2016 IEEE International Conference on Computer and Information Technology

Reasonably Migrating Virtual Machine in NFV-featured Networks

Jing Xia, Deming Pang, Zhiping Cai, Ming Xu, Gang Hu
College of Computer
National University of Defense Technology
Changsha, 410073, China
{jingxia, pang3724, zpcai, xuming, hugang}@nudt.edu.cn

***Abstract*—Combining with software-defined networking and virtualization technology, Network Function virtualization (NFV) has been proposed as an important technology for constructing scalable network. VNF (Virtual Network Function) redeployment is a critical step for dealing with network evolve. Due to the inner state consistency constraints, VNF migration to a new location is major challenge for redeployment. Previous works about VNF migration primarily focused on VNF internal state transfer, which required reprograming VNF software and designed new transfer protocols. Extra work and more bugs could be introduced due to software develop workload. In this paper, we propose to migrate the entire VM (Virtual Machine) rather than VNF to simplify the development workload. Combining with Consolidated Middlebox, it is reasonable to migrate the VM which contains corresponding VNF to adapt to network evolve. However, transfer overhead introduced by VM migration should be minimized. Therefore, we tend to fulfill good VNF migration and minimize the time cost for the migration, meet the network bandwidth requirement as well. As this optimization problem is NP-hard, two heuristic algorithms are proposed aimed at various network scenarios. The paper is concluded with the simulation results and future direction.**

***Index Terms*—SDN; NFV; Migration; Virtualization;**

I. Introduction

Network functions (NFs) are network traffic processing modules that examine and modify packets and flows. Many functions such as performance and security improvement (e.g., proxies, firewalls and intrusion detection system (IDS)), bandwidth costs reduction (e.g., wide area network (WAN) optimizers), or policy compliance capabilities (e.g., network address translation (NAT) and content filters) can be provided through the implementation of NFs which have played critical role in dynamic networks.

A major challenge in dynamic networks is how to maintain appropriate NFs deployment to adapt to the changes of network environment. Traditionally, such challenge can be resolved by network function virtualization(NFV). In NFV-featured network, Virtual Network Functions (VNFs) are kinds of VMs (virtual machines) which running network function applications. Combined with Software-Defined Networking (SDN) technology, VNF makes the redeployment of NF easier by removing VNF from original physical machine and deploy another one with better location [1].

However, many network functions, such as firewall or IDS, need to track the operating state and characteristics of flows traversing through it, which means the history information must be kept in these VNFs. In other words, these VNFs are stateful, and the redeployment of these stateful VNFs will break the continuity of the flow state tracking. Thus, VNF internal flow state must be transferred along with NF redeployment, which leads us to VNF migration.

Several virtual network function migration mechanisms have been proposed recently [2] [3]. These mechanisms need to introduce new programs and protocols to extract and transfer VNF internal state during the migration process. However, VNF applications are provided by large amount of third part software supplier, the internal state extraction interface is not available or not compatible in most of these applications. To adopt new migration mechanisms, huge develop resources would be consumed. Besides, extra software development work may introduce unnecessary bugs to systems.

To deal with this problem, we propose a new mechanism in this paper. Rather than extract and transfer internal state of VNF, which introduces complicated flow state processing problem, we propose to migrate VM to adapt to network variations. As the VM contains VNF, the migration of VM transfer all needed data to migrate VNF, and due to the large research of virtualization technology, live VM migration has become a feasible and reliable technology.

In this novel mechanism, whenever a VNF is needed to be moved to another location, we will migrate the VM that the VNF sojourns in. VM migration can achieve the VNF redeployment without introducing new software development cost.

However, migrating VM rather than VNF may introduces other challenges. At first, there could be multiple VNFs in one VM, and not all of them need to be migrated at the same time. To resolve this problem, we adopt a one-to-one service model in a VM, which means each VM provides all required service functions to one network service class, rather than servicing multiple network traffic flows with one VM. This model can be implemented easily by exploiting the CoMb (Consolidated Middlebox) [4]. After that, VNF will be associated to network service rather than network flow, which means fixed VNF number for fixed services classes, and there is no VNF splitting or merging occur in this network model. Even if the network introduces more service classes, we simply deploy more VMs and vice versa.

978-1-5090-4314-9/16 $25.00 © 2016 IEEE
DOI 10.1109/CIT.2016.96

Secondly, existing study indicates that the performance could be compromised in live VM migration [5], and the cost will be different depending on the migration strategy. During a migration, the memory of a VM should be transmitted from source node to new target node through a multi-hop path. Different links along with the path may have different capacity, which will influence the finishing time of a VM migration. Furthermore, multiple migrations could be transmitted through the same link. Therefore, the bandwidth assigned to each VM migrations should be considered carefully. Since the link bandwidth is limited and valuable, the bandwidth constraints on each link must be satisfied when we design the migration plan. All these factors above make the designing of VM migration plan to be a challenging problem.

To resolve this problem, we define the migration time of a VM as the VNF migration cost, and formulate the path selection and capacity assignment as an integer programming problem to minimize the migration cost. As this problem is shown to be NP-hard, we design two heuristic algorithms for different application scenario and evaluate their performance by simulation.

The rest of the paper is organized as follows: Related work is discussed in Section II. In Section III, we describe the problem formulation. And we give two heuristic algorithms to solve the optimized migration plan problem for changing network environment in next section. The effectiveness of those algorithms is validated by simulations evaluations in Section V. Finally, conclusions are presented in the last section.

II. Related Work

There are a number of studies to address the deployment and migration of VNF. Sekar et al. used CoMb to reduce the complexity of the middlebox classes [4], Gember et al. realized a software-defined middlebox networking framework to simplify the management of complex and diverse functionalities [6]. In the scenarios of NFV and SDN, Gember et al. designed a control plane called OpenNF [2] which could provide efficient, coordinated control of both internal middlebox state and network forwarding state. Olteanu et al. proposed a two phases algorithm to avoid state transfer error in stateful middlebox migration [7].

Besides the implementation of the deployment and migration mechanism, many studies also focus on optimal deploy strategies of the VNF. Cheng et al. used the simulating annealing algorithm for the combinational problem of service chains, which could manage network services in an efficient and scalable way [8]. Xiong et al. designed a quantum genetic algorithm to resolve the service placement and minimize the network transport delay [9]. Ye et al. proposed an algorithm to address the JTDM (Joint Topology Design and Mapping of Service Function Chains) problem, and proposed a scalable and reliable strategy which can significantly reduce the network reconfigurations and enhance the service reliability [10]. Wang et al. used a heuristic algorithm to calculate the proper middlebox position [11]. Olteanu et al. proposed a heuristic algorithm to calculate the state transfer path [12].

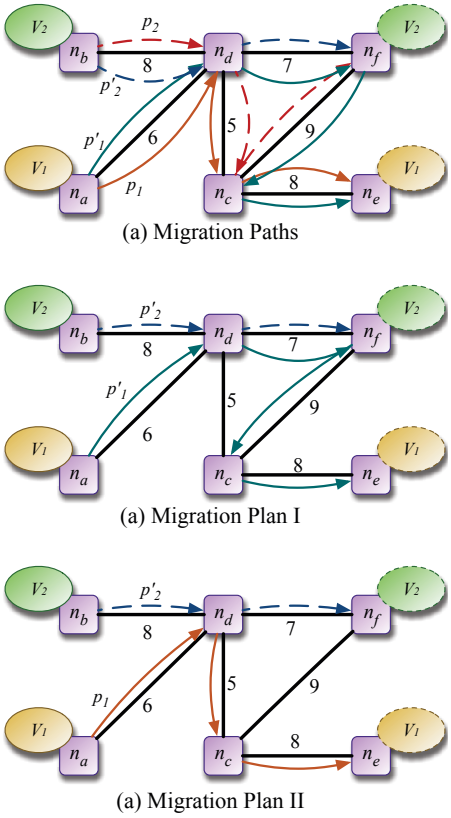

Fig. 1. VNF Migration

All of the above approaches do not try to migrate the virtual machine which contains the NF. With the stateful VNF migration, the two-step approach which separate the VNF redeploy and the inner state transfer is redundant and clearly increase the software complexity. To migrate the VM containing corresponding NF, the migration process is more simple and easier to construct the software. Thus, there is a need to migrate virtual machine instead of VNF and transfer VNF inner state. To the best of our knowledge, this paper is the first attempt to migrate the VM containing the NF for dynamic network environment.

III. System Model and Problem Formulation

Fig. 1 shows a typical scenario of VNF migration in part of a network. There are six network nodes and two VNFs are in the network, and VNF V_1, V_2 needs to be migrated from source location n_a, n_b to target location n_e, n_f, respectively. Fig. 1(a) shows the candidate paths of migrate V_1 and V_2. For VNF V_1, the migration path can be $p_1 = (n_a, n_d, n_c, n_e)$ and $p'_1 = (n_a, n_d, n_f, n_c, n_e)$. For VNF V_2, the migration path can be $p_2 = (n_b, n_d, n_c, n_f)$ and $p'_2 = (n_b, n_d, n_f)$. Numbers next to network edge represent the bandwidth capacity of this edge.

Fig. 1(b) and (c) show two different migration plan, as shown in Fig. 1(b), migration plan I choose path p'_1 and p'_2 as the migration path, both of the path include edge e_{df},

so the aggregation bandwidth of migrations cannot greater than 7 units. Migration plan II choose path p_1 and p_2', as shown in Fig. 1(c), In this plan, there is no shared edge between migrations, so the migration of V_1 and V_2 can respectively have 5 and 7 units of bandwidth in migrating path. Apparently, different migration path can give different aggregation migration bandwidth, which is the key factor to effect migration time.

A. Physical Network

We use one directed graph, $G = (N, E)$, to represent the physical network, where N is the set of its nodes and E is the set of its edges. Each node corresponds to an OpenFlow switch, and each edge is a link connecting two switches. For simplicity, we assume that every PM (Physical Machine) is connected with a single switch. So, we do not present PM in our formulation, and use the switch connected to PM as the migration source and target.

In this paper, we consider link bandwidth capacity as link attribute. We use constant b_e to represent this value, which is a measure (estimate) of total capacity that traffic flow gets on the link. The capacity of each link can be measured in advance. We assume that the physical network is full-duplex, so we have $b_{e_{uv}} = b_{e_{vu}}$, where nodes $u, v \in N$ are endpoints of edge e.

B. VM Migration

We use triple $m = (s, t, a)$ to represent a VM migration, where s and t are source and target node of migration m respectively, and a is the size of memory which will be moved in the migration. So we have $s \in N$, $t \in N$ and $a > 0$.

A set M is used to denote the collection of all the migrations. n denotes the number of the migration, i.e. $|M| = n$. So the ith migration m_i can be presented as $m_i = (s_i, t_i, a_i)$, where $0 < i \leq n$.

For a migration, VM usually need to be transmitted along a multi-hop path. Path is sequence of edges which connects source and target node in a network. It is assumed that all paths we discussed are loop-free since a VM needs to be moved to a different place. We use p to represent one path, and a set P represents the set of paths which shares the same sources and targets.

In this paper, we only discuss the paths associated to migrations. Each migration has only one pair of source and target nodes, so a set P_i is used to represent all the paths between source node s_i and target node t_i of a migration m_i. Let h_i represents the number of the paths in path set Pi, so the jth path of the ith migration m_i can be denoted as $p_{ij} \in P_i$, where $0 < j < h_i$.

Let q_p represents the number of edges in a path p, and we denote the edges in the path p as e^p, the kth edge in the jth path of the ith migration m_i can be denoted as $e_k^{p_{ij}}$, where $0 < k \leq q_{p_{ij}}$.

For a path p, a capacity function $B(p)$ is defined as the capacity assigned to this path, Obviously, the capacity of a path is determined by the minimal capacity of all edges, so we have the path capacity equation:

$$B(p_{ij}) = \min_{0<k\leq q_{p_{ij}}} \{b_{e_k^{p_{ij}}}\} \tag{1}$$

We call the edge with minimal capacity in a path as critical edge.

C. Cost Function and Optimal Objectives

We assume that each migration uses only one path and consumes fixed bandwidth. A variable x_i is introduced to represent the path selected by migration i in the path set P_i, and variable λ_i is used to represent the ratio migration consume against the path capacity. Let $f(m_i)$ be such consumption for the migration m_i. We should constraint the bandwidth consumption on paths. The function $f(m_i)$ can be calculated by the equation below:

$$f(m_i) = \lambda_i \times B(p_{ix_i}) \tag{2}$$

When the migration m_i occurred, memory of the VM size of a_i should be transfer through the chosen path p_{ix_i} at a fix bandwidth consumption $f(m_i)$. so we have the migration time:

$$t(m_i) = a_i / f(m_i) \tag{3}$$

Based on the equation above, we know that the selection of transmitting path for each VM migration impacts the time cost since different paths have different capacity. At the same time, ratio λ_i assigned to migration i also need to be taken into account carefully.

We assume the migration started simultaneously for simplicity. Since the migration imposes important effect on network performance once it started, the time duration for all the migrations should be minimized. Therefore, we formally define the cost function as follow

$$C(G, M) = \max\{t(m_i) | 0 < i \leq n\} \tag{4}$$

D. Formulation of optimal VM migration

The optimal VM migration planning in evolving network can be stated as following:

Given a set of VM migration request M and a physic network G, how to determine a VM migration plan $X = \{x_i, \lambda_i\}$ in G such that:

(1) The time cost for all VM migrations is minimum;

(2) All migration must perform correctly;

(3) The path bandwidth request of all migration should be satisfied;

(4) Each migration takes only one path.

The optimal VM migration planning can naturally be expressed as an optimization problem. The objective is:

$$\min_{y_{ie}, \lambda_i} C(G, M) \tag{5}$$

TABLE I
MAIN NOTATIONS

Notation	Description
$G = (N, E)$	Physical network
b_e	Capacity of link e in network
M	Set of all the migrations
n	Number of the migration in set M
$m_i = (s_i, t_i, a_i)$	Migration i with source s_i, destination t_i, and memory need to be migrated o_i
P_i	Path set of the ith migration m_i
h_i	Number of paths in the Path set P_i
p_{ij}	Path j of the ith migration m_i
$B(p_{ij})$	Bandwidth capacity of path p_{ij}
x_i	Path selected by migration i in the path set P_i
λ_i	Ratio of the migration m_i consume against the path capacity
$f(m_i)$	Bandwidth consumption of migration m_i
$t(m_i)$	Migration time of migration m_i
$C(G, M)$	Cost function of the migrations M in network G
y_{ie}	Whether edge e belongs to path x_i

where y_{ie} represents edge e belongs to path x_i or not, and the subjects are below:

$$p_{ix_i} \in P_i \tag{6}$$

$$0 < \lambda_i \leq 1 \tag{7}$$

$$y_{ie} = \begin{cases} 0(e \notin p_{ix_i}) \\ 1(e \in p_{ix_i}) \end{cases} \tag{8}$$

$$\sum_{i=1}^{n} f(m_i) \times y_{ie} \leq b_e(\forall e | e \in E) \tag{9}$$

Equation (9) constrains that the sum of all bandwidths assigned to migration which select edge e should not exceed the capacity of e.

All above notations are listed in Table I.

E. Computational Complexity

The optimal VM migration is of no doubt NP-hard problem. We can prove it via a reduction from the multiple commodity flow problem to this optimal problem. Thus, we have no hope to get a polynomial time algorithm for it.

IV. The Heuristic Algorithms for Solving the Problem

We design polynomial time algorithm in this section to minimize the cost function, i.e. the longest migration time of all the middlebox migration. A key step of the algorithm is to determine the value of λ which is the ratio of the bandwidth migration take against the capacity of the path.

The variant size and complexity of an evolved networks influence the manner of determining λ. We design two heuristic algorithms which can work well in deferent network environments.

Algorithm 1 Preprocessing

```
All paths set AP ← ∅
Critical edge set CE ← ∅
for i = 1 to n do
    DFS search all paths from s_i to t_i,
    Store them in P_i
    AP[i] ← P_i
    ce ← Maximal capacity
    for j = 1 to h_i do
        for k = 1 to q_P[i, j] do
            if b_{e_k^P[i,j]} < ce then
                ce ← b_{e_k^P[i,j]}
                CE[ij] ← e_k^P[i, j]
            end if
        end for
    end for
end for
return AP, CE
```

A. Preprocessing

At the beginning of the plan, we should calculate all the needed variants, and we called this step preprocessing. In this step, we propose an algorithm based on DFS to calculate the paths for each migration, and find out the critical edge for each path. The detail of preprocess algorithm is shown in Algorithm 1.

B. Critical Edge Pre-allocating approach

The first approach to solve the path allocation is to pre-allocate the critical edge bandwidth. In a network with sparse connections and density, the flows of multiple migrations usually pass the same edge. Therefore, if we constraint the bandwidth assigned to each migration to a small value, we can avoid conflict even that the paths selected by all migrations share the same edge as their critical edges.

1) Caculate the value of λ: We use the following equation to calculate the value of parameter λ:

$$\lambda_i = a_i / \sum_{j=1}^{n} a_j \tag{10}$$

2) Layout Paths: When we get the value of λ, we can layout the migration paths to the physical network. In this step, we simply choose the fastest path for migration due to that the capacity of edges will never been used up. The detail of the path layout algorithm has been shown in Algorithm 2.

C. Backtracking Approach

The algorithm designed above can work well in a simple network due to that the sharing of critical edge by multiple migrations happens commonly. But when the connections of physic network increase to a very large number, even far more than the migration number, this algorithm may cause large bandwidth waste.

Algorithm 2 Critical Edge Bandwidth Pre-allocating

Chosen path set $CP \leftarrow \emptyset$
for $i = 1$ to n **do**
  $CP[i] = AP[i][1]$
  for $j = 1$ to h_i **do**
    if $B(AP[i][j]) > B(CP[i])$ **then**
      $CP[i] \leftarrow P[i][j]$
    end if
  end for
end for
return CP

In the first approach, the parameter λ constraint the bandwidth consumption of each migration to a small value, which can avoid the problem of using up bandwidth of edge even that all the paths share the same critical edge. But in another way, this conservative design could waste lots of the bandwidth. In a network with dense connections and small number of migrations, which are more common in the practical network, the path of most migrations can be laid on different sets of edges. Therefor we design the second algorithm based on a backtracking approach.

The backtracking approach use a greedy strategy which is to allocate migration based on the migrating memory size a. We introduce two variants to control the backtracking process. The first variant is the initial bandwidth allocation ratio: α. At the beginning, we make λ_i equals α, and allocate the bandwidth in $B(p_{ij}) \times \lambda$ to a migration. When the following allocation use up the bandwidth of an edge, we use the second variant, back off ratio: β, to backtrack the last allocation, and make $\alpha \times \beta$ to new λ, to decrease the allocation. The proposed backtracking path layout algorithm has been detailed in Algorithm 3.

In this algorithm, once we lay a path for a migration in physical network, we update all the edges capacity by subtracting bandwidth consumption. We use b_e^r to denote the new bandwidth of an edge at the beginning, $b_e^r = b_e$. When the capacity of an edge is updated, all the paths should update their capacities too.

V. Simulations

In this section, we evaluate the performance of the proposed heuristic algorithms on several different topologies and parameter settings.

A. Simulation Preliminary

We generate the physical network topologies by using the GT-ITM tool [13], and considering the migration problems with typical scenario, including the customized source, target node and the memory sizes. Different network topologies and migrations are generated to evaluate these two algorithms and different parameters α and β. Simulation results presented here are averaged over 5 different topologies.

The physical network is configured to have 100 nodes and around 500 links, a scale that corresponds to a medium sized ISP. The link capacity of the network follow a uniform distribution from 100 to 1000 Mbps. The numbers of migration are changed by stepwise increase 5 migrations. The source and target nodes are randomly chosen and supposed to be connected by at least one path. The memory size needed to be moved by the migration is generated randomly and follows a uniform distribution from 2G to 4G bytes.

Algorithm 3 Backtracking

Chosen path set $CP \leftarrow \emptyset$
Sort the migration with decrease by the migrating memory size of the migration set
$i \leftarrow 1$
while $i \leq n$ **do**
  Sort the path with decrease by the critical edge capacity in the path set of migration m_i
  $j \leftarrow 1$, $\lambda_i \leftarrow \alpha$
  while $j \leq h_i$ **do**
    if $B(p_{ij}) > 0$ **then**
      $CP[i] \leftarrow p_{ij}$
      Update edge and other path capacity by minis $B(p_{ij}) \times \lambda_i$
      break
    else
      $j \leftarrow j + 1$
    end if
  end while
  if $j > h_i$ **then**
    $i \leftarrow i - 1$
    $\lambda_i \leftarrow \lambda_i \times \beta$
  else
    $i \leftarrow i + 1$
  end if
end while
return CP

TABLE II
Detail Simulation Results

n	$t_{backtracking}$	$t_{pre-allocating}$	t_{random}
10	112.3	120.2	150.3
20	130.4	146.2	212
30	170.3	193	261.4

B. The Effect of the Algorithm for Scalable Situation

We compare the performance of our two algorithms which uses randomly picking strategy to layout the migration path. Fig. 2(a) shows the total migration time costs of finishing all the migration, which is a function of the stepwise random 5 migrations. Table II presents a set of simulation results. These three curves represent different approaches to plan the migration: complete random, critical edge pre-allocating and backtracking, respectively. It is shown the total migration time is rapidly increased with the increasing of the number of migrations. The difference of performance about the three approaches shows that our approaches are reasonably efficient.

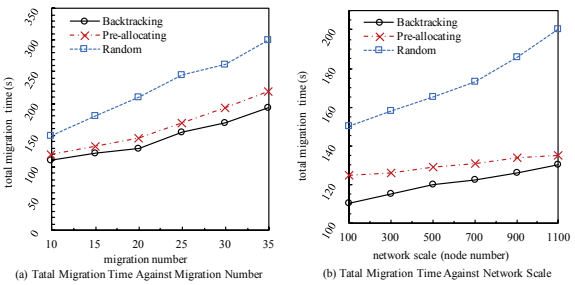

Fig. 2. Performance evaluation for scalable

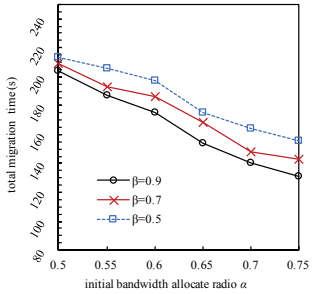

Fig. 3. Performance against backtracking parameter

Fig. 2(b) shows the total migration time costs for perform all the migration as a function of the stepwise 200 network node. It is shown that the total migration time is increased by the increase of the network scale. And the performance of our proposed approaches is better than the random approach.

C. Evaluation of Backtracking Parameter

To determine the parameter used in the backtracking approach, we compare the performance of algorithm with different parameter values. Fig. 3 shows the total migration time costs for finishing all the migration as a function of the initial bandwidth allocate ratio α. These three curves represent the instances with different back off ratio β equal to 0.9, 0.7 and 0.5, respectively. It is shown that total migration time is slowly decreased by increasing α, and is slightly increased by decreasing back off ratio β.

VI. Conclusion

This paper deals with migrating VM instead of VNF to avoid flow state transmission problem. In order to minimize the transmission cost of VM migration, we formulate the joint path selection and capacity assignment as an integer programming problem. Due to this problem is NP hard, we propose two heuristic algorithms for different application scenarios. A critical edge pre-allocating approach is designed for the network with sparse connections, and backtracking approach works well for high destiny networks. Simulations validate the effectiveness of the proposed algorithms for computing an optimized path. Further research would be conducted to design adaptive algorithms for multiple objectives and satisfying multiple constraints. It would allow our approach to be applied in practice easily.

Acknowledgments

The authors would like to thank the National Natural Science Foundation of China under Grant Nos. 61379144, 61379145, 61501482 for their supports of this work.

References

[1] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, Feb. 2015.

[2] A. Gember-Jacobson, R. Viswanathan, C. Prakash, 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*. New York, NY, USA: ACM, 2014, pp. 163–174. [Online]. Available: http://doi.acm.org/10.1145/2619239.2626313

[3] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield, "Split/Merge: System Support for Elastic Execution in Virtual Middleboxes," in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*. Berkeley, CA, USA: USENIX Association, 2013, pp. 227–240. [Online]. Available: http://dl.acm.org/citation.cfm?id=2482626.2482649

[4] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi, "Design and Implementation of a Consolidated Middlebox Architecture," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*. Berkeley, CA, USA: USENIX Association, 2012, pp. 24–24. [Online]. Available: http://dl.acm.org/citation.cfm?id=2228298.2228331

[5] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, "Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation," in *Cloud Computing: First International Conference, CloudCom 2009, Beijing, China, December 1-4, 2009. Proceedings*, M. G. Jaatun, G. Zhao, and C. Rong, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 254–265. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-10665-1_23

[6] 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*. New York, NY, USA: ACM, 2012, pp. 7–12. [Online]. Available: http://doi.acm.org/10.1145/2390231.2390233

[7] V. A. Olteanu and C. Raiciu, "Efficiently Migrating Stateful Middleboxes," in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. New York, NY, USA: ACM, 2012, pp. 93–94. [Online]. Available: http://doi.acm.org/10.1145/2342356.2342376

[8] G. Cheng, H. Chen, H. Hu, Z. Wang, and J. Lan, "Enabling network function combination via service chain instantiation," *Computer Networks*, vol. 92, Part 2, pp. 396–407, Dec. 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128615003254

[9] G. Xiong, Y. Hu, L. Tian, J. Lan, J. Li, and Q. Zhou, "A virtual service placement approach based on improved quantum genetic algorithm," *Frontiers of Information Technology & Electronic Engineering*, vol. 17, no. 7, pp. 661–671, 2016.

[10] Z. Ye, X. Cao, J. Wang, H. Yu, and C. Qiao, "Joint topology design and mapping of service function chains for efficient, scalable, and reliable network functions virtualization," *IEEE Network*, vol. 30, no. 3, pp. 81–87, May 2016.

[11] S. Wang and S. Mahalingam, "Dynamic Scaling for Software Middleboxes," 2013. [Online]. Available: http://www.cs.berkeley.edu/~kubitron/courses/cs262a-F12/projects/reports/project18_report.pdf

[12] V. A. Olteanu, F. Huici, and C. Raiciu, "Lost in Network Address Translation: Lessons from Scaling the World's Simplest Middlebox," in *Proceedings of the 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization*. New York, NY, USA: ACM, 2015, pp. 19–24. [Online]. Available: http://doi.acm.org/10.1145/2785989.2785994

[13] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *Proceedings IEEE INFOCOM '96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation*, vol. 2, Mar. 1996, pp. 594–602 vol.2.