# Leveraging Tenant Flexibility in Resource Allocation for Virtual Networks

Sheng Zhang[†], Zhuzhong Qian[†], Jie Wu[§], and Sanglu Lu[†]
[†]State Key Lab. for Novel Software Technology, Nanjing University, China
[§]Department of Computer and Information Sciences, Temple University, USA
[†]zhangsheng@dislab.nju.edu.cn, qzz@nju.edu.cn, sanglu@nju.edu.cn, [§]jiewu@temple.edu

***Abstract*—Virtual networks that allow tenants to explicitly specify their computing as well as networking resources are recently proposed to be better interfaces between cloud providers and tenants. Many virtual networks have time-varying resource demands, as evidenced in prior studies [1–3]. New opportunities emerge when such variation is exploited. In this paper, we design a novel resource demand model for tenants to flexibly trade off between application performance and cost, and propose a work-conserving allocation algorithm, *WCA*, for deploying virtual networks with time-varying resource demands. WCA places virtual nodes in a first-fit fashion, and places virtual links through path-splitting. In each physical node or link, by opportunistically sharing physical resources among multiple variable parts of resource demands, physical utilization can be improved, and more virtual networks can be deployed concurrently. Our evaluation results show that WCA achieves a 4% higher physical resource utilization and rejects 18% less virtual network requests than a state-of-the-art algorithm [4].**

## I. Introduction

As cloud computing becomes pervasive, data centers have evolved as key infrastructures for computation-intensive applications and business service backends [5]. The more recent emergence of virtual desktop [6], which exclusively relies on remote data centers for access, has further elevated the importance of the latter. Today's public data centers (*e.g.*, Amazon EC2, Microsoft Azure, and Google AppEngine) concentrate on a computation-oriented resource reservation model, which only allows tenants to specify computing and memory demands, but ignores networking completely. In other words, almost all current data centers just offer best-effort networking service, which is largely influenced by a multiplicity of various factors including virtual machines (VMs) placement, data center load and architecture.

Considering the scarcity and, not surprisingly, the common oversubscription [7] of data center bandwidth resources, this model results in high unpredictability of tenants' performance, which leads to, among others, two major negative consequences [8]. First, tenants' expenses are increased, since data center providers charge tenants based on the duration of a request, which depends on networking as well as computing resources; second, providers' revenues get lowered, since unpredictability impacts cloud applicability, and further limits the adoption of data centers.

To provide networking guarantee, prior works [1, 8, 9] have proposed several novel virtual network (VNet) abstractions that allow tenants to explicitly specify networking as well as computing demands. However, these researches have fallen into two broad types: (i) only tree topologies are considered [1, 8], contradicting the various dependencies among virtual machines that are observed in practice; (ii) fixed resources are reserved throughout the duration of a request [10–12], ignoring the time-varying resource demands. Although these works make a good start, they restricted the solution space at the expense of limiting the practical applicability.

We observed that many virtual networks have time-varying resource demands, as evidenced in prior studies [1–3]. To exploit such variations, we propose a novel resource demand model that allows a tenant to split the resource demand of a virtual node or link into three parts: one basic part and two variable parts. As shown in Section III, the probabilistic combination of these parts makes it easy for a tenant to control the trade-off between application performance and cost. A tenant can adjust its resource demands according to its expected application performance and budget.

Based on the proposed model, we study the problem of deploying virtual networks with time-varying resource demands to achieve high physical resource utilization, and propose a work-conserving allocation algorithm, WCA. It includes two stages: the *global* stage places virtual nodes in a first-fit fashion, and places virtual links through path-splitting; the *local* stage optimizes resource utilization in each physical node or link, through opportunistically sharing physical resources among multiple variable parts of resource demands. In doing so, physical utilization can be improved, and more virtual networks can be deployed concurrently. Our evaluation results show that WCA achieves a 4% higher physical resource utilization and rejects 18% less virtual network requests than a state-of-the-art algorithm [4]. This paper makes the following contributions:

(1) We propose a novel resource demand model for tenants to flexibly control the trade-off between application performance and cost.
(2) We design a work-conserving allocation algorithm for virtual networks. WCA leverages tenant flexibility in sharing multiple variable parts of resource demands.
(3) We demonstrate the efficiency and effectiveness of the proposed algorithm through simulations.

The remainder of this paper is organized as follows. We go over related work in Section II. Section III introduces the proposed resource demand model and the generation strategy. Problem formulation is presented in Section IV. We present WCA in Section V. Before we conclude this paper in Section VII, we perform evaluations in Section VI.

## II. Related Work

Our work is related to resource allocation in clouds. The capability of providing efficient resource allocation is central to data centers. Virtualization multiplexes and shares physical resources among tenants' applications, which reduces energy consumption, and finally translates into increased data center revenues and decreased tenant expenses. Prior works [2, 13] sought to minimize the number of running physical machines through VM consolidation [14], which is often regarded as a variant of the bin packing problem [15]. However, networking guarantee was rarely considered. Virtual networks were then proposed to allow tenants to specify their networking and computing demands explicitly [1, 8, 9]. To provide scalable network structure, recent studies [7, 16–18] developed several high-performance and scalable data center architectures, *e.g.*, VL2, Fat-tree, BCube, and DCell. Different from these works, we study resource allocation with a focus on leveraging tenant flexibility and achieving work-conserving allocations.

This paper is also related to previous works on network virtualization [19]. A key challenge in this field is the virtual network embedding problem, which deals with embedding multiple virtual networks with resource constraints in substrate networks, so as to efficiently utilize substrate resources. To cope with its NP-completeness [20], meta-heuristic-based algorithms are designed in [21]. The study of embedding with unlimited substrate resources is conducted in [4], in which load balancing and reconfiguration are also considered. Substrate supports for path splitting and computation parallelization are envisioned in [10] and [22], respectively. A subgraph isomorphism detection-based embedding algorithm is proposed in [11]. Linear programming and deterministic/randomized rounding-based algorithms are developed in [12] to deal with virtual networks with location constraints. Comparatively, in this paper, we design a novel model that captures the dynamic resource demands and provides the flexibility of controlling the trade-off between performance and cost to tenants.

## III. The Proposed Resource Demand Model

### A. Probabilistic Combination as Resource Demand

Tenants request resources in the form of VNets from data center providers to install their applications/services. A VNet is usually considered as a undirected graph, where nodes represent computing resource demands and links represents networking resource demands. The nodes and links in a VNet are often called *virtual nodes (VNs)* and *virtual links (VLs)*, respectively. When receiving a VNet request, a data center provider tries to map a VN and VL to a *physical machine (PM)* and a physical path, respectively, while respecting capacity constraints.

Tenants usually target potential end-users all over the world, so it is extremely difficult to predict the trend of workload changing. As the resource demand of a VNet at a particular time is generally proportional to the amount of workload at that time, it is also hard to forecast resource demands in the future. In the other hand, the variability of a virtual machine workload widely exists in modern data centers, as shown in prior works [14]. And in a previous profiling experiment [1], which measured networking traffic between pairs of VMs in a data center, it was found that inter-VM traffic also fluctuates over time. Therefore, we only explain the computing resource demand model in the rest of this section; the model can be applied to the networking resource demand without any major changes.

We assume that the data center is based on logical partitioning [23], *i.e.*, the underlying physical resources are time-multiplexed between different VMs, and time is partitioned into slots of equal length. We denote the computing resource demand of a VN $v$ at time $t$ by $R(v,t)$, and assume it is the probabilistic combination of three parts: a basic part $R_0(v,t)$, which exists throughout the lifetime of the VN, and two variable parts $R_1(v,t)$ and $R_2(v,t)$, which occur with probabilities of $p_1^v$ and $p_2^v$, respectively, *i.e.*, $R_1(v,t)$ and $R_2(v,t)$ follow Bernoulli distribution. More formally, $R(v,t) = R_0(v,t)+R_1(v,t)+R_2(v,t)$, where $R_0(v,t) = r_0^v$, $R_1(v,t) \backsim B(r_1^v, p_1^v)$, and $R_2(v,t) \backsim B(r_2^v, p_2^v)$. Thus, $R(v,t)$ can be characterized by a tuple $< r_0^v, r_1^v, p_1^v, r_2^v, p_2^v >$. Fig. 1 shows the probability distribution of $R(v,t)$.

| $R(v,t)$ | $r_0^v$ | $r_0^v + r_1^v$ | $r_0^v + r_2^v$ | $r_0^v + r_1^v + r_2^v$ |
|---|---|---|---|---|
| $P$ | $(1-p_1^v)(1-p_2^v)$ | $p_1^v(1-p_2^v)$ | $(1-p_1^v)p_2^v$ | $p_1^v p_2^v$ |

Fig. 1. Probability distribution of $R(v,t)$.

### B. Model Generation Strategy

This subsection introduces how a tenant generates the tuple $< r_0^v, r_1^v, p_1^v, r_2^v, p_2^v >$ for each VN and VL in its VNet. There are two challenges that we have to solve.

First, how to get the computing and networking usage traces and guarantee them to be consistent with the realistic deployment in data centers. We envision that the data center providers offer profiling runs for tenants to obtain their resource usage traces. Specifically, a tenant can tentatively deploy its VNet request in a data center for a relatively short time period; the data center provider collects the computing and networking usages over time, and feeds them back to the tenant.

Second, given the usage traces, how to generate an appropriate tuple for a VN or VL. We provide here a strategy for a tenant to flexibly control the trade-off between application performance and cost through tuning $p_1$ and $p_2$. It is better to illustrate the strategy using an example. In Fig. 2, the solid black curve shows the computing resource demand of a VN over time, while the dashed red curve represents our model. Given $p_1$ and $p_2$, we just have to find appropriate $t_1$, ..., and $t_{12}$, such that:

$$\frac{(t_{10}-t_9)+(t_4-t_3)}{t_{13}-t_0} = p_1 p_2$$
$$\frac{(t_3-t_2)+(t_5-t_4)+(t_9-t_8)+(t_{11}-t_{10})}{t_{13}-t_0)} = (1-p_1)p_2$$
$$\frac{(t_2-t_1)+(t_6-t_5)+(t_8-t_7)+(t_{12}-t_{11})}{t_{13}-t_0)} = p_1(1-p_2)$$

After we have the values of $t_1$, ..., and $t_{12}$, we can easily determine $r_0$, $r_1$, and $r_2$. Denote by $rdp(t)$ the resource demand at time $t$. We have $r_1 = max_t(rdp(t)) - rdp(t_3)$ and $r_2 = r_1 + rdp(t_3) - rdp(t_2)$.

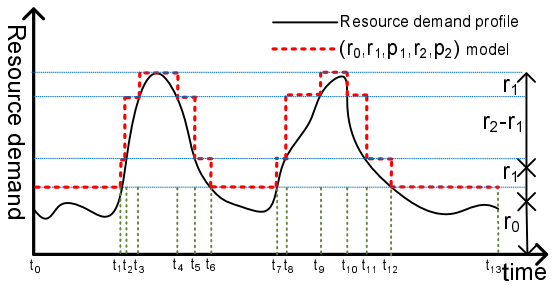



Fig. 2. An example of the model generation strategy. The parameters can be easily generated once $p_1$ and $p_2$ are specified.

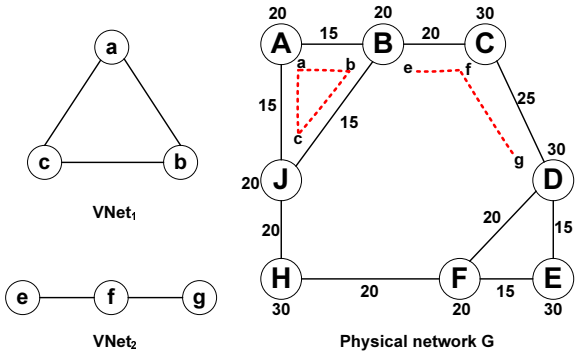



Fig. 3. Examples of VNets and physical network. The dashed red lines indicate a possible placement of the two VNets. For example, VN $b$ is placed in PM $B$; VL $(e, f)$ is placed in PL $(B, C)$.

### C. Tenant Flexibility

This model provides great flexibility for tenants to control trade-off between performance and cost through tuning $p_1$ and $p_2$ for each VN and VL. If a tenant has plenty of funds and only cares about performance, the tenant can set $p_1 = p_2 = 0$ for all VNs and VLs; otherwise, the tenant can adjust $p_1$ and $p_2$ to best suit its performance/cost objective. Overall, if a tenant wants to maximize its VNet performance, $p_1$ and $p_2$ should be small; if a tenant wants to minimize the placement cost that a cloud provider may charge him/her, $p_1$ and $p_2$ should be large.

Besides tenant flexibility, the proposed model exhibits some other desirable properties. First, traditional models can be seen as special cases of our model, *i.e.*, when $p_1 = p_2 = 0$ and $r_0$ equals the peak demand. It ensures that our system is backwards-compatible with the other forms of resource demands, *e.g.*, *virtual data center* [9], and *virtual cluster* [8]. Second, our model is also a tradeoff between modeling complexity and precision. When the number of parts in our model (currently 3) increases, the model precision increases, and hence, can express the realistic resource demands more accurately. However, the complexity in generating the model, not surprisingly, increases as well, which also greatly complicates the interactions between data center providers and tenants.

One limitation of our model is that, modeling generation incurs some profiling overhead and seems to be impractical in today's data center business architectures; however, this overhead can be drastically reduced if tenants have to reserve resources for the same type of VNets repeatedly and lastingly. For example, about 40% of applications are recurring in Bing's production data center [24]. For the same type of VNets, the data center provider only needs to offer one profiling run, and the same results could be fed back to tenants who want to deploy that type of VNet. In doing so, the profiling overhead for data center providers is greatly reduced.

## IV. Problem Statement

*VNet.* A VNet is denoted by a weighted undirected graph, $VNet = (V, E)$. $V$ is the set of virtual nodes (VNs); $E$ is the set of virtual links (VLs). Each VN $v$ is associated with a computing resource demand $R(v, t)$, and each VL $e$ is associated with a networking resource demand $R(e, t)$. The lifetime or duration of a VNet is denoted by $lt$. The left of Fig. 3 shows two examples of VNets.

*Physical network.* The cloud physical network is modeled as a weighted undirected graph, $G = (V^p, E^p)$, where $V^p$ denotes the set of PMs, and $E^p$ denotes the set of physical links (PLs). The computing resources in a PM $n \in V$ is denoted by $C(n)$. A PL $e_{ij}$ connects two PMs $n_i$ and $n_j$, *i.e.*, $e_{ij} = (n_i, n_j)$. The networking resources in a PL $e_{ij} = (n_i, n_j) \in E$ is denoted by $B(e)$. We use $P(n_i, n_j)$ to represent the set of loop-free physical paths between $n_i$ and $n_j$. We also denote by $P$ the set of all loop-free paths in the physical network. For example, in Fig. 3, there are 4 loop-free paths between $B$ and $J$ in the physical network, *i.e.*, $P(B, J) = \{BAJ, BJ, BCDFHJ, BCDEFHJ\}$. Fig. 4 summarizes the main notations in this paper for reference.

*Resource allocation.* Following existing work [9, 10, 12], we assume that one VN maps to one PM. When a tenant prefers to deploy multiple VNs in one PM, we treat all these VNs as one large VN by summing up their resource demands. The placement of a VNet can be decomposed into two phases, namely, *VN mapping* and *VL mapping*. The VN mapping phase $\mathcal{M}_V : V \rightarrow V^p$ maps a VN to a PM; the VL mapping $\mathcal{M}_E : E \rightarrow P$ maps a VL to a set of loop-free physical paths. Fig. 3 shows a placement of the two VNets. For example, the VN mapping for $VNet_1$ is $\{a \rightarrow A, b \rightarrow B, c \rightarrow J\}$, and the VL mapping is $\{(a, b) \rightarrow \{(A, B)\}, (b, c) \rightarrow \{(B, J)\}, (c, a) \rightarrow \{(J, A)\}\}$.

*Collision threshold.* Since different VNets are operated by different tenants and offer different services, it is reasonable to assume that the resource demands of VNs from different VNets are mutually independent. To maximize the resource utilization, we propose to share physical resources opportunistically among multiple variable parts of resource demands. However, capacity violations accompany sharing: when more than one variable part occurs simultaneously, a violation happens. To provide probabilistic performance guarantee, a cloud provider should at least *bound the maximum collision probability*. We denote this threshold by $p_{th}$. For example, in Fig. 3, VN $b$ from $VNet_1$ and VN $e$ from $VNet_2$ are placed on the same PM. Suppose that $R(b, t) =< 8, 1, 0.1, 2, 0.1 >$ and $R(e, t) =< 6, 1, 0.2, 2, 0.2 >$. If resource sharing is not exploited as in prior studies, these two VNs would occupy a total of 20 units of resources on PM $B$. However, when resource sharing is allowed, and we assume that the cloud provider guarantees that $p_{th} = 0.1$; since $p_1^b \cdot p_1^e = p_2^b \cdot p_2^e = 0.1 \times 0.2 = 0.02 < p_{th}$, we only have to allocate 3 units of resources to the variable parts of resource demands. We find that, in this way, they only occupy a total of $8 + 6 + 3 = 17$ units of resources on $B$.

| Notation | Meaning |
|---|---|
| $VNet = (V, E)$ | a virtual network |
| $R(v, t)$ | the resource demand of a VN $v$ |
| $lt$ | the lifetime of a VNet |
| $G = (V^p, E^p)$ | the physical network |
| $C(n)$ | capacity of PM $n$ in terms of computing resources |
| $B(e)$ | capacity of PL $e$ in terms of networking resources |
| $AC(n)$ | the available computing resources in PM $n$ |
| $AB(e)$ | the available networking resources in PL $e$ |
| $P(n_i, n_j)$ | the set of loop-free paths between PMs $n_i$ and $n_j$ |
| $p_{th}$ | collision threshold |

Fig. 4. Summary of main notations in this paper.

*Objective.* This paper focuses on designing an allocation algorithm for placing VNets that arrive and depart one by one over time. Upon the arrival of a VNet request, the allocation algorithm must decide whether or not accept it. Batch processing is not the focus of the paper.

Our goal is to maximize the cloud provider's revenue through efficiently utilizing physical resources. Following prior research [3, 10, 12], the revenue of embedding a *VNet* should be proportional to both the amount of allocated resources and its lifetime $lt$. Therefore, it can be defined as follows:

$$\mathbb{R}(VNet) = [\alpha \sum_{n \in V} (r_0^n + p_1^n r_1^n + p_2^n r_2^n) + \beta \sum_{e \in E} (r_0^e + p_1^e r_1^e + p_2^e r_2^e)] \cdot lt \quad (1)$$

where $\alpha$ and $\beta$ are the weights for cloud providers to control the trade-off between utilizing computing and networking resources. Therefore, the total revenue of a cloud provider can be denoted by $\sum_{VNet} \mathbb{R}(VNet)$. To maximize it, the physical resources must be efficiently utilized, and VNets must be properly placed. We formally define our problem below.

***Problem 1:*** (VNet Placement Problem) Given a physical network $G = (V^p, E^p)$ and a $VNet = (V, E)$, find a placement for the VNet to maximize the cloud provider's revenue while respecting capacity constraints.

## V. The Solution

### A. Overview

We provide an overview of our work-conserving allocation algorithm, namely, WCA, which leverages tenant flexibility to improve physical resource utilization through opportunistic sharing. In general, WCA consists of two stages: the global and local stages. The global stage produces the mapping from VNs and VLs to PMs and physical paths, respectively; the local stage deals with resource sharing in each PM and PL.

Concretely, in the *global* stage, we sort VNs in the descending order of their respective expected resource demands. Then, we place each VN in that order in the unused PM with the most residual resource. This kind of "maximum-first" placement fashion has several advantages, *e.g.*, it is beneficial to future VNets that may request some kind of bottleneck resource. After all VNs have their destinations, we proceed to map VLs to physical paths. For each VL, we try to map it to the shortest path with sufficient networking resource between the PMs that the two endpoint VNs of the VL are placed in. If we cannot find such a path, we then divide the networking demand of the VL into two equal portions, and map them separately. If such paths cannot be found again, we continue to split the demand into three equal portions; however, we restrict the number of portions to be no more than $K$ for efficiency concerns.

**Algorithm 1** WCA

**Require:** $G = (V^p, E^p)$, $VNet = (V, E)$
1: $Q \leftarrow$ sorted $V$ with decreasing $(r_0^v + r_1^v p_1^v + r_2^v p_2^v)$
2: $Q^p \leftarrow$ sorted $V$ with increasing $AC(n)$
3: **for** $i = 1$ to $Q.length$ **do**
4: for increasing $j$, map $Q[i]$ to the first $Q^p[j]$ if SharingFeasibility($Q^p[j], Q[i]$) returns true
5: **end for**
6: $Q^e \leftarrow$ sorted $E$ with decreasing $(r_0^e + r_1^e p_1^e + r_2^e p_2^e)$
7: **for** $h = 1$ to $Q^e.length$ **do**
8: **for** $splitRaio = 1$ to $K$ **do**
9: $G' \leftarrow$ GetReducedGraph($G, \frac{R(Q^e[h],t)}{splitRatio}$)
10: **if** $Q^e[h]$'s end-hosts are connected in $G'$ **then**
11: map $Q^e[h]$ to the shortest path(s) between
12: the two end-hosts, and **break**
13: **end if**
14: **end for**
15: **if** $splitRatio > K$ **return false**
16: **end for**
17: **return true**

In the *local* stage, we seek to minimize the amount of physical resource used for placing multiple variable parts of resource demands. Through sharing physical resource between multiple variable parts of demands, the resource allocation becomes work-conserving. Given multiple variable parts of resource demands and collision threshold $p_{th}$, we must decide how to divide the variable parts of demands into groups, such that the capacity violation probability in each group is no more than the threshold. We formulate this problem as a variant bin-packing problem [15], except that the bin size is now the collision threshold.

### B. The Global Stage

We denote by $AC(n)$ and $AB(e)$ the available resource of PM $n$ and PL $e$, respectively. The global stage of WCA includes two phases, *i.e.*, VN and VL mapping, as shown in Alg. 1. In the VN mapping phase (lines 1–5), we sort VNs in the descending order of their respective expected computing resource demands, *i.e.*, $(r_0^v + r_1^v p_1^v + r_2^v p_2^v)$, and sort PMs in the ascending order of their respective available resources. VN $Q[i]$ is mapped to the first $Q^p[j]$ if the local stage algorithm SharingFeasibility($Q^p[j], Q[i]$) returns true. SharingFeasibility handles local resource sharing in each PM and PL. For simplicity of presentation, we omit the failure detection in the pseudocode shown in Alg. 1: if there are not enough computing resources, the VNet would be rejected. The reason for sorting PMs and VNs is implicit, if somewhat subtle: VNs with larger resource demands are more difficult to map. In the case that we cannot find an appropriate PM for a VN, we can fail faster and switch to the next VNet. Sorting also helps us to minimize the number of running PMs and cut down data center energy consumption via turning off idle PMs.

In the VL mapping phase (lines 6–16), we sort VLs in the descending order of their respective expected networking

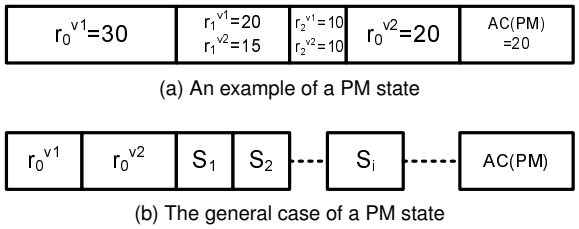



Fig. 5. An example and the general case of the resource allocation state of a PM, where the collision threshold $p_{th}$ is 0.1.

demands for the same reason as above, and map a VL $Q^e[h]$ to the shortest physical path with sufficient networking resource between the PMs that the two endpoint VNs of the VL are placed in. If such a physical path does not exist, *i.e.*, $Q^e[h]$'s end-hosts are disconnected in the reduced graph $G'$, which is obtained by removing physical links that cannot accommodate $Q^e[h]$ from $G$. We then split the resource demand of $Q^e[h]$ into two equal portions and try to map them to the shortest paths in a new reduced graph. We repeat this kind of split-and-map process at most $K$ times for computational efficiency concerns.

### C. The Local Stage

It is better to illustrate how physical resources are shared using an example. Fig. 5(a) shows the current resource allocation state of a PM, which has 100 units of computing resources in total, and hosts two computing resource demands: $< 30, 20, 0.4, 10, 0.3 >$ from $v1$ and $< 20, 15, 0.2, 10, 0.1 >$ from $v2$. The collision threshold $p_{th}$ is 0.1. Since $0.4 \times 0.2 = 0.08 < 0.1$, we let the first variable parts of $v1$ and $v2$ share $max(r_1^{v1}, r_1^{v2}) = 20$ units of computing resources; similarly, the second variable parts of $v1$ and $v2$ share another 10 units of computing resources, as shown in the figure.

Now we want to check whether a third demand $< 20, 15, 0.3, 5, 0.1 >$ from $v3$ can be placed in this PM. We see that the amount of available resource $AC(PM)$ is enough to host the basic part, which is 20 units. However, is it feasible for the first variable part of $v3$ to share the 20 units of computing resources with $v1$ and $v2$? The answer is negative, since the collision probability is 0.212, which is larger than 0.1. Therefore, the third demand cannot be placed in the PM. Fig. 5(b) shows the general case of a PM state, where $S_i$ denotes the set of variable parts of resource demands that share $max(r_k^v \in S_i)$ units of computing resources. We denote the general state of a PM by $< r_0^{v_1}, ..., r_0^{v_n}, S_1, ..., S_m, AC(PM) >$.

Given multiple variable parts of resource demands and collision threshold $p_{th}$, we must decide how to divide the variable parts of demands into groups, such that the capacity violation probability in each group is no more than the threshold. We formulate this problem as a variant bin-packing problem [15], except that the bin size is now the collision threshold. First-fit is an approximation algorithm with a factor of 2 for bin-packing. First-fit attempts to place an item to the first bin that can accommodate the item; if this is not possible, the item is places into a new bin. We applying the core idea of first-fit to our problem. As we deal with the online case, it is enough to show: given a PM state and a computing resource demand, whether is it feasible for us to place the demand in the given PM?

**Algorithm 2** SharingFeasibility(PM $p$,VN $v$)

**Require:** PM state $< r_0^{v_1}, ..., r_0^{v_n}, S_1, ..., S_m, AC(p) >$, and a computing resource demand $< r_0^v, r_1^v, p_1^v, r_2^v, p_2^v >$

```
1: if AC(p) < r_0^v return false
2: AC(p) ← AC(p) − r_0^v
3: j ← 1
4: for i = 1 to m do
5:    if Pr(S_i ∪ {r_j^v}) ≤ p_th
6:       diff ← r_j^v − max(r_k^v ∈ S_i)
7:       if diff > 0
8:          AC(p) ← AC(p) − diff
9:       S_i ← S_i ∪ {r_j^v}
10:      j ← j + 1
11:      if j > 2 return true
12: end for
13: if j == 1 & AC(p) > r_1^v + r_2^v
14:    AC(p) ← AC(p) − r_1^v − r_2^v
15:    return true
16: if j == 2 & AC(p) > r_2^v
17:    AC(p) ← AC(p) − r_2^v
18:    return true
19: return false
```

Alg. 2 shows the sharing feasibility checking algorithm. The inputs include a PM state and a computing resource demand. In lines 1–2, we checks whether there is enough resources for the basic part $r_0^v$; in lines 3–12, SharingFeasibility tries to map $r_1^v$ and $r_2^v$ to existing resource blocks. Here, a first-fit-based strategy to search an existing resource block that can accommodate $r_1^v$ in terms of capacity violation probability (line 5). Denote by $Pr(S_i)$ the capacity violation probability of a set $S_i$ of variable parts of resource demands, which is defined as

$$Pr(S_i) = 1 - \prod_{r_h^{v_j} \in S_i} (1 - p_h^{v_j}) - \sum_{r_h^{v_j} \in S_i} (p_h^{v_j} \prod_{r_h^{v_k} \in S_i, k \neq i} (1 - p_h^{v_k})) \quad (2)$$

where $h = 1$ or 2. When such a block is found, we may need to resize the block (lines 6–8) and continue to find another existing block for $r_2^v$. Here, we emphasize that $r_1^v$ and $r_2^v$ are dependent, and hence they cannot be mapped to the same existing block. In lines 13–18, SharingFeasibility then tries to allocate new resources from $AC(p)$ to the one (or even two) variable parts.

SharingFeasibility linearly scans resource blocks in a PM, and it costs $O(C_m)$ time, where $C_m = max_{n \in V^p}(C(n))$. The time complexities of sorting VNs, PMs, and VLs are $O(|V|log|V|)$, $O(|V^p|log|V^p|)$, and $O(|E^p|log|E^p|)$, respectively. The VN mapping phase takes $O(|V||V^p|C_m)$ time. The function *GetReducedGraph* takes $O(|E^p|C_m)$ time; breadth-first-search for checking connectivity costs $O(|V^p| + |E^p|)$ time; Dijkstra shortest path algorithm [25] takes $O(|E^p|)log(|V^p|))$ time; hence, the VL mapping phases take $O(K|E|(|E^p|C_m + |V^p| + |E^p| + |E^p|log(|V^p|))) = O(K|V|^2||V^p|^2(C_m + log|V_p|))$ time. Here we have simplified the summations by using $|E| = O(|V|^2)$ for any simple graph. And the overall time complexity of WCA is $O(K|V|^2|V^p|^2(C_m + log|V^p|))$.

### D. Discussions

We summarize this section by providing a few implementation issues. As discussed in [9], we may choose to

adopt source routing as its routing strategy. Since, otherwise, switches have to maintain bandwidth reservation states and MAC address tables, which impacts the scalability of WCA. To enforce bandwidth reservations, hypervisors and switches should perform the rate limiting function.

To further improve the computational efficiency of SharingFeasibility, we can partition the resource in a PM or a PL into multiple elementary blocks of equal size, each of which should be taken as a whole (thus cannot be partitioned into even smaller pieces) in the resource allocation process. However, this may reduce physical resource utilization, providing a trade-off for system designers. The maximum number of repetitions, *i.e.*, $K$, should be kept small, both for computational concerns and resource utilization. Because the overhead in the packet header also increases when the number of portions increases.

Recall that SharingFeasibility is invoked whenever we want to check whether a given resource demand can be embedded in a PM or PL. In fact, we can conservatively round $p_1^v$ and $p_2^v$ to several fixed values, then enumerate all of the possible combinations of these fixed values. In doing so, the invoking of SharingFeasibility is reduced to looking up in a predetermined table, making WCA more efficient and scalable.

The resource type in our algorithm is one-dimensional, and here we outline its adaption to the scenario of multi-dimensional resources. There are two cases: correlated and independent resources. The former case can be easily solved by transforming them into one-dimensional compositive resources, while, in the latter case, we can run our algorithm for each type of resource separately. For the off-line and batch arrival scenarios, we only have to sort VNs in the descending order of their total resource demands, and then allocate physical resources to them sequentially.

One of our previous studies envisions substrate support for parallelization [22], *i.e.*, data centers support parallel computation and allows a VN to be mapped to multiple PMs. Parallelization not only enables data centers to achieve higher resource utilizations through making efficient use of fragmented physical resources, but also makes data centers more reliable, since computation tasks can quickly migrate to other PMs in the case that a PM crashes. We leave incorporating parallelization support into WCA as future work.

## VI. Evaluation

In this section, we provide the evaluation results of the proposed algorithm and make some remarks.

### A. Simulation Setup

We use simulations to study the performance of the proposed algorithm. All the experiments are performed on a Lenovo T410RT5 PC with two 2.67Hz Intel i7 CPUs and 4G memory. The simulation settings follow prior work [3, 10, 12]. The physical network contains 60 PMs. Each pair of them is connected with a probability of 0.3. The capacity of every PM is 100 units of CPU, and the capacity of every PL is 100 units of bandwidth. By default, the threshold of capacity violation probability $p_{th}$ is set to 0.2.

The number of VNs in a VNet is determined by a uniform distribution between $(Avg-4)$ and $(Avg+4)$, where $Avg$ denotes the average VNet size. Each pair of VNs are also connected with a probability of 0.3. We also check whether the physical network and VNet topologies are connected; if not, we just regenerate them until they are connected. The peak resource demand of a VN or a VL is randomly chosen between 20 and $HR$, where $HR$ denotes the high bound. Recall that our resource demand model allows a tenant to control its trade-off between application performance and cost through tuning $p_1$ and $p_2$. By default, $p_1 = 0.3$, and $p_2 = 0.1$. The lifetime of each VNet is assumed to be exponentially distributed with an average of 300 seconds. The arrivals of VNets are modeled as a Poisson process with parameter $\lambda$. By default, $Avg = 6$, $HR = 30$, $\lambda = 1/12$ seconds, and the maximum number of portions that a networking resource demand can be divided into is 3. Fig. 6 summarizes main simulation parameters and their values by default.

| Notation and its value by default | Definitions |
|---|---|
| $K = 3$ | the maximum number of portions that a networking demand can be split into |
| $\lambda = 1/12$ | the average interval between two consecutive VNets' arrivals |
| $p_{th}$=0.2 | collision threshold |
| $Avg$=6 | the average number of VNs in a VNet |
| $HR$=30 | the maximum resource demand of a VN or VL |
| $p_1$=0.3 | the occurring probability of the 1st variable part |
| $p_2$=0.1 | the occurring probability of the 2nd variable part |

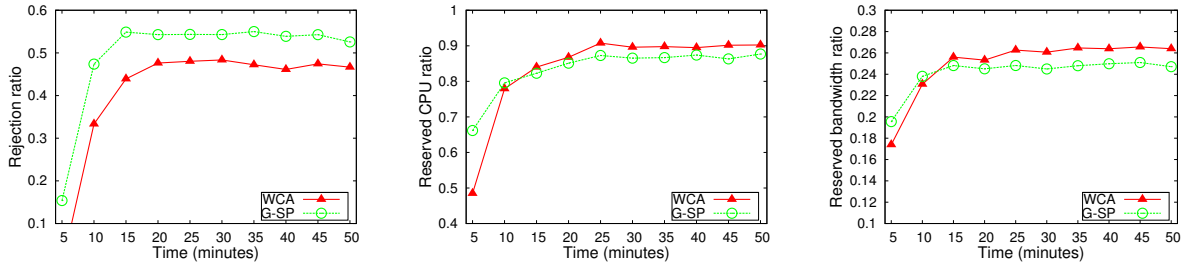
Fig. 6. Simulation parameters and their values by default.

### B. Simulation Results

We compare the proposed allocation algorithm with $G-SP$ [4], which does not share physical resources between resource demands, and adopts a similar heuristic as ours to deal with VN and VL mappings. The performance metrics we use for comparison include: *rejection ratio*, which is the ratio of the number of rejected VNets to all VNets; *reserved CPU ratio*, which is the ratio of the amount of reserved computing resources to the total computing resources; and *reserved bandwidth ratio*, which is the ratio of the amount of reserved networking resources to the total networking resources.

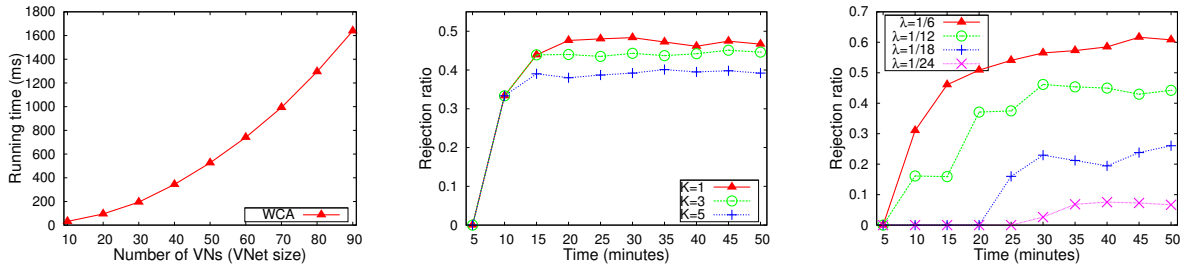
Fig. 7(a) shows the comparison results of rejection ratio between WCA and G-SP. We observe that the rejection ratio of WCA first increases, then reaches its stable value after 15 minutes. This is because the data center is empty and has plenty of resource at first, since we begin to generate VNets from time 0. We note that WCA achieves a up to 4% higher physical resource utilization and rejects up to 18% less VNets than G-SP, implying that opportunistic resource sharing improves physical resource utilization efficiency.

The comparison results of reserved CPU and bandwidth ratios are shown in Figs. 7(b) and (c), respectively. Generally speaking, WCA reserves more resources than G-SP most of the time, indicating that WCA achieves a higher CPU resource utilization. In both figures, we notice an interesting observation: G-SP reserves more resources than WCA during the first ten to fifteen minutes. To understand this, we only




(a) Percentage of rejected requests (b) Reserved CPU resource (c) Reserved bandwidth resource

Fig. 7. Performance comparison between WCA and G-SP [4].




(a) Running time with varying VNet size (b) Impact of $K$ (c) Rejection ratio with varying $\lambda$

Fig. 8. The running time of WCA, and the impact of $K$ and $\lambda$.

have to know, given a fixed number of VNets, WCAreserves less resources than G-SP.

Fig. 8(a) demonstrates the running time of WCA with a varying number of VNs in a VNet, where we set the number of PMs to be 100. For a VNet with 90 VNs, we can perform the deployment within 1.6 seconds. The result shows that the deployment time only grows linearly with the number of VNs, which shows the scalability of the proposed algorithm.

We are also interested in the impact of some parameters, including $K$, $\lambda$, $p_{th}$, $Avg$, $HR$, $p_1$, and $p_2$. We run simulations with one of the parameters in Fig. 6 varying while keeping the other parameters to be their respective by-default values.

Fig. 8(b) shows the comparison results when $K$ takes three different values. Not surprisingly, as $K$ increases, a VL resource demand can be split into more portions, each of which becomes easier to be placed, hence, the rejection ratio decreases. We note that $K$ also provides a tradeoff between resource utilization and computational overhead, since path splitting incurs additional packet headers. We also observe that, when data center load is slight (e.g., the first ten minutes), three settings perform almost the same, due to the plethora of networking resources.

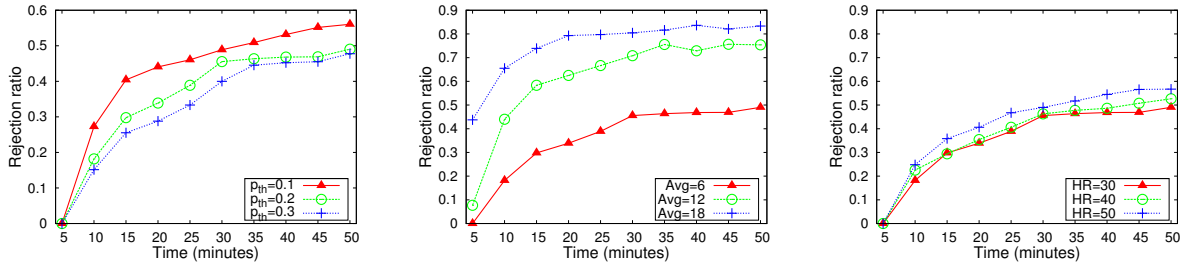
Fig. 8(c) shows how WCA performs under different loads. The Poisson process parameter $\lambda$ is used to control the deploying load. Generally speaking, when $\lambda$ increases, the rejection ratio also goes up. The main reason is that, as $\lambda$ goes up, the expected time interval between two consecutive VNets' arrivals, which equals $1/\lambda$, becomes shorter; thus, more VNets will arrive in a fixed period of time.

Fig. 9(a) illustrates the impact of $p_{th}$ on rejection ratio. When $p_{th}$ increases, the rejection ratio decreases. This is reasonable, because a large $p_{th}$ allows physical resources to be shared among more variable parts of resource demands than small ones.

We then keep the probability of two VNs connecting unchanged, and investigate the impact of VNet size on rejection ratio, shown in Fig. 9(b). As a whole, when the size of a VNet increases, its resource demand also becomes larger, and the rejection ratio hence goes up. For example, at time 50, the rejection ratios of $Avg$ = 6, $Avg$ = 12, and $Avg$ = 18 are 0.49, 0.75, and 0.83, respectively. Fig. 9(c) shows the impact of $HR$, where the rejection ratio of $HR$ = 30 is the smallest. When we enlarge $HR$, the resource demand of a single VN or VL increases on average, hence the total resource demands of a VNet also increase, which makes it more difficult for the physical network to accept upcoming VNets.

Fig. 10 shows the rejection ratio of WCA against varying $p_1$ and $p_2$. The proposed demand model exposes choice of parameters to tenants. We notice that, as expected, when $p_1$ and $p_2$ become larger, WCA rejects less VNets. This is because, when $p_1$ and $p_2$ become larger, it is more likely that a larger portion of resource demands would become variable parts. However, we note that, when the occurring probabilities of variable parts of resource demands increase, the average number of variable parts that can share the same amount of physical resources would decrease. The rejection ratios of four different settings of $p_1$ and $p_2$ demonstrate that our model indeed allows tenants to control the trade-off between performance and cost, and thus choose parameters to best suit their objectives.

In summary, WCA performs well and exhibits good scalability in a variety of settings. Sharing resources opportunistically and path splitting improve resource utilization. We wish our simulation will provide some potential guidelines for the future design of such algorithms.




(a) Rejection ratio with varying $p_{th}$ (b) Rejection ratio with varying $Avg$ (c) Rejection ratio with varying $HR$

Fig. 9. Sensitivity analysis: the impact of $p_{th}$, $Avg$, and $HR$.

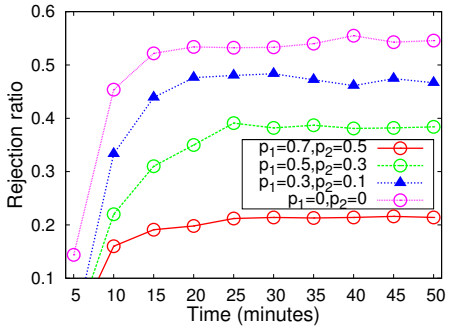



Fig. 10. The impact of varying $p_1$ and $p_2$.

## VII. Conclusion

In this paper, we study the problem of placing virtual networks with time-varying resource demands, and we propose a novel resource demand model that allows a tenant to flexibly control the trade-off between application performance and cost. Leveraging such flexibility, the proposed algorithm WCA achieve work-conserving resource allocation, and thus improves physical resource utilization through resource sharing and path splitting. Simulation results demonstrate the effectiveness and efficiency of our model and algorithms.

In retrospect, our work leaves several issues open for future research. While we are limited to modeling the resource demand of a VN or VL as a probabilistic combination of three parts, we do not know whether cyclic resource demand patterns exist. Another open problem is to incorporate live migration and parallelization support into WCA. We are also going to improve the proposed solution through exploiting the unique features of data center networks, *e.g.*, topology regularity, application behavior, and so on.

## Acknowledgements

We thank the ICCCN reviewers for feedback on earlier drafts of the paper. This work was supported in part by NSFC Grants (61073028, 61202113, 61321491, and 91218302), Key Project of Jiangsu Research Program Grant (BE2013116), Jiangsu NSF Grant (BK2011510), Research and Innovation Program for Jiangsu Graduates Grant (CXZZ12_0055), Program A for Outstanding PhD candidate of Nanjing University (201301A08), HUAWEI Project (YBIN2011056), US NSF grants (ECCS 1128209, CNS 1065444, CCF 1028167, CNS 0948184, and CCF 0830289), and EU FP7 IRSES MobileCloud Project Grant (612212).

## References

[1] D. Xie, N. Ding, Y. C. Hu, and R. Kompella, "The only constant is change: incorporating time-varying network reservations in data centers," in *ACM SIGCOMM 2012*.

[2] M. Wang, X. Meng, and L. Zhang, "Consolidating virtual machines with dynamic bandwidth demand in data centers," in *IEEE INFOCOM 2011*.

[3] S. Zhang, Z. Qian, J. Wu, S. Lu, and L. Epstein, "Virtual network embedding with opportunistic resource sharing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 3, pp. 816–827, March 2014.

[4] Y. Zhu and M. Ammar, "Algorithms for assigning substrate network resources to virtual network components," in *IEEE INFOCOM 2006*.

[5] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *CACM*, vol. 53, no. 4, pp. 50–58, 2010.

[6] S. Kim, D. Kim, S. Kim *et al.*, "Method and architecture for virtual desktop service," 2013, US Patent 20,130,007,737.

[7] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: a scalable and flexible data center network," in *ACM SIGCOMM 2009*.

[8] H. Ballani, P. Costa, T. Karagiannis, and A. I. Rowstron, "Towards predictable datacenter networks." in *ACM SIGCOMM 2011*.

[9] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "Secondnet: a data center network virtualization architecture with bandwidth guarantees," in *ACM CoNEXT 2010*.

[10] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: substrate support for path splitting and migration," *ACM SIGCOMM CCR*, vol. 38, no. 2, pp. 17–29.

[11] J. Lischka and H. Karl, "A virtual network mapping algorithm based on subgraph isomorphism detection," in *ACM VISA 2009*.

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

[13] D. Breitgand and A. Epstein, "Improving consolidation of virtual machines with risk-aware bandwidth oversubscription in compute clouds," in *IEEE INFOCOM 2012*.

[14] W. Vogels, "Beyond server consolidation," *Queue*, vol. 6, no. 1, pp. 20–26, 2008.

[15] V. V. Vazirani, *Approximation Algorithms*. Springer, 2003.

[16] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *ACM SIGCOMM 2008*.

[17] C. Guo, G. Lu, 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 *ACM SIGCOMM 2009*.

[18] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: a scalable and fault-tolerant network structure for data centers," in *ACM SIGCOMM 2008*.

[19] N. Chowdhury and R. Boutaba, "A survey of network virtualization," *Computer Networks*, vol. 54, no. 5, pp. 862–876, 2010.

[20] D. G. Andersen, "Theoretical approaches to node assignment," Dec. 2002, Computer Science Department, Carnegie Mellon University.

[21] R. Ricci, C. Alfeld, and J. Lepreau, "A solver for the network testbed mapping problem," *ACM SIGCOMM CCR*, vol. 33, no. 2, pp. 65–81, 2003.

[22] S. Zhang, J. Wu, and S. Lu, "Virtual network embedding with substrate support for parallelization," in *IEEE Globecom 2012*.

[23] T. L. Borden, J. P. Hennessy, and J. W. Rymarczyk, "Multiple operating systems on one processor complex," *IBM Systems Journal*, vol. 28, no. 1, pp. 104–123, 1989.

[24] S. Agarwal, S. Kandula, N. Bruno, M.-C. Wu, I. Stoica, and J. Zhou, "Re-optimizing data-parallel computing," in *USENIX NSDI 2012*.

[25] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. The MIT Press.