# PLAN: A Policy-Aware VM Management Scheme for Cloud Data Centres

Lin Cui*, Fung Po Tso†, Dimitrios P. Pezaros‡, Weijia Jia§
*Department of Computer Science, Jinan University, Guangzhou, China
†School of Computing & Mathematical Science, Liverpool John Moores University, L3 3AF, UK
‡School of Computing Science, University of Glasgow, G12 8QQ, UK
§Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai China
Email: tcuilin@jnu.edu.cn; p.tso@ljmu.ac.uk; dimitrios.pezaros@glasgow.ac.uk; jia-wj@cs.sjtu.edu.cn;

***Abstract*—Policies play an important role in network configuration and therefore in offering secure and high performance services especially over multi-tenant Cloud Data Center (DC) environments. At the same time, elastic resource provisioning through virtualization often disregards policy requirements, assuming that the policy implementation is handled by the underlying network infrastructure. This can result in policy violations, performance degradation and security vulnerabilities.**

**In this paper, we define *PLAN*, a PoLicy-Aware and Network-aware VM management scheme to jointly consider DC communication cost reduction through Virtual Machine (VM) migration while meeting network policy requirements. We show that the problem is NP-hard and derive an efficient approximate algorithm to reduce communication cost while adhering to policy constraints. Through extensive evaluation, we show that *PLAN* can reduce topology-wide communication cost by 38% over diverse aggregate traffic and configuration policies.**

***Keywords*-Data center networks; virtual machine; migration; policies; middleboxes;**

## I. Introduction

Network configuration and management is a complex task often overlooked by research that focuses on improving resource usage efficiency. However, providing secure and balanced distributed services while maintaining high application performance is a major challenge for providers. In Cloud Data Centers (DC)s in particular, this challenge is amplified by the collocation of diverse services over a centralized infrastructure, as well as by virtualization that decouples services from the physical hosting platforms. Applications over DC networks have complex communication patterns which are governed by a collection of network policies regarding security and performance. In order to implement these policies, network operators typically deploy a diverse range of network appliances or "middleboxes", including firewalls, traffic shapers, load balancers, Intrusion Detection and Prevention Systems (IDS/IPS), and application enhancement boxes [1]. Across all network sizes, the number of middleboxes is on par with the number of routers in a network, hence such deployments are large and require high up-front investment in hardware on the order of thousands to millions of dollars [2][3].

Network policies demand traffic to traverse a sequence of specified middleboxes. As a result, network administrators are often required to manually install middleboxes in the data path of end points or significantly alter network partition and carefully craft routing in order to meet policy requirements. There is a consequent lack of flexibility that makes DC networks prone to misconfiguration, and it is no coincidence that there is emerging evidence demonstrating that up to 78% of DC downtime is caused by misconfiguration [2] [4].

On the other hand, Cloud applications can be rapidly deployed or scaled on-demand, fully exploiting resource virtualization. Consolidation is the most common technique used for reducing the number of servers on which VMs are hosted to improve server-side resource fragmentation, and is typically achieved through VM migration. When a VM migrates, it retains its IP address, and the standard 5-tuple (source and destination addresses, source and destination ports, protocol) used to describe a flow remain the same. This implies that migrating a VM from one server to another will inevitably alter the end-to-end traffic flow paths requiring subsequent dynamic change or update of the affected policy requirements [5]. Clearly, change of the point of network attachment as a result of VM migrations substantially increases the risk of breaking predefined sequence of middlebox traversals and lead to violations of policy requirements. It has been demonstrated in PACE [1] that deployment of applications in Cloud DC without considering network policies may lead to up to 91% policy violations.

It is common in DCs that an application involving multiple VMs (e.g., indexing, document, web, etc.) is hosted in non-collocated servers. The underlying traffic flows need to traverse distinct firewalls and IPSes that are attached to different switches and routers, making the true end-to-end paths longer than shortest paths due to middlebox traversals (see Fig. 1). Therefore, when deciding where to migrate any one of these VMs, locations of these middleboxes have to be taken into considerations. Failing to do so, will not only lead to sub-optimal performance due to much longer middlebox traversal paths, but also cause service disruption and unreachability as a result of being unable to follow a predefined sequence of middlebox traversal rules. However, to date, network policy and VM management are treated disjointly. PACE [1] is the only work that jointly considers middlebox traversal and VM placement in a Cloud DC environment, however it only considers static placement and does not provide any reliable mechanisms to facilitate subsequent dynamic VM migration. In contrast, our initial effort has shown that policy-aware dynamic VM consolidation can remarkably improve network utilization [6].

Besides PACE, existing proposals that aim to dynamically manage network policies can be broadly classified into the following two categories, both of which fail to address the aforementioned problem.

- *Virtualization and Consolidation*: Software-centric middlebox applications have been proposed to separate policy from reachability (i.e., virtualization) [3][4] and middlebox functions can be consolidated [3] dynamically. But consolidation of middleboxes may not always be feasible, as different capabilities (e.g., proxy, firewall) arse provided by separate hardware middleboxes. Furthermore, if not dealt with carefully when migrating VMs, middlebox consolidation can put VMs on the risk of policy violation. In addition, the requirement for high-speed, often hardware-accelerated operation, as well as the need for in-network deployment, does not currently allow for a fully virtualized policy implementation [7].
- *SDN-based policy enforcement*: Software-Defined Networking (SDN) [8] has enabled a new paradigm for enforcing middlebox policies [7][9][10][11]. SDN abstracts a logically centralized global view of the network and can be exploited to programmatically ensure correctness of middlebox traversal. Some simple middlebox functions (e.g., NAT, load balancers) are implemented directly in (SDN) switches to ensure speed and flexibility, but these switches have limited on-board memory to store a potentially very large number of forwarding rules. Moreover, SDN cannot solely make flow decisions to adhere to policies, specially when VMs migrate.

In this paper, we explore the policy-aware VMs migration problem, and present an efficient PoLicy-Aware VM maNagement (*PLAN*) scheme, which, (a) adheres to policy requirements, and (b) reduces network-wide communication cost in DC networks. The communication cost is defined with respect to policies associated to each VM. In order to attain both goals, we model the *utility* (i.e., the reduction ratio of communication cost) of VM migration under middlebox traversal requirements and aim to maximize it during each migration decision. To the best of our knowledge, this is the first joint study on policy-aware performance optimization through elastic VM management in DC networks.

The contributions of this paper are three-fold:

1) The formulation of the policy-aware VM management problem (*PLAN*), the first study that jointly considers policy-aware VM migration and performance optimization in DC networks;
2) An efficient distributed algorithm to optimize network communication cost and guarantee network policy compliance;
3) An extensive performance evaluation demonstrating that *PLAN* can effectively reduce communication cost while meeting policy requirements.

The remainder of this paper is organized as follows. Section II describes the model of policy-aware VM management (*PLAN*), and defines the communication cost and utility for VM migration. An efficient, distributed algorithm and implementation are proposed in Section III. Section IV evaluates the performance of *PLAN*. Section V outlines related work on VM migration and policy implementations. Finally, Section VI concludes the paper.

## II. Problem Modeling

### A. Motivating Example

We describe a common DC Web service application as an example to demonstrate that migrating VMs without policy-awareness will lead to unexpected results and application performance degradation.

*1) Topology and Application:* Fig. 1 depicts a typical Fat-tree DC network topology [12] that consists of a number of network switches and several distinct types of middleboxes. *Firewall* $F_1$ will filter unwanted or malicious traffic and protect tenants' networks in the DC from the Internet. *Intrusion Prevention Systems* (IPS), e.g., $IPS_1$ and $IPS_2$, are configured with a ruleset, monitoring the network for malicious activity, and subsequently log and block/stop it. They also provide a detailed view and checking of how well each middlebox is performing for the traffic flow. A *Load Balancer*, e.g., $LB_1$, provides one point of entry to the web service, but forwards traffic flows to one or more hosts, e.g. $v_1$, which provide the actual service. In this example, $v_1$ is a web server, which accepts HTTP requests from an Internet client (denoted by $u$). After receiving such requests, $v_1$ will query data server $v_2$ (i.e., a database), perform some computation based on the fetched data, and feed results back to the client.

*2) Policy Configurations:* Polices are identified through a 5-tuple and a list of middleboxes (A more formal definition is given in Section II-B). The following policies are configured through the *Policy Controller* to govern traffic related to the web application in this example:

- $p_1 = \{u, LB_1, *, 80, HTTP\} \rightarrow \{F_1, LB_1\}$
- $p_2 = \{u, v_1, *, 80, HTTP\} \rightarrow \{IPS_1\}$
- $p_3 = \{v_1, v_2, 1001, 1002, TCP\} \rightarrow \{LB_2, IPS_2\}$
- $p_4 = \{v_2, v_1, 1002, 1001, TCP\} \rightarrow \{IPS_2, LB_2\}$
- $p_5 = \{v_1, u, 80, *, HTTP\} \rightarrow \{IPS_1, LB_1\}$
- $p_6 = \{LB_1, u, 80, *, HTTP\} \rightarrow \{\}$

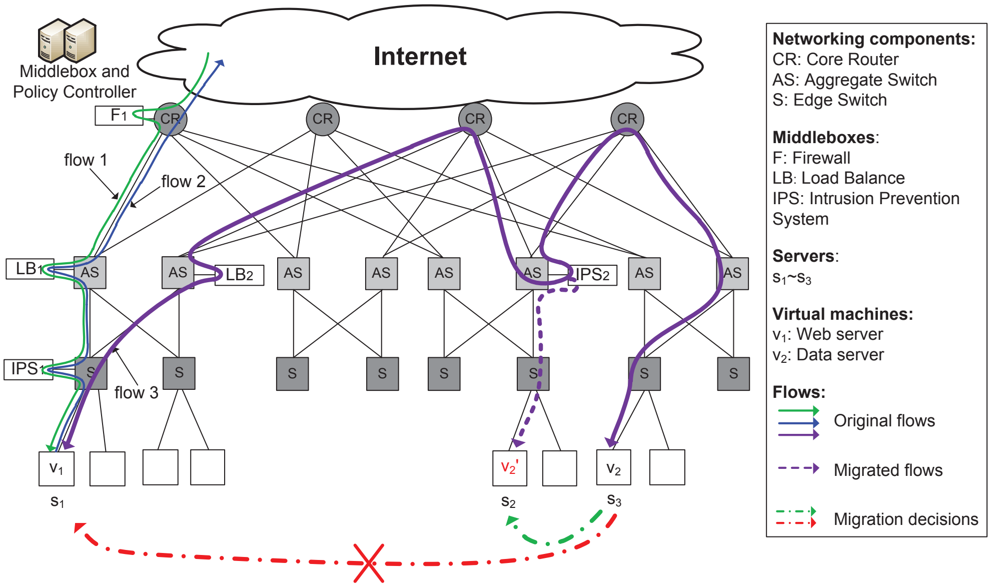


Figure 1: Flows traversing different sequences of middleboxes in DC networks. Without policy-awareness, $v_2$ will be migrated to $s_1$, resulting in longer paths for flow 1 and wasting network resources.

Policy $p_1$: The Internet client first sends a HTTP request to the public IP address of $LB_1$. All traffic from the Internet must traverse firewall $F_1$, which is in charge of the first line of defense and configured to allow only HTTP traffic.

Policy $p_2$: $LB_1$ will load-balance among several web servers and change the destination to web server $v_1$ in the example. Traffic will need to traverse $IPS_1$, which protects web servers.

Policy $p_3, p_4$: $v_1$ will communicate with a data server to fetch the required data, which is in turn protected by $IPS_2$. This traffic will be forwarded to $LB_2$ for load-balancing first, and then reach the data server $v_2$ after traversing $IPS_2$. The response traffic from $v_2$ to $v_1$ also needs to traverse both $IPS_2$ and $LB_2$.

Policy $p_5, p_6$: Upon getting the required data from the data server, the web server will send computed results to client. The reply traffic is sent to $LB_1$ first, traversing $IPS_1$, and then forwarded to the Internet client by $LB_1$. Any traffic originating from $v_1$ and destined to an Internet client needs no further checks, and hence does not need to traverse $F_1$.

*3) Migration Rule:* The DC network is often increasingly oversubscribed from bottom to core layers in a bid to reduce total investment. In order to reduce congestion in the core layers of DC network, effective VM management schemes cluster VMs to confine traffic in lower layers of the network such that as much traffic as possible is only routed over the edge layer (which is not oversubscribed) [13][14]. As a result, VMs as well as middleboxes which exchange packets more often and intensively are collocated in order to keep traffic within the edge layer boundaries.

Consider the migration of $v_2$ in the above example application. $v_2$ was originally hosted by server $s_2$. A large traffic volume needs to be exchanged between web server $v_1$ and data server $v_2$. This would cost precious bandwidth on core routers. Without policy awareness, in order to consolidate VMs on servers and keep traffic in the edge layer, $v_2$ may be migrated to $s_1$ so that $v_1$ and $v_2$ are close to each other. However, it will increase the route length of flow 3 and waste more network bandwidth. This is because that all traffic between $v_1$ and $v_2$ need to traverse $LB_2$ and $IPS_2$, according to the policy rules (i.e., $p_3$ and $p_4$). Considering policy configurations and traffic patterns in this example, when migrating $v_2$, it should be migrated to server $s_2$ to reduce the cost generated between $v_2$ and $IPS_2$.

Clearly, policy-aware VM migration will require finding an optimal placement whilst satisfying network bandwidth and policy requirements. Unless stated otherwise, our discussion and problem formulation in the rest of this paper focus on policy-aware VM migration with an aim to reduce over communication cost.

### B. Communication Cost with Policies

We consider a multi-tier DC network which is typically structured under a multi-root tree topology (canonical [15] or fat-tree [12]).

Let $\mathbb{V} = \{v_1, v_2, \ldots\}$ be the set of VMs in the DC network hosted by the set of servers $\mathbb{S} = \{s_1, s_2, \ldots\}$. Let $\lambda_k(v_i, v_j)$ denote the *traffic load* (or rate) in data per time unit exchanged between VM $v_i$ and $v_j$ (from $v_i$ to $v_j$) following policy $p_k$.

For a group of middleboxes $MB = \{mb_1, mb_2, \ldots\}$, there are various deployment points in DC networks. They can be on the networking path or off the physical network [4]. Without loss of generality, we consider that middleboxes are attached to switches for improved flexibility and scalability of policy deployment [4]. These middleboxes may belong to different applications, deployed and configured by a *Middlebox Controller*, see Fig. 1. The centralized *Middlebox Controller* monitors the liveness of middleboxes and informs the switches regarding the addition or failure/removal of a middlebox. Network administrators can specify and update policies, and reliably disseminate them to the corresponding switches through the *Policy Controller* in Fig. 1.

The set of policies is $\mathbb{P} = \{p_1, p_2, \ldots\}$. Each policy $p_i$ is defined in the form of $\{flow \rightarrow sequence\}$. $flow$ is represented by a 5-tuple: $\{src_{ip}, dst_{ip}, src_{port}, dst_{port}, proto\}$ (i.e., source and destination IP addresses and port numbers, and protocol type). $sequence$ is a list of middleboxes that all flows matching policy $p_i$ should traverse them in order: $p_i.sequence = \{mb_1^i, mb_2^i, \ldots\}$. We denote $p_i^{in}$ and $p_i^{out}$ to be the first (ingress) and last (egress) middleboxes respectively in $p_i.sequence$. Let $P(v_i, v_j)$ be the set of all policies defined for traffic from $v_i$ to $v_j$, i.e., $P(v_i, v_j) = \{p_k | p_k.src = v_i, p_k.dst = v_j\}$.

We denote $L(n_i, n_j)$ to be the routing path between nodes (e.g., VM, middlebox or switch) $n_i$ and $n_j$. $l \in L(n_i, n_j)$ if link $l$ is on the path. If a flow from VM $v_i$ to $v_j$ is matched to policy $p_k$, its actual routing path is:

$$\begin{aligned} L_k(v_i, v_j) = \ & L(v_i, p_k^{in}) \\ & + \sum_{mb_s^k \neq p_k^{out}} L(mb_s^k, mb_{s+1}^k) \\ & + L(p_k^{out}, v_j) \end{aligned} \tag{1}$$

Not all DC links are equal, and their cost depends on the particular layer they interconnect. High-speed core router interfaces are much more expensive (and, hence, oversubscribed) than lower-level ToR switches [13]. 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, we can assign a representative link weight $\omega_i$ for an $i$th-level link per data unit. Without loss of generality, in this case $\omega_1 < \omega_2 < \omega_3$.

Hence, the *Communication Cost* of all traffic from VM $v_i$ to $v_j$ is defined as

$$\begin{aligned} C(v_i, v_j) &= \sum_{p_k \in P(v_i, v_j)} \lambda_k(v_i, v_j) \sum_{l_s \in L_k(v_i, v_j)} c_s \\ &= \sum_{p_k \in P(v_i, v_j)} (C_k(v_i, p_k^{in}) + C_k(p_k^{in}, p_k^{out}) \\ & \qquad + C_k(p_k^{out}, v_j)) \end{aligned} \tag{2}$$

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 $C_k(p_k^{in}, p_k^{out})$ is the communication cost between $p_k^{in}$ and $p_k^{out}$.

Since we jointly consider compliance of network policies and minimization of network communication cost through VM migration, $C_k(p_k^{in}, p_k^{out})$ in (2) can be ignored as it makes no contribution to the minimization of the communication cost.

### C. Policy-Aware VM Allocation Problem

We denote $MB^{in}(v_i)$ to be the set of ingress middleboxes of all outgoing flows from $v_i$, i.e., $MB^{in}(v_i) = \{mb_j | mb_j = p_k^{in}, p_k.src = v_i\}$. Similarly, $MB^{out}(v_i) = \{mb_j | mb_j = p_k^{out}, p_k.dst = v_i\}$ is the set of egress middleboxes of all incoming flows to $v_i$.

As each server is connected to an edge switch, and each edge switch can retrieve the global graph of all middleboxes from the Policy Controller, we define all the servers that can reach middlebox $mb_k$ as $S(mb_k)$. Thus, to preserve the policy requirements, the acceptable servers that a VM $v_i$ can migrate to are:

$$S(v_i) = \bigcap_{mb_k \in MB^{in}(v_i) \cup MB^{out}(v_i)} S(mb_k) \tag{3}$$

Hence, for traffic not governed by any policies, $S(v_i)$ is all servers that can be reached by $v_i$, i.e., possible destinations where $v_i$ can be migrated to.

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 resources such as CPU cycles, memory size, and I/O operations, respectively. Accordingly, the amount of physical resource provisioning by host server $s_j$ is given by a vector $H_j$. And $R_i \preceq H_j$ means all types of resource of $s_j$ are enough to accept $v_i$.

We denote $A$ to be an allocation of all VMs. $A(v_i)$ is the server which hosts $v_i$ in $A$, and $A(s_j)$ is the set of VMs hosted by $s_j$. 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:

$$\mathcal{S}_i = \{\hat{s} | (\sum_{v_k \in A(\hat{s})} R_k + R_i) \preceq H_j, \hat{s} \in S(v_i)\} \tag{4}$$

Let $\mathcal{C}_i(s_j)$ be the total communication cost induced by $v_i$ between $s_j$ and $MB^{in}(v_i) \cup MB^{out}(v_i)$, where $s_j = A(v_i)$.

$$\mathcal{C}_i(s_j) = \sum_{p_k \in P(v_i, *)} C_k(v_i, p_k^{in}) + \sum_{p_k \in P(*, v_i)} C_k(v_i, p_k^{out}) \tag{5}$$

Migrating a VM also generates network traffic between the source and destination hosts of the migration, as it involves copying the in-memory state and the content of

CPU registers between the hypervisors. The live migration allows moving a continuously running VM from one physical host to another. To enable that, modern DC networks use a technique called pre-copy and it is comprised of three phases: pre-copy phase, pre-copy termination phase and stop-and-copy phase [16]. 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].

The estimated migration cost defined in [17] is:

$$C_m(v_i) = M \cdot \frac{1-(R/L)^{n+1}}{1-(R/L)} \tag{6}$$

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 [17].

Such migration overhead can be measured by the hypervisor hosting the VM and should not outweigh the reduction in the overall communication cost. We then consider the *utility* in terms of the expected benefit (of migrating a VM to a server) minus the expected cost incurred by such operation. The *utility* of the migration $A(v_i) \to \hat{s}$ is defined as:

$$U(A(v_i) \to \hat{s}) = \mathcal{C}_i(A(v_i)) - \mathcal{C}_i(\hat{s}) - C_m(v_i) \tag{7}$$

Specifically, $U(A(v_i) \to \hat{s}) = 0$ if no migration takes place, i.e., $A(v_i) = \hat{s}$. The *total utility* $\mathcal{U}_{A \to \hat{A}}$ is the summation of *utilities* for all migrated VMs from allocation $A$ to $\hat{A}$.

The *PoLicy-Aware VM maNagement (*PLAN*)* problem is defined as follows:

**Definition 1.** *Given the set of VMs $\mathbb{V}$, servers $\mathbb{S}$, policies $\mathbb{P}$, and an initial allocation A, we need to find a new allocation $\hat{A}$ that maximizes the total utility:*

$$\begin{aligned} &\max \mathcal{U}_{A \to \hat{A}} \\ &s.t. \ \mathcal{U}_{A \to \hat{A}} > 0 \\ &\qquad \hat{A}(v_i) \in S_i, \forall v_i \in \mathbb{V} \end{aligned} \tag{8}$$

*PLAN* can be treated as a restricted version of the Generalized Assignment Problem (GAP) [18]. However, the GAP is APX-hard to approximate [18]. The existing centralized approximation algorithms are too complex and infeasible to implement over a DC environment, which could include thousands or millions of servers, VMs, switches and traffic flows.

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

*Proof:* To show the non-polynomial complexity of *PLAN*, we will show that the Multiple Knapsack Problem (MKP) [19], 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 allocation $A_0$, in which all VMs are allocated to one server $s_0$, then the *PLAN* problem is to find a new allocation $\hat{A}$ for migrating VMs that maximizes the total utility $\mathcal{U}_{A_0 \to \hat{A}}$. We denote $S' = \mathbb{S} \setminus \{s_0\}$ to be the set of destination servers for migration. For a VM $v_i$, suppose the computed communication cost induced by $v_i$ on all candidate servers is the same, i.e., $C_i(\hat{s}) = \delta_i, \forall \hat{s} \in S'$, where $\delta_i$ is a constant. Consider each VM to be an item with size $R_i$ and profit $U(A(v_i) \to \hat{s}) = \mathcal{C}_i(A(v_i)) - \delta_i - C_m(v_i)$, each server $s_j \in S'$ to be knapsack with capacity $H_j$. The *PLAN* problem becomes finding a feasible subset of VMs to be migrated to servers $S'$, maximizing the total profit. Therefore, the MKP problem is reducible to the *PLAN* problem in polynomial time, and hence the *PLAN* problem is NP-hard.

$\blacksquare$

## III. PLAN Algorithms

The *PLAN* problem is a restricted version of the Generalized Assignment Problem (GAP), which has been proved APX-hard to approximate [18]. We can use some existing centralized algorithms to approximately maximize the total gained *utility* by migration, e.g., [20], [21]. However, the computation times of those algorithms are unacceptable for DCs, specially considering the large scales of servers, VMs, switches and millions of traffic flows [14]. In this section, we design a decentralized heuristic scheme to perform policy-aware VMs migration.

### A. Policy-Aware Migration Algorithms

Server hypervisors (or SDN controller, if used) will monitor all traffic load for each collocated VM $v_i$. A migration decision phase will be triggered periodically during which $v_i$ will compute the appropriate destination server $\hat{s}$ for migration. If no migration is needed, $U(A(v_i) \to \hat{s}) = 0$. Otherwise, the total *utility* is increased after migration when $A(v_i) \neq \hat{s}$.

Algorithm 1 and Algorithm 2 show the corresponding routines for VMs (*PLAN-VM*) and servers (*PLAN-Server*), respectively. *PLAN-VM* is only triggered for a migration decision every $T_m + \tau$ time, where $\tau$ is a random value to avoid synchronization of VM migrations. *PLAN-VM* operations will be suppressed for $T_s$ time period if $v_i$ is migrated to a new server, avoiding too frequent migration or oscillation among servers. The value of $T_s$ depends on the traffic patterns, e.g., smaller value for a DC with more stable traffic. *PLAN-Server* is designed for hypervisors on servers which can accept requests from VMs based on the residual resources of the corresponding server and prepare for migration of remote (incoming) VMs.

Several control messages will be exchanged for both *PLAN-VM* and *PLAN-Server*. The interface *sendMsg(type, destination, resource)* sends a control message of a specified type and resource declaration to the destination. The

**Algorithm 1** PLAN-VM for $v_i$

```
    /* Triggered every $T_m + \tau$ period*/
 1: $L = \emptyset$
 2: DECISION-MIGRATION($v_i$, $L$)
 3: loop
 4:     $msg \leftarrow$ getMsg()
 5:     switch $msg.type$ do
 6:         case reject
 7:             $L = L \cup \{msg.sender\}$
 8:             DECISION-MIGRATION($v_i$, $L$)
 9:         case accept
10:             sendMsg(migrate, $msg.sender$, $R_i$)
11:             perform migration: $v_i \rightarrow s$
12:     end switch
13: end loop

14: function DECISION-MIGRATION($v_i$, $L$)
15:     $s_0 \leftarrow A(v_i)$
16:     $\mathcal{S}_i \leftarrow$ feasible servers in Equation (4)
17:     $X \leftarrow \arg\max_{x \in \mathcal{S}_i \setminus L} U(A(v_i) \rightarrow x)$
18:     if $X \neq \emptyset$ && $s_0 \notin X$ then
19:         $s \leftarrow$ the one with most residual resources in $X$
20:         sendMsg(request, $s$, $R_i$)
21:     else
22:         exit        ▷ exit whole algorithm if no migration
23:     end if
24: end function
```

**Algorithm 2** PLAN-Server for $s_j$

```
 1: loop
 2:     $msg \leftarrow$ getMsg()
 3:     switch $msg.type$ do
 4:         case request
 5:             $v_i = msg.sender$
 6:             $R_i = msg.resouce$
 7:             if $\sum_{v_k \in A(s_j)} R_k + R_i \leq H_j$ then
 8:                 sendMsg(accept, $v_i$)
 9:             else
10:                 sendMsg(reject, $v_i$)
11:             end if
12:         case migrate
13:             if $\sum_{v_k \in A(s_j)} R_k + R_i \leq H_j$ then
14:                 provisionally resource reservation etc.
15:             else
16:                 sendMsg(reject, $v_i$)
17:             end if
18:     end switch
19: end loop
```

interface *getMsg()* reads such messages when received. The *request* message is a probe from VM to a destination server for migration. A server can respond by sending back an *accept* or *reject* message, according to the residual resource of the server and the requirements of the VM. If the server accepts the request from the distant VM, a *migrate* message will be sent back as confirmation.

For each VM $v_i$, the *PLAN-VM* algorithm starts with checking feasible servers, in a greedy manner, for improving *utility* by calling the function *Decision-Migration()*, e.g., line 2 and 7. The function *Decision-Migration()* will find a potential destination server for $v_i$ to perform migration. A blacklist $L$ is maintained during each execution of *PLAN-VM* to avoid repeating request for the same servers which reject $v_i$ previously. If a feasible server $s$ accepts $v_i$'s request, $v_i$ will be migrated to $s$, e.g., line 10 $\sim$ 11. For each server $s_j$, the *PLAN-Server* algorithm keeps listening incoming migration request from VMs. For a request from $v_i$, $s_j$ will check its residual resources and send back an *accept* message if it has enough resource to host $v_i$, e.g., line 5 $\sim$ 8. Otherwise, it will reject the migration request of $v_i$, e.g., line 16.

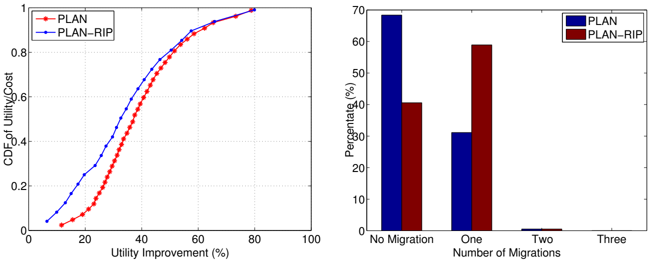
The *PLAN* scheme described in Algorithms 1 and 2 can decrease the total communication cost and will eventually converge to a stable state:

**Theorem 2.** *Algorithms 1 and 2 will converge after a finite number of iterations.*

*Proof:* The cost of each VM $v_i$ is determined by its hosting server and related ingress/egress middleboxes in $MB^{in}(v_i)$ and $MB^{out}(v_i)$. Hence, under the policy scheme described in the previous section, the migrations of different VMs are independent. Furthermore, each time a migration occurs in *line 11* of Algorithms 1, say, $A(v_i) \rightarrow s$, the utility gained from the migration is always larger than zero, i.e., $U(A(v_i) \rightarrow s) > 0$. Thus, the total induced communication cost, which is always a positive value, is strictly decreasing while VMs are migrating among servers. So, the two algorithms will converge after a finite number of steps. ∎

### B. Initial Placement

Policy-aware initial placement of VMs is also critical for new VMs in DC networks. When a VM instance, say $v_i$, is to be initialized, the DC network controller needs to find a suitable server to host the VM. Initially, predefined application-specific policies should be known for $v_i$. Together with $v_i$'s resource requirement $R_i$ and all servers' residual resources in the DC network, the feasible decision space $\mathcal{S}_i$ can be obtained through Equation (4). Since the VM has just been initialized, its traffic load might not be available. However, we can still choose the best server to host $v_i$ by considering traffic of all policies for $v_i$ equally, e.g., $\lambda_k = 1, \forall p_k \in P(v_i, *) \cup P(*, v_i)$. In particular, the migration cost $C_m(v_i), \forall v_i \in \mathbb{V}$, is set to be zero during



(a) CDF: ratio of utility to communication cost

(b) Number of migrations before convergence

Figure 2: Performance of PLAN

initial placement. Then, the destined server to host $v_i$ is $\arg\max_{s\in\mathcal{S}_i} \mathcal{C}_i(s)$.

## IV. Evaluation

### A. Experimental Setup

We have implemented *PLAN* in ns-3 [22] and evaluated it under a fat-tree DC topology. In our simulation environment, a single VM is modeled as a collection of socket applications communicating with one or more other VMs in the DC network. For each server, we have implemented a VM hypervisor application to manage all collocated VMs on the server. The hypervisor also supports migration of VMs among different servers in the network. Fat-tree is a representative DC topology and hence, results from this topology should extend to other types of DC networks without loss of generality.

In order to model a typical DC server's capability, we have limited the CPU and memory resources for accommodating a certain number of VMs. For example, a server equipped with 16GB RAM and 8 cores can safely allow 8 VMs running concurrently if each VM occupies one core and 1GB RAM (the CPU and memory occupied by VMs can be varied). Throughout the simulation, we created 2320 VMs on the 250 servers. Each VM has average 10 random outgoing socket connections, which are CBR traffic with a randomly generated rate. We have considered practical bandwidth limitations such that the aggregate bandwidth required by all VMs in a host does not exceed the network capacity of its physical interface. Therefore, a VM migration is only possible when the target host has sufficient system resources and bandwidth, i.e., a feasible server as defined in Equation (4).

We have also implemented the policy scheme described in Section III. In all experiments, we have set 10% of flows to be policy-free, meaning that they are not subject to any of the existing network policies in place. For the other 90% of flows, they have to traverse a sequence of middleboxes as required by policies before being forwarded to their destination [4]. Specially, each policy-constrained flow is configured to traverse 1$\sim$3 middleboxes, including *Firewall*, *IPS* or *LB*.

To demonstrate the benefit of *PLAN*, we compare it with S-CORE [14], a similar but non policy-aware VM management scheme which has been shown to outperform other schemes, e.g., Remedy [17]. S-CORE is a live VM migration scheme that reduces the topology-wide communication cost through clustering VMs without considering any underlying network policies. A VM migration takes place so long as it yields a positive *utility*, the communication cost reduction outweighs the migration cost, and the target server has sufficient resources to accommodate the new VM. In addition, *PLAN* by default is used with the initial placement algorithm described in Section III-B. In contrast, S-CORE initially starts with a set of randomly allocated VMs. In order to offset such a bias, we have also simulated *PLAN* without using the initial placement algorithm (which is referred to as *PLAN* with Random Initial Placement or *PLAN-RIP* in the sequel).

Alongside the communication cost, we also consider the impact of policies on average route length and link utilization. Route length is defined as the number of hops for each flow, including the additional route for traversing middleboxes. Link utilization is calculated on each layer of links in the fat-tree topology, i.e., *Edge Layer* links interconnect servers and edge switches, *Aggr Layer* links interconnect edge and aggregation switches, and *Core Layer* links interconnect aggregation switches to core routers.

### B. Experimental Results

We first evaluate the performance of *PLAN*. Fig. 2 demonstrates some unique properties of *PLAN* in progress towards convergence in terms of communication cost improvement as well as number of migrations. Fig. 2a depicts the improvement of individual VM's communication cost after each migration through calculating the ratio of *utility* to the communication cost of that VM before migration. It can be observed that each migration can reduce communication cost by 39.06% on average for *PLAN* and 34.19% for *PLAN-RIP*, respectively. Nearly 60% of measured migrations can effectively reduce their communication cost by as much as 40%. Such improvements are more significant when *PLAN* is used without an initial placement scheme in which VMs are allocated randomly at initialization. Fig. 2b shows the number of migrations per VM as *PLAN* converges. In *PLAN*, as a result of initial placement, only 30% of VMs need to migrate only once to achieve stable state throughout the whole experimental run. In comparison, 60% of VMs in *PLAN-RIP* need to migrate once when it converges. Nevertheless, in both schemes (with and without initial placement), we observe that very few ($<$ 1%) VMs need to migrate twice and no VM needs to migrate three times or more. These results demonstrate that low-cost, low-overhead

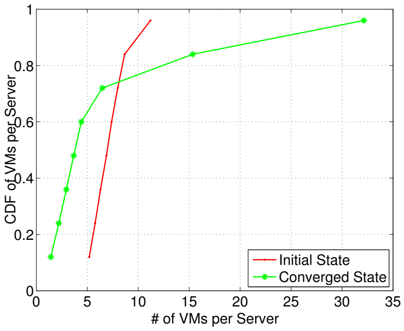


Figure 3: VMs clustering on servers at different states

initial placement can significantly reduce migration overhead in general.

We also study the transitioning state behavior of *PLAN* to reveal its intrinsic properties. Fig. 3 shows the snapshot of VM allocations at both the initial and the converged states of *PLAN*. Initially, before *PLAN* is running, VMs are nearly randomly distributed on servers, e.g., each server hosts 5∼12 VMs. After *PLAN* converges, plenty of VMs are clustered into several groups of servers, e.g., nearly 16% of servers host 56.55% of the total VMs. Moreover, an important property we can exploit is that 3.2% of servers are idle when *PLAN* converges and they can be safely shutdown to, e.g., save power.

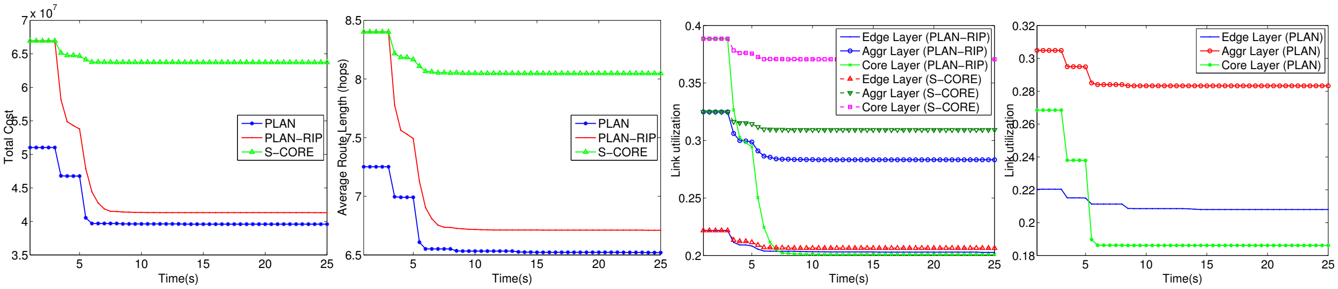
Next, we present performance results of *PLAN* when compared to *S-CORE*. Fig. 4 shows the overall communication cost reduction (measured in terms of number of bytes using network links), average end-to-end route length, as well as link utilization for all layers, for all the three schemes. Fig. 4a demonstrates that *PLAN* and *PLAN-RIP* can efficiently converge to a stable allocation. *PLAN* reduces the total communication cost by 22.42% while *PLAN-RIP* achieves an improvement of up to 38.27% which is a factor of nearly eight times better than *S-CORE* whose improvement is a mere 4.79%. The reason that *PLAN-RIP* has higher improvement is that the initial random VM placement offers more space for optimization than the already policy-aware initial placement of *PLAN*. However, it is evident that this potential is not exploited by *S-CORE*. More importantly, as shown in Fig. 4b, by migrating VMs, the average route length can be significantly reduced by as much as 20.12% and 10.08% for *PLAN-RIP* and *PLAN*, respectively, while *S-CORE* only improves it by 4.22%. Being able to reduce the average route length is an important feature of *PLAN* as it implies that flows can be generally completed faster and are less likely to create congestion in the network. Both Fig. 4a and 4b show that *PLAN* can effectively optimize the network-wide communication cost by localizing frequently communicated VMs as well as to reduce the length of the end-to-end path.

For the same reasons, Fig. 4c and 4d demonstrate that *PLAN* can mitigate link utilization at the core and aggregation layers by 30.55% and 7.01%, respectively. For *PLAN-RIP*, because it starts with random allocation of VMs which is non-optimal and inefficient compared to *PLAN* with initial placement, it can reduce link utilization across the core and aggregate layer links by 42.87% and 12.81%, respectively. The corresponding reduction for *S-CORE* is only 4.6% and 4.8%, respectively. On the other hand, utilization improvement on edge links is marginal for all three schemes, since they try to fully utilize lower-layer links where bisection bandwidth is maximum. Mitigation of link utilization at core and aggregation layers means that *PLAN* can effectively create extra topological capacity headroom for accommodating larger number VMs and services. Meanwhile, Fig. 4 also reveals that *PLAN*'s initial placement algorithm can greatly improve the communication cost, route length and link utilization. Then, the algorithm itself can continue to adaptively optimize network resource usage as it evolves after initial placement.

In order to examine *PLAN*'s adaptability to dynamic changes in policy configuration and traffic patterns, Fig. 5 presents the algorithm's performance results when policies are changed at 50s, 100s, and 150s, respectively, and after the algorithm had initially converged. Since *S-CORE* does not consider the underlying network policies, its performance is independent of policy configurations and is thus omitted. Throughout the experiments shown in Fig. 5, 10% of policies are removed at 50s, making the corresponding flows policy-free. This leads the DC to an non-optimized state, leaving room for further optimizing VMs allocations. Due to policy-awareness, *PLAN* can promptly adapt to new policy patterns, reducing the total communication cost, route length and link utilization to a great extent. The sudden drop at 50s is due to policy-free flows not needing to traverse through any middleboxes, hence causing the total communication cost to fall immediately. The same phenomenon can be observed when new policies are added at 100s and existing policies are modified at 150s. In particular, disabling some policies produces new policy-free flows so *PLAN* can localize their hosting VMs, greatly improving bandwidth. So, core-layer link utilization is promptly reduced when some policies are disabled at 50s. All the above results demonstrate that *PLAN* is highly adaptive to dynamism in policy configuration.

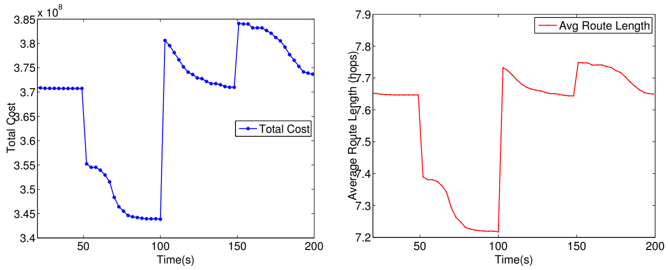
## V. Related Work

Network policy management research to date has either focused on devising new policy-based routing/switching mechanisms or leveraging Software-Defined Networking (SDN) to manage network policies and guarantee their correctness [11][23]. Joseph et al. [4] proposed *PLayer*, a policy-aware switching layer for DCs consisting of interconnected policy-aware switches (*pswitches*). Vyas et al. [3] proposed a middlebox architecture, CoMb, to actively con-



(a) Total communication cost (b) Average route length (c) Link utilization (d) Link utilization

Figure 4: Performance comparison of *PLAN* and *S-CORE* VM migration schemes



(a) Change of total communication cost (b) Change of route length

Figure 5: Performance of *PLAN* with dynamic policies.

solidate middlebox features and improve middlebox utilization, reducing the number of required middleboxes for operational environments.

Recent developments in SDN enable more flexible middlebox deployments over the network while still ensuring that specific subsets of traffic traverse the desired set of middleboxes [7]. Zafar et al. [10] proposed *SIMPLE*, a SDN-based policy enforcement scheme to steer DC traffic in accordance to policy requirements. Similarly, Fayazbakhsh et al. presented FlowTags [9] 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 [24][25][26][14][27]. Song et al.[26] proposed a multi-objective optimization model based on detailed analysis of the impact of CPU temperature, resource usage and power consumption in VM selection. Zhang et al.[27] present a Metadata based VM migration approach (Mvmotion) to reduce the amount of transferred data during migration by utilizing memory de-redundant technique between two physical hosts. However, none of these research efforts consider network policy in their design. The closest work to *PLAN* is a framework for Policy-Aware Application Cloud Embedding (PACE) [1] to support application-wide, in-network policies, and other realistic requirements such as bandwidth and reliability. However, PACE only considers one-off VM placement in conjunction with network policies and hence fails to deal with and further improve resource utilization in the face of dynamic workloads.

## VI. Conclusion

In multi-tenant Cloud Data Center (DC), network policies are popular used to provide secure and high performance services. In this paper, we have studied the optimization of DC network resource usage while adhering to a variety of policies governing the flows routed over the infrastructure. We have presented *PLAN*, a policy-aware VM management scheme that meets both efficient DC resource management and middleboxes traversal requirements. Through the definition of communication cost that incorporates policy, we have modeled an optimization problem of maximizing the utility (i.e., reducing total communication cost in DC) of VM migration, which is then shown to be NP-hard. We have subsequently derived a distributed heuristic approach to approximately reduce communication cost while preserving policy guarantees. Our results show that *PLAN* can reduce network-wide communication cost by 38% over diverse aggregate traffic loads and network policies, and is adaptive to changing policy and traffic dynamics.

## Acknowledgment

This work is partly supported by the National China 973 Project No. 2015CB352401, Shanghai Science Research Program Project No. 15JC1402400, "Fundamental Research Funds for the Central Universities" No. 21614330 and NSFC No. 61402200, Shanghai Scientific Innovation Act of STCSM(No.15JC1402400).

# References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] S. Sivakumar, G. Yingjie, and M. Shore, "A framework and problem statement for flow-associated middlebox state migration," 2012.

[6] L. Cui, F. P. Tso, D. P. Pezaros, W. Jia, and W. Zhao, "Policy-aware virtual machine management in data center networks," in *IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2015.

[7] 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.

[8] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn," *Queue*, vol. 11, no. 12, p. 20, 2013.

[9] 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.

[10] 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.

[11] A. Gember, 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.

[12] 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.

[13] 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.

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

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

[16] 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.

[17] 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.

[18] D. G. Cattrysse and L. N. Van Wassenhove, "A survey of algorithms for the generalized assignment problem," *European Journal of Operational Research*, vol. 60, no. 3, pp. 260–272, 1992.

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

[20] R. Cohen, L. Katzir, and D. Raz, "An efficient approximation for the generalized assignment problem," *Information Processing Letters*, vol. 100, no. 4, pp. 162–166, 2006.

[21] H. Ramalhinho and D. Serra, "Adaptive search heuristics for the generalized assignment problem," *Mathware & soft computing*, vol. 9, no. 3, pp. 209–234, 2008.

[22] "NS-3." [Online]. Available: http://www.nsnam.org

[23] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, "Extending sdn to handle dynamic middlebox actions via flowtags," *Presented as part of the Open Networking Summit*, vol. 2014, 2014.

[24] M. Wang, X. Meng, and L. Zhang, "Consolidating virtual machines with dynamic bandwidth demand in data centers," in *2011 Proceedings IEEE INFOCOM*. IEEE, 2011, pp. 71–75.

[25] 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.

[26] 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.

[27] Z. Zhang, L. Xiao, M. Zhu, and L. Ruan, "Mvmotion: a metadata based virtual machine migration in cloud," *Cluster Computing*, pp. 1–12, 2013.