allows multiple service providers to coexist. However, limitations of current data center architectures hinder a deployment of specific classes of applications that depend on network constraints other than computing and storage. For example, *MapReduce* applications have strict constraints in terms of latency and throughput, which server virtualization cannot guarantee [4].

Extending the benefits of virtualization to network infrastructures is an emerging trend toward fully virtualized data center architectures [5]. Similarly to virtualization of wide area networks [6], two main entities cooperate with each other in virtualized data center networks : *infrastructure providers* (InPs), which owns the physical data center, and *service providers* (SPs), which in turn run applications on virtual data centers (VDC) built on the top of the physical substrate. An VDC consists not only of VMs but also on multiple resources related to the inter-connection of the VMs (especially switches and links) One of the main objectives of InPs is to accommodate as many VDC requests as possible.

Defining the mapping of virtual resources to physical ones is commonly known as *embedding*, and has been the subject of extensive research in the context of network virtualization [7]–[10]. An efficient VDC embedding is crucial to support a high number of tenants, which in turn impacts the revenue of an InP. Data center architectures such as SecondNet [11] and Oktopus [12] have proposed heuristics to cope with the NP-hardness of the embedding problem. However, these approaches are heavily topology dependent and cannot be generally applied to arbitrary data center topologies. For example, Oktopus uses a greedy algorithm for the resource allocation to a VDC; however, it supports only two types of VDC requests, and is applicable only for tree-like physical topologies. In addition, and more importantly, data center embedding solutions usually focus on VM placement [11] and do not explicitly consider other types of resources (*e.g.*, storage servers, switches, routers, links) and their associated constraints, which can limit the applicability of current data center embedding solutions in realistic scenarios.

We revisit the problem of VDC embedding in this paper. We define a VDC in a generic way, where we distinguish multiple resource types on multiple equipments. We typically observe that the providers of interactive multimedia applications require GPU in their VMs while other SPs are interested in very fast memory hardware (SRAM). An InP should now be able to differentiate these different hardware in order to serve a vast range of SPs. We also go further for the definition of virtual switches and virtual links. We believe that a SP running applications with strong network requirements is interested in renting a very precise network infrastructure for the connection between its VMs. Protocols related to Software-Defined Networks (SDN) can provide some guarantees but a full isolation of applications can only be offered by the virtualization of switches.

Our formulation of the VDC embedding problem reveals

that the problem is intractable, thus we present a high-level approach for tackling the problem. We design a heuristic, which is based on three embedding phases: VM mapping, link mapping, and switch mapping. We explicitly include in the core of our heuristic the different data center resources such as switches and storage. Moreover, we clearly distinguish physical servers and network nodes (switches). To increase the chances of successful VDC embedding, our solution includes two features. First, it allows multiple resources of a single VDC request to be embedded in the same physical resource, which is not considered in other proposals. Second, we leverage the coordination between switch mapping and link mapping phases.

The rest of this paper is organized as follows. In Section II, we formalize the data center network model and the VDC embedding problem itself. In Section III, we present a general approach for tackling the VDC embedding problem. In Section IV, we propose the particular algorithm for VDC embedding based on coordinated switch and link mapping. In Section V, we present the performance evaluation of the proposed algorithm. Finally, we conclude the paper in Section VI.

II. Problem Description

We provide now our network model and problem description of the VDC embedding problem. All the notations are shown in Table I and Table II for physical network infrastructure and VDC request respectively.

A. Physical Data Center

The InP owns a physical data center network, which is modelled as a weighted undirected graph, $G^p(S^p, X^p, E^p)$, where S^p is the set of physical servers, X^p is the set of physical switches and E^p is the set of physical links.

We associated with each physical server in S^p a set of featured hardware, which typically includes CPU, data storage memory, Graphical Processing Unit (GPU), and fast SRAM memory. A part of each of these hardware can be reserved by a tenant. To preserve the generality of our model, we associate with each physical server $s^p \in S^p$ a set of featured capacities $c_i(s^p), i \in \{1, \ldots, k\}$ where k is the number of different reservable specific hardware capacities. For example, $c_1(s^p)$ and $c_2(s^p)$ can refer to CPU and memory capacities respectively. We also keep track of the residual capacity for each hardware after some parts of these hardware have been rented by a tenant. We note $\bar{c}_i(s^p)$ the residual capacity for hardware $i \in \{1, \ldots, k\}$.

We also associate with each physical link e^p in E^p a bandwidth capacity, noted $b(e^p)$, and the residual bandwidth capacity, noted $\bar{b}(e^p)$. Please note that this capacity can be extended with a subscript in the same manner as for the capacity of physical servers.

One of the novelties of our model is that we introduce also resources for switches. We consider each buffer of each switch port and we allows a SP to partially rent each of these buffers. Thus, a SP is able to very precisely adjust packet losses and processing in every port. To simplify, we use the same notations as for the virtual servers, so $c_i(x^p)$ is the capacity of the hardware i in the virtual switch x^p.

TABLE I
NOTATIONS FOR PHYSICAL NETWORK INFRASTRUCTURE.

$G^p(S^p, X^p, E^p)$	The physical network infrastructure
S^p	set of physical servers
X^p	set of physical switches
E^p	set of physical links
$c_i(s^p)$	capacity of hardware i of server $s^p \in S^p$
$\bar{c}_i(s^p)$	residual capacity of hardware i of server $s^p \in S^p$
$b(e^p)$	bandwidth capacity of link $e^p \in E^p$
$\bar{b}(e^p)$	residual bandwidth capacity of link $e^p \in E^p$
$c_i(x^p), \bar{c}_i(x^p)$	capacity hardware i (and residual capacity) of switch x^p

B. VDC Request

The clients of the InP are entities that would like to reserve a subset of InP's physical infrastructure. We call a VDC request what a *tenant* wants to reserve. The VDC request is also modelled as a weighted undirected graph, which we note $G^v(S^v, X^v, E^v)$, where S^v is the set of VMs, X^v is the set of virtual switches and E^v is the set of virtual links.

TABLE II
NOTATIONS FOR VDC REQUEST.

$G^v(S^v, X^v, E^v)$	VDC request
S^v	set of virtual machines
X^v	set of virtual switches
E^v	set of virtual links
$c_i(s^v)$	capacity of hardware i in VM $s^v \in S^v$
$b(e^v)$	bandwidth of virtual link $e^v(i,j) \in E^v$
$c_i(x^v)$	capacity of hardware i in virtual switch $x^v \in X^v$

We keep the same notations as for the physical infrastructure for the specific required capacities of VMs and virtual links.

C. Objective and Problem Formulation

The goal of an InP is basically to maximize its profits through an optimal exploitation of its infrastructure. This objective is related to maximizing the number of VDC requests that are successfully embedded in the physical infrastructure subject to embedding constraints.

Another challenge for InP is that tenants do not coordinate to request VDC. Thus, VDC requests do not arrive at the same time. Hence the problem faced by InPs is an *online* problem where the inputs (here the VDC requests) have to be processed iteratively, without knowledge of the future VDCs. Let consider a VDC request G^v, which is received by the InP while a part of its infrastructure has already been reserved. We express in the following the constraints for the embedding of G^v. We first have to define the output variables.

Let x_{uv} be a set of binary variables for the mapping of equipment $u \in S^v$ (respectively $u \in X^v$) into a physical

server $v \in S^p$ (respectively $v \in X^p$).

$$x_{uv} = \begin{cases} 1 & \text{if } u \text{ is embedded into } v \\ 0 & \text{otherwise} \end{cases}$$

Let $y_{(uu')(vv')}$ be a set of binary variables for the mapping of virtual link $(uu') \in E^v$ into physical link $(vv') \in E^p$.

$$y_{(uu')(vv')} = \begin{cases} 1 & \text{if } (uu') \text{ is embedded into } (vv') \\ 0 & \text{otherwise} \end{cases}$$

An embedding can be done if the following constraints are fulfilled for the mapping of equipments.

$$\sum_{v \in S^p \cup X^p} x_{uv} = 1, \quad \forall u \in S^v \cup X^v \tag{1}$$

$$\sum_{u \in S^v \cup X^v} x_{uv} \times c_i(u) \leq \bar{c}_i(v), \quad \forall v \in S^p \cup X^p, \forall i \tag{2}$$

Constraints (1) ensure that a VM is embedded into one and only one physical server. It also makes sure that all VMs are embedded. Constraints (2) guarantee that the capacities of physical servers are not overused by the embedded VMs.

For the link mapping, here are the constraints:

$$\sum_{(vv') \in E^p} y_{(uu')(vv')} \geq 1, \quad \forall (uu') \in E^v \tag{3}$$

$$\sum_{(uu') \in E^v} y_{(uu')(vv')} \times b((uu')) \leq \bar{b}((vv')), \quad \forall (vv') \in E^p \tag{4}$$

Constraints (3) ensure that a virtual link can be mapped to several physical links or a physical path. Constraints (4) guarantee that physical links have enough bandwidth to host embedded virtual links.

For one given VDC and one infrastructure configuration, several embeddings can fulfill all above constraints. But some of them succeed in preserving "nearby" resources for future VDCs although others barely allow the next VDCs to be embedded. The quality of the online embedding algorithm for VDCs directly relates to the capacity of the embedding algorithm to be friendly for the next VDCs.

III. Iterative Heuristics: Main Principle

Optimization problems for embeddings that include both nodes and links are known to be NP-hard. For practical implementations, heuristics are needed. We discuss in this Section the main principles of an iterative three-steps algorithm for the embedding. We go into the details of our main proposal in Section IV.

A. Introducing the Three Phases

For the processing of a new VDC G^v, an iterative heuristic consists of the three following rounds.

Phase I: VMs mapping – in this first step, the objective is to map each VM in S^v into one physical server. The mapping should respect the constraints (2), that is, the physical server that host a VM should have enough capacity. Various strategies can be implemented. With regard to the following steps, the proximity of the selected physical servers is critical.

Phase II: Virtual switches mapping – the second step is about mapping each virtual switch in X^v to a physical switch. Besides constraints (2), the main idea here is to associate virtual switches to physical ones while taking into account the previous VM mapping. It makes more sense to map a virtual switch between two already embedded VM so that the distance between the chosen physical switch and the physical servers that host the VM is small.

Phase III: Virtual links mapping – the final step is to map the virtual links into physical links, with respect to constraints (4). Many paths can exist between two physical servers in the real topology. The goal here is to find the paths between the physical servers that host the virtual equipments that are given in E^v so that all virtual links can be embedded. paths

In this heuristic, each step critically impacts the following steps because the input of each phase depends on the output of the previous phase. Therefore, we have to implement a mechanism that allows to get back to a previous phase if ever one phase cannot be fulfilled. For example, the impossibility to find a mapping for virtual switches does not mean that the whole VDC cannot be embedded. Another VM mapping may enable a successful switch mapping. On the other hands, a heuristic does not explore all possible solutions. A trade-off should be found between the number of tested solutions allowed by such a *retro-mechanism* and the efficiency of the whole algorithm, which should be fast.

B. A Randomized Heuristic

We illustrate the presentation of the iterative heuristic by an example: a simple algorithm based on randomized processes. For each VM, a physical server is randomly picked and if it has the capacity to host the VM, the algorithm processes the next VM, otherwise another physical server is picked. This algorithm iterates on the whole set of VMs until all VMs are successfully mapped. The switch mapping is done in the same manner. Finally, the link mapping is based on a random walk between the endpoints of each virtual links.

For this randomized heuristic, it is obvious that some of the mappings, even successful ones, can result in poor mappings, which do not allow the other mapping to be successful. When any phase fails, the retro-mechanism that can be implemented is based on a threshold. We define δ as the maximum times a new mapping can be computed. When any of the phase fails more than δ times, the algorithm stops and rejects the VDC request.

IV. Our Heuristic

We present now our heuristic. First, we present the rationale of our solutions. Then, we describe our implementations of the three steps.

A. Rationale

The main idea of our heuristic is to reduce server fragmentation, minimize communication costs between VMs and virtual switches, increase the residual bandwidth of the network, and offer load balancing capabilities to the network. For the sake of simplicity, we consider only one resource for the server (e.g. CPU).

Reduce server fragmentation. We use the variance of the residual CPU capacity of the physical servers. The rationale behind using variance in server defragmentation is explained in Figure 1 with basic physical topology. Initially, two physical servers have residual capacity of 3 CPU units each. After receiving a VM request (1) of 1 CPU unit, the residual CPU of the servers become 2 CPU units and 3 CPU units. A future VM request of 2 CPU units has two possible mappings. In the first one (2), the residual CPU of one server is 3 CPU units, which allows a future VM request requiring 3 CPU units. In the second mapping (3), the residual CPU of the servers are 2 CPU units and 1 CPU unit. In this latter case, a future VM request with 3 CPU cannot be accepted. The variance of the residual CPU capacity in the first allocation is 4.5 and 0.5 in the second allocation. We observe that the higher the variance is, the higher the resource defragmentation is and higher the possibility of accepting higher number of VDC requests.

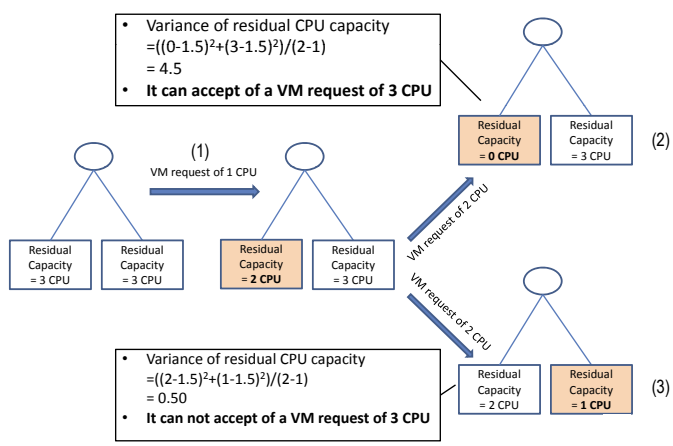

Fig. 1. Using variance for resource defragmentation.

Let $\overline{(\bar{c}_i)}$ be the average residual capacity of hardware i (here we have only CPU, represented by c_1) over all nodes s^p. Formally, the solution that we select among all possible solutions satisfying the previously defined constraints should take into account $\mathcal{V}_{cpu}$, which is defined as:

$$\mathcal{V}_{cpu} = \frac{|S^p|}{\sum_{u \in S^p} \left(\bar{c}_1(u) - \overline{\bar{c}_1}\right)^2} \tag{5}$$

Minimize communication cost and increase residual bandwidth. We consider the hop-count between the virtual nodes (*i.e.*, VM or virtual switch) multiplied by the corresponding requested bandwidth of the virtual link connecting the two virtual nodes. For example, in Figure 2 the cost of mapping virtual switch x_1^v (Fig. 2(a)) to physical switch x_2^p (Fig. 2(b)) is 2 (hop_count from physical switch x_2^p to the *red* physical server) $\times 3$ (bandwidth required between the virtual switch x_1^v and the *red* virtual server) $+2$ (hop_count from physical switch x_2^p to the *yellow* physical server) $\times 2$ (bandwidth required between the virtual switch x_1^v and the *yellow* virtual server) $= 10$, if we consider only the *red* and *yellow* virtual machines, which are the closest ones to the virtual switch x_1^v.

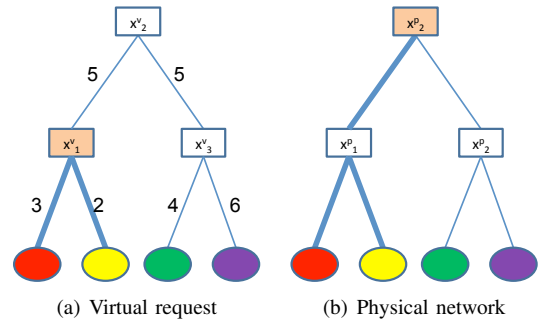

Fig. 2. Minimizing communication cost

A solution that matches our needs takes into account $\mathcal{V}_{sw}$, which is defined as:

$$\mathcal{V}_{sw} = \sum_{(vv') \in E^p} \sum_{(uu') \in E^v} y_{(uu')(vv')} \times b((uu')) \times hop_count(vv') \tag{6}$$

Avoid bottleneck link. Mapping virtual switches to other locations can possibly require a high number of virtual links and result in network bottlenecks. In order to avoid bottleneck of physical links, we also use load balancing for virtual links having the same communication cost by minimizing the variance of the residual network bandwidth.

In the same manner as for $\mathcal{V}_{cpu}$, we define $\mathcal{V}link$ as:

$$\mathcal{V}_{link} = \frac{\sum_{(vv') \in E^p} \left(\bar{b}(vv') - \overline{(\bar{b})}\right)^2}{|E^p|} \tag{7}$$

Overall Optimization. In order to consider all aforementioned objectives, we define one unique objective, which is to minimize:

$$\alpha \times \mathcal{V}_{sw} + \beta \times \mathcal{V}_{cpu} + \gamma \times \mathcal{V}_{link} \tag{8}$$

Since such optimization problem of VDC embedding is NP hard [11], we present a heuristic (shown in Algorithm 1) to solve the problem that has three phases: VM mapping, virtual switch mapping, and virtual link mapping. In order to achieve locality of the VMs, we try to start the mapping process from one physical server. If any of the three phases fails, we increase the number of physical servers adding one adjacent server and try the mapping process again, until we consider all the physical servers in the data center.

B. VM Mapping

In the VM mapping, we map VMs to the physical servers; we do it sequentially so that more than one VM can be mapped

Algorithm 1 VDC Embedding Algorithm

1: **for** $(|S^p| = 1; |S^p| \leq TotalNumberofServers; S^p = S^p \cup \{AdjacentServer\})$ **do**
2: **VM Mapping:**
3: Sort the VMs, $s^v \in S^v$ according the requested
4: resource capacity
5: **for** each $s^v \in S^v$ **do**
6: **for** each $s^p \in S^p$ **do**
7: Add an edge from s^v to s^p if s^p satisfies the
8: capacities of s^v
9: **end for**
10: Add a source node and add an edge from source
11: node to s^v.
12: Add a destination node and add an edge from
13: each of s^p to destination node.
14: Solve min-cost flow problem and goto line 1 if fails
15: **update** residual capacities of s^p.
16: **end for**
17: **Switch Mapping:**
18: **for** each $x^v \in X^v$ **do**
19: **for** each $x^p \in X^p$ **do**
20: Add an edge from x^v to x^p if x^p satisfies the
21: capacities of x^v
22: **end for**
23: **end for**
24: Add a source node and add an edge from source
25: node to each of s^v.
26: Add a destination node and add an edge from each
27: of s^p to destination node.
28: Solve min-cost flow problem and go to line 1 if fails
29: **update** residual capacities of x^p.
30: **Link Mapping:**
31: **for** each $e^v \in E^v$ **do**
32: Remove each $e^p \in E^p$ where $b(e^p) < b(e^v)$
33: Run BFS to compute the shortest path and go
34: to line 1, if no path found
35: **Update** capacity of each link e^p in that path
36: **end for**
37: If all mapping phases are successful, **break**
38: **end for**

to one physical server. We rely on the used bandwidth of physical servers for load balancing, and on the difference between the residual resource capacity and the average resource capacity of physical servers for server defragmentation. In the beginning, requested VMs are sorted according to a requested resource capacity in a descending order. Thus, we can reject a request quickly if there is not enough capacity to satisfy the request. Then we map VMs sequentially so that more than one VM can be mapped to a physical server. First, we take the first VM in the sorted list and build a bipartite graph putting the VM on the left side and all physical servers on the right side. We add an edge from the VM to a physical server if the server can satisfy the resource constraints of the selected VM. Then, we add a source node in the left of the VM and a destination node in the right of the physical servers. After that, we add an edge from the source node to the VM and an edge from each of the physical servers to the destination node. Thus it forms a min-cost flow problem from source to the destination node.

$$\text{Minimize} \sum_{(i,j)\in E} a(i,j).x(i,j) \tag{9}$$

$$\begin{aligned}
\text{where, } x(i,j) &\in \{0,1\} \\
\sum_{j\in V} x(i,j) &= 0, \text{for all } i \neq s,t \\
\sum_{j\in V} x(s,j) &= \text{number of virtual machines} \\
\sum_{i\in P} x(i,t) &= \text{number of physical servers} \\
E &= \text{Set of edges in the graph} \\
V &= \text{Set of virtual machines in the graph} \\
P &= \text{Set of physical servers in the graph} \\
s &= \text{source node of the graph} \\
t &= \text{destination node of the graph}
\end{aligned} \tag{10}$$

The cost function $a(i,j)$ for each edge is defined by,

$$a(i,j) = b(j) \times \frac{1}{|\bar{c}_1(j) - \overline{\bar{c}_1}|} \tag{11}$$

where $b(j)$ is the used bandwidth of the server j and $\overline{(\bar{c}_1)}$ is the mean of the residual CPU of the physical servers. This cost function is used as weight for each of the edge in the graph of min-cost flow problem. The variable $x(i,j)$ gives the mapping of VM to physical servers. We rely on the used bandwidth of physical servers for load balancing, and on the difference between the residual resource capacity and the average resource capacity of physical servers for server defragmentation, that will satisfy our objectives mentioned in Section IV-A.

C. Switch Mapping

After mapping VMs, we proceed to switch mapping. Similar to VM mapping, we build a bipartite graph keeping virtual switches in the left and the physical switches in the right. Then, we add an edge from a virtual switch i to a physical switch j if the residual capacity of the physical switch j can satisfy the requested capacity of the virtual switch i. We add a source node s at the left of the virtual switches and a destination node d at the right of the physical switches. Following that, we add an edge from the source node s to each of the virtual switches and an edge from each of the physical switches to the destination node d, as shown in Figure 3. This reduced the switch mapping problem to finding min-cost flow from source s to destination d.

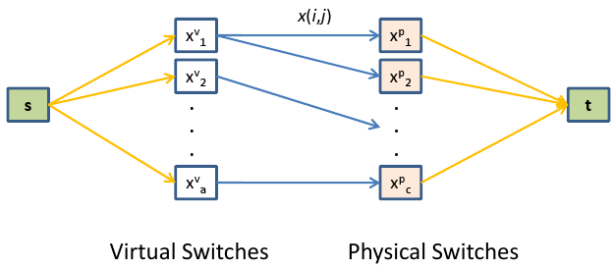

Fig. 3. Graph of min-cost flow problem for switch mapping.

$$\text{Minimize} \sum_{(i,j)\in E} a(i,j).x(i,j) \tag{12}$$

$$\begin{aligned}
\text{where, } x(i,j) &\in \{0,1\} \\
\sum_{j\in V} x(i,j) &= 0, \text{for all } i \neq s,t \\
\sum_{j\in V} x(s,j) &= \text{number of virtual switches} \\
\sum_{i\in P} x(i,t) &= \text{number of physical switches} \\
E &= \text{Set of edges in the graph} \\
V &= \text{Set of virtual switches in the graph} \\
P &= \text{Set of physical switches in the graph} \\
s &= \text{source node of the graph} \\
t &= \text{destination node of the graph}
\end{aligned} \tag{13}$$

The cost function $a(i,j)$ for each edge between a virtual switch i and a physical switch j is defined by $a(i,j) = \sum [hop_count(j,m) \times bandwidth(i,n)]$, where $m \in S^p$, $n \in S^v$, and n in mapped into m. The intuition behind the cost function is to map the virtual switch to the physical switch that offers the lowest communication cost ($hop_count \times bandwidth$) for VMs of the same VDC connecting to the virtual switch i. The cost function described above is used as the weight for each edge in the graph of min-cost flow problem. The variable $x(i,j)$ gives the mapping of virtual switches to physical switches. If the value of $x(i,j)$ is 1, it indicates that the virtual switch i is mapped to the physical switch j. By considering the bandwidth of the links when mapping switches, the algorithm increases the residual bandwidth of the network, which ultimately results in higher acceptance ratio of VDC requests.

D. Link Mapping

Finally, we start link mapping after finishing the node mapping and switch mapping. We map each of the virtual links to physical links one by one. Before allocating any link, we first remove all the physical links having residual bandwidth capacity less than the requested bandwidth capacity of the virtual link. Then, we calculate the shortest path between the physical nodes, where the source and destination nodes of the virtual link are mapped, to reduce the communication cost. We use Breadth First Search (BFS) algorithm to find the unweighted shortest path.

V. Performance Evaluation

We studied the performance of our VDC allocation algorithm through simulation. In this section, we describe the simulation environments followed by performance metrics and result analysis. We have shown that our VDC algorithm can embed VDC requests that include virtual switches. As no other existing work consider switch mapping in VDC embedding algorithm, we have showed the advantages of our algorithm for VDC allocation by comparing acceptance ratio, network and CPU utilization with random VM mapping and random switch mapping.

A. Simulation Environment

We implemented a discrete event simulator for the evaluation of our proposed algorithm using C++ and used open source C++ template library LEMON(Library for Efficient Modeling and Optimization in Networks) [13] to solve minimum-cost flow problem.

We used VL2 [3] topology for our physical data center network topology. Our physical data center has 24 servers, 22 switches and 144 links. The links between servers and ToR (top-of-rack) switches have bandwidth 1000 unit whereas the links between ToR switches and aggregate switches have 10 000 units and the links between aggregate switches and intermediate switches have the capacities of 10 000 units. The normalized values of CPU and storage were real numbers, uniformly distributed between 0.5 and 1.

For the VDC requests, we used VL2 topology, star topology and mixed of them. For all the topologies, the CPU and storage of the VMs were real numbers uniformly distributed between 0.08 and 0. For the VL2 topology, the bandwidth of the links between servers and ToR switches, ToR switches and aggregate switches, and aggregate switches and intermediate switches, were uniformly distributed from 1 to 100, 1 to 1000, and 1 to 1000 respectively. For the start topology, the number of VMs are uniformly distributed from 3 to 10 and the links between the switch and the servers have bandwidth, uniformly distributed from 1 to 100. The VDC requests arrived in a Poisson process. The durations of the VDC requests were following a Poisson distribution having average of 500 time units. Our embedding algorithm serves the VDC requests online and with the passage of time, VDC requests, whose duration have passed, leave the physical data center and new VDC requests arrive.

B. Performance Metrics

1) Acceptance Ratio: The acceptance ratio of a certain point of time is the ratio of the number of accepted VDC requests and the total number of VDC requests until that time. This gives a sense of how well an algorithm is performing,

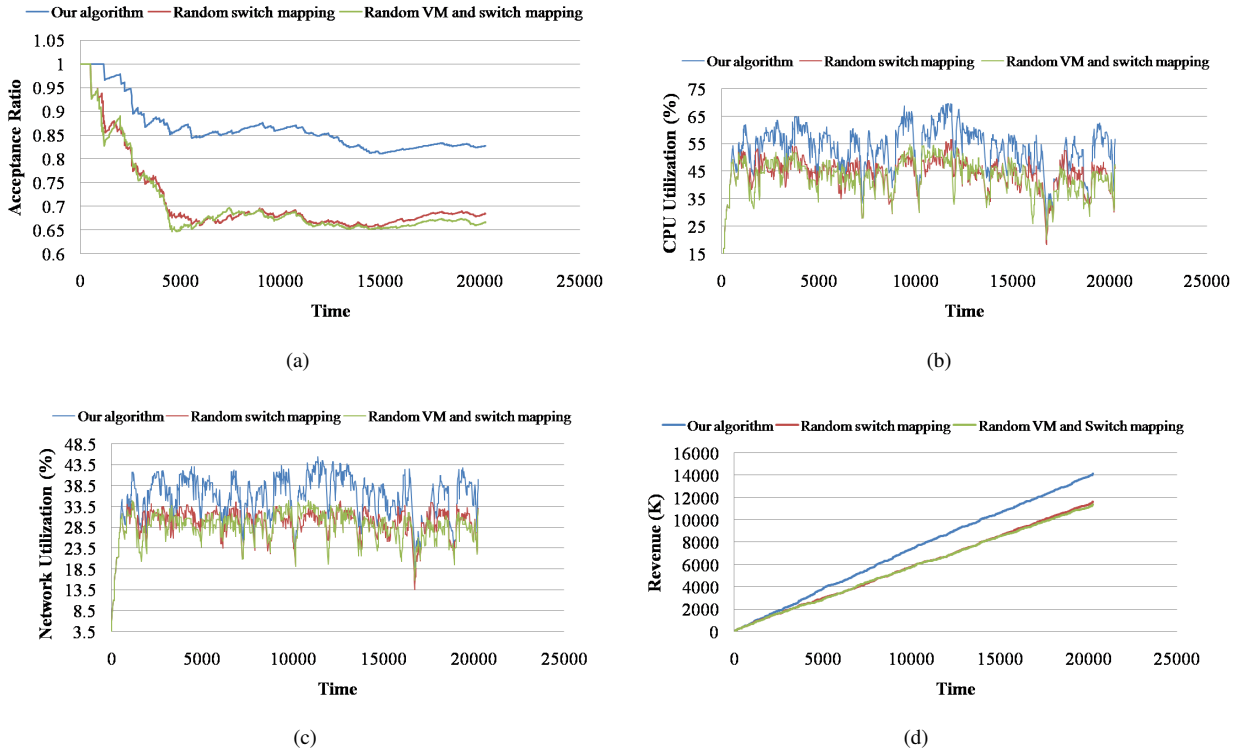

Fig. 4. Comparison of acceptance ratio, resource utilization and revenue over time for VDC requests of VL2 topology. (a) Acceptance Ratio. (b) CPU Utilization. (c) Network Utilization. (d) Revenue

but can not completely capture the whole picture of the performance. An algorithm may accept more number of smaller VDC requests which can end up less revenue.

2) CPU Utilization: The CPU utilization is the total CPU used for embedding VDC requests divided by the total CPU capacity of the physical data center at a certain point of time, which is expressed as percentage.

3) Network Utilization: The network utilization is the total bandwidth of the links used for embedding VDC request divided by the total bandwidth capacity of the physical data center at a certain point of time, which is also expressed as percentage.

4) Revenue: We also computed the revenue generated for an embedding algorithm over time. Revenues are expressed in terms of allocated resources. We have used a revenue model [7] where revenue is considered as sum of bandwidth and CPU. α is used to strike a balance between the two resources.

$$Revenue(G^v) = \sum_{e^v \in E^p} b(e^v) + \alpha \sum_{s^p \in S^p} c_1(s^p)$$

C. Results analysis

After running the simulation, we observed that our algorithm was able to embed the VMs as well as the virtual switches to the physical resources. It also embeds both types of topologies which shows the ability of our algorithm to embed VDC of any topology. To show the effectiveness of our algorithm, we compared our algorithm with random switch mapping and random VM mapping. Figure 4 shows the comparison of performance metrics of our algorithm with random switch mapping and random VM and switch mapping when VL2 topologies were used for input VDC requests. Our algorithm outperforms the random algorithms in terms of each of the metrics. For example, at 20 000 time unit, our algorithm has 0.83 acceptance ratio whereas the random switch mapping and random VM and switch mapping algorithms have acceptance ratio of 0.68 and 0.66 respectively. Our algorithm also has higher CPU and network utilization and revenue. As our algorithm considers the used CPU and bandwidth of the servers while doing VM mapping and bandwidth of the links while doing switch mapping, it can accept higher number of VDC requests as well as higher revenue.

Figure 5 shows the comparison of performance metrics of the algorithms when both VL2 topologies and star topologies were used for input VDC requests. Here, again we see that our algorithm outperforms the random algorithms in terms of each of the metrics. But here the random algorithms comparatively perform better than for the VL2 only topologies. Because, VDC requests of star topology were added to the input. Star topology has only one virtual switch and all the VMs are connected to the switch. So, there is less diversity for the embedding algorithm, specially for the switch mapping. Our algorithm has higher acceptance ratio, CPU and network utilization and revenue than the other two algorithms. We also observe that random switch mapping performs better

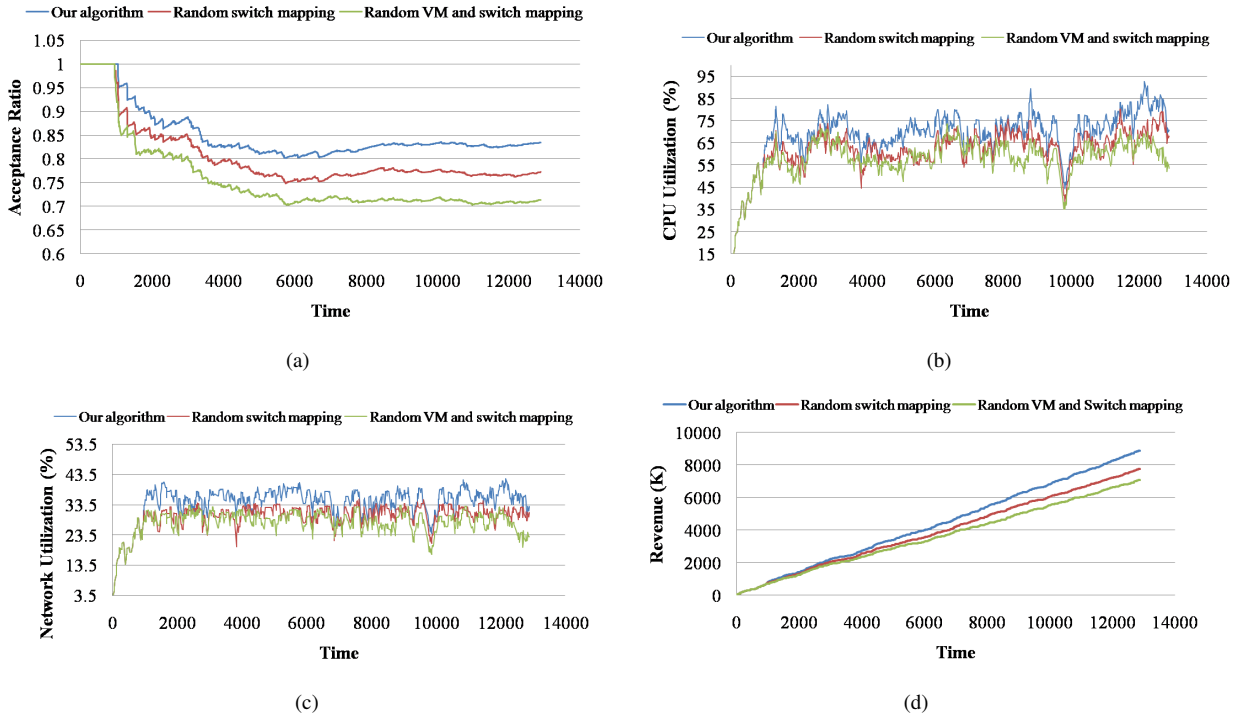

Fig. 5. Comparison of acceptance ratio, resource utilization and revenue over time for VDC requests of mix of VL2 and star topology. (a) Acceptance Ratio. (b) CPU Utilization. (c) Network Utilization. (d) Revenue

than random VM and switch mapping because random switch mapping utilizes our VM mapping phase for its VM mapping. In both of the above cases, our algorithm has higher acceptance ratio, as well as, higher revenue, which demonstrates the advantages of our VDC embedding algorithm.

VI. Conclusion

We have proposed a new formulation of VDC embedding problem where along with VMs and virtual links, we also considered virtual switches in the VDC requests. We also proposed a three-phase minimum-cost-flow-based heuristic to solve the embedding problem, considering residual bandwidth, server defragmentation, communication costs and load balancing, which can efficiently embed VDC requests of any topology. A discrete event simulator was developed to evaluate our proposed VDC embedding algorithm. We have showed that our embedding algorithm was able to embed VMs and virtual links, as well as virtual switches and our algorithm has higher acceptance ratio, CPU and network utilization, as well as higher revenue. We are planning to extend our work by considering VDC embedding across multiple data centers. Another possible extension of this work is to consider energy-aware VDC embedding.

References

[1] M. Al-Fares, A. Loukissas, and A. Vahdat, "A Scalable, Commodity Data Center Network Architecture," in *Proceedings ACM SIGCOMM*, August 2008.

[2] C. Guo, G. Lhttp://matwbn.icm.edu.pl/ksiazki/amc/amc21/amc2125.pdfu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers," in *Proceedings ACM SIGCOMM*, August 2009.

[3] A. Greenberg, J. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, and S. Sengupta, "VL2: A Scalable and Flexible Data Center Network," in *Proceedings ACM SIGCOMM*, August 2009.

[4] G. Wang and E. Ng, "The Impact of Virtualization on Network Performance of Amazon EC2 Data Center," in *Proceedings IEEE INFOCOM*, March 2010.

[5] M. Bari, R. Boutaba, R. Esteves, L. Granvilley, M. Podlesny, M. Rabbani, Q. Zhang, and F. Zhani, "Data Center Network Virtualization: A Survey," *to appear in IEEE Communications Surveys and Tutorials*, 2012.

[6] N. M. M. K. Chowdhury and R. Boutaba, "Network Virtualization: State of the Art and Research Challenges," *IEEE Comm. Mag.*, vol. 47, pp. 20–26, Jul. 2009.

[7] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking Virtual Network Embedding: Substrate Support for Path Splitting and Migration," *ACM Computer Communication Review*, vol. 38, no. 2, pp. 17–29, April 2008.

[8] M. Chowdhury, M. Rahman, and R. Boutaba, "Vineyard: Virtual network embedding algorithms with coordinated node and link mapping," *Networking, IEEE/ACM Transactions on*, vol. 20, no. 1, pp. 206 –219, feb. 2012.

[9] M. R. Rahman, I. Aib, and R. Boutaba, "Survivable Virtual Network Embedding," in *Proceedings IFIP Networking*, May 2010.

[10] N. F. Butt, N. M. M. K. Chowdhury, and R. Boutaba, "Topology-Awareness and Reoptimization Mechanism for Virtual Network Embedding," in *Proceedings IFIP Networking*, May 2010.

[11] C. Guo, G. Lu, H. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "SecondNet: A Data Center Network Virtualization Architecture with Bandwidth Guarantees," in *Proceedings ACM CoNEXT*, December 2010.

[12] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards Predictable Datacenter Networks," in *Proceedings ACM SIGCOMM*, August 2011.

[13] LEMON Graph Library. http://lemon.cs.elte.hu/trac/lemon.