Virtual Machine Migration Planning in Software-Defined Networks

Huandong Wang*,Yong Li*, Ying Zhang†, Depeng Jin*
*Tsinghua National Laboratory for Information Science and Technology
Department of Electronic Engineering, Tsinghua University, Beijing 100084, China
†Ericsson Research
Email: liyong07@tsinghua.edu.cn

***Abstract*—Live migration is a key technique for virtual machine (VM) management in data center networks, which enables flexibility in resource optimization, fault tolerance, and load balancing. Despite its usefulness, the live migration still introduces performance degradations during the migration process. Thus, there has been continuous efforts in reducing the migration time in order to minimize the impact. From the network's perspective, the migration time is determined by the amount of data to be migrated and the available bandwidth used for such transfer. In this paper, we examine the problem of how to schedule the migrations and how to allocate network resources for migration when multiple VMs need to be migrated at the same time. We consider the problem in the Software-defined Network (SDN) context since it provides flexible control on routing.**

More specifically, we propose a method that computes the optimal migration sequence and network bandwidth used for each migration. We formulate this problem as a mixed integer programming, which is NP-hard. To make it computationally feasible for large scale data centers, we propose an approximation scheme via linear approximation plus fully polynomial time approximation, and obtain its theoretical performance bound. Through extensive simulations, we demonstrate that our fully polynomial time approximation (FPTA) algorithm has a good performance compared with the optimal solution and two state-of-the-art algorithms. That is, our proposed FPTA algorithm approaches to the optimal solution with less than 10% variation and much less computation time. Meanwhile, it reduces the total migration time and the service downtime by up to 40% and 20% compared with the state-of-the-art algorithms, respectively.

I. Introduction

The modern cloud computing platform has leveraged virtualization to achieve economical multiplexing benefit while achieving isolation and flexibility simultaneously. Separating the software from the underlying hardware, virtual machines (VMs) are used to host various cloud services [1]. VMs can share a common physical host as well as be migrated from one host to another. Live migration, $i.e.$, moving VMs from one physical machine to another without disrupting services, is the fundamental technique that enables flexible resource management in the virtualized data centers. By adjusting the locations of VMs dynamically, we can optimize various objective functions to provide better services, such as improving performance, minimizing failure impact and reducing energy consumption [2].

While there are continuous efforts on the optimal VM placements to reduce network traffic [3], [4], VM migration has received relatively less attention. We argue that careful planning of VM migration is needed to improve the system performance. Specifically, the migration process consumes not only CPU and memory resources at the source and the migrated target's physical machines [4], [5], but also the network bandwidth on the path from the source to the destination [4]. The amount of available network resource has a big impact on the total migration time, $e.g.$, it takes longer time to transfer the same size of VM image with less bandwidth. As a consequence, the prolonged migration time should influence the application performance. Moreover, when multiple VM migrations occur at the same time, we need an intelligent scheduler to determine which migration tasks to occur first or which ones can be done simultaneously, in order to minimize the total migration time.

More specifically, there can be complex interactions between different migration tasks. While some independent migrations can be performed in parallel, other migrations may share the same bottleneck link in their paths. In this case, performing them simultaneously leads to longer total migration time. In a big data center, hundreds of migration requests can take place in a few minutes [6], where the effect of the migration order becomes more significant. Therefore, we aim to design a migration plan to minimize the total migration time by determining the orders of multiple migration tasks, the paths taken by each task, and the transmission rate of each task.

There have been a number of works on VM migration in the literature. Work [9], [12] focused on minimizing migration cost by determining an optimal sequence of migration. However, their algorithms were designed under the model of one-by-one migration, and thus cannot perform migration in parallel simultaneously, leading to a bad performance in terms of the total migration time. Bari $et\ al.$ [7] also proposed a migration plan of optimizing the total migration time by determining the migration order. However, they assumed that the migration traffic of one VM only can be routed along one path in their plan. Compared with single-path routing, multipath routing is more flexible and can provide more residual bandwidth. Thus, we allow multiple VMs to be migrated simultaneously via multiple routing paths in our migration plan.

In this paper, we investigate the problem of how to reduce the total migration time in Software Defined Network (SDN) scenarios [8], [9]. We focus on SDN because with a centralized

978-1-4799-8381-0/15/$31.00 ©2015 IEEE

controller, it is easier to obtain the global view of the network, such as the topology, bandwidth utilization on each path, and other performance statistics. On the other hand, SDN provides a flexible way to install forwarding rules so that we can provide multipath forwarding between the migration source and destination. In SDN, the forwarding rules can be installed dynamically and we can split the traffic on any path arbitrarily. We allow multiple VMs to be migrated simultaneously via multiple routing paths. The objective of this paper is to develop a scheme that is able to optimize the total migration time by determining their migration orders and transmission rates. Our contribution is threefold, and is summarized as follows:

- We formulate the problem of VM migration from the network's perspective, which aims to reduce the total migration time by maximizing effective transmission rate in the network, which is much easier to solve than directly minimizing the total migration time. Specifically, we formulate it as a mixed integer programming (MIP) problem, which is NP-hard.
- We propose an approximation scheme via linear approximation plus fully polynomial time approximation, termed as FPTA algorithm, to solve the formulated problem in a scalable way. Moreover, we obtain its theoretical performance bound.
- By extensive simulations, we demonstrate that our proposed FPTA algorithm achieves good performance in terms of reducing total migration time, which reduces the total migration time by up to 40% and shorten the service downtime by up to 20% compared with the state-of-the-art algorithms.

The rest of the paper is organized as follows. In Section II, we give a high-level overview of our system, and formulate the problem of maximizing effective transmission rate in the network. In Section III, we propose an approximation scheme composed of a linear approximation and a fully polynomial time approximation to solve the problem. Further, we provide its performance bound. In Section IV, we evaluate the performance of our solution through extensive simulations. After presenting related works in Section V, we draw our conclusion in Section VI.

II. System Model and Problem Formulation

A. System Overview

We first provide a high-level system overview in this section. As shown in Fig. 1, in such a network, all networking resources are under the control of the SDN controller, while all computing resources are under the control of some cloud management system, such as OpenStack. Our VM migration plan runs at the Coordinator and it is carried out via the OpenStack and SDN controller.

More specifically, devices in the network, switches or routers, implement forwarding according to their obtained forwarding tables and do some traffic measurement. The SDN controller uses a standardized protocol, OpenFlow, to communicate with these network devices, and gather link-state information measured by them. Meanwhile, the SDN

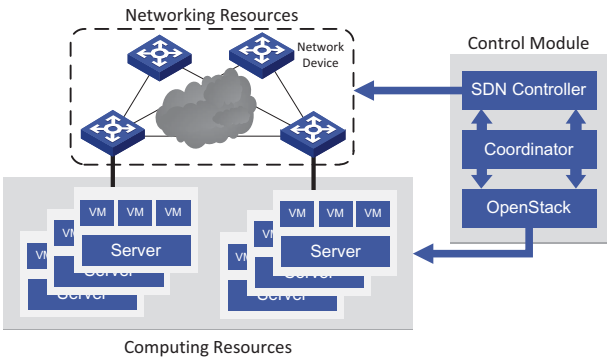


Fig. 1. System Overview

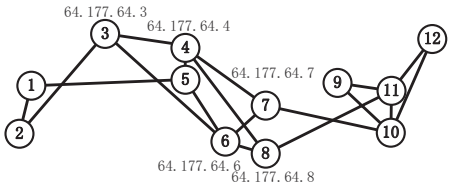
controller is responsible for computing the forwarding tables for all devices. On the other hand, the cloud controller, OpenStack, is responsible for managing all computing and storage resources. It keeps all the necessary information about virtual machines and physical hosts, such as the memory size of the virtual machine, the residual CPU resource of the physical host. Meanwhile, all computing nodes periodically report their up-to-date information to it. Besides, OpenStack also provides general resource management functions such as placing virtual machines, allocating storage, etc.

The processes of VM migration are described as follows. Firstly, migration requests of applications are sent to the Coordinator. Based on the data collected from the OpenStack and SDN controller, the VM migration plan outputs a sequence of the VMs to be migrated with their corresponding bandwidths. After reconfiguring the network and providing a bandwidth guarantee by SDN controller, the VM migration plan is carried out at the corresponding time by OpenStack. By this way, it realizes the control and management of VM migrations.

To compute the migration sequence of VMs, we need network topology and traffic matrix of the data center. Besides, memory sizes and page dirty rates of VMs, and residual physical resources such as CPU and memory are also needed. Most of them can be obtained directly from the SDN controller or OpenStack, but measurements of page dirty rate and traffic matrix need special functions of the platform. We next present the approach to measure them in details:

Page Dirty Rate Measurement: We utilize a mechanism called shadow page tables provided by Xen [1] to track dirtying statistics on all pages [2]. All page-table entries (PTEs) are initially read-only mappings in the shadow tables. Modifying a page of memory would result a page fault and then it is trapped by Xen. If write access is permitted, appropriate bit in the VMs dirty bitmap is set to 1. Then by counting the dirty pages in an appropriate period, we obtain the page dirty rate.

Traffic Measurement: We assume SDN elements, switches and routers, can spilt traffic on multiple next hops correctly, and perform traffic measurements at the same time [10], [11]. To aid traffic measurements, an extra column in the forwarding table is used to record the node in the network that can reach



(a) Google's inter-datacenter WAN

Prefix	Node	Next Hop	Traffic
195.112/16	64.177.64.8	64.177.64.6	α
195.027/16	64.177.64.3	64.177.64.4	β
...	...	...	...

(b) Modified Forwarding Table

Fig. 2. Google's inter-datacenter WAN and the modified forwarding table for example.

the destination IP address as in work [10]. Take Fig. 2(a), which is the topology of the inter-datacenter WAN of google, as an example, where all nodes are SDN forwarding elements. For instance, we assume node 8 (IP address 64.177.64.8) is the node that can reach the subset 195.112/16, and the shortest path from node 7 to node 8 goes through node 6 (IP address 64.177.64.6). Then, the forwarding table of node 7 is shown in Fig. 2(b), where the first entry is corresponding to the longest matched prefix 195.112/16. When a packet with the longest matched prefix 195.112/16 is processed by node 7, α showed in the figure increases by the packet length. Thus, it tracks the number of bytes routed from node 7 to node 8 with the longest matched prefix 195.112/16. Using these data, the SDN controller easily obtains the traffic between arbitrary two nodes as well as residual capacity of each link.

B. Problem Overview

In the system, we assume there is no alternate network dedicated to VM migrations, because of the cost of its deployment, especially in large-scale infrastructures. Thus, only residual bandwidth can be used to migrate VMs. Then, our goal is to determining the VMs' migration orders and transmission rates that satisfy various constraints, such as capacity constraints for memory and links, to optimize the total migration time.

Now we give an example in Fig. 3. In this network, there are 2 switches (S_1 and S_2) and 4 physical machines (H_1 to H_4) hosting 4 VMs (V_1 to V_4). Assume the capacity of each link is 100MBps and memory size of each VM is 500MB. We want to migrate V_1 from H_1 to H_2, V_2 from H_2 to H_3, and V_4 from H_3 to H_4. The optimal plan of migration orders and transmission rates is that first migrate V_1 and V_4 simultaneously, respectively with paths $\{(H_1, S_1, H_2)\}$ and $\{(H_3,S_2,H_4)\}$ and the corresponding maximum bandwidths of 100MBps. Then migrate V_2 with paths $\{(H_2, S_1, H_3), (H_2, S_2, H_3)\}$ and the corresponding maximum bandwidth of 200MBps. It totally takes 7.5s to finish all the migrations. Then, take random migration orders for example, *i.e.*, first migrate V_1 and V_2 simultaneously, respectively with paths $\{(H_1, S_1, H_2)\}$ and $\{(H_2, S_2, H_3)\}$ and the corresponding maximum bandwidths of 100MBps. Then migrate V_4 with path $\{(H_3, S_2, H_4)\}$ and the corresponding maximum bandwidth of 100MBps. It totally takes 10s to finish all the migrations.

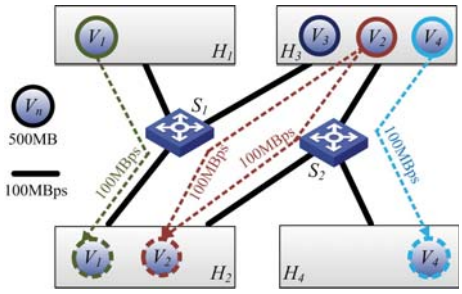


Fig. 3. An example of migration request and plan.

In this example, V_1 and V_4 can be migrated in parallel, while V_2 can be migrated with multipath. However, V_1 and V_2, V_4 and V_2 share same links in their paths, respectively. By determining a proper order, these migrations can be implemented making full use of the network resources. Thus, the total migration time is reduced by 25% in the example, illustrating the effect of the migration plan.

C. Mathematical Model for Live Migration

In this section, we present the mathematical model of live migration. We use M to represent the memory size of the virtual machine. Let R denote the page dirty rate during the migration and L denote the bandwidth allocated for the migration. Then, the process of the live migration is shown in Fig. 4. As we can observe, live migration copies memory in several rounds. Assume it proceeds in n rounds, and the data volume transmitted at each round is denoted by V_i $(0 \leq i \leq n)$. At the first round, all memory pages are copied to the target host, and we have $V_0 = M$. Then in each round, pages that have been modified in the previous round are copied to the target host. The transmitted data can be calculated as $V_i = R \cdot T_{i-1},\ i > 0$. Thus, the elapsed time at each round can be calculated as $T_i = V_i/L = R \cdot T_{i-1}/L = M \cdot R^i/L^{i+1}$.

Let λ denote the ratio of R to L, that is $\lambda = R/L$. Combining the above analysis, the total migration time can be represented as:

$$T_{mig} = \sum_{i=0}^{n} T_i = \frac{M}{L} \cdot \frac{1-\lambda^{n+1}}{1-\lambda}. \quad (1)$$

Let V_{thd} denote the threshold value of the remaining dirty memory that should be transferred at the last iteration. We can calculate the total rounds of the iteration by the inequality $V_n \leq V_{thd}$. Using the previous equations we obtain:

$$n = \left\lceil log_\lambda \frac{V_{thd}}{M} \right\rceil. \quad (2)$$

In this model, the downtime caused in the migration can be represented as $T_{down} = T_d + T_r$, where T_d is the time spent on transferring the remaining dirty pages, and T_r is the time spent on resuming the VM at the target host. For simplicity, we assume the size of remaining dirty pages is equal to V_{thd}.

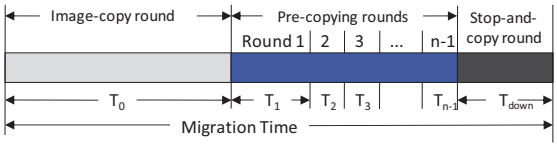


Fig. 4. Illustration of live migration performing pre-copy in iterative rounds.

D. Problem Formulation

The network is represented by a graph $G = (V, E)$, where V denotes the set of network nodes and E denotes the set of links. Let $c(e)$ denote the residual capacity of the link $e \in E$. Let a migration tuple (s_k, d_k, m_k, r_k) denote that a virtual machine should be migrated from the node s_k to the node d_k with the memory size m_k and the page dirty rate r_k. There are totally K migration tuples in the system. For the migration k, l_k represents the bandwidth allocated for it. Let P_k denote the set of paths between s_k and d_k. The flow in path p is represented by the variable $x(p)$. Besides, as different migrations are started at different times, we define binary variable X_k to indicate whether migration k has been started at the current time.

We first discuss the optimization objective. To obtain an expression of the total migration time is difficult in our model, because we allow multiple VMs to be migrated simultaneously. Thus, the total migration time cannot simply be represented as the sum of the migration time of each VM like work [9], [12], whose migration plans were designed under the model of one-by-one migration. Moreover, even though we obtain the expression of the total migration time, the optimization problem is still difficult and cannot be solved efficiently. For example, work [7] gives an expression of the total migration time by adopting a discrete time model. However, they did not solve the problem directly, instead, they proposed a heuristic algorithm independent with the formulation without any theoretical bound. Thus, we try to obtain the objective function reflecting the total migration time from other perspectives.

On the other hand, since the downtime of live migration is required to be unnoticeable by users, the number of the remaining dirty pages in the stop-and-copy round, $i.e.$ V_n, need to be small enough. According to the model provided in the last subsection, we have $V_n = M \cdot \lambda^n$. Thus, λ^n must be small enough. For example, if migrating a VM, whose memory size is 10GB, with the transmission rate of 1GBps, to reduce the downtime to 100ms, we must ensure $\lambda^n \leq 0.01$. Thus, by ignoring λ^n in the equation (1), we have:

$$T_{mig} \approx \frac{M}{L} \cdot \frac{1}{1-\lambda} = \frac{M}{L-R}. \tag{3}$$

We call the denominator as $net\ transmission\ rate$. From an overall viewpoint, the sum of memory sizes of VMs is reduced with the speed of $\sum_{k=1}^{K}(l_k - X_k r_k)$, which is the total net transmission rate in the network. In turn, the integration of the net transmission rate respect to time is the sum of memory sizes. By maximizing the total net transmission rate, we can reduce the total migration time efficiently. Thus, it is reasonable for us to convert the problem of reducing the migration time to maximizing the net transmission rate, which is expressed as $\sum_{k=1}^{K}(l_k - X_k r_k)$.

We now analyze constraints of the problem. A VM is allowed to be migrated with multipath in our model. Thus, we have a relationship between l_k and $x(p)$:

$$\sum_{p \in P_k} x(p) = l_k, \quad k = 1, ..., K.$$

Besides, the total flow along each link must not exceed its capacity. Thus, we have:

$$\sum_{p \in P_e} x(p) \leq c(e), \quad \forall e \in E.$$

For a migration that has not been started, there is no bandwidth allocated for it. Thus, we have constraints expressed as follow:

$$l_k \leq \beta \cdot X_k, \quad k = 1, ..., K,$$

where β is a constant large enough so that the maximum feasible bandwidth allocated for each migration cannot exceed it. Then, the problem of maximizing the net transmission rate can be formulated as follows:

$$\begin{aligned} \textbf{max}\ & \textstyle\sum_{k=1}^{K}(l_k - X_k r_k) \\ \textbf{s.t.}\ & \begin{cases} \sum_{p \in P_k} x(p) = l_k, & k = 1, ..., K \\ \sum_{p \in P_e} x(p) \leq c(e), & \forall e \in E \\ l_k \leq \beta \cdot X_k, & k = 1, ..., K \\ X_k \in \{0, 1\}, & k = 1, ..., K \\ x(p) \geq 0, & p \in P \end{cases} \end{aligned} \tag{4}$$

which is a mixed integer programming (MIP) problem.

When some new migration requests come or old migrations are finished, the input of the problem changes. Thus, we recalculate the programming under the new updated input. We notice that migrations that have been started cannot be stopped. Otherwise, these migrations must be back to square one because of the effect of the page dirty rate. Thus, when computing this problem next time, we add the following two constraints to it:

$$\begin{cases} X_k \geq X_k^0, & k = 1, ..., K \\ l_k \geq l_k^0, & k = 1, ..., K \end{cases} \tag{5}$$

where X_k^0 and l_k^0 are equal to the value of X_k and l_k in the last computing, respectively. It means a migration cannot be stopped and its bandwidth does not decrease.

By solving the programming, we obtain the VMs that should be migrated with their corresponding transmission rates, maximizing the total net transmission rate under the current condition. By dynamically determining the VMs to be migrated in tune with changing traffic conditions and migration requests, we keep the total net transmission rate maximized, which is able to significantly reduce the total migration time.

III. Approximation Algorithm

Solving the formulated MIP problem, we obtain a well-designed sequence of the VMs to be migrated with their

corresponding bandwidths. However, the MIP problem is NP-hard, and the time to find its solution is intolerable on large scale networks. For example, we implement the MIP problem using YALMIP – a language for formulating generic optimization problems [13], and utilize the GLPK to solve the formulation [14]. Then, finding the solution of a network with 12 nodes and 95 VMs to be migrated on a Quad-Core 3.2GHz machine takes at least an hour. Therefore, we need an approximation algorithm with much lower time complexity.

A. Approximation Scheme

1) Linear Approximation: Let us reconsider the formulated MIP problem (4). In this problem, only $X_k,\ k = 1, ..., K$, are integer variables. Besides, the coefficient of X_k in the objective function is r_k. In practical data center, r_k is usually much less than l_k, *i.e.*, the migration bandwidth of the VM. Thus, we ignore the part of $\sum_{k=1}^{K} X_k r_k$ in the objective function, and remove variables $X_k,\ k = 1, ..., K$. Then, we obtain a linear programming (LP) problem as follows:

$$\textbf{max}\ \sum_{k=1}^{K} l_k \quad \textbf{s.t.}\ \begin{cases} \sum_{p\in P_k} x(p) = l_k, & k = 1, ..., K \\ \sum_{p\in P_e} x(p) \le c(e), & \forall e \in E \\ x(p) \ge 0, & p \in P \end{cases} \tag{6}$$

We select the optimal solution l^* for (6) with most variables that are equal to zero as our approximate solution. Then we let N^* denote the number of variables that are not zero in our approximate solution l^*, and the corresponding binary decision variables X_k are then set to be 1, while the other binary decision variables are set to be 0. Then the final approximate solution is denoted by (l_k^*, X_k^*).

As for the primary problem with the additional constraints shown in (5), by a series of linear transformations, the problem is converted to a LP problem with the same form as (6) except for a constant in the objective function, which can be ignored. Thus we obtain a linear approximation for the primary MIP problem.

2) Fully Polynomial Time Approximation: The exact solution of the LP problem (6) still cannot be found in polynomial time, which means unacceptable computation time for large scale networks. Thus, we further propose an algorithm to obtain the solution in polynomial time at the cost of accuracy.

Actually, ignoring the background of our problem and removing the intermediate variable l_k, we can express the LP problem (6) as:

$$\textbf{max}\ \sum_{p\in P} x(p) \quad \textbf{s.t.}\ \begin{cases} \sum_{p\in P_e} x(p) \le c(e), & \forall e \in E \\ x(p) \ge 0, & p \in P \end{cases} \tag{7}$$

This is a maximum multicommodity flow problem, that is, finding a feasible solution for a multicommodity flow network that maximizes the total throughput.

Fleischer *et al.* [15] proposed a Fully Polynomial-time Approximation Scheme (FPTAS) algorithm independent of the number of commodities K for the maximum multicommodity flow problem. It can obtain a feasible solution whose objective function value is within $1+\epsilon$ factor of the optimal, and the computational complexity is at most a polynomial function of the network size and $1/\epsilon$.

Specifically, the FPTAS algorithm is a primal-dual algorithm. We denote $u(e)$ as the dual variables of this problem. For all $e \in E$, we call $u(e)$ as the length of link e. Then, we define $\text{dist}(p) = \sum_{e\in p} u(e)$ as the length of path p. This algorithm starts with initializing $u(e)$ to be δ for all $e \in E$ and $x(p)$ to be 0 for all $p \in P$. δ is a function of the desired accuracy level ϵ, which is set to be $(1+\epsilon)/((1+\epsilon)n)^{1/\epsilon}$ in the algorithm. The algorithm proceeds in phases, each of which is composed of K iterations. In the r_{th} phase, as long as there is some $p \in P_k$ for some k with $\text{dist}(p) < \min\{\delta(1+\epsilon)^r, 1\}$, we augment flow along p with the capacity of the minimum capacity edge in the path. The minimum capacity is denoted by c. Then, for each edge e on p, we update $u(e)$ by $u(e) = u(e)(1+\frac{\epsilon c}{c(e)})$. At the end of the r_{th} phase, we ensure every (s_j, d_j) pair is at least $\delta(1+\epsilon)^r$ or 1 apart. When the lengths of all paths belonging to P_k for all k are between 1 and $1+\epsilon$, we stop. Thus, the number of phases is at most $\lceil log_{1+\epsilon}\frac{1+\epsilon}{\delta}\rceil$. Then, according to theorem in [15], the flow obtained by scaling the final flow obtained in previous phases by $\log_{1+\epsilon}\frac{1+\epsilon}{\delta}$ is feasible. We modified the FPTAS algorithm by adding some post-processes to obtain the feasible (l_k, X_k) and $x(p)$ to the primal MIP problem, and the modified algorithm is given in more detail in Algorithm 1. The computational complexity of the post-processes is only a linear function of the number of the VMs

Algorithm 1: FPTA Algorithm.

Input: network $G(V, E)$, link capacities $c(e)$ for $\forall e \in E$, migration requests (s_j, d_j)
Output: Bandwidth l_k, binary decision variable X_k for each migration k, and the amount of flow $x(p)$ in path $p \in P$.

```
Initialize u(e) = δ ∀e ∈ E, x(p) = 0 ∀p ∈ P
for r = 1 to ⌈log_{1+ε}((1+ε)/δ)⌉ do
    for j = 1 to K do
        p ← shortest path in P_j
        while u(p) < min{1, δ(1+ε)^r} do
            c ← min_{e∈p} c(e)
            x(p) ← x(p) + c
            ∀e ∈ p, u(e) ← u(e)(1 + εc/c(e))
            p ← shortest path in P_j
for each p ∈ P do
    x(p) = x(p)/log_{1+ε}((1+ε)/δ)
for j = 1 to K do
    l_j = Σ_{p∈P_j} x(p)
    X_j = 0
    if l_j ≠ 0 then
        X_j = 1
Return (l_k, X_k) and x(p)
```

to be migrated. In addition, the computational complexity of the FPTAS algorithm is at most a polynomial function of the network size and $1/\epsilon$ [15]. Thus, the computational complexity of our approximation algorithm is also polynomial. and we obtain a fully polynomial time approximation (termed as FPTA) to the primal MIP problem.

B. Bound Analysis

To demonstrate the effectiveness of our proposed algorithm, we now analyze the bound of it. We first analyze the bound of the linear approximation compared with the primary MIP problem (4), then analyze the bound of the FPTA algorithm compared with the linear approximation (6). With these two bounds, we finally obtain the bound of the FPTA algorithm showing in Algorithm 1 compared with the primary MIP problem (4).

1) Bound of the Linear Approximation: We discuss the bound of the linear approximation compared with the primary MIP problem in normal data center network scenarios. Common topologies of data center networks, such as fat tree, usually provide full bisection bandwidth, which enables all hosts communicating with each other with full bandwidth at the same time. Thus, we can ignore the routing details, and only guarantee the traffic at each host not exceeds its maximum bandwidth. Then, the LP problem (6) becomes:

$$\textbf{max} \ \sum_{k=1}^{K} l_k \qquad \textbf{s.t.} \ \begin{cases} \sum_{s_k=i} l_k \le C_i^s, & i = 1, ..., H \\ \sum_{d_k=i} l_k \le C_i^d, & i = 1, ..., H \\ l_k \ge 0, & k = 1, ..., K \end{cases} \tag{8}$$

where C_i^s is the maximum amount of traffic that can be received at host i, while C_i^d is the maximum amount of traffic that can be sent at host i. Besides, there are H hosts in the data center. Then, we let L_0 be the minimum of C_i^s and C_i^d. That is, $\min\{C_1^s, ..., C_H^s\} \ge L_0$ and $\min\{C_1^d, ..., C_H^d\} \ge L_0$. Similarly, we let R_0 be the maximum of r_k. That is, $\max\{r_1, ..., r_K\} \le R_0$.

We now provide some supplement knowledge about linear programming. For a linear programming with standard form, which can be represented as:

$$\textbf{max} \ b^T x \qquad \textbf{s.t.} \ \begin{cases} Ax = c \\ x \ge 0 \end{cases} \tag{9}$$

where $x, b \in R^n$, $c \in R^m$, $A \in R^{m \times n}$ has full rank m, we have the following definitions and lemmas.

Definition 1 (Basic Solution) Given the set of m simultaneous linear equations in n unknowns of $Ax = c$ in (9), let B be any nonsingular $m \times m$ submatrix made up of columns of A. Then, if all $n - m$ components of x not associated with columns of B are set equal to zero, the solution to the resulting set of equations is said to be a basic solution to $Ax = c$ with respect to the basis B. The components of x associated with columns of B are called basic variables, that is, $Bx_B = c$ [16].

Definition 2 (Basic Feasible Solution) A vector x satisfying (9) is said to be feasible for these constraints. A feasible solution to the constraints (9) that is also basic is said to be a basic feasible solution [16].

Lemma 1 (Fundamental Theorem of LP) Given a linear program in standard form (9) where A is an $m \times n$ matrix of rank m. If there is a feasible solution, there is a basic feasible solution. If there is an optimal feasible solution, there is an optimal basic feasible solution [16].

These definitions and the lemma with its proof can be found in the textbook of linear programming [16]. With these preparations, we have the following lemma:

Lemma 2 There exists an optimal solution for (8), such that there are at least N^* equalities that hold in inequality constraints of (8).

Proof: Problem (8) can be represented in standard form as:

$$\textbf{max} \ b^T l \qquad \textbf{s.t.} \ \begin{cases} [A \ I] \begin{bmatrix} l \\ s \end{bmatrix} = c \\ l, s \ge 0 \end{cases} \tag{10}$$

where $c \in R^{2H}$, $l = (l_1, l_2, ..., l_K)^T$, $s = c - Al$, $b = (1, 1, ..., 1)^T \in R^K$, $I \in R^{K \times K}$ is the identity matrix of the order K, $A \in R^{2H \times K}$ is composed of 0 and 1, and each column of A has and only has two elements of 1. Besides, $[A \ I] \in R^{2H \times K + 2H}$ has full rank $2H$.

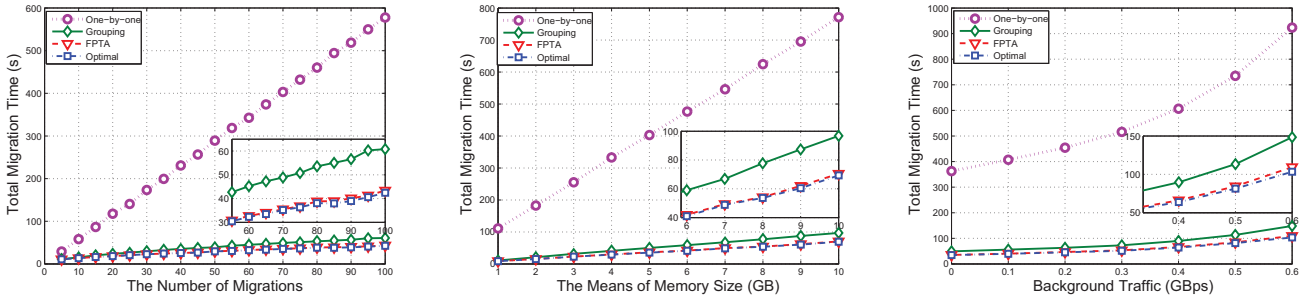
By Lemma 1, if the LP problem (10) has an optimal feasible solution, we can find an optimal basic feasible solution $(\hat{l}, \hat{s})$ for (10). By the definition of basic solution, the number of nonzero variables in $(\hat{l}, \hat{s})$ is less than $2H$. Meanwhile, By the definition of N^*, the number of nonzero variables in $\hat{l}$, which is represented by $\hat{N}$, is greater than N^*. Thus the number of nonzero variables in $\hat{s}$ is less than $2H - N^*$. Then there are at least N^* variables that are equal to zero in $\hat{s}$. Meanwhile, $\hat{s}_j = 0, \ j \in \{1, ..., 2H\}$ means the equality holds in the inequality constraint corresponding j_{th} row in A. Therefore, we have at least N^* equalities that hold in inequality constraints of (8).■

Theorem 1 Assume $R_0 = \eta L_0$. Let U be the optimal value of the primal MIP problem (4), and V be the optimal value of the LP problem (8). Then we have $V - N^* R_0 \ge (1 - \sigma)U$, where $\sigma = \frac{2\eta}{1 - 2\eta}$.

Proof: We first prove $V \ge \frac{1}{2} N^* L_0$. By lemma 2, we know that there exists an optimal solution of (8) such that there are at least N^* equalities that hold in inequality constraints of (8). We select the corresponding rows $a_1, ...a_{N^*}$ of A and corresponding elements $c_1, ...c_{N^*}$ of c. Then we have $a_i^T \hat{l} = c_i, \ i = 1, ..., N^*$. Because each column of A has and only has two elements of 1, elements of $\sum_{i=1}^{N^*} a_i$ are at most 2. Thus, we have $V = \sum_{k=1}^{K} \hat{l}_k \ge \frac{1}{2} \sum_{i=1}^{N^*} a_i^T \hat{l} = \frac{1}{2} \sum_{i=1}^{N^*} c_i \ge \frac{1}{2} N^* L_0$.

By definition of U and V, we have $U \le V$ and $V - N^* R_0 \le U$. Then we have $|U - (V - N^* R_0)| = U - V + N^* R_0 \le N^* R_0$. Besides, by the last paragraph, we have $U \ge V - N^* R_0 \ge \frac{1}{2} N^* L_0 - N^* R_0$. Thus, we have $\frac{|U - (V - N^* R_0)|}{U} = \frac{U - (V - N^* R_0)}{U} \le \frac{N^* R_0}{\frac{1}{2} N^* L_0 - N^* R_0} = \frac{2R_0}{L_0 - 2R_0} = \frac{2\eta}{1 - 2\eta} = \sigma$, *i.e.*, $V - N^* R_0 \ge (1 - \sigma)U$. ■

By the definitions of N^* and R_0, we have that the net



(a) (b) (c)

Fig. 5. Total migration time vs different parameters in one datacenter under the topology of PRV1.

transmission rate corresponding to the selected solution of (8) is at least $V - N^*R_0$. Thus, we obtain the bound of the linear approximation compared with the primary MIP problem.

2) Bound of the FPTA Algorithm: We next analyze the bound of the FPTA algorithm. According to theorem in [15], we have the following lemma:

Lemma 3 If p is selected in each iteration to be the shortest (s_i, d_i) path among all commodities, then for a final flow value $W = \sum_{p \in P} x(p)$ obtained from the FPTAS algorithm, we have $W \geq (1 - 2\epsilon)V$, where ϵ is the desired accuracy level.

Because the value of $x(p)$ is unchanged in our post-processes of Algorithm 1, W is also the final flow value of our proposed FPTA algorithm. Note that it is not the bound of the FPTA algorithm compared with the LP problem (6), because our objective function is the net transmission rate, while W is only the transmission rate of the solution of the FPTA algorithm. Besides, V is not the maximum net transmission rate as well. The bound of the FPTA algorithm is given in the following theorem:

Theorem 2 Let F be the net transmission rate corresponding to the solution of Algorithm 1. In the data center networks providing full bisection bandwidth, we have $F \geq (1 - 2\epsilon - \sigma)U$, where U is the optimal value of the primal MIP problem (4).

Proof: By the definitions of N^* and R_0, we have that the net transmission rate corresponding to the solution of the FPTA algorithm is at least $W - N^*R_0$, *i.e.*, $F \geq W - N^*R_0$. Thus we have $F \geq (1 - 2\epsilon)V - N^*R_0 = (1 - 2\epsilon)(V - N^*R_0) - 2\epsilon N^*R_0$. Meanwhile, by $U \geq \frac{1}{2}N^*L_0 - N^*R_0 \geq \frac{1}{2\eta}N^*R_0 - N^*R_0$, we have $N^*R_0 \leq \frac{2\eta}{1-2\eta}U = \sigma U$.

By Theorem 1, we have $F \geq (1 - 2\epsilon)(1 - \sigma)U - 2\epsilon\sigma U = (1-2\epsilon-\sigma)U$. Thus we obtain the bound of the FPTA algorithm compared with the primal MIP problem.■

IV. Performance Evaluation

A. Simulation System Set Up

With the increasing trend of owning multiple datacenter sites by a single company, migrating VMs across datacenters becomes a common scenario. Thus, to evaluate the performance of our proposed migration plan inside one datacenter and across datacenters, we select the following two topologies to implement our experiments: (1) The topology of a private enterprise data center located in Midwestern United States (PRV1 in [18]). (2) B4, Google's inter-datacenter WAN with 12 data centers interconnected with 19 links [17] (showing in Fig. 2(a)). In B4, each node represents a data center. Besides, the network provides massive bandwidth. However, to evaluate the performance of our proposed algorithm under relatively hard conditions, we assume the capacity of each link is only 1GBps. On the other hand, the capacities of links in RPV1 are set ranging from 1GB to 10GB according to [18]. The page dirty rate is set to 100MBps. Besides, V_{thd} and T_r are set to 100MB and 20ms, respectively. The memory sizes of VMs are also set ranging from 1GB to 10GB unless stated otherwise.

In our experiments, we evaluate the performance of our proposed FPTA algorithm compared with the optimal solution of the MIP problem (referred to as optimal algorithm) and two state-of-the-art algorithms. In the two state-of-the-art algorithms, one is the algorithm based on one-by-one migration scheme (referred to as one-by-one algorithm), which is proposed in [9], [12]. The other is the algorithm that migrates VMs by groups (referred to as grouping algorithm), just as the algorithm proposed in [7]. In this algorithm, VMs that can be migrated in parallel are divided into the same group, while VMs that share the same resources, such as the same link in their paths, are divided into different groups. Then VMs are migrated by groups according to their costs [7]. We further set the function of the cost as the weighted value of the total migration time and the number of VMs in each group.

B. Results and Analysis

1) Migration Time: In our first group of experiments, we compare the total migration time of our proposed FPTA algorithm with that of other algorithms introduced above, with the variation of different parameters, *i.e.*, the number of VMs to be migrated, the amount of background traffic, the average memory size of VMs, in PRV1 and B4, respectively. The results are shown in Fig. 5 and Fig. 6.

As we can observe from the Fig. 5, the performance of the one-by-one algorithm is much worse than that of the other three algorithms: when there are 100 VMs to be migrated, the total migration time it takes is about 10 times more than that of the other three algorithms, illustrating its inefficiency

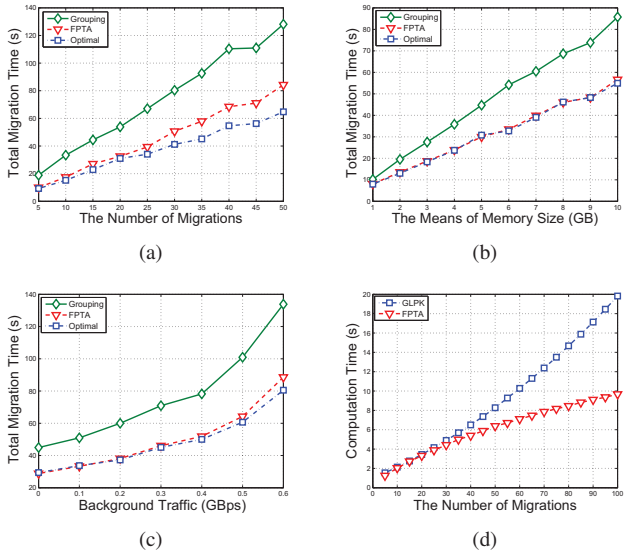


Fig. 6. Total migration time or computation time vs different parameters in inter-datacenter network under the topology of B4.

in reducing the total migration time. Since the performance gap between one-by-one algorithm and the other algorithms is huge, we do not show its performance in Fig. 6. Besides, from Fig. 6(d) we can observe that the computation time of our proposed FPTA algorithm is at most a polynomial function of the number of the migrations, much less than that of using GLPK to solve the LP problem (6).

As for the performance of the other three algorithms, their total migration time vs different parameters in data center networks and inter-datacenter WAN has a similar trend: the total migration time of FPTA algorithm is very close to that of the optimal algorithm, and much less than that of the grouping algorithm. Take Fig. 5(a) and Fig. 6(a) for example. In PRV1 (showing in Fig. 5(a)), total migration time of the FPTA algorithm and the optimal algorithm almost cannot be distinguished, while in B4 (showing in Fig. 6(a)) the gap is less than 15% relative to the optimal algorithm. Meanwhile, FPTA algorithm performs much better than the grouping algorithm: its migration time is reduced by 40% and 50% in comparison with the grouping algorithm in PRV1 and B4, respectively. Thus, the solution of our proposed FPTA algorithm approaches to the optimal solution and outperforms the state-of-the-art solutions.

2) Net Transmission Rate: To illustrate the effectiveness of maximizing the net transmission rate, we implement the second group of experiments in the scenario where there are 40 VMs to be migrated in B4. Net transmission rates of the FPTA algorithm and the grouping algorithm are evaluated, as functions of time. The result is shown in Fig. 7.

According to previous theoretic analysis, we know that the sum of memory sizes of VMs to be migrated is approximately equal to the integration of the net transmission rate with respect to time. In the experiments, the sum of memory sizes of the 40 VMs to be migrated are 203GB. Meanwhile, in Fig. 7,

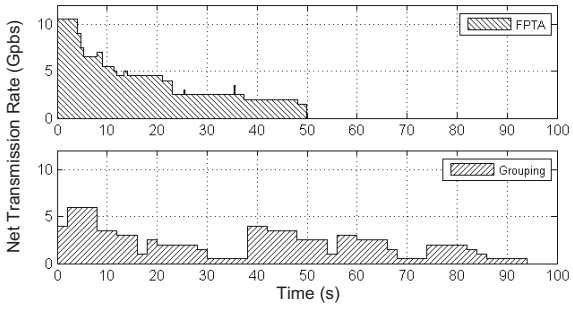


Fig. 7. The net transmission rates vs time of FPTA algorithm and grouping algorithm in inter-datacenter network under the topology of B4 with 40 VMs to be migrated.

the shadow areas of the FPTA and grouping algorithm, which can represent the integrations of the net transmission rates with respect to time, are 203.0GB and 212.0GB, respectively. The relative errors are less than 5%. It proves the correctness of our theoretic analysis. Besides, from the figure we observe that the net transmission rate with the FPTA algorithm remains a relatively high level in the process of migrations, about 2 times higher than that of the grouping algorithm on average. Thus, the integration of the net transmission rate can reaches $\sum_{k=1}^{40} m_k$ with less time. Specifically, in this group of experiments, the total migration time of FPTA algorithm is reduced by up to 50% compared with grouping algorithm. Thus, our FPTA algorithm significantly reduces the total migration time by maximizing the net transmission rate.

3) Application Performance: The scenarios of this group of experiments are to optimize the average delay of services in B4. Assume there are some VMs located randomly in the data centers in B4 at the beginning, and they are providing services to the same user, who is located closely to the node 8 (data center 8). Thus we need to migrate these VMs to data centers as close to the node 8 as possible. However, memory that each data center provides is not unlimited, which is set to be 50GB in our experiments. Besides, there are 11, 19, 27, 41 VMs in the network, respectively. We find the final migration sets by minimizing the average delay. Then we use the FPTA and grouping algorithm to implement these migrations. The results are shown in Fig. 8.

Fig. 8(a) and (b) show the total migration time and downtime, respectively. As we observe, FPTA algorithm reduces the total migration time and downtime by 43.7% and 22.6% on average compared with those of the grouping algorithm, respectively. Thus, our proposed FPTA algorithm outperforms the grouping algorithm uniformly, which provides better services for the user.

V. Related Work

Works related to our paper can be divided by two topics: live migration and migration planning.

Since Clark proposed live migration [2], there have been plenty of works that have been done in this field. Ramakrishnan *et al.* [19] advocated a cooperative, context-aware

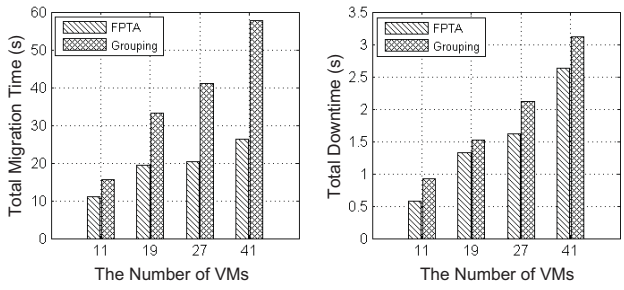


Fig. 8. Total migration time and downtime for optimizing delay in inter-datacenter network under the topology of B4.

approach to data center migration across WANs to deal with outages in a non-disruptive manner. Wood *et al.* [20] presented a mechanism that provides seamless and secure cloud connectivity as well as supports live WAN migration of VMs. On the other hand, VM migration in SDNs has made some progress. Mann *et al.* [21] presented CrossRoads – a network fabric that provides layer agnostic and seamless live and offline VM mobility across multiple data centers. Boughzala *et al.* [8] proposed a network infrastructure based on OpenFlow that solves the problem of inter-domain VM migration. Meanwhile, Keller *et al.* [22] proposed LIME, a general and efficient solution for joint migration of VMs and the network. These works indicate that SDN has big advantages in implementing VM migration. In contrast, we focus on developing a VM migration plan to reduce the total migration time in Software Defined Network (SDN) scenarios.

Meanwhile, there have been some works about VM migration planning. However, most of them were designed under the model of one-by-one migration [9], [12] or their main focuses were not to optimize the total migration time [9], [23]. Ghorbani *et al.* [9] proposed a heuristic algorithm of determining the ordering of VM migrations and corresponding OpenFlow instructions. However, they concentrated on bandwidth guarantees, freedom of loops, and their algorithm is based on the model of one-by-one migration. Al–Haj *et al.* [23] also focused on finding a sequence of migration steps. Their main goal was to satisfy security, dependency, and performance requirements.

VI. Conclusion

In this work, we focus on reducing the total migration time by determining the migration orders and transmission rates of VMs. Since solving this problem directly is difficult, we convert the problem to another problem, *i.e.*, maximizing the net transmission rate in the network. We formulate this problem as a mixed integer programming problem, which is NP-hard. Then we propose a fully polynomial time approximation (FPTA) algorithm to solve the problem. Results show that the proposed algorithm approaches to the optimal solution with less than 10% variation and much less computation time. Meanwhile, it reduces the total migration time and the service downtime by up to 40% and 20% compared with the state-of-the-art algorithms, respectively.

Acknowledgment

This work is supported by National Basic Research Program of China (973 Program) (No. 2013CB329105), National Nature Science Foundation of China (No. 61301080, No. 61171065 and No. 61273214), National High Technology Research and Development Program (No. 2013AA013501 and No. 2013AA013505), Chinese National Major Scientific and Technological Specialized Project (No. 2013ZX03002001), Chinas Next Generation Internet (No. CNGI-12-03-007)

References

[1] P. Barham, B. Dragovic, K. Fraser, "Xen and the art of virtualization", *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 164-177, 2003.
[2] C. Clark, K. Fraser, and S. Hand, "Live migration of virtual machines," in *Proc. of 2nd NSDI*, 2005, pp. 273–286.
[3] M. F. Zhani, Q. Zhang, G. Simona, "VDC Planner: Dynamic migration-aware virtual data center embedding for clouds", in *Proc. of IFIP/IEEE IM*, 2013, pp. 18-25.
[4] T. Wood, P. J. Shenoy, A. Venkataramani, "Black-box and Gray-box Strategies for Virtual Machine Migration", in *NSDI*, 2007, pp. 17.
[5] K. Ye, X. Jiang, D. Huang, "Live migration of multiple virtual machines with resource reservation in cloud computing environments" in *Proc. of IEEE CLOUD*, 2011, pp. 267-274.
[6] C. Mastroianni, M. Meo, G. Papuzzo, "Self-economy in cloud data centers: Statistical assignment and migration of virtual machines", *Euro-Par 2011 Parallel Processing*, 2011, pp. 407-418.
[7] M. F. Bari, Zhani M F, Zhang Q, "CQNCR: Optimal VM Migration Planning in Cloud Data Centers".
[8] B. Boughzala, R. Ben Ali, M. Lemay, "OpenFlow supporting inter-domain virtual machine migration", in *Proc. of WOCN*, 2011, pp. 1-7.
[9] S. Ghorbani, M. Caesar, "Walk the line: consistent network updates with bandwidth guarantees", in *Proc. of HotSDN*, 2012, pp. 67-72.
[10] S. Agarwal, M. Kodialam, T. V. Lakshman, "Traffic engineering in software defined networks", in *Proc. of IEEE INFOCOM*, 2013, pp. 2211-2219.
[11] A. Sridharan, R. Guerin, C. Diot, "Achieving near-optimal traffic engineering solutions for current OSPF/IS-IS networks", *IEEE/ACM Transactions on Networking (TON)*, vol. 13, no. 2, pp. 234-247, 2005.
[12] X. Li, Q. He, J. Chen, K. Ye, and T. Yin, "Informed live migration strategies of virtual machines for cluster load balancing", in *Proc. of NPC 2011*, (Berlin, Heidelberg), 2011, pp. 111-122, Springer-Verlag.
[13] J. Lofberg, "YALMIP: A toolbox for modeling and optimization in MATLAB", in *IEEE CACSD*, 2004, pp. 284-289.
[14] A. Makhorin, "GLPK (GNU linear programming kit)", 2008.
[15] L. K. Fleischer, "Approximating fractional multicommodity flow independent of the number of commodities", *SIAM Journal on Discrete Mathematics*, vol. 13, no. 4, pp. 505-520, 2000.
[16] D. G. Luenberger, Y. Ye, "Linear and nonlinear programming", *Springer*, 2008.
[17] S. Jain, A. Kumar, S. Mandal, "B4: Experience with a globally-deployed software defined WAN", in *Proc. of ACM SIGCOMM*, 2013, pp. 3-14.
[18] T. Benson, A. Akella, D. A. Maltz, "Network traffic characteristics of data centers in the wild", in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, 2010, pp. 267-280.
[19] K. K. Ramakrishnan, P. Shenoy, J. Van der Merwe, "Live data center migration across WANs: a robust cooperative context aware approach",in *Proc. of INM 2007*", pp. 262-267.
[20] T. Wood, K. K. Ramakrishnan, P. Shenoy, "CloudNet: dynamic pooling of cloud resources by live WAN migration of virtual machines",in *ACM SIGPLAN Notices*, 2011, pp. 121-132.
[21] V. Mann, A. Vishnoi, K. Kannan, "CrossRoads: Seamless VM mobility across data centers through software defined networking", in *Proc. of IEEE NOMS*, 2012, pp. 88-96.
[22] E. Keller, S. Ghorbani, M. Caesar, "Live migration of an entire network (and its hosts)", in *Proc. of HotNets*, 2012, pp. 109-114.
[23] S. Al-Haj, E. Al-Shaer, "A formal approach for virtual machine migration planning", in *CNSM*, 2013, pp. 51-58.