Synergistic Policy and Virtual Machine Consolidation in Cloud Data Centers

Lin Cui*, Richard Cziva‡, Fung Po Tso†, Dimitrios P. Pezaros‡
*Department of Computer Science, Jinan University, Guangzhou, China
†School of Computing & Mathematical Science, Liverpool John Moores University, UK
‡School of Computing Science, University of Glasgow, Glasgow, UK
Email: tcuilin@jnu.edu.cn; p.tso@ljmu.ac.uk; richard.cziva, dimitrios.pezarios@glasgow.ac.uk

***Abstract*—In modern Cloud Data Centers (DC)s, correct implementation of network policies is crucial to provide secure, efficient and high performance services for tenants. It is reported that the inefficient management of network policies accounts for 78% of DC downtime, challenged by the dynamically changing network characteristics and by the effects of dynamic Virtual Machine (VM) consolidation. While there has been significant research in policy and VM management, they have so far been treated as disjoint research problems.**

In this paper, we explore the simultaneous, dynamic VM and policy consolidation, and formulate the *Policy-VM Consolidation* (PVC) problem, which is shown to be NP-Hard. We then propose *Sync*, an efficient and synergistic scheme to jointly consolidate network policies and virtual machines. Extensive evaluation results and a testbed implementation of our controller show that policy and VM migration under *Sync* significantly reduces flow end-to-end delay by nearly 40%, and network-wide communication cost by 50% within few seconds, while adhering strictly to the requirements of network policies.

I. Introduction

As Cloud computing sees widespread adoption, data centers (DCs), the underpinning infrastructures, are challenged by the increased complexity of network management in which the configuration of (virtualized) server connectivity is dictated by numerous of network policies. In order to implement the desired network policies to ensure security and high performance, operators typically deploy a diverse range of "middleboxes" (MBs), such as firewalls, load balancers, Intrusion Detection and Prevention Systems (IDS/IPS), and application acceleration boxes [1]. In particular, one of the design requirements for today's Cloud DCs is to support the insertion of new MBs [2][3]. As a result, the number of MBs is on par with the number of routers and switches in enterprise networks [4]. Recent studies have also shown that the advent of diverse consumer devices will further increase the demand for in-network services [5]. Clearly, the added number of intermediate networking devices has in turn added a sheer degree of difficulty to network management. Research literature has demonstrated that deploying applications in Cloud DCs without considering in-network policies can lead to up to 91% policy violations, since network policies demand traffic to traverse a sequence of specified MBs [6]. Policy violations will potentially lead to severe consequences, including network outage, performance degradation [7], security vulnerabilities (e.g., firewall bypassing) or data leakage [4][6]. We argue that this challenge is amplified by the dynamism of traffic relocation as a result of dynamic Virtual Machine (VM) migration in today's virtualized DCs since when a VM is migrated, all related flows must be updated across the network [2].

Significant amount of research has been put into the areas of network policy and VM management, respectively. However, dynamic VM and network policy consolidation have been so far addressed in isolation. In the area of policy management, recent studies have focused primarily on exploiting Software-Defined Networking (SDN) [8] and Network Function Virtualization (NFV) [9], assuming a static allocation of compute resources. The centralized global view of the network provided by SDN can be exploited to programmatically ensure correctness of MB traversal. Coupled with SDN, software-based, virtualized network functions can be consolidated on demand. SDN and NFV have enabled a new paradigm for enforcing and dynamically migrating MB policies [8][10][11][12][13]. At the same time, VM management has largely concentrated on the efficient placement, consolidation and migration of Virtual Machines (VMs) to maximize server-side resources (such as CPU, RAM and network I/O [14], energy efficiency [15]), and optimize application-related Service Level Agreements (SLA) [16]. However, there has been no work on dynamic VM consolidation in conjunction with dynamic policy reconfiguration to optimize the network-wide communication cost. We argue that treating VM and policy management separately can lead to suboptimal network utilization and policy violations.

To see why, consider the following example scenarios as depicted in Fig. 1 where the traffic flows from v_1 to v_2 are configured to be checked by an IPS (the path is shown in solid blue line). (1) *Violation - MB capacity overloaded*: As shown in Fig. 1a, since the capacity of each IPS is constrained by their processing rate, migrating policies from IPS_1 to an overloaded IPS_2 will lead to the packets of this flow being rejected. (2) *Violation - route unreachable*: As demonstrated in Fig. 1b, assuming s_2 and LB_1 are behind VLAN1, while s_3 is behind VLAN2, then flows from v_1 to v_2 are configured to be checked by IPS_1 and load balancer LB_1. If we migrated v_2 to s_3, packets of the flow will fail to be sent to v_2 if the policy requirement is enforced. (3) *No violation - suboptimal network utilization*: We demonstrate two cases in Fig. 1c. If we only consider VM migration, for example using S-CORE [14], a network-aware communication cost reduction scheme, v_2 will be migrated to s_1 to be collocated with v_1 in a bid to reduce the VM-to-VM communication cost. Alternatively, only policy migration might be considered, using, for example, CoMb [11] to migrate policies from IPS_1 to IPS_2. However, in neither

of the above cases the true communication cost is improved, since the traversal path between v_1 and v_2 will remain the same length and use overloaded links. An ideal solution is to consider both policy and VM migration by migrating VM v_1 to s_1 and also migrating the traffic inspection policies concerning v_1 and v_2 from IPS_1 to IPS_2.

The above observations call for a new policy management scheme which can adapt to dynamic policy re-configurations as a result of VM migrations, and reinforce network policy in Cloud DC environments.

In this paper, we study the joint dynamic policy and VM migration problem on top of a SDN-based environment. We initially formulate and model the problem based on legacy hardware-based MBs due to three reasons. Firstly, they are ubiquitously available in today's DCs. Secondly, there have been concerns regarding the efficiency of fully virtualized implementations (i.e., NFV) [10]. Lastly, legacy hardware-based MBs can support in-network policy deployment [2][6]. Nevertheless, we also show that our SDN-based scheme can be easily extended to support a NFV environment in Section III-F. We demonstrate that joint optimization of dynamic VM and policy migration can achieve significant network cost savings while still adhering to network policy requirements. By modelling the communication cost among MBs and applying stable matching theory in the allocation of VMs, we propose *Sync*, a Synergistic policY and virtual machiNe Consolidation scheme, which enables consolidation of both policies and VMs while reducing the network-wide communication cost. To the best of our knowledge, this is the first study on joint policy and VM migration optimization in DC environments.

The remainder of this paper is structured as follows. Section II describes the model of joint policy and VM consolidation (*PVC*), and defines the communication cost and utility for both VM and policy migration. An efficient *Sync* migration scheme is proposed in Section III. Section IV evaluates the performance of the proposed scheme demonstrating significant benefits in network-wide communication cost and flow delay reduction. Section V outlines related work, and Section VI concludes the paper.

II. Problem Modeling

A. Overview

We consider a multi-tier DC network which is typically structured under a multi-root tree topology such as canonical [17] or folded clos network [18][19].

Let $\mathbb{V} = \{v_1, v_2, \ldots\}$ be the set of VMs in the DC hosted by the set of servers $\mathbb{S} = \{s_1, s_2, \ldots\}$. The vector r_i denotes the physical resource requirements of VM v_i. For instance, r_i could have three components that capture three types of physical hardware resources, such as CPU cycles, memory size, and I/O operations [1]. Accordingly, the available amount of physical resource of server s_j is given by a vector h_j. Hence we use $\sum_{v_k \in A(s_j)} r_k + r_i \preceq h_j$ to denote that s_j has sufficient physical resource to accommodate v_i, where $\sum_{v_k \in A(s_j)} r_k$ is the total requirements of all VMs hosted by s_j.

Let $\mathbb{M} = \{m_1, m_2, \ldots\}$ be the set of all MBs in DC. Each MB m_i has several important properties $\{type, state, capacity\}$. The property $m_i.type$ defines the function of m_i, e.g., *IPS* (Intrusion Prevention System), *RE* (Redundancy Elimination), or *FW* (Firewall). MBs are usually stateful and need to process both directions of a session for correctness. $m_i.state$ is used to store the internal state and processing logic for m_i. The $m_i.capacity$ is essentially the throughput of m_i.

There are various deployment points for MBs in DC. They can be on the networking path or off the physical network [7]. Following the recent Cisco guidelines [3], we consider MBs are attached to aggregation switches for improved flexibility and scalability of policy deployment. These MBs may belong to different applications, deployed and configured by a *Policy Controller*. The centralized *Policy Controller* monitors the liveness of MBs, including addition or failure/removal of a MB. Network administrators can specify and update policies through the *Policy Controller*.

Traffic in DC is largely flow-based [19]. In light of this, we define DC traffic as $\mathbb{F} = \{f_1, f_2, \ldots\}$. For each flow $f_i \in \mathbb{F}$, the properties $f_i.src$ and $f_i.dst$ specify the source and destination VMs of f_i respectively, e.g., $f_i.src = v_1$ and $f_i.dst = v_2$. The data rate of $f_i.rate$ is represented by data exchanged from VM $f_i.src$ to VM $f_i.dst$ per time unit[2].

The set of policies is $\mathbb{P}$. In reality, one policy can be applied to multiple flows and vice versa. However, for ease of discussion, we assume that flows and policies are one to one correspondence. For each $f_i \in \mathbb{F}$, there is a policy p_i. $p_i.seq$ defines the sequence of MB types that all flows matching policy p_i should traverse in order, e.g., $p_i.seq = \{FW, IPS, Proxy\}$. $p_i.len$ denotes the size of the MB list. $p.list$ is the list of MBs that are assigned to p_i to fulfill the traversal requirement defined in $pi.seq$. Specially, $p_i = \emptyset$ means f_i is not governed by any policies.

We denote $p_i.in$ and $p_i.out$ to be the first (ingress) and last (egress) MBs, respectively, in $p_i.list$. Let $P(v_i, v_j)$ be the set of all policies defined for traffic from v_i to v_j, i.e., $P(v_i, v_j) = \{p_k | p_k \neq \emptyset, f_k.src = v_i, f_k.dst = v_j, \forall p_k \in \mathbb{P}\}$.

Policy p_i is called *satisfied*, if and only if all required MBs are allocated to p_i with the correct types and order:

$$m.type == p_i.seq[j], \quad \forall\ m = p_i.list[j], j = 1, \ldots, p_i.len \tag{1}$$

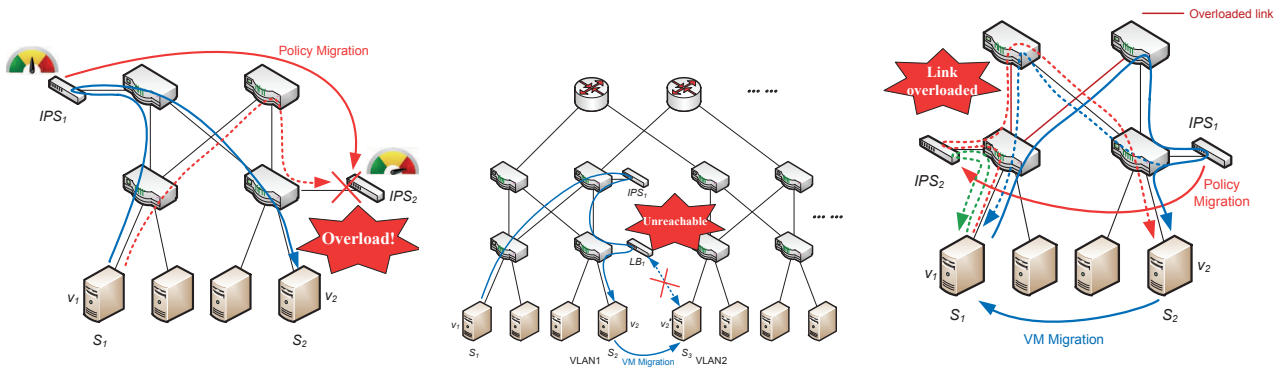
where $p_i.seq[j]$ is the jth type of MB that need to be traversed in $p_i.seq$.

B. Communication Cost with Policies

We denote $R(n_i, n_j)$ as the routing path between nodes (i.e., servers, MBs or switches) n_i and n_j. $L \in R(n_i, n_j)$ if

[1]In this paper, we assume that the size of a slice is a multiple of an atomic VM. For example, if the atomic VM has one 1 GHz CPU core, 512 MB memory and 10 GB storage, then a VM of size 2 means it effectively has a 2 GHz CPU core, 1 GB memory and 20 GB hard disk storage. Such atomic sizing is common among large-scale public clouds to reduce the overhead of managing hundreds of thousands of VMs, and is widely adopted in research literature [20] to reduce the dimensionality of the problem.

[2]There are a handful of research literature, e.g., [21], about deriving real time traffic matrices in DC networks.



(a) Scenario 1: MB capacity overloaded (b) Scenario 2: Route unreachable (c) Scenario 3: Suboptimal network utilization

Fig. 1: Examples of policy violations: Migrating policy or VMs separately make no improvement on efficiency. Solid blue: original flow path; dash blue: only migrate VM; dash red: only migrate policy; dash green: migrate both VM and policy.

link L is on the path. For a flow f_i, where $p_i \neq \emptyset$, its actual routing path is:

$$\begin{aligned} R_i(f_i.src, f_i.dst) = \; & R(f_i.src, p_i.in) \\ & + \sum_{j=1}^{p_i.len-1} R(p_i.list[j], p_i.list[j+1]) \\ & + R(p_i.out, f_i.dst) \end{aligned} \quad (2)$$

The cost of each link in DC networks varies on the particular layer that they interconnect. High-speed core router interfaces are much more expensive (and, hence, oversubscribed) than lower-level ToR switches [22]. Therefore, in order to accommodate a large number of VMs in the DC and at the same time keep providers' investment cost low from a providers perspective, utilization of the "lower cost" switch links is preferable to the "more expensive" router links. Let c_i denote the *link weight* for L_i. In order to reflect the increasing cost of high-density, high-speed (10 Gb/s) switches and links at the upper layers of the DC tree topologies, and their increased over-subscription ratio [18], we can assign a representative link weight ω_i for an ith-level link per data unit. Without loss of generality, in this case $\omega_1 < \omega_2 < \omega_3$.

Hence, we define the *Communication Cost* of all traffic from VM v_i to v_j as

$$\begin{aligned} C(v_i, v_j) &= \sum_{p_k \in P(v_i, v_j)} f_k.rate \sum_{L_s \in R_k(v_i, v_j)} c_s \\ &= \sum_{p_k \in P(v_i, v_j)} \{C_k(v_i, p_k.in) \\ &\quad + \sum_{j=1}^{p_k.len-1} C_k(p_k.list[j], p_k.list[j+1]) \\ &\quad + C_k(p_k.out, v_j)\} \end{aligned} \quad (3)$$

where $C_k(v_i, p_k.in) = f_k.rate \sum_{L_s \in R(v_i, p_k.in)} c_s$ is the communication cost between v_i and $p_k.in$ for flows which matched p_k. Similarly, $C_k(p_k.out, v_j)$ is the communication cost between $p_k.out$ and v_j for p_k, and $C_k(p_k.list[j], p_k.list[j+1])$ is the communication cost between $p_k.list[j]$ and its successor MB in $p_k.list$.

For a free flow f_i, which is not governed by any policies, i.e., $p_i = \emptyset$, its communication cost is calculated directly between the source and destination VMs. Unless otherwise stated, we only consider policy flows for ease of discussion in the rest of the paper.

C. Policy and VM Consolidation Problem

We denote A to be an allocation of both VMs and MBs. $A(v_i)$ is the server which hosts v_i, and $A(s_j)$ is the set of VMs hosted by s_j. $A(p_k)$ is the set of MBs which are allocated to policy p_k, i.e., $p_k.list$. $A(m_l)$ refers to all flow policies that use m_l as a node on its path.

The *Policy-VM Consolidation* (PVC) problem is defined as follows:

Definition 1. *Given the set of VMs $\mathbb{V}$, servers $\mathbb{S}$, policies $\mathbb{P}$ and MBs $\mathbb{M}$, we need to find an allocation A that minimizes the total communication cost:*

$$\begin{aligned} \min \; & \sum_{v_i \in \mathbb{V}} \sum_{v_j \in \mathbb{V}} C(v_i, v_j) \\ s.t. \; & A(v_i) \neq \emptyset \;\&\&\; |A(v_i)| = 1, \forall v_i \in \mathbb{V} \\ & p_k \text{ is satisfied}, \forall\, p_k \in \mathbb{P} \\ & \textstyle\sum_{v_i \in A(s_j)} r_i \preceq h_j, \forall s_j \in \mathbb{S} \\ & \textstyle\sum_{p_k \in A(m_i)} f_k.rate \leq m_i.capacity, \forall m_i \in \mathbb{M} \end{aligned} \quad (4)$$

The first constraint ensures that each VM is hosted by one server. The second constraint is to fulfill all policy requirements on MBs traversal. The third and fourth constraints are the capacity requirements for both servers and MBs.

Theorem 1. *The* PVC *problem is NP-Hard.*

Proof: To show that *PVC* problem is NP-Hard, we will show that the Multiple Knapsack Problem (MKP) [23], whose decision version has already been proven to be strongly NP-complete, can be reduced to this problem in polynomial time.

Consider a special case of the *PVC* problem: there are only two servers, i.e., s_1 and s_2, connecting to two different edge switches. Their capacity are all equal to n, i.e., $h_1 = h_2 = n$. All $2n$ VMs are divided into two groups, each one has n VMs.

The resource requirement of each VM is 1, i.e., $r_i = 1, \forall i = 1 \dots 2n$. There are n flows, and each flow is from a distinct VM of group 1 to another distinct VM of group 2. All flows are policy flows and need to traverse three MBs in sequence, e.g., *LB*, *RE* and *IPS*. There are only one *LB* box, one *IPS* box and multiple *RE* boxes. The *LB* and *IPS* boxes are attached to the edge switches connected to s_1 and s_2 respectively. Suppose the capacity of *LB* and *IPS* are enough to accept all flows. Thus, a reasonable solution is to migrate all VMs of group 1 to s_1 and all VMs of group 2 to s_2. Then, the *PVC* problem becomes to find an appropriate *RE* box for each flow.

Consider each flow f_i to be an item, where $f_i.rate$ is item size. Each *RE* box is a knapsack with limited capacity. The profit of assigning f_i to each *RE* box is the negative of the communication cost defined in Equation (3). The *PVC* problem becomes finding an allocation of all flows to *RE* boxes, maximizing the total profit. Therefore, the MKP problem is reducible to the *PVC* problem in polynomial time, and hence the *PVC* problem is NP-hard. ■

III. Synergistic Policy and VM Consolidation

In this section, we introduce *Sync*, a Synergistic policY and virtual machiNe Consolidation scheme.

A. VM Migration and Cost

Considering a migration for VM v_i from its current allocated server $A(v_i)$ to another server $\hat{s}$: $A(v_i) \rightarrow \hat{s}$, the feasible space of candidate servers for v_i is characterized by:

$$\mathcal{S}(v_i) = \{\hat{s} | (\sum_{v_k \in A(\hat{s})} r_k + r_i) \preceq \hat{h}, \forall \hat{s} \in \mathbb{S} \setminus A(v_i)\} \quad (5)$$

Considering that v_i is hosted on s_j, i.e., $A(v_i) = s_j$, let $\mathcal{C}_i(s_j)$ be the total communication cost induced by v_i between s_j and all ingress & egress MBs associated with v_i:

$$\mathcal{C}_i(s_j) = \sum_{p_k \in P(v_i, *)} C_k(v_i, p_k.in) + \sum_{p_k \in P(*, v_i)} C_k(v_i, p_k.out) \quad (6)$$

Migrating a VM also generates network traffic between the source and destination hosts. The amount of traffic depends on the memory size of the VM, its page dirty rate, the available bandwidth for the migration and some other hypervisor-specific constants [24]. We use the model for estimating migration cost defined in [24]:

$$C_m(v_i) = M \cdot \frac{1 - (R/L)^{n+1}}{1 - (R/L)} \quad (7)$$

where $n = \min(\lceil \log_{R/L} \frac{T \cdot L}{M} \rceil, \lceil \log_{R/L} \frac{X \cdot R}{M \cdot (L-R)} \rceil)$ is the number of pre-copy cycles, M is the memory size of v_i, R is the page dirty rate, and L is the bandwidth used for migration. X and T are user settings for the minimum required progress for each pre-copy cycle and the maximum time for the final stop-copy cycle, respectively [24].

Such migration cost should not outweigh the reduction in the overall communication cost. We then define the *utility* of migration $A(v_i) \rightarrow \hat{s}$ to be the expected benefit through migration:

$$U(A(v_i) \rightarrow \hat{s}) = \mathcal{C}_i(A(v_i)) - \mathcal{C}_i(\hat{s}) - C_m(v_i) \quad (8)$$

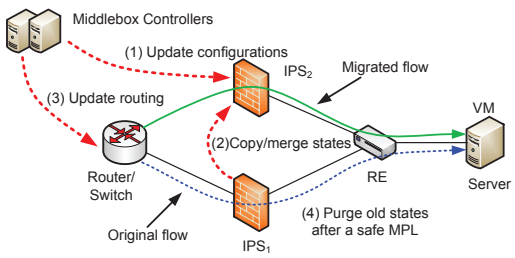


Fig. 2: Policy migration among MBs

Specifically, $U(A(v_i) \rightarrow \hat{s}) = 0$ if no migration takes place.

B. Policy Migration and Cost

When performing policy migration, to preserve the correctness and fidelity of active flows, the destination MB must receive the internal MB state associated with the migrated flows, while the old MB still keeps the internal state associated with remaining flows. Clearly, the MB states must be able to be cloned, shared, moved and merged. To support this, we adopt the architecture of *OpenNF* [12], which is a control plane with carefully designed APIs for managing MBs and policies.

Fig. 2 presents an example of migrating a policy flow from IPS_1 to IPS_2. The internal logic state of IPS_1 will be first migrated to IPS_2, e.g., step (2). Next, the network configuration will be updated to forward all new traffic to IPS_2, e.g., step (3). Some clean-up work may also required to maintain the consistence after migration, e.g., step (4). Denote σ_{ik} to be the total traffic induced to transmit the internal states size of m_i for policy p_k.

Let $(p_k, i) \rightarrow \hat{m}$ denote migrating the ith MB of p_k, i.e., $m' = p_k.list[i]$, to a new MB of $\hat{m}$. The feasible space of candidate MBs for $\hat{m}$ is characterized by:

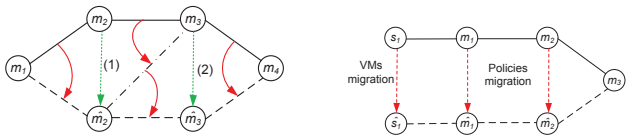
$$M(p_k, i) = \{\hat{m} | \hat{m}.type == m'.type, \sum_{p_j \in A(\hat{m})} f_j.rate \leq \hat{m}.capacity - f_k.rate, \forall \hat{m} \in \mathbb{M} \setminus m'\} \quad (9)$$

We assume that the MBs assignment for policy p_k is an *atomic* operation, i.e., either all required MBs are assigned for p_k or none is assigned. In the following, we suppose p_k is assigned, and consider policy migration on *intermediate* MBs (i.e., $p_k.list[i], \forall i = 2, \dots, p_k.len - 1$) and *end* MBs (i.e., $p_k.in$ and $p_k.out$) of $p_k.list$, respectively.

1) Migration on Intermediate MBs Associated Policies: We start from the simplest case that performing migration of p_k on only one attached MB.

Define the *utility* of migration $(p_k, i) \rightarrow \hat{m}$ as the communication cost reduction gained substract the cost induced by the policy migration:

$$U((p_k, i) \rightarrow \hat{m}) = C_k(p_k.list[i-1], m') + C_k(m', p_k.list[i+1]) - C_k(p_k.list[i-1], \hat{m}) - C_k(\hat{m}, p_k.list[i+1]) - \sigma_{ik} \quad (10)$$



(a) (b)

Fig. 3: (a) Multiple MBs migration on a policy path equals to migrating them one by one: old path $m_1 \rightarrow m_2 \rightarrow m_3 \rightarrow m_4$, new path $m_1 \rightarrow \hat{m_2} \rightarrow \hat{m_3} \rightarrow m_4$. We can first migrate m_2 to $\hat{m_2}$, then migrate m_3 to $\hat{m_3}$, and vice versa. (b) Decomposable migration among VMs and MBs

If migration of p_k involves two or more attached MBs along the policy path, we can migrate p_k on these MBs one by one. *Utilities* of different migration orders can be easily proved to be equal through Fig. 3a.

2) Migration on End MBs Associated Policies: End MBs for p_k involves the migration of either $p_k.in$ ($p_k.in \rightarrow \hat{m}$) or $p_k.out$ ($p_k.out \rightarrow \hat{m}$). The key difference of *utility* between policy migration on end MBs and intermediate MBs is that VMs are included in calculating the cost, as end MBs of policies communicate directly with src/dst VMs. Hence, the *utility* for migration of $p_k.in \rightarrow \hat{m}$ is:

$$\begin{aligned} U(p_k.in \rightarrow \hat{m}) = & C_k(f_k.src, p_k.in) + C_k(p_k.in, p_k.list[2]) \\ & - C_k(f_k.src, \hat{m}) - C_k(\hat{m}, p_k.list[2]) \\ & - \sigma_{ik} \end{aligned} \tag{11}$$

The *utility* for migration of $p_k.out \rightarrow \hat{m}$ is similar to Equation (11).

C. Decomposable Migration

Based on above analysis, we can easily observe that, for a single flow, the migration of VMs and policies are all *decomposable*. It means the migration order on src/dst VMs and each MB are independent and the final utilities of all migrations remain the same.

Without loss of generality, we use a flow for example in Fig. 3b. We consider migration on portion of a flow, i.e., migration of a source VM which is hosted on s_1, and policy migration on the ingress MB m_1 and one intermediate MB m_2. According to the *utility* definitions of both VM and policy migrations in Equations (8), (10) and (11), we can easily show that:

$$\begin{aligned} & U((s_1, m_1, m_2) \rightarrow (\hat{s_1}, \hat{m_1}, \hat{m_2})) = \\ & \quad U(s_1 \rightarrow \hat{s_1}) + U(m_1 \rightarrow \hat{m_1}) + U(m_2 \rightarrow \hat{m_2}) \end{aligned} \tag{12}$$

Equation (12) describes an important property of VM and Policy migration: we can treat all MBs and VMs at both endpoints of the flow independently during migration.

D. Communicating VMs Groups

Operating our *Sync* scheme on all VMs and policies in a DC would be computationally impractical and would introduce intolerable delays. In this section, we consider simplifying the problem by dividing VMs into isolated groups according to their pairwise communication patterns.

Algorithm 1 *GetNextCommunicatingVMsGroup()*

Input: $V_r, \mathbb{F}$
Output: Next communicating VMs group G, if any

$G = \emptyset$
if $V_r \neq \emptyset$ and $\exists v_i, v_i \in V_r$ **then**
  $G = \{v_i\}$
  $F = \{$flows related to $v_i\}$
  while $F \neq \emptyset$ and $\exists f_j, f_j \in F$ **do**
    if $f_j.src \notin G$ **then**
      $v' = f_j.src$
    else if $f_j.dst \notin G$ **then**
      $v' = f_j.dst$
    else
      continue;
    end if
    $G = G \cup \{v'\}$
    $F = F \cup \{$flows related to $v'\}$
    $F = F \setminus \{f_j\}$
  end while
end if
$V_r = V_r \setminus G$
Output a communicating VMs group G

A *Communicating VMs Group* G is defined as a set of VMs where every VM is communicating with at least one other VM in the group and none of them is communicating with VMs outside the group, i.e.,

$$\begin{aligned} & \forall v_i \in G, \quad \exists f_j, v_i \in \{f_j.src, f_j.dst\} \\ & \text{And} \\ & \nexists f_j, \qquad f_j.src \in G \wedge f_j.dst \notin G \quad \text{or} \\ & \qquad\qquad f_j.src \notin G \wedge f_j.dst \in G \end{aligned} \tag{13}$$

Because we consider a multi-tenant DC environment, such isolated groups always exist. In the worst case, all VMs of a tenant belong to a single group. Those groups can be easily found by either depth first search (DFS) or breadth first search (BFS) operating on the active flows at the DC. Algorithm 1 shows an example using BFS. The set of VMs V_r refers to all VMs that remain to be processed. Initially, V_r can be all VMs. Any dynamics, e.g., VM or traffic changes, will cause related VMs to be added to V_r.

E. Sync Migration Algorithms

In the following, we propose a *Sync* scheme utilizing the property of *decomposability*. When $V_r \neq \emptyset$, a communicating VM group G will be obtained through Algorithm 1 and the *Sync* migration algorithms will be triggered. The whole scheme is comprised of two phases - migration of policies and VMs, respectively - to reduce the total communication cost.

1) Phase I: Policy Migration: We have shown that migration of policies among MBs for a single flow is decomposable. Nevertheless, a VM usually has multiple concurrent active flows, making it difficult to determine optimal VM migration. Therefore, in Phase I, we only migrate policies, while preparing for the migrations of VMs by building the preference matrix for servers.

For a flow f_i which needs to traverse $n = p_i.len$ MBs, we define its *Cost Network*, which is a $(n+2)$-tier directed graph.

Algorithm 2 Phase I: Policy Migration

Input: A communicating VMs group $G, \mathbb{S}, \mathbb{F}, \mathbb{P}, \mathbb{M}$
Output: New allocation of MBs for policies

```
1: F ← related flows of group G
2: for each f_k ∈ F do
3:     p_k =the policy applied on f_k
4:     Construct the Cost Network N
5:     (s_src, s_dst, mlist) = SPF(N)
6:     for i = 1 to mlist.len do
7:         if p_k.list[i] ≠ mlist[i] then
8:             Perform policy migration: p_k.list[i] → mlist[i]
9:         end if
10:    end for
11:    Update routing for policies
12:    ρ(s_src, f_k.src) += 1        ▷ Update preference matrix
13:    ρ(s_dst, f_k.dst) += 1
14: end for
```

Flows originate from the *source* ($f_i.src$) and terminate at the *sink* ($f_i.dst$). The first (or the last) tier includes all possible servers that can accept the $f_i.src$ (or $f_i.dst$) VM for migration according to Equation (5), as well as its current host server. Similarly, the middle n tiers are all possible MBs defined in Equation (9). The weight of each edge is initialized as the corresponding communication cost between two connected nodes, plus migration cost. More specifically, the weight of edges connected to *source/sink* are the migration cost of source/destination VMs. Fig. 4 shows an example of Cost Network for flow f_k, which needs to traverse $\{IPS, RE\}$. v_0 is currently hosted on s_i, so the weight from *source* to s_i is 0, and weight to s_j is migration cost $C_m(v_0)$.

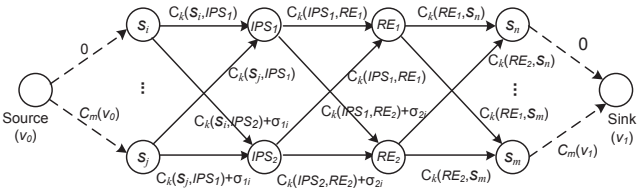


Fig. 4: Cost Network Example: Suppose the current path for flow f_k from v_0 to v_1 is $s_i \rightarrow IPS_1 \rightarrow RE_1 \rightarrow s_n$.

Clearly, the route with the largest utility for a flow is the shortest path from *source* to *sink*. Thus, we propose the *Policy Migration Algorithm*, shown in Algorithm 2, to minimize the total communication cost through the migration of policies. The function call $(s_{src}, s_{dst}, mlist)$ = $SPF(N)$ returns the shortest path for f_k, where s_{src} is the source server, s_{dst} is the destination server and $mlist$ is the list of MBs, e.g., $(s_1, s_2, \{m_1, \ldots, m_2\})$.

A $S \times V$ preference matrix ρ is maintained to help future VM migration, where S and V are the number of input servers and VMs. $\rho(s, v)$ is the score of VM v, which is given by server s. $\rho(s, v)$ is initialized to be zero, and will be increased each time if a flow f_k, either to or from v, chooses s for its shortest path in Algorithm 2.

2) Phase II: VM Migration: According to Equation (8), migrating v_i to a different server will yield different *utility*, which means v_i can rank order candidate servers for migration. In the mean time, during policy migration, each server also has presented their preference to all VMs through the preference matrix $\rho(s, v)$. Those preferences can be presented in a ranked order list. For example, denote $l_i = \{v_1, v_2, \ldots\}$ to be the preference list of s_i over all possible VMs, and $v_2 \prec_{l_i} v_1$ means s_i prefer v_1 to v_2. Moreover, each VM also has resource requirements when it is assigned to a server whose availability of resources is limited. The preferences of servers and VMs might be inconsistent. This makes it difficult to determine migration destination of each VMs.

To overcome this, we model the VM migration problem above to be a typical *many-to-one stable matching*, hence Stable Marriage [25]. The key concept of stable matching is *stability*. Before explaining the stability, we will first define the *blocking pair*.

Definition 2. *For an allocation A, a VM-server pair (v_i, s_j) is a blocking pair if $A(v_i) \neq s_j$ and* [3]

$$\sum\nolimits_{v_k \in A(s_j) \ \&\& \ v_i \prec_{l_j} v_k} r_k + r_i \preceq h_j \tag{14}$$

where l_j is the preference list of s_j over all VMs, $v_i \prec_{l_j} v_k$ means s_j prioritize v_k over v_i according l_i.

A *stable matching* means no *blocking pair* exists in the final matching of VMs to servers. Thus, an *unstable* matching between VMs and servers will always leave room to minimizing the total communication cost, while a *stable* matching is the optimal assignment for both VMs and servers.

We then apply our modified *Gale-Shapley algorithm* [25] to address the conflict of preferences and efficiently output stable matching between VMs and servers, shown in Algorithm 3. Initially, all VMs are unmatched. For such a VM, say v_i, $\hat{A}(v_i) = \emptyset$. v_i will be first matched to its most preferred server in $\mathcal{S}(v_i)$, say server s_j, which has not yet rejected v_i and can gain the largest utility for v_i (line 5∼6). If s_j has sufficient capacity in the matching $\hat{A}$, it accepts v_i. Otherwise, it sequentially rejects less preferable VMs, which were allocated to s_j previously (line 9∼10). Whenever s_j rejects a VM, it updates the *best_rejected* variable (line 11), which indicates the most preferred VM that rejected by s_j currently. In the end, all VMs ranked lower than *best_rejected* will remove s_j from their preference list by adding s_j to their blacklists (line 13∼15).

Theorem 2. *Algorithm 3 can always output a stable matching in $O(VS)$, where V and S are the number of input VMs and servers respectively.*

Proof: We can prove the stability of the output matching by contradiction. Suppose that the Algorithm 3 produces a matching $\hat{A}$ with a blocking pair (v_i, s_j), i.e., there is at least one VM $v' \in \hat{A}(s_j)$ worse than v_i to s_j. v_i must have proposed to s_j and been rejected by s_j. v' should have either been

[3]Considering the complexity and existence of a stable matching, we only consider one type of blocking pair. For more information about the blocking pair and stability, please refer to [20]

Algorithm 3 Phase II: VM Migration

Input: A Communicating VMs Group $g, \mathbb{S}, \rho$
Output: New allocation of servers-VMs $\hat{A}$

1: Obtain preference list l_i according to ρ, $\forall s_i \in \mathbb{S}$
2: Initialize blacklist $b_j = \emptyset$, $\forall v_j \in g$
3: $\hat{A} = \emptyset$
4: **while** $\exists v_i$, and $\hat{A}(v_i) = \emptyset$ **do**
5: $s_j \leftarrow \arg\max_{s \in \mathcal{S}(v_i) \setminus b_i} U(A(v_i) \rightarrow s)$ ▷ $U > 0$
6: $\hat{A}(v_i) = s_j$
7: **if** $\sum_{v_k \in \hat{A}(s_j)} r_k \succ h_j$ **then**
8: **repeat**
9: $v_k \leftarrow$ last VM in $\hat{A}(s_j)$ according to l_j
10: $\hat{A}(v_k) = \emptyset$ ▷ s_j rejects v_k
11: *best_rejected*$\leftarrow v_k$
12: **until** $\sum_{v_k \in \hat{A}(s_j)} r_k \preceq h_j$
13: **for each** $v_k \in l_j$, $v_k \preceq_{l_j}$ *best_rejected* **do**
14: $b_k = b_k \cup s_j$ ▷ Add s_j to the blacklist of v_k
15: **end for**
16: **end if**
17: **end while**
18: Perform VMs migration: $A \rightarrow \hat{A}$
19: Update routing for VMs

rejected by s_j before, or s_j should have been added to its blacklist when v_i was rejected (in line 14). Thus, $v' \notin \hat{A}(s_j)$, which contradicts the assumption.

In the worst case, each VM is rejected by every server. So, Algorithm 3 will always be terminated and output a stable matching within $O(VS)$. ■

F. Extension to NFV

A NFV infrastructure enables the execution of traditional, hardware-based middleboxes as software-based virtual network functions, typically encapsulated in VMs [9] and hosted on the same hypervisors as traditional VMs. While placing the network functions to the lowest layer of the network infrastructure (hypervisors) gives more flexibility for their dynamic placement, it also makes network-wide resource management crucial to avoid sub-optimal network utilization. In order to apply our *Sync* algorithms to NFV infrastructures, the migration cost of the policies and the calculation of the cost networks (Section III-B) should reflect the cost of migrating the VMs hosting the network functions instead of transferring only MB states between hardware-based MBs. *We note that this is the only difference between applying Sync to legacy hardware and NFV software MBs.* The communication cost (Section II-B) and migration cost of the VMs (Section III-A) can be obtained in the same way as with hardware-based MBs.

Since SDN aggregates the network-wide control logic into a logically centralized software component, it can enable policies, configuration, and network resource management to be programmed in a convenient and simplified way. When a new flow arrives to the network, our SDN-based *Policy Controller* checks for matching policies, allocates a path to ensure the network function traversal requirements are met, and installs the corresponding configuration information to flow tables of all switches/routers along the path. Similar to the situation in the network of legacy hardware MBs, the initially allocated path might not be the most efficient (e.g., a VM uses a network function hosted on a distant server), we ensure that our controller collects flow statistics and runs our *Sync* scheme periodically to mitigate inefficiencies.

IV. Evaluation

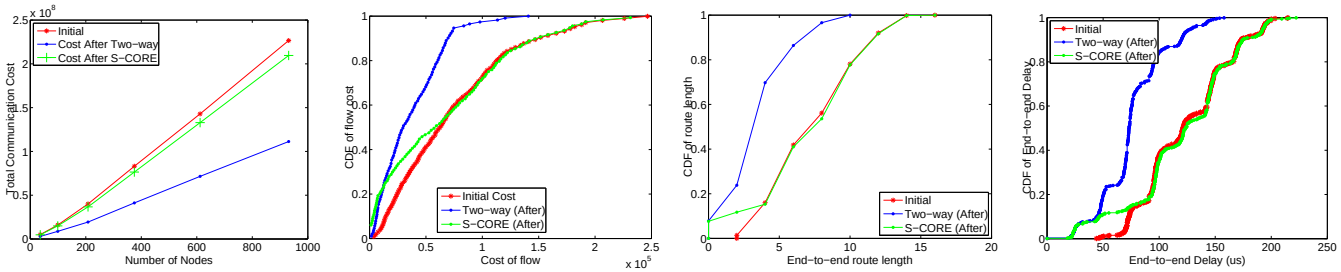
In this section, we present the performance of our *Sync* scheme in both simulation and testbed environment.

A. Simulation Setup and Results

We have evaluated the performance of *Sync* scheme over a simulated fat-tree DC topology with $k = 14$ (i.e., 931 nodes, including 686 servers and 245 switches) in *ns-3*. VMs are modeled as a collection of socket applications communicating with one or more other VMs in the DC network. We define *policy flows* as traffic flows that have to traverse a sequence of MBs as specified in their governing policies, and *policy-free flows* that are not subject to any network policies. In all experiments, all traffic flows are randomly generated during initialization and are composed of 20% *policy-free flows* and 80% *policy flows*. Each *policy flow* is configured to traverse 1∼3 MBs, such as, a firewall, IPS and/or LB. As a result, the average path length for each flow is 8.3 hops. A centralized controller is implemented to collect all network information and perform the *Sync* scheme. In order to compare against policy-agnostic VM management, we have also implemented S-CORE [14], which is a network-aware dynamic VM migration scheme that reduces network-wide communication cost through localizing heavy-bandwidth communicating VM pairs.

Fig. 5 shows the performance of *Sync* with a focus on two key factors of DC networks: communication cost and end-to-end delay between hosts. We can easily observe from Fig. 5a that *Sync* significantly reduces network-wide communication cost by about 50% for all topology scales, i.e. $k = 4$ to 14. Specifically, Fig. 5b shows the CDF of communication cost for each flow when $k = 14$. In comparison, *S-CORE* only reduces the communication cost by 7.43% due to its intrinsic policy-agnostic nature. This is evidenced by Fig. 5c, which illustrates the CDF of the length of flow paths after running *Sync* and *S-CORE*, respectively. Fig. 5c shows that *Sync* significantly reduces the average flow path length from 8.3 hops to 4.3 hops, a nearly 48% improvement, as opposed to merely 3.2% of that of *S-CORE* which fails to consider that traffic flows of migrated VMs might have to traverse even longer paths if policies are not migrated accordingly. As a result of shorter flow path length, Fig. 5d demonstrates that *Sync* can reduce the average end-to-end delay from 125 *us* to 76 *us* (38.8% improvement), compared to 121 *us* for *S-CORE* (2.6% improvement). We note that being able to reduce end-to-end delay is an important feature since it implies that *Sync* can potentially improve flow completion time to a similar extent.

Fig. 6 reveals more details of the performance of *Sync* on policy and VM migration. Fig. 6a shows that the number of VM and policy migrations for the *Sync* increases linearly with the number of active VMs. More interestingly, if we contrast this with that of *S-CORE*, which also increases linearly, it is obvious that *Sync* only migrates half the number of VMs required by *S-CORE*. This is crucial since the network overhead for migrating VMs is a lot more expensive than migrating



(a) Total Cost (b) Cost of flows for $k = 14$ (c) Route length of flows (d) End-to-end Delay

Fig. 5: *Sync* Performance

network policies. Fig. 6b shows the total utility gained for each policy (Equation 10 and 11) and VM migration (Equation 8). More interestingly, we can see from Fig. 6b that the utility gained through policy migrations is almost 3 times higher than migrating VMs, contributing 72.87% on average for the overall utility. In comparison, while *S-CORE*'s utility on VM migration as defined by Equation (8) is comparable to its counterpart in *Sync*, its overall utility is only a small fraction, 14.6%, of that of the *Sync* scheme.

Next, we explore the cases of policy violation as depicted in Fig. 1. Intuitively, we can observe from Fig. 6c that *Sync* scheme, by considering only feasible servers and MBs in constructing the Cost Network for both VM and policy migrations, can successfully avoid policy violations. On the contrary, *S-CORE* violates 8.75% of policies on average as a result of its policy-agnostic VM migration decisions. Since one policy is often implemented by multiple MBs, we have recorded 5.7% of all MBs being part of the violations. Albeit small fraction, this actually translates to enormous number, i.e. hundreds, of MBs. Apart from potential security vulnerabilities, this also implies that it could be very difficult to pinpoint the problems manually when policy violations happen.

B. Testbed Results on Controller Performance

We have implemented the central controller for the proposed system on top of the Ryu SDN controller, running on an CentOS 6 host with Intel 2.1GHz CPU and 4GB of memory. The controller is responsible for the collection of flow statistics from the network, running the *Sync* migration algorithms and initiating the policy and VM migrations. Flow statistics are collected from all software switches (Open vSwitch 2.3.1) operating at the hypervisors to be able to account for all VM communication [26]. We considered two ways of collecting flow statistics. One could periodically pull OpenFlow flow statistics from all hosts to retrieve fine-grained statistics. According to our measurements, this is a reasonable solution with a single controller for mid-sized infrastructures, giving around 5.0s to retrieve flow statistics from 631 hosts each hosting 20 VMs, as shown in Fig. 7a. For larger infrastructures ($>$ 1000 hosts), we refer to [26] and suggest using multiple SDN controllers to collect flows or sampling them by using sFlow or DevoFlow [27].

The running time of the *Sync* migration algorithm has also been evaluated on the controller. The number of VMs within a communicating group, as an important input parameter in the algorithms, is scaled from 100 to 1000 VMs. The times consumed for *GetNextCommunicatingVMsGroup()* in Algorithm 1, Phase I for policy migration in Algorithm 2, and Phase II for VM migration in Algorithm 3 are shown in Fig. 7b. As shown in the figure, *GetNextCommunicatingVMsGroup()* is very efficient and can be completed within 0.06s even with 1000 communicating VMs. The running time of Phase I for policy migration is similar to Phase II when the number of communicating VMs is small. However, Phase I outperforms Phase II after the number of VMs increased above 500. With 1000 VMs, Phase I takes 1.5s, while Phase II takes 2.4s, and the total running time for *Sync* is around 4.0s. Combining the collection of flow statistic (5s) and *Sync* migration algorithms (4s), our control loop takes only around 9s to initiate VM and policy migration operations on a fat-tree DC topology with $k = 14$ (i.e., 931 nodes, including 686 hosts and 245 switches).

V. Related Work

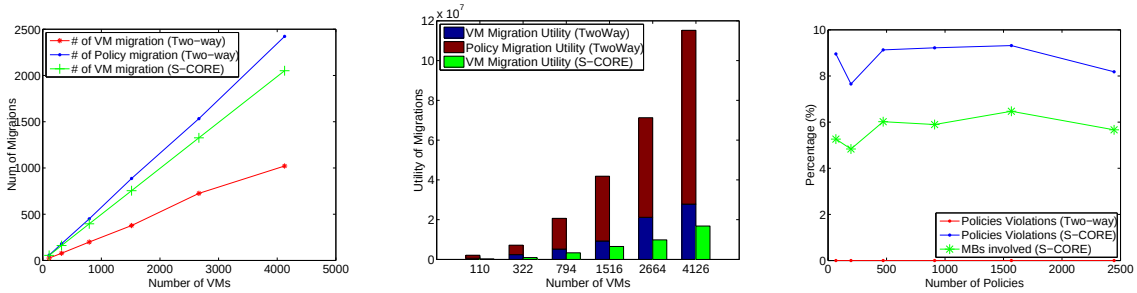
Recent developments in SDN enable more flexible MB deployment over the network while still ensuring that specific subsets of traffic traverse the desired set of MBs [8][10][12][13]. Zafar et al. [8] proposed *SIMPLE*, a SDN-based policy enforcement scheme to steer DC traffic in accordance to policy requirements. Similarly, Fayazbakhsh et al. presented FlowTags [13] to leverage SDN's global network visibility and guarantee correctness of policy enforcement. However, these proposals are not fully designed with VMs migration in consideration, and may put migrated VMs on the risk of policy violation and performance degradation.

Multi-tenant Cloud DC environments require more dynamic application deployment and management as demands ebb and flow over time. As a result, there is considerable literature on VM placement, consolidation and migration for server, network, and power resource optimization [14][28][29]. However, none of these research efforts consider network policy in their design.

A similar work to ours is Policy-Aware Application Cloud Embedding (PACE) [6] which is a framework to support application-wide and in-network policies VM placement. However, PACE only considers one-off VM placement, and hence fails to deal with and further improve resource utilization in the face of dynamic workloads. A recent work, *PLAN*, has been proposed to provide a joint policy and network-aware virtual machine migration scheme [30], but it does not migrate network policies, limiting the the scope for substantial performance improvement.

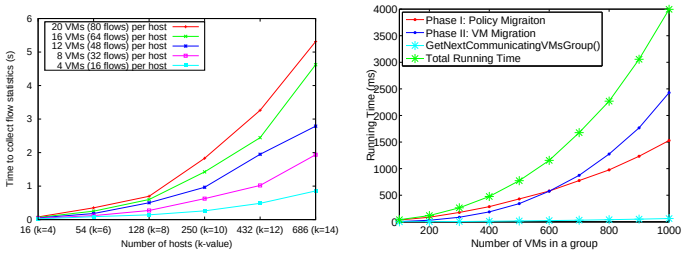
VI. Conclusion

Network policies and virtual machines are at the heart of DC network design today. In this paper, we have studied



(a) Number of Migrations (b) VM & Policy Migration Utility (c) Policy violations and MBs involved

Fig. 6: Policy and VM Migrations



(a) Flow collection time (b) *Sync* algorithms running time

Fig. 7: Controller Performance

the network communication cost reduction in DC topologies by jointly considering virtual machine and network policy dynamic (re)allocation. We first proved that this jointly optimization problem is NP-Hard, and then proposed a *Sync* migration scheme to minimize the communication cost by performing policy and VM migration in two phases. Extensive results have shown that *Sync* significantly reduces the total communication cost in DC by 50% within few seconds and effectively improve end-to-end delay by 38.8%, while strictly satisfying requirements of network policies.

REFERENCES

[1] Z. Liu, X. Wang, W. Pan, B. Yang, X. Hu, and J. Li, "Towards efficient load distribution in big data cloud," in *IEEE ICNC*, 2015, pp. 117–122.

[2] L. Avramov and M. Portolani, *The Policy Driven Data Center with ACI: Architecture, Concepts, and Methodology*. Cisco Press, 2014.

[3] Cisco virtualized multi-tenant data center, version 2.0 compact pod design guide. [Online]. Available: http://hyperurl.co/hpj2xt

[4] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: Network processing as a cloud service," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 13–24, 2012.

[5] World enterprise network and data security markets. [Online]. Available: http://hyperurl.co/hgr9qz

[6] L. E. Li, V. Liaghat, H. Zhao, M. Hajiaghayi, D. Li, G. Wilfong, Y. R. Yang, and C. Guo, "PACE: Policy-aware application cloud embedding," in *Proceedings of 32nd IEEE INFOCOM*, 2013.

[7] D. A. Joseph, A. Tavakoli, and I. Stoica, "A policy-aware switching layer for data centers," in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4. ACM, 2008, pp. 51–62.

[8] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "Simplefying middlebox policy enforcement using SDN," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 27–38, 2013.

[9] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "Clickos and the art of network function virtualization," in *Proc. USENIX NSDI*, 2014, pp. 459–473.

[10] A. Gember, P. Prabhu, Z. Ghadiyali, and A. Akella, "Toward software-defined middlebox networking," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*. ACM, 2012, pp. 7–12.

[11] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi, "Design and implementation of a consolidated middlebox architecture." in *NSDI*, 2012, pp. 323–336.

[12] A. Gember-Jacobson, C. P. Raajay Viswanathan, R. Grandl, J. Khalid, S. Das, and A. Akella, "OpenNF: enabling innovation in network function control," in *Proc. of ACM SIGCOMM*, 2014, pp. 163–174.

[13] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, "Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags," in *Proc. USENIX NSDI*, 2014.

[14] F. P. Tso, K. Oikonomou, E. Kavvadia, and D. Pezaros, "Scalable traffic-aware virtual machine management for cloud data centers," in *IEEE ICDCS*, 2014.

[15] A. Verma, P. Ahuja, and A. Neogi, "pMapper: power and migration cost aware application placement in virtualized systems," in *Proc. ACM/IFIP/USENIX Int. Conf. on Middleware*, 2008, pp. 243–264.

[16] D. Breitgand and A. Epstein, "SLA-aware placement of multi-virtual machine elastic services in compute clouds," in *IFIP/IEEE Int. Symp. on Integrated Network Management (IM'11)*, May 2011, pp. 161–168.

[17] Cisco, "Data center: Load balancing data center services," 2004.

[18] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4. ACM, 2008, pp. 63–74.

[19] 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 computer communication review*, vol. 39, no. 4. ACM, 2009, pp. 51–62.

[20] H. Xu and B. Li, "Anchor: A versatile and efficient framework for resource management in the cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1066–1076, 2013.

[21] Z. Hu, Y. Qiao, and J. Luo, "Coarse-grained traffic matrix estimation for data center networks," *Computer Communications*, vol. 56, pp. 25–34, 2015.

[22] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," *ACM SIGCOMM computer communication review*, vol. 39, no. 1, pp. 68–73, 2008.

[23] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack problems*. Springer Verlag, 2004.

[24] V. Mann, A. Gupta, P. Dutta, A. Vishnoi, P. Bhattacharya, R. Poddar, and A. Iyer, "Remedy: Network-aware steady state vm management for data centers," in *NETWORKING 2012*. Springer, 2012, pp. 190–204.

[25] D. Gale and L. S. Shapley, "College admissions and the stability of marriage," *American mathematical monthly*, pp. 9–15, 1962.

[26] V. Mann, A. Vishnoi, and S. Bidkar, "Living on the edge: Monitoring network flows at the edge in cloud data centers," in *5th International Conference on Communication Systems and Networks (COMSNETS)*. IEEE, 2013, pp. 1–9.

[27] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: Scaling flow management for high-performance networks," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 254–265.

[28] J. W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, "Joint VM placement and routing for data center traffic engineering," in *2012 Proceedings IEEE INFOCOM*. IEEE, 2012, pp. 2876–2880.

[29] A. Song, W. Fan, W. Wang, J. Luo, and Y. Mo, "Multi-objective virtual machine selection for migrating in virtualized data centers," in *Pervasive Computing and the Networked World*. Springer, 2013, pp. 426–438.

[30] L. Cui, F. P. Tso, D. P. Pezaros, W. Jia, and W. Zhao, "Policy-aware virtual machine management in data center networks," in *IEEE ICDCS*, 2015.