Joint Virtual Machine and Network Policy Consolidation in Cloud Data Centers

Lin Cui*, Fung Po Tso†
*Department of Computer Science, Jinan University, Guangzhou, China
†School of Computing & Mathematical Science, Liverpool John Moores University, UK
Email: tcuilin@jnu.edu.cn; p.tso@ljmu.ac.uk;

***Abstract*—Correct implementation of network policies provides secure and high performance network environment for multi-tenant Cloud Data Center (DC), but at the same time, requires substantial effort for manual configuration. While it is reported that 78% of DC downtime is caused by misconfiguration [1], the dynamic policy configuration and user demands on server consolidation have amplified the challenge of correct implementation. In this paper, we propose joint virtual machine and network policy migrations - *Two-way migration* - to facilitate the simultaneous need of dynamic server consolidation and policy management. Extensive simulation analysis shows that the proposed *Two-way migration* can mitigate link utilization at the core and aggregation layers by 90% and 28% respectively while adhering strictly to the network policies.**

I. Introduction

As cloud computing has seen proliferated adoption in recently years, data centres (DCs), the underpinning infrastructures, are challenged with increased complexity of 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 [2]. In order to implement desired network policies, network operators typically deploy a diverse range of "middleboxes" (MBs), such as firewalls, load balancers, Intrusion Detection and Prevention Systems (IDS/IPS), and application enhancement boxes, the number of which is on par with the number of routers in a network [3][4].

Network policies demand traffic to traverse a sequence of specified MBs. Consequently, emerging evidence demonstrating that up to 78% of DC downtime is attributed to misconfiguration of network policies and paths [1][3]. 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 DC. Existing research treat dynamic VM and network policy consolidation as disjoint research problems [1][5][6]. Fig. 1 shows an example that traffic between two VMs need to be checked by an IPS. If we only consider VM migration, the second VM might be migrated to s_1 or s_2, or if only policy migration considered, the traffic can be watched by another IPS. In either case, there is actually no help on reducing bandwidth cost, while increasing the risk of policy violation due to the longer path.

In this paper, we study both the policy migration and VM migration problem. We show that joint optimization of dynamic VM and policy migration can archive significant network-cost savings whilst still satisfying network policy requirements. Based on building the cost network among MBs

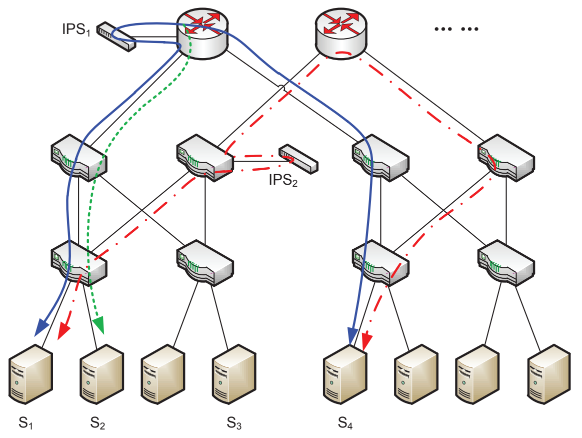


Fig. 1: Migrate policy or VMs separately make no improvement on efficiency. Solid blue: original flow; dash green: only migrate policy flow; dot dash red: only migrate VM

.

and applying the stable matching theory in the allocation of VMs, we proposed a simple and efficient Two-way migration scheme, which enables consolidation of both policies and VMs, while reducing the total communication cost of whole DC. 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 organised 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 *Two-way migration* algorithm is proposed in Section III. Section IV evaluates the performance of the proposed scheme 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 [7] or folded clos network [8][9].

Let $\mathbb{V} = \{v_1, v_2, \ldots\}$ be the set of VMs in the DC network 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 host 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 VM v_i, where $\sum_{v_k \in A(s_j)} r_k$ is the total requirements of all VMs hosted by s_j currently.

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), 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 networks. They can be on the networking path or off the physical network [1][12]. We consider the MBs are attached to switches for improved flexibility and scalability of policy deployment [1]. These MBs may belong to different applications, deployed and configured by a *MB Controller*, as shown in Fig. 1. The centralized *MB Controller* monitors the liveness of MBs and informs the switches regarding the addition or failure/removal of a MB. As the topology of MBs is relatively static after deployment, so that edge switches can easily retrieve the global information of MBs from the controller. Network administrators can specify and update policies, and reliably disseminate them to the corresponding switches through the *MB Controller*.

Traffic in DC is largely flow-based [8]. 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]

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. The set of policies is $\mathbb{P}$. For each $f_i \in \mathbb{F}$, there is a policy p_i. The $p_i.seq$ defines the sequence of MB types that all flows matching policy p_i should traverse in order, e.g., $p_i.seq = \{FW, IPS, Proxy\}$. The $p_i.len$ denotes the size of the MB list. $list$ is the list of MBs that are assigned to p_i to fulfil its requirement. 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 *satisfied*, if and only if all required MBs are allocated to p_i with the correct types and order:

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

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 $L(n_i, n_j)$ as the routing path between nodes (e.g., VMs, MBs or switches) n_i and n_j. $l \in L(n_i, n_j)$ if link l is on the path. For a flow f_i, where $p_i \neq \emptyset$, its actual routing path is:

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

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 [15]. Therefore, in order to accommodate a large number of VMs in the DC and at the same time keep 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 [9], 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

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

where $C_k(v_i, p_k.in) = \lambda_k(v_i, v_j) \sum_{l_s \in L(v_i, p_k.in)} c_s$ 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 m_l and its successor MB in $p_k.list$.

C. Policy and VM Migration 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.

[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 [10], [11] to reduce the dimensionality of the problem.

[2]There are a handful of research literature, e.g., [13][14], about deriving real time traffic matrices in DC networks.

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 a allocation A that minimizes the total communication cost:*

$$\begin{aligned} \min & \sum_{v_i \in \mathbb{V}} \sum_{v_j \in \mathbb{V}} C(v_i, v_j) \\ s.t. \; & A(v_i) \neq \emptyset, \forall v_i \in \mathbb{V} \\ & \sum\nolimits_{v_i \in A(s_j)} r_i \leq h_j, \forall s_j \in \mathbb{S} \\ & p_k \textit{ is satisfied}, \forall \; p_k \in \mathbb{P} \\ & \sum\nolimits_{p_k \in \mathcal{A}(m_i)} f_k.rate \leq m_i.capacity, \forall m_i \in \mathbb{M} \end{aligned} \tag{4}$$

PVC can be easily proven to be *NP-Hard* as it can be treated as a restricted version of the Generalized Assignment Problem (GAP) [16]. However, the GAP is APX-hard to approximate [16]. The existing centralized approximation algorithms are too complex and infeasible to implement over a DC environment which could include tens of thousands servers.

III. Two-way Migrations

The *PVC* problem involves placement of both VMs and policies. To solve it, we introduce a *Two-way* migration: *VMs migrations* among servers and *policy migration* among MBs.

A. VMs Migration and Cost

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

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

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:

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

Migrating a VM also generates network traffic between the source and destination hosts of the migration. 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 [17]. We borrowed the model for estimating migration cost defined in [18]:

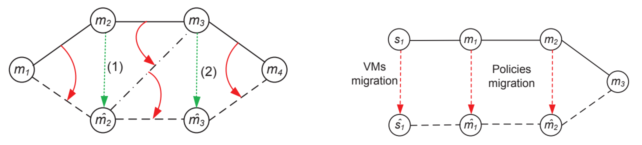
$$C_m(v_i) = M \cdot \frac{1 - (R/L)^{n+1}}{1 - (R/L)} \tag{7}$$

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 [18].

Such migration cost should not outweigh the reduction in the overall communication cost. We then consider *utility* as the gain of a migration. Hence, the *utility* of the migration $A(v_i) \to \hat{s}$ is defined as:

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

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



(a) (b)

Fig. 2: (a) Multiple MBs migration on a policy path equals to migrating them one by one: old path $m_1 \to m_2 \to m_3 \to m_4$, new path $m_1 \to \hat{m_2} \to \hat{m_3} \to m_4$. We can first migrate m_2 to $\hat{m_2}$, then migrate m_3 to $\hat{m_3}$. (b) Decomposable migration among VMs and MBs

B. Policy Migration and Cost

To preserve the correctness and fidelity of in-progress flows for policy migrations, 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* [19], which is a control plane with carefully designed APIs for managing MBs and policies.

Let $(p_k, i) \to \hat{m}$ 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:

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

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

1) Migration on Intermediate MBs Associated Policies: If migration of p_k involves two or more attached MBs along the policy path, we can migrate p_k from these MBs one by one. This can be easily proved to be equal through Fig. 2a. Thus, in the following analysis, we only need to consider the migration of p_k for one attached MB each time.

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

$$\begin{aligned} U((p_k, i) \to \hat{m}) = & \mathcal{C}_k(p_k.list[i-1], m') + \mathcal{C}_k(m', p_k.list[i+1]) \\ & - \mathcal{C}_k(p_k.list[i-1], \hat{m}) - \mathcal{C}_k(\hat{m}, p_k.list[i+1]) \\ & - \sigma_{ik} \end{aligned} \tag{10}$$

2) Migration on End MBs Associated Policies: End MBs for p_k involves the migration of either $p_k.in$ ($p_k.in \to \hat{m}$) or $p_k.out$ ($p_k.out \to \hat{m}$). The key difference of *utility* between migration of end MBs and intermediate MBs associated policies is that VMs is included for calculating the cost. Hence, the *utility* for migration of $p_k.in \to \hat{m}$ is:

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

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

C. Two-way Migration Algorithm

Based on above analysis, we can easily observe that, for a single flow, the migration of VMs and policies are all *decomposable*, which means the migration sequence for src/dst VMs and each MBs are independent. Without loss of generality, we use a flow for example in Fig. 2b. We consider migration on portion of a flow, i.e., migration of source VM which is hosted by s_1, policy migration on the ingress MB m_1 and one intermediate MB m_2.

According to utility definitions of both VMs and policy migration in the Equations (8), (11) and (10), we can easily show that:

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

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

In the following, we propose a Two-Way Migration scheme utilizing the property of decomposable. The whole scheme is comprised of two phases - to perform migration on policies and VMs respectively in order to reduce the total communication cost.

1) Phase I: Policy Migration: We have shown that migration of policies amongst MBs for a single flow is decomposable. Nevertheless, a VM usually has multiple active flows at a time. Therefore, in Phase I, we only migrate policies, while preparing for the migrations of VMs by building the preference matrix of servers on VMs.

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. Flow originates from the source ($f_i.src$) and terminate at the sink ($f_i.dst$). The first (or the last) tier is all possible servers that can accept $f_i.src$ (or $f_i.dst$) for VM migration defined in Equation (5), as well as its current host server. The middle n tiers are all possible MBs defined in Equation 9. Fig. 3 shows an example of a flow needing to traverse $\{IPS, RE\}$. The weight of each edge is initialized as the corresponding cost for migration. Specially, the weight of edges connected to source/sink are the migration cost of source/destination VMs.

Clearly, the route with largest utility for a flow through migration is the shortest path from source to sink. Thus, we proposed the *Policy Migration Algorithm*, shown in Algorithm 1, to minimizing the total communication cost through the

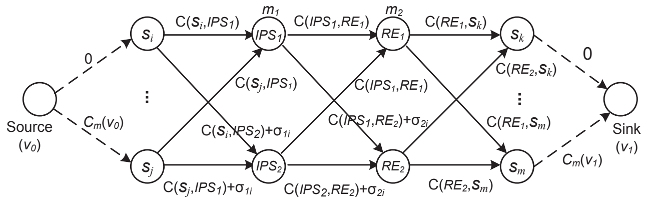


Fig. 3: Example: Suppose the current path for flow f_i path is $s_i \to IPS_1 \to RE_1 \to s_k$.

Algorithm 1 Phase I: Policy Migration

```
Input:  $\mathbb{V}, \mathbb{S}, \mathbb{F}, \mathbb{P}, \mathbb{M}$
Output: New allocation of MBs for all policies
 1: Sort $\mathbb{F}$ by rate in ascending order
 2: for each $f \in \mathbb{F}$ do
 3:     $p$ =the policy applied on $f$
 4:     Construct the cost network $N$
 5:     $(s_{src}, s_{dst}, mlist)$ = $SPF(N)$
 6:     for $i = 1$ to $mlist.len$ do
 7:         if $p.list[i] \neq mlist[i]$ then
 8:             Perform Policy migration: $p.list[i] \to mlist[i]$
 9:         end if
10:     end for
11:     Update routing for policies
12:     $\rho(s_{src}, f.src)$ += 1        ▷ Update preference matrix
13:     $\rho(s_{dst}, f.dst)$ += 1
14: end for
```

migration of policies. The function call $(s_{src}, s_{dst}, mlist)$ = *SPF*(N) returns the shortest path for f, 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. $\rho(s, v)$ is the scores of VM v, which is given by server s. $\rho(s, v)$ is initialized to be zero, and will be increased by one each times if a flow f, to/from v, chooses s for its shortest path in Algorithm 1.

2) Phase II: VM Migration: According to Equation (8), migrating v_i to different server will yield different *utility*, which means v_i has preference over 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, VMs also has a resource requirement when it is assigned to a server whose availability of resources is limited. These preference might be inconsistent, making it difficult to determine migration destination of each VMs.

To overcome this, we model the VMs migration problem above to be a typical *many-to-one stable matching*, hence Stable Marriage[20]. We then apply our modified *Gale-Shapley algorithm*[20] to address the conflict of preferences and efficiently match VMs to servers. The key concept of stable matching is the *stability*. Before explain the stability, we will

Algorithm 2 Phase II: VM Migration

```
Input: $\mathbb{V}, \mathbb{S}, \rho$
Output: New allocation of servers-VMs $\hat{A}$
 1: Obtain preference list $l_i$ according to $\rho$, $\forall s_i \in \mathbb{S}$
 2: Initialize blacklist $b_j = \emptyset$, $\forall v_j \in \mathbb{V}$
 3: $\hat{A} = \emptyset$
 4: while $\exists v_i$, and $\hat{A}(v_i) = \emptyset$ do
 5:     $s_j \leftarrow \arg\max_{s \in S \backslash b_i} U(A(v_i) \to s)$
 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 $v_k \in l_j$, $v_k \prec_{l_j}$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: $A \to \hat{A}$
19: Update routing for VMs
```

first define the *blocking pair*.

Definition 2. *For an allocation $\mathcal{A}$, a VM-server pair (v_i, s_j) is a* blocking pair *if $\mathcal{A}(v_i) \neq s_j$, $U(A(v_i) \to s_j) > 0$, and any of the two conditions holds* [3]

$$\forall v_k \in A(s_j) \text{ and } v_i \prec_{l_j} v', \sum r_k + r_i \prec h_j \qquad (13)$$

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

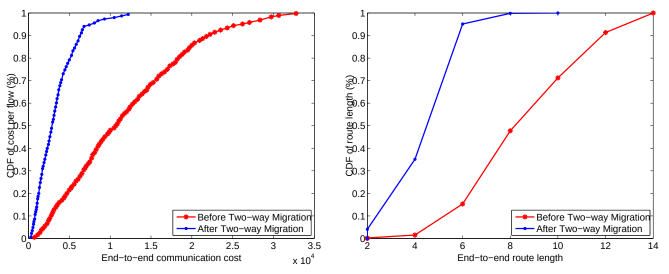
A stable matching means no *blocking pair* exists in the final matching of VMs and servers. An *unstable* matching between VMs and servers will always leave rooms to minimizing the total communication cost, while a *stable* matching is the optimal assignment for both VMs and servers.

The Phase II algorithm for VM migration is shown in Algorithm 2. Initially, all VMs are unmatched. For such a VM, say v_i, $\hat{A}(v_i) = \emptyset$. v_i will be first matched to the server which has not yet rejected v_i and can gain the largest utility (line 5∼6), say server s_j. If s_j has sufficient capacity, it accepts v_i. Otherwise, it sequentially rejects less preferable VMs, which are allocated to s_j previously. Whenever s_j rejects a VM, it updates the best_rejected variable, and at the end all VMs ranked lower than best_rejected are removed from its preference.

We can prove that Algorithm 2 can always output a stable matching.

Theorem 1. *The Algorithm 2 can always output a stable matching in $O(\mathcal{VS})$, where $\mathcal{V}$ and $\mathcal{S}$ are the number of input VMs and servers.*

[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 [10]



(a) Cost of flows (b) Route length of flows

Fig. 4: Performance of Two-way migration

Proof: We can prove the stability of the output matching by contradiction. Suppose that the Algorithm 2 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 either be rejected by s_j before, or add s_j to its blacklist when v_i is rejected (in line 14). Thus, $v' \notin \hat{A}(s_j)$, which is contradicts with the assumption.

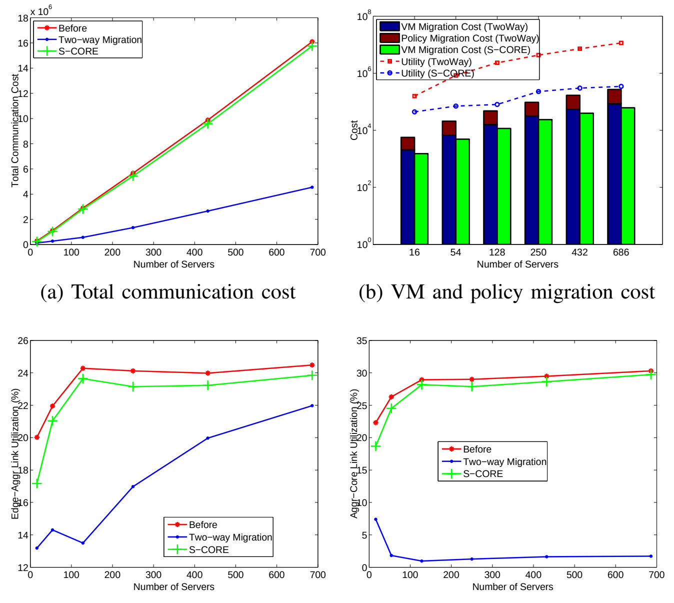
In the worst case, each VM is rejected by every server. So, Algorithm 2 will always be terminated and output a stable matching within $O(\mathcal{VS})$. ■

IV. Evaluation

We have evaluated the performance of the two-way migration through extensive simulations in a fat-tree DC topology with $k = 14$ (i.e., 686 servers and 245 switches). In the simulation, VMs are modeled as a collection of socket applications communicating with one or more other VMs in the DC network. In all experiments, we have set 10% of flows to be policy-free, meaning that they are not governed by any existing network policies. For the other 90% of flows, they have to traverse a sequence of MBs specified by policies before being forwarded to their destination. Each policy-constrained flow is configured to traverse 1∼3 MBs, including firewall, IPS, RE or LB. A centralized controller is implemented to collect all network information and perform the two-way migration algorithm.

Fig. 4 shows the evaluation results of two-way migration. Fig. 4a demonstrates that, throughout the simulation, the two-way migration method successfully reduces total communication cost by 63.64%. In addition, by combining migration of both VMs and policies, the average route length can also be significantly reduced by as much 40%, as exhibited in Fig. 4b.

Fig. 5 shows the performance comparison between two-way migration and S-CORE [5] which is distributed communication cost reduction but policy-agonistic scheme. Obviously, Fig. 5b illustrates that policy-agonistic S-CORE can only marginally reduce communication cost. More interestingly, we can see from Fig. 5b that the VM migration cost of two-way is only slightly higher than that of S-CORE, but the utility received by two-way is 20 times better than S-CORE. Fig. 5c and Fig. 5d demonstrate that two-migraiton can mitigate link utilization at the core and aggregation layers by 90% and 28%, respectively, compared to that of 6% and 5.2% for S-CORE.



(a) Total communication cost

(b) VM and policy migration cost

(c) Edge-Aggr link utilization

(d) Aggr-Core link utilization

Fig. 5: Comparision of Two-way migration and S-CORE

V. Related Work

Recent researches on network policy focus on leveraging SDN to enable more flexible middlebox management over the network while ensuring traversal requirement. Zafar et al. [21] proposed *SIMPLE*, a SDN-based policy enforcement scheme to steer DC traffic in accordance to policy requirements. Fayazbakhsh et al. [22] proposed FlogTags to guarantee correctness of policy enforcement. Gember et al. [19] proposed *OpenNF*, which is a control plane with carefully designed APIs for managing MBs and policies, e.g., cloning, sharing, moving and merging MB states. However, current approaches on policy and middleboxes management ignore elastic resource provisioning through server virtualization.

Cloud DC environments require more dynamic application management, and considerable works have been proposed on VM placement, consolidation and migration for server and network resource optimization [5][2]. In this paper, we have demonstrated in that schemes such as S-CORE [5] perform poorly due to ignoring network policies. The framework PACE (Policy-Aware Application Cloud Embedding) [2] can support application-wide, in-network policies, and other realistic requirements such as bandwidth and reliability. However, it only considers one-off VM placement and fails to deal with dynamic workloads, which is common in Cloud DC.

VI. Conclusion

Network policies and virtual machines are the basics of DC technologies today. In this paper, we study the cost reduction in DC by jointly considering both virtual machine and network policy migrations. We first prove that the optimization problem is NP-Hard, and then proposed a Two-way migration scheme to minimize the cost by performing policy migration and VM migration in two phases. Extensive simulation results show that the two-way migration can significantly reduce the total cost and link utilization at the higher layers of the network architecture.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

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

[6] Y. Zhao, Y. Huang, K. Chen, M. Yu, S. Wang, and D. Li, "Joint vm placement and topology optimization for traffic scalability in dynamic datacenter networks," *Computer Networks*, vol. 80, pp. 109–123, 2015.

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

[8] 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.

[9] 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.

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

[11] M. Korupolu, A. Singh, and B. Bamba, "Coupled placement in modern data centers," in *IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*. IEEE, 2009, pp. 1–12.

[12] L. E. Li, M. F. Nowlan, Y. R. Yang, and M. Zhang, "Mosaic: policy homomorphic network extension," in *Proceedings of the 4th International Workshop on Large Scale Distributed Systems and Middleware*. ACM, 2010, pp. 38–43.

[13] Z. Hu, Y. Qiao, and J. Luo, "Coarse-grained traffic matrix estimation for data center networks," *Computer Communications*, 2014.

[14] Y. Qiao, Z. Hu, and J. Luo, "Efficient traffic matrix estimation for data center networks," in *IEEE IFIP Networking Conference*, 2013, pp. 1–9.

[15] 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.

[16] "Generalized assignment problem." [Online]. Available: http://en.wikipedia.org/wiki/Generalized_assignment_problem

[17] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association, 2005, pp. 273–286.

[18] 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.

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

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

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

[22] S. K. Fayazbakhsh, V. Sekar, M. Yu, and J. C. Mogul, "Flowtags: enforcing network-wide policies in the presence of dynamic middlebox actions," in *ACM SIGCOMM HotSDN*. ACM, 2013, pp. 19–24.