PLAN: Joint Policy- and Network-Aware VM Management for Cloud Data Centers

Lin Cui, *Member, IEEE*, Fung Po Tso, Dimitrios P. Pezaros, *Senior Member, IEEE*, Weijia Jia, *Senior Member, IEEE*, and Wei Zhao, *Fellow, IEEE*

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.

Index Terms—Data centers, Policy, Virtual Machine, Migration.

1 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 totalling thousands to millions of dollars [2] [3].

- *Lin Cui is with Department of Computer Science, Jinan University, Guangzhou, China.
E-mail: tcuilin@jnu.edu.cn*
- *Fung Po Tso is with Department of Computer Science, Liverpool John Moores University, L3 3AF, UK.
E-mail: p.tso@ljmu.ac.uk.*
- *Dimitrios P. Pezaros is with School of Computing Science, University of Glasgow, G12 8QQ, UK.
E-mail: dimitrios.pezaros@glasgow.ac.uk.*
- *Weijia Jia is with Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai China.
E-mail: jia-wj@cs.sjtu.edu.cn.*
- *Wei Zhao is with Department of Computer and Information Science, University of Macau, Macau SAR, China.
E-mail: weizhao@umac.mo.*

Network policies demand that traffic 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 [3]. 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].

In order to combat the complexity of middleboxes management, a body of research works have been proposed to dynamically manage network policies. These works can be broadly classified into the two categories. *Virtualization and Consolidation*: Software-centric middlebox applications, including network function virtualization (NFV) [5], have been proposed to separate policy from reachability (i.e., virtualization) [3] [4] and middlebox functions can be consolidated [3] dynamically. *SDN-based policy enforcement*: Software-Defined Networking (SDN) [6] has enabled a new paradigm for enforcing middlebox policies [7]. SDN abstracts a logically centralized global view of the network and can be exploited to programmatically ensure correctness of middlebox traversal [8] [9] [10].

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 remains 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 [11]. Clearly, the 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 leads to violations of policy requirements. It has been demonstrated that, in PACE [1], VM placements in Cloud DC without considering network policies may lead to up to 91% policy violations.

It is common in DCs that a multi-tier 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 the shortest paths due to middlebox traversals (see Fig. 1), incurring redundant cross traffic between switches. Therefore, when deciding where to migrate any one of these VMs, not only dependency of VMs but also the locations of these middleboxes have to be taken into consideration. Failing to do so will not only lead to sub-optimal performance due to much longer middlebox traversal paths, but will also cause service disruption and unreachability as a result of being unable to follow a predefined sequence of middlebox traversal rules.

Using the SDN+NFV paradigm described above, such as OpenNF [8] and FlowTags [9], may be able to implement the correct sequence of traversal even when VMs migrate, *but they neither ensure shortest traversal paths nor reduce network communication cost*. S-CORE [12] has demonstrated that the shortest path between any two communicating VMs gives minimal communication cost in data center network environments. PACE [1] 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 comparison, we focus on dynamic VM management, and our initial effort has shown that policy-aware dynamic VM consolidation can remarkably improve network utilization [13].

In this paper, we explore the joint policy-aware and network-aware VMs migration problem, and present an efficient PoLicy-Aware and Network-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 [13].

In short, 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; and
3) A real-life implementation of *PLAN*[1] and 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 2 describes the model of policy-aware VM management (*PLAN*), and defines the communication cost and utility for VM migration. An efficient distributed algorithm is proposed in Section 3, followed by presentation of our python-based testbed implementation in Section 4. Section 5 evaluates the performance of *PLAN*. Section 6 outlines related work on VM migration and policy implementations. Finally, Section 7 concludes the paper and discusses future works.

2 Problem Modeling

2.1 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.

2.1.1 Topology and Application

Fig. 1 depicts a typical Fat-tree DC network topology [14] 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.

2.1.2 Policy Configurations

Polices are identified through a 5-tuple and a list of middleboxes (A more formal definition is given in Section 2.2). The following policies are configured through the *Policy Controller* to govern traffic related to the web application in this example:

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

Policy p_1: The Internet client first sends an 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

1. The source code for our implementation is available at: https://github.com/posco/policy-aware-plan

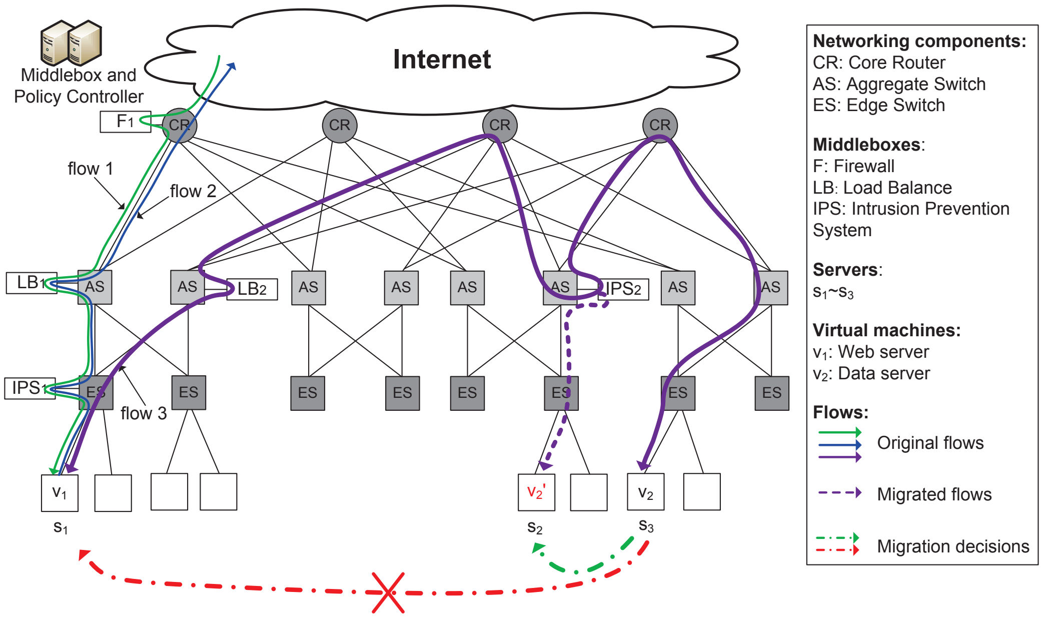


Fig. 1: Flows traversing different sequences of middleboxes in DC networks. Without policy-awareness, v_2 will be migrated to s_1, resulting in longer paths for flow 1 and wasting network resources.

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 obtaining the required data from the data server, the web server will send computed results to the 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.

2.1.3 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 routed only over the edge layer (which is not oversubscribed) [15] [12]. As a result, VMs as well as middleboxes, which communicate and 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 communication cost.

2.2 Communication Cost with Policies

For simplicity, in the following, we use a multi-tier DC network which is typically structured under a multi-root tree topology (canonical [16] or fat-tree [14]) as example. However, with appropriate link weight assignments (see Section 4.2.3), the model and solution described in this paper can be extended to any other data center topologies.

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. Here, $\lambda_k(v_i, v_j)$ can be the average pairwise traffic rates over a certain temporal interval, which can be set suitably long to match the dynamism of the environment while not responding to instantaneous traffic bursts. This is reasonable that many existing DC measurement studies suggest that DC traffic exhibits fixed-set hotspots that change slowly over time [17] [18] [19].

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

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

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, which will affect both $C_k(v_i, p_k^{in})$ and $C_k(p_k^{out}, v_j)$ above. The value of $C_k(p_k^{in}, p_k^{out})$ in (2) remain unchanged when migrating either v_i or v_j, and can be ignored as it makes no contribution to the minimization of the communication cost during VM migration.

For *policy-free* flows, which are not governed by any policies, their communication cost can be 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.

2.3 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:

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

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 [20], which is comprised of three phases: pre-copy phase, pre-copy termination phase and stop-and-copy phase.

According to [21], the amount of traffic generated during migration depends on the VMs image size M_s, its page dirty rate P_r, and the available bandwidth B_a. X_{min} and T_{max} are user setting for minimum required progress for each pre-copy cycle and the maximum time for the final stop-and-copy cycle respectively [21]. Let N_j denote the traffic on the jth pre-copy cycle. The first iteration will result in migration traffic equal to the entire memory size $N_0 = M_s$, and time M_s/B_a. During that time, $M_s \frac{P_r}{B_a}$ amount of memory becomes dirty. Then, the second iteration results in $N_2 = M_s \frac{P_r}{B_a}$ amount of traffic. It is easy to obtain that $N_j = M_s (\frac{P_r}{B_a})^{j-1}$ for $j \geq 1$. Thus if there are n *pre-copy* cycles and one final *stop-and-copy* cycle, the estimated migration cost, i.e., total traffic generated during migration, for VM v_i is [21]:

$$C_m(v_i) = M_s \cdot \frac{1 - (P_r/B_a)^{n+1}}{1 - (P_r/B_a)} \tag{6}$$

There are two conditions for stopping the pre-copy cycle corresponding to the minimum required progress for each pre-copy cycle and the maximum time for the final stop-copy cycle respectively [21]: (1) $M_s(\frac{P_r}{B_a})^n < T_{max} \cdot B_a$, and (2) $M_s(\frac{P_r}{B_a})^{n-1} - M_s(\frac{P_r}{B_a})^n < X_{min}$. Thus, we can derive that the number of pre-copy cycles $n = \min(\lceil \log_{P_r/B_a} \frac{T_{max} \cdot B_a}{M_s} \rceil, \lceil \log_{P_r/B_a} \frac{X_{min} \cdot P_r}{M_s \cdot (B_a - P_r)} \rceil)$.

Such migration overhead in (6) 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 $A(v_i) \rightarrow \hat{s}$, where $\hat{s} \in \mathcal{S}_i$, is defined as:

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

And the *total utility* $\mathcal{U}_{A \rightarrow \hat{A}}$ 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:*

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

PLAN can be treated as a restricted version of the Generalized Assignment Problem (GAP) [22]. However, the GAP is APX-hard to approximate [22]. 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) [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 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 $\mathcal{U}_{A_0 \rightarrow \hat{A}}$. 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 δ_i is a constant. Consider each VM to be an item with size R_i and profit $U(A(v_i) \rightarrow \hat{s}) = \mathcal{C}_i(A(v_i)) - \delta_i - C_m(v_i)$, 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. □

3 PLAN Algorithms

The *PLAN* problem is a restricted version of the Generalized Assignment Problem (GAP), which has been proven APX-hard to approximate [22]. We can use some existing centralized algorithms to approximately maximize the total gained *utility* by migration, e.g., [24], [25]. However, the computation times of those algorithms are unacceptable for DCs, especially considering the large scales of servers, VMs, switches and millions of traffic flows [12]. In this section, we design a decentralized heuristic scheme to perform policy-aware VMs migration.

3.1 Policy-Aware Migration Algorithms

Server hypervisors (or SDN controller, if used, see Section 4) 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, $U(A(v_i) \rightarrow \hat{s}) = 0$. 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. *PLAN-VM* operations will be suppressed for T_m time period if v_i is migrated to a new server, avoiding too frequent migration or oscillation among servers. The value of T_m depends on the traffic patterns, e.g., smaller value for a DC with more stable traffic. τ is a random positive value to avoid synchronization of migrations for different VMs. Its value range can be determined by the scale of VMs, e.g., smaller range for few number of VMs. *PLAN-Server* is designed for hypervisors on servers which can accept requests from VMs based on the residual resources of the corresponding server and prepares 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 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

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
```

utility by calling the function *Decision-Migration()*, i.e., 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. The function *Decision-Migration()* tries to find a set of servers with maximum migration *utility*, i.e., line 17. If the current server, which hosts v_i, is not included in this set, the one with most residual resources will be chosen as a candidate server for migration, i.e., line 18 $\sim$ 20. Otherwise, no migration action will take place. This will avoid oscillation of VM migration among the same set of servers. 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 requests 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, i.e., line 5 $\sim$ 8. Otherwise, it will reject the migration request of v_i, i.e., line 16. For accepted VMs, s_j will prepare for the incoming migration by provisionally reserve resources for them, i.e., line 14.

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 varied and always larger than zero during each migration, 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. Furthermore, the amount of decreased cost (i.e., utilities) is variable for each migration. So, the two algorithms will finally converge after a finite number of steps. □

Suppose the total number of VMs is m and total number of servers is n. In the worst case, each VM needs to send a request to all the other servers and is rejected by all of them except the last one. Thus, the total message complexity of *PLAN* is $O(mn)$.

In Fig. 2, we use the same scenario in Fig. 1 as an example to show operations of PLAN algorithms. We consider the same policies configuration as described in Section 2.1.2. Link weights are assigned to grow exponentially for each layer, i.e., e^0 for edge layer links, e^1 for aggregation layer links, e^2 for core layer links respectively. Fig. 2a lists all example flows with src/dst, rates and applied policies. Fig. 2b is the initial placement. Numbers within brackets are total resources (or requirements) of servers (or VMs). Initially, v_1 and v_2 are hosted on s_1 and s_3 respectively. Based on the flow rates and route path, the total communication cost is 10692.15. Assume v_1 first called *PLAN-VM* in time 1, and find its current hosted server s_1 is the best choice. v_1 exit *PLAN-VM* on line 22. At time 2, v_2 called *PLAN-VM*. Both s_1 and s_2 can accept v_2, but migrating to v_2 receives more utility (line 17). So, v_2 sent a request to s_2 which would accept the request since s_2 has enough space to host v_2 (line 8 in *PLAN-Server*). Finally, v_2 is migrated to s_2 and the total communication cost becomes 6997.62.

Flow: src→dst(rate)	Policies
$f_1 : u \to v_1(30kbps)$	p_1, p_2
$f_2 : v_1 \to v_2(50kbps)$	p_3
$f_3 : v_2 \to v_1(200kbps)$	p_4
$f_4 : v_1 \to u(100kbps)$	p_5, p_6

(a) Example flows

Server(res.)	VMs(req.)
$s_1(20)$:	$v_1(10)$
$s_2(15)$:	$-$
$s_3(28)$:	$v_2(8)$

(b) Initial placement (total cost = 10692.15)

Server(res.)	VMs(req.)
$s_1(20)$:	$v_1(10)$
$s_2(15)$:	$-$
$s_3(28)$:	$v_2(8)$

(c) Time 1 (total cost = 10692.15)

Server(res.)	VMs(req.)
$s_1(20)$:	$v_1(10)$
$s_2(15)$:	$v_2(8)$
$s_3(28)$:	$-$

(d) Time 2 (total cost = 6997.62)

Fig. 2: Example of *PLAN* Algorithms

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

4 Implementation

In this section, we discuss a real-world implementation of the *PLAN* scheme, highlighting the rationale as well as the operational and design details of the individual components.

The foremost principle for the implementation of *PLAN* is to be efficient and deployable in production DC - *should we adopt fully or partly decentralized implementation?* Although a fully decentralized *PLAN* implementation is possible, it requires substantial modification to middleware for it to be able to collect relevant VM and flow statistics. Given most of the required information such as temporal network topology, per-flow and per-port based traffic statistics are readily available in SDN paradigm, we have, hence, adopted a partially decentralized approach, i.e. computing communication cost –most computationally expensive part as defined in Equation (7) – in the hypervisors and monitoring flow statistics using a centralized SDN-based orchestration in [26].

4.1 Decentralized Modules

4.1.1 VM vs Hypervisor

While conceptually, *PLAN* relies on VMs to make their own migration decisions, in practice this is unsuitable because the hosts being virtualized should not be aware that they are in fact running within a virtualized environment preventing VMs from communicating directly with the underlying hypervisor. Since migration is a facility provided by the hypervisor, we have decided to implement our solution within Dom0, a privileged domain [20], of the Xen [27].

4.1.2 Distributed Cost Computation

Ubuntu 14.04 was used for Dom0 and the Python-based *xm* was used as the management interface. Open vSwitch [28] is enabled in Xen hypervisor as it only provides more flexible retrieval of local traffic flow tables, but allows Dom0 to interact with a SDN controller through OpenFlow protocol. Therefore, in order to compute the communication defined in Equation (7), this module needs to query SDN controller for the pair-wise flow statistics, λ, and the link weight, c, along the network path. Upon receiving requests, the SDN controller will return a JSON object containing all necessary information.

4.1.3 Server Resource Measurement

Similar to the distributed cost computation described above, Dom0's privilege can be exploited to measure residual server resources such as CPU utilization as well as available memory size. The measurement should be reported to and aggregated in the controller as part of a JSON object being exchanged between Open vSwitch and the controller.

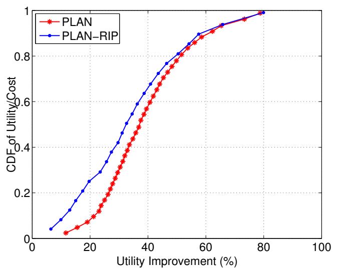
4.1.4 Flow Monitoring

In order to keep track of communicating VMs, flow history gathering is required. However, Open vSwitch only maintains flows for as long as they are active and discards any inactive flows after 5 seconds, hindering the accumulation of any long-term history. To overcome this limitation, we have extended the flow table for storing flow-level statistics. For the purposes of *PLAN*, the flow table must support the following operations: *Fast addition of new flows*; *Retrieval of a subset of flows, by IP address*; *Access to the number of bytes transmitted per flow*; *Access to flow duration, for calculation of throughput*. The flow table will be periodically updated through polling Open vSwitch for datapath statistics allowing for the storage of flows for as long as it is required. Flows will be stored from when they start up till a migration decision is made for a VM.

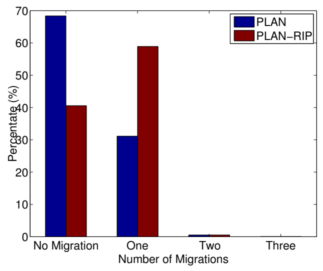
4.2 Centralized Modules

4.2.1 Policy Implementation

A centralized Policy Controller that stores and disseminates policies to the corresponding switches/routers, and accepts queries from them is a common deployment model in both non SDN [4] and SDN-based implementation [9]. We adopted SDN-based FlowTags [9] architecture to ensure correct sequence of middlebox traversals regardless of the location of VMs. In the following discussion we assume that, as a result of FlowTags, network policies are strictly followed no matter wherever VMs are migrated to and whenever new flows are emitted to the DC network.



(a) CDF: ratio of utility to communication cost



(b) Number of migrations before convergence

Fig. 3: Performance of PLAN

4.2.2 Topology Discovery

The knowledge of the network topology is crucial for *PLAN* to assign link-cost each individual link. By utilising the OpenFlow Discovery Protocol (OFDP) and Link Layer Discovery Protocol (LLDP) we can easily construct the network topology. Through the *SwitchEnter* and *SwitchLeave* events invoked by OpenFlow enabled switches, active switches within the topology can be discovered. Similarly, *LinkAdd* and *LinkDelete* events are triggered on the addition and removal of network links, allowing us to keep track of interconnected switches and physical ports.

4.2.3 Link Weights

In order to reflect the higher communication cost for using links in the higher layers of the topology, increasing weights with respect to the topology are adopted. In our implementation, link weights are set to grow exponentially for each layer, e.g., e^0 for edge layer links, e^1 for aggregation layer links, e^2 for core layer links respectively. However, in the general case, link weight assignment can be based on DC operator's policy to reflect diverse metrics, such as energy consumption, performance, fault tolerance, and so on.

4.2.4 Flow Statistics

To collected pairwise utilization, we used OpenFlow's *Statistic Request* messages to periodically collect the number of packets and bytes processed by the flow entry since the flow was installed. Both edge switches at the source and destination have similar flow entries for a particular VM-to-VM communication, therefore during collection of flow statistics, it is important to collect the metrics from the same switch for a particular VM-to-VM flow. In the current implementation, the first time a new flow is discovered from the flow statistics, the Data Path ID (DPID) of the switch is stored and subsequent measurements must originate from the same switch or will otherwise be discarded.

5 Evaluation

5.1 Simulation Setup

We have implemented *PLAN* in ns-3 [29] 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. 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 of 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 3. 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]. 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 [12], a similar but non policy-aware VM management scheme which has been shown to outperform other schemes, e.g., Remedy [21]. 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 3.2. 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.

5.2 Simulation Results

We first evaluate the performance of *PLAN*. Fig. 3 demonstrates some unique properties of *PLAN* in progress towards

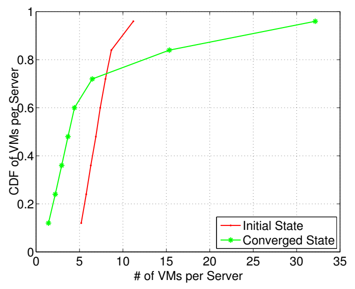


Fig. 4: VMs clustering on servers at different states

convergence in terms of communication cost improvement as well as number of migrations. Fig. 3a 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. 3b 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 a 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 initial placement can significantly reduce migration overhead in general.

We also study the transitioning state behavior of *PLAN* to reveal its intrinsic properties. Fig. 4 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.

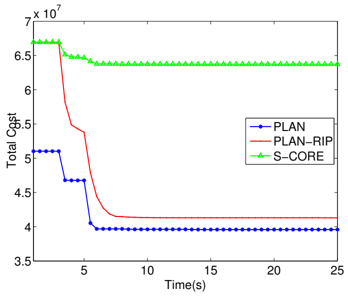
Next, we present performance results of *PLAN* when compared to *S-CORE*. Fig. 5 shows the overall communication cost reduction (measured in terms of number of bytes using network links), average end-to-end route length, as well as link utilization for all layers, for all the three schemes. Fig. 5a demonstrates 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. 5b, 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. 5a and 5b 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. 5c and 5d 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. 5 also reveals that *PLAN*'s initial placement algorithm can greatly improve the communication cost, route length and link utilization, and the algorithm itself can then continue to adaptively optimize network resource usage as it evolves.

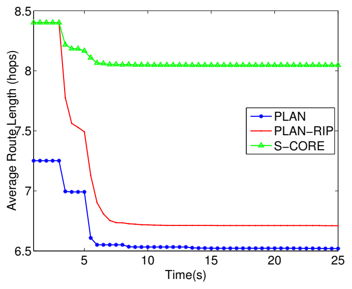
In order to examine *PLAN*'s adaptability to dynamic changes in policy configuration and traffic patterns, Fig. 6 presents the algorithm's performance results when policies are changed at 50s, 100s, and 150s, respectively, and after the algorithm had initially converged. Since *S-CORE* does not consider the underlying network policies, its performance is independent of policy configurations and is, thus, omitted. Throughout the experiments shown in Fig. 6, 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.

5.3 Testbed Results on VM Migration

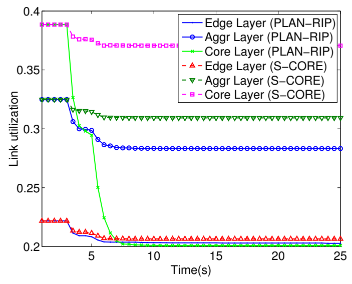
We have also set up a testbed environment to assess the algorithm's footprint and the performance of actual VM migrations. The servers used all have an Intel Core i5 3.2GHz



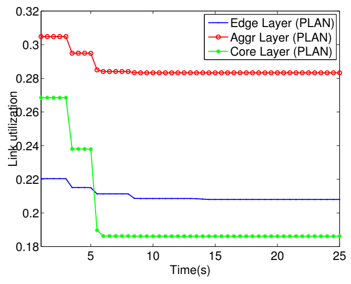
(a) Total communication cost



(b) Average route length

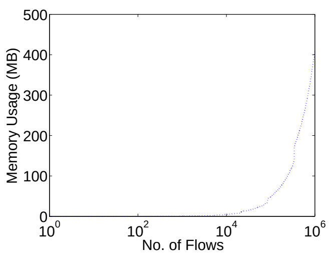


(c) Link utilization

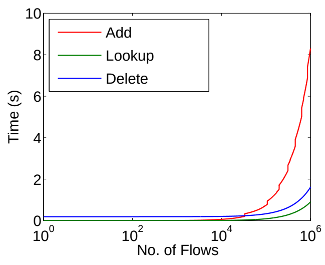


(d) Link utilization

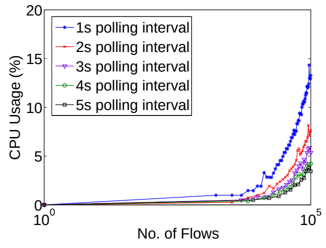
Fig. 5: Performance comparison of *PLAN* and *S-CORE* VM migration schemes



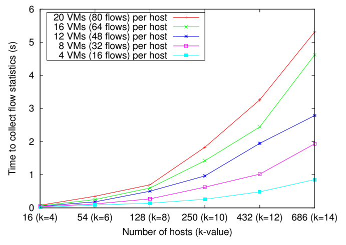
(a) Flow table memory usage



(b) Flow table operation times for up to 1 million unique flows

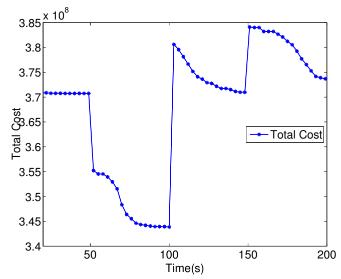


(c) CPU utilization when updating flow table at varying polling intervals

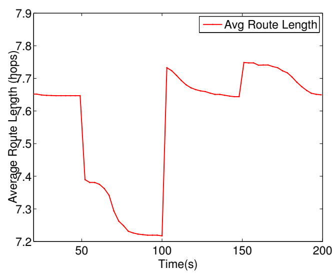


(d) Controller's flow collection time

Fig. 7: Testbed Results



(a) Change of total communication cost



(b) Change of route length

Fig. 6: Performance of *PLAN* with dynamic policies.

CPU with 8GB RAM running Xen hypervisor Ver. 4.2 with Ubuntu server 12.04 as Dom0. VMs are Ubuntu 12.04 with 1GB RAM allocated to each VM. We mainly stress-tested our decentralized module to see if it will be the performance bottleneck. We omitted testing our SDN controller since most functions are readily available in OpenFlow specification and hence will not create bottlenecks.

We first created 1 million flows with all source IP addresses being unique to examine flow tables (Section 4.1.4). This results in a new entry being created at the root of the flow table for each flow. As shown in Fig. 7a, the size of the flow table scales sub-linearly. With 10,000 and 100,000 entries, the flow table has a memory footprint of only 16MB and 91MB respectively. However, a number of studies have reported that the total number of concurrently active flows between VMs is much more contained: a busy server rarely talks to servers outside the rack [19] and the number of active concurrent flows going in and out of a machine is 10 [17]. With a more realistic scenario in which every virtual server concurrently sends or receives 10 flows, with 100 in the worst case, we anticipate that actual memory consumption of the flow table will be between 24.75 KB - 186.47 KB for a hypervisor hosting 16 VMs.

To understand the time taken to perform the different operations on the flow table, we have measured the time to add, lookup and delete flows, summing the times over the number of flows, for the same sets of flows. Fig. 7b shows the time to perform various flow table operations with differing numbers of flows in a single operation. Nevertheless, addition, lookup and deletion operations will not need more than 100ms for a realistic DC production workload of 100 concurrent flows.

Next, we measured the CPU usage for manipulating the flow table. This experiment was done by measuring the CPU usage for gradually adding and looking up the flow table with an interval between 1 second to 5 seconds, as shown in Fig. 7c. It is evident that the performance impact for adding up to 10,000 flows is negligible for any polling interval accounting for less than 5% CPU utilization. In the best case for 10,000 flows added or updated each time, CPU utilization was only around 1% at a polling rate of 5 seconds, while the worst case CPU utilization was 3.6% at a polling rate of 1 second. For a more realistic load of 1,000 flows, the best and worst cases are 0.002% and 0.01%, respectively.

Next, we test the centralized controllers performance on collecting flow statistics from the network. 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 as described in Sec. 4. The controller periodically pulls 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. 7d. For larger infrastructures, for example, when there are > 1000 hosts, multiple SDN controllers can be deployed to collect flows or sampling them using, e.g., DevoFlow [30].

Similar to other DC management schemes, PLAN will inevitably impose control overhead on the network, such as querying for flow statistics. An improperly designed control scheme may overwhelm the network with additional –control– load, but *how much overhead will PLAN create?* As we described in Section 4, fully distributed network is limited to a few control messages for triggering migration as described in Algorithm 1 and 2. The partial centralized approach will inevitably have higher overhead. With the best and worst scenario in a production DC (assuming 500 clusters) described above, the will be 5MB and 500MB respectively for every $T_m + \tau$ (used to set sparse intervals).

6 Related Works

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. Joseph et al. [4] proposed *PLayer*, a policy-aware switching layer for DCs consisting of inter-connected policy-aware switches (*pswitches*). Vyas et al. [3] proposed a middlebox architecture, CoMb, to actively consolidate middlebox features and improve middlebox utilization, reducing the number of required middleboxes for operational environments. Abujoda et al. [31] present MIDAS, an architecture for the coordination of middlebox discovery and selection across multiple network functions providers.

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]. Kazemian et al. [32] presented *NetPlumber*, a real-time policy-checking tool with sub-millisecond average run-time per rule update, and evaluated it on three production networks including Google's SDN, the Stanford backbone and Internet2. 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. Yaniv et al. [33] designed *EnforSDN*, a management approach that exploits SDN principles to decouple the policy resolution layer from the policy enforcement layer in network service appliances. Based on SDN, Chaithan et al. [34] tackled the problem of automatic, correct and fast composition of multiple independently specified network policies by developing a high-level Policy Graph Abstraction (PGA). Anat et al. [35] presented *OpenBox*, which decouples the control plane of middleboxes from their data plane, and unifies the data plane of multiple middlebox applications using entities called service instances. Shameli et al. [36] studied the problem of placing virtual security appliances within the data center in order to minimize network latency and computing costs while maintaining the required traversing order of virtual security appliances. 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 [12] [37] [38] [39] [40] [41]. 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.

Some other research works focus on scheduling of middleboxes. Duan et al. [42] studied latency behaviours of software middleboxes, and proposed Quokka which can schedule both traffic and middlebox positions to reduce transmission latencies of the network. Based on a Clos network design, Tu et al. [43] studied programmable middlebox that can distribute traffic more evenly to improve QoS. A SDN controller is used to collect global information to improve bandwidth utilization and reduce latency of middleboxes. Tamás et al. [44] focused on theoretical analysis of middleboxes placement problem. They presented a deterministic and greedy approximation algorithm for incremental deployment scenarios and showed that it can achieve near optimal performance in the offline scenario. However, all those works above ignore the influence of network polices on VM migration. They present detailed theoretical analysis on incremental middlebox deployment

Our another work in [45] proposed *Sync*, which also studies policy-awareness VMs management but focuses on a different problem. The *PLAN* proposed in this paper mainly focuses on VMs management with both policy-awareness and network-awareness. However, *Sync* studies the synergistic between VMs and middleboxes and focuses on the consolidation of both computing and networking resources in datacenters.

7 Conclusion and Future Work

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 communication cost) 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.

In this paper, we mainly focus on VM migration and hardware middleboxes. For virtualized middlebox or NFV, new software MB instances can be easily launched on commodity servers located within the network, and can be scheduled dynamically. By utilizing the capability of virtualized Network Function, the communication cost and efficiency of VM migration can be further improved, which is our following work currently.

Acknowledgments

This work is partially supported by Chinese National Research Fund (NSFC) Chinese National Research Fund (NSFC) Project No.: 61402200 and 61373125; NSFC Key Project No. 61532013; National China 973 Project No. 2015CB352401; Shanghai Scientific Innovation Act of STCSM No.15JC1402400 and 985 Project of Shanghai Jiao Tong University with No. WF220103001; Macau FDCT project 009/2010/A1 and 061/2011/A3; University of Macau project MYRG112; US National Science Foundation (NSF) under grant 0963979; the UK Engineering and Physical Sciences Research Council (EPSRC) grants EP/N033957/1, EP/L026015/1, and EP/L005255/1.

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] "Network functions virtualisation white paper #3." [Online]. Available: http://portal.etsi.org/NFV/NFV_White_Paper3.pdf

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

[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] 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.

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

[12] 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.

[13] 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.

[14] 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.

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

[17] 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 *Proc. ACM SIGCOMM'09*, 2009, pp. 51–62.

[18] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 2010, pp. 267–280.

[19] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proc. ACM SIGCOMM Internet Measurement Conference (IMC'09)*, 2009, pp. 202–208.

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.

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

[26] R. Cziva, D. Stapleton, F. P. Tso, and D. Pezaros, "SDN-based virtual machine management for cloud data centers," in *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, Oct 2014, pp. 388–394.

[27] "The xen project." [Online]. Available: http://www.xenproject.org/

[28] "Open vSwitch." [Online]. Available: http://openvswitch.org/

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

[30] 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.

[31] A. Abujoda and P. Papadimitriou, "MIDAS: Middlebox discovery and selection for on-path flow processing," *IEEE COMSNETS, Bangalore, India*, 2015.

[32] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, and S. Whyte, "Real time network policy checking using header space analysis," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2013.

[33] Y. Ben-Itzhak, K. Barabash, R. Cohen, A. Levin, and E. Raichstein, "EnforSDN: Network policies enforcement with SDN," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2015, pp. 80–88.

[34] C. Prakash, J. Lee, Y. Turner, J.-M. Kang, A. Akella, S. Banerjee, C. Clark, Y. Ma, P. Sharma, and Y. Zhang, "PGA: Using graphs to express and automatically reconcile network policies," in *ACM SIGCOMM Computer Communication Review*. ACM, 2015, pp. 29–42.

[35] A. Bremler-Barr, Y. Harchol, and D. Hay, "Openbox: Enabling innovation in middlebox applications," in *Proceedings of the 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization*. ACM, 2015, pp. 67–72.

[36] A. Shameli-Sendi, Y. Jarraya, M. Fekih-Ahmed, M. Pourzandi, C. Talhi, and M. Cheriet, "Optimal placement of sequentially ordered virtual security appliances in the cloud," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2015, pp. 818–821.

[37] 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.

[38] 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.

[39] 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.

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

[41] M. H. Ferdaus, M. Murshed, R. N. Calheiros, and R. Buyya, "Network-aware virtual machine placement and migration in cloud data centers," *Emerging Research in Cloud Distributed Computing Systems*, pp. 42–61, 2015.

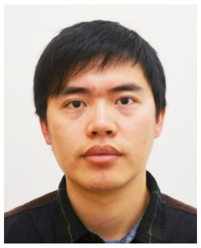
[42] P. Duan, Q. Li, Y. Jiang, and S. T. Xia, "Toward latency-aware dynamic middlebox scheduling," in *24th International Conference on Computer Communication and Networks (ICCCN)*, Aug 2015, pp. 1–8.

[43] R. Tu, X. Wang, J. Zhao, Y. Yang, L. Shi, and T. Wolf, "Design of a load-balancing middlebox based on sdn for data centers," in *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April 2015, pp. 480–485.

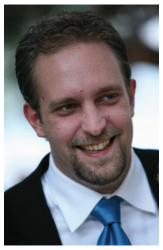
[44] T. Lukovszki, M. Rost, and S. Schmid, "It's a match!: Near-optimal and incremental middlebox deployment," *SIGCOMM Comput. Commun. Rev.*, vol. 46, no. 1, pp. 30–36, Jan. 2016.

[45] L. Cui, R. Cziva, F. P. Tso, and D. P. Pezaros, "Synergistic policy and virtual machine consolidation in cloud data centers," in *IEEE International Conference on Computer Communications (INFOCOM)*, April 2016, pp. 217–215.

Lin Cui is currently with the Department of Computer Science at Jinan University, Guangzhou, China. He received the Ph.D. degree from City University of Hong Kong in 2013. He has broad interests in networking systems, with focuses on the following topics: cloud data center resource management, data center networking, software defined networking (SDN), virtualization, distributed systems as well as wireless networking.



Fung Po Tso received his BEng, MPhil and PhD degrees from City University of Hong Kong in 2006, 2007 and 2011 respectively. He is currently Lecturer in the School of Computer Science at the Liverpool John Moores University (LJMU). Prior to joining LJMU, he worked as SICSA Next Generation Internet Fellow at the School of Computing Science, University of Glasgow. His research interests include: network measurement and optimisation, cloud data centre resource management, data centre networking, software defined networking (SDN), virtualisation, distributed systems as well as mobile computing and system.



Dimitrios P. Pezaros is Senior Lecturer at the Embedded, Networked and Distributed Systems (ENDS) Group in the School of Computing Science, University of Glasgow, which he joined in 2009. He is leading research on next generation network and service management mechanisms for converged and virtualised networked infrastructures, and is director of the Networked Systems Research Laboratory (netlab) at Glasgow. His research is being funded by the Engineering and Physical Sciences Research Council (EPSRC), the London Mathematical Society (LMS), the University of Glasgow, and the industry.

Weijia Jia is currently a full-time Zhiyuan Chair Professor at Shanghai Jiaotong University. He is leading currently several large projects on next-generation Internet of Things, environmental sensing, smart cities and cyberspace sensing and associations etc. He received BSc and MSc from Center South University, China in 82 and 84 and PhD from Polytechnic Faculty of Mons, Belgium in 1993 respectively. He worked in German National Research Center for Information Science (GMD) from 93 to 95 as a research fellow. From 95 to 13, he has worked in City University of Hong Kong as a full professor. He has published over 400 papers in various IEEE Transactions and prestige international conference proceedings.

Wei Zhao is currently the rector of the University of Macau, China. Before joining the University of Macau, he served as the dean of the School of Science, Rensselaer Polytechnic Institute. Between 2005 and 2007, he served as the director for the Division of Computer and Network Systems in the US National Science Foundation when he was on leave from Texas A&M University, where he served as senior associate vice president for research and professor of computer science. As an elected IEEE fellow, he has made significant contributions in distributed computing, real-time systems, computer networks, cyber security, and cyber-physical systems.