Energy Cost Optimization in Dynamic Placement of Virtualized Network Function Chains

Binayak Kar, Eric Hsiao-Kuang Wu, *Member, IEEE*, and Ying-Dar Lin, *Fellow, IEEE*

***Abstract*—Network function virtualization (NFV), with its virtualization technologies, brings cloud computing to networking. Virtualized network functions (VNFs) are chained together to provide the required functionality at runtime on demand. It has a direct impact on power consumption depending on where and how these VNFs are placed and chained to accomplish certain demands as the power consumption of a physical machine (PM) depends on its traffic load. One of the advantages of VNF placement over traditional virtual machine placement is that virtualization is not limited solely to servers. The PMs, including the servers and varying loads to these machines and their utilization, are critical issues related to the network's energy consumption. In this paper, we designed a dynamic energy-saving model with NFV technology using an M/M/c queuing network with the *minimum capacity* policy where a certain amount of load is required to start the machine, which increases the utilization of the machine and avoids frequent changes of the machines' states. We formulate an energy-cost optimization problem with capacity and delay as constraints. We propose a dynamic placement of VNF chains (DPVC) heuristic solution to the NP-hard problem. The results show that the DPVC solution performs better and saves more energy. It uses 45%–55% less active nodes to satisfy the requested demands and increases the utilization of the active nodes by 40%–50% compared to other algorithms.**

***Index Terms*—Virtualized network function, service chaining, Markov model, energy consumption, cost optimization.**

I. Introduction

THE ENERGY crisis has been a significant issue for years, as most resources are non-renewable such as oil, coal, and gas [1]. As the report [2] suggested, by 2008, global electrical generation from solar sources was less than 1%. Furthermore, solar power has a very poor infrastructure worldwide. In 2015, it barely passed 1 % [3]. The majority of the world's energy comes from non-renewable resources, which, due to the large amounts of greenhouse gases emitted (such as carbon dioxide), lead to global warming. To add to this concern, energy consumption and greenhouse gas emissions resulting from Internet usage have been steadily increasing in recent years. For example, data center Web servers, such as those used by Google and Facebook, account for 2% of the greenhouse gas emissions – about the same as air travel [4]. This rate is going to increase substantially as people of highly populated developing countries like India and China become more inclined to use the Internet. In the U.S., which hosts approximately 40% of the world's data center servers, it is estimated that server farms consume close to 3% of the national power supply [5]. Greenpeace's 2010 "Make IT Green" report estimates that the global demand for electricity from data centers was on the order of 330bn kWh in 2007, close to the equivalent of the entire electrical demand of the U.K. This demand is projected to triple or quadruple by 2020.

Fortunately, virtualization technology can spur a "green" revolution in the communication network. Server virtualization in data center networks is a big example of this revolution. A great deal of research [7]–[9] has focused on minimizing energy consumption in data center networks that use virtual machines. However, this virtualization mechanism has been limited to servers only, and some hitches, such as excessive resource fragmentation [10] and migration issues [11], still exist. Additionally, virtualization technology has a very high migration cost both in terms of time and energy [13]. Most importantly, its management is only limited to the service provider.

Network function virtualization (NFV) [12], [14], [15] technology has emerged as a new alternative, which can overcome the pitfalls. NFV offers a new way to design, deploy, and manage networking services by decoupling the network functions, such as network address translation, firewalls, intrusion detection, domain name service, etc., from dedicated hardware devices so they can run in software. These network functions are called virtualized network functions (VNFs), and they are placed on physical machines as "virtual machine (VM) instances." However, VNFs can be placed on other types of containers like Docker or Linux container (LXC) [6]. A network service chain consists of a chain of such VNFs that can be connected across the network using software provisioning. An example of service chain placement in the network is presented in Figure 1. The network consists of nine nodes considered as the physical machines of the network. We have four different network functions: A, B, C, D, which are available in different nodes of the network, as shown in Figure 1. Table I shows four service chain demands, with the source and destination paths of four different flows. The virtual links and physical path of each flow from source to destination are given in the table. The first service chain, SC1 (B–C–A), at source node 1, will be placed in the sequence of nodes 2, 4, 7.

Manuscript received September 6, 2017; revised November 17, 2017; accepted November 24, 2017. Date of publication December 12, 2017; date of current version March 9, 2018. This work was supported by the Ministry of Science and Technology under Grant 106-3114-E-002-005. The associate editor coordinating the review of this paper and approving it for publication was F. De Turck. *(Corresponding author: Eric Hsiao-Kuang Wu.)*

B. Kar and E. H.-K. Wu are with the Department of Computer Science and Information Engineering, National Central University, Chung-Li 32001, Taiwan (e-mail: binayakar@wmlab.csie.ncu.edu.tw; hsiao@csie.ncu.edu.tw).

Y.-D. Lin is with the Department of Computer Science, National Chiao Tung University, Hsinchu 300, Taiwan (e-mail: ydlin@cs.nctu.edu.tw).

Digital Object Identifier 10.1109/TNSM.2017.2782370

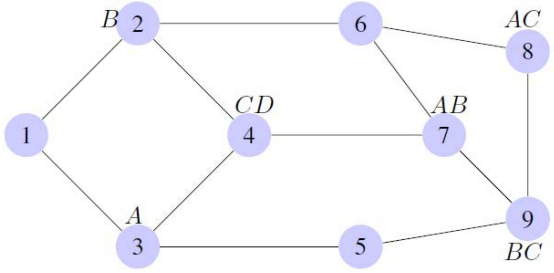

Fig. 1. Illustrative example of service chain placement in the network.

TABLE I
VNF PLACEMENT OF THE SERVICE CHAINS

SC number	Source	Destination	SC Demand	Virtual Path	Physical Path
SC1	1	6	B–C–A	1–2–4–7–6	1–2–4–7–6
SC2	5	1	C–A–D	5–9–7–4–1	5–9–7–4–2–1
SC3	6	1	A–D–B	6–7–4–2–1	6–7–4–2–1
SC4	5	6	B–D–B	5–9–4–2–6	5–9–7–4–2–6

That is, the first VNF 'B' will be placed on node 2, then the second VNF 'C' will be placed on node 4, where the function 'C' is available. The final VNF of SC1 will be placed on node 7 and the chain will terminate at destination node 6. Similarly, the second service chain, *SC2* (C–A–D), will start from source node 5, terminate at destination node 1, and place VNFs on nodes 9, 7, 4. The third service chain, *SC3* (A–D–B), will start from source node 6, terminate at destination node 1, and place VNFs on nodes 7, 4, 2. However, for the fourth service chain, *SC4* (B–D–B), from the source node 5, after placement of the first VNF 'B' on node 9, the next function 'D' is not available in the neighboring nodes. Therefore, it will be placed on node 4, the last VNF 'B' will be placed on node 2, and finally, it will terminate at destination node 6.

As we discussed earlier, energy is a big issue within the Internet world. Unlike VM placement, we can minimize energy consumption using NFV technology, which can extend virtualization to other PMs, such as routers, switches, etc. Physical machines consume maximum power only during peak demand times, and average servers remain idle over 90% of the time [17]. However, the power consumption of an idle machine is nearly 60% of the peak load power consumption of the machine [18]. By reducing the number of active machines, and turning off the idle machines, we can reduce the energy consumption of the network.

In our design, we use the minimum capacity mechanism [44] to minimize the frequent change of the machines' states. According to this mechanism, we required a minimum capacity to transit an OFF node to an ACTIVE node. This helps to increase the utilization of the machine. Fifty percent of the power consumed by the PMs is to reduce heat generated during processing [20]. Hence, depending on the PM load, the cooling load also varies [45]. Therefore, in this paper, we normalize the energy consumption cost of the PMs, and the respective VM instances on those machines, which will help with performing a better analysis of the network's energy consumption issues. We consider the dynamic service chain placement, which is a more realistic scenario than static placement. However, in this work, we do not consider the energy consumption of the link, as the difference of the energy consumption of the link from idle to full utilization is very minimal [47].

Since NFV extends the virtualization beyond servers, it can be applied to data centers, wide areas, and backbone networks. Hence, our design is not limited to any predefined topology of the network. In this paper, we tried to find the most suitable node for the placement of VNF of the service chain, in order to minimize the total energy consumption cost with certain constraints. Our novel contribution is summarized as follows:

1) First, we design an energy-saving model using an M/M/c queuing network [*discussed in Section III (B)*] for the placement of multiple service chains' functions in the network.
2) We formulate an optimization problem to minimize the total energy consumption cost of the network with capacity and delay as the constraints and prove that NP-hard.
3) We propose an efficient dynamic placement of VNF chains (DPVC) heuristic algorithm for the dynamic placement of VNFs in the network. Via MATLAB experimentation, we demonstrate that our algorithm significantly minimizes the cost of energy consumption.

The remainder of the paper is organized as follows. In Section II we will discuss some related works. Design and modeling is presented in Section III. We propose the optimization problem in Section IV, and the heuristic solution in Section V. In Section VI, we present an analysis of results and discuss the conclusions in Section VII.

II. RELATED WORKS

Works related to our paper can be divided into two categories: Energy saving models using VM placement; and different VNF placement methods and how our paper differs from them.

A. Energy Saving Using Virtual Machines

Energy consumption minimization is one of the most studied objective functions, with several modeling approaches proposed in VM placement [22]. A much-studied approach is to consolidate VMs on the minimum number of PMs based on the assumption that the use of fewer PMs will bring less energy consumption [23]. Ding *et al.* [25] proposed an energy-efficient scheduling algorithm of VMs to reduce the total energy consumed by the cloud. A VM placement algorithm for the distributed data centers was proposed in [24] to enhance environmental sustainability. Chiang *et al.* [26] proposed power-saving methods using VM placement by reducing the number of unnecessary power-consuming machines in cloud systems. In [8], an energy-aware Virtual Machine Allocation Algorithm is presented to reduce data center power consumption. This is accomplished by switching idle nodes to sleep mode and allow Cloud providers to optimize resource usage. Reference [7] Proposes a two-stage scheme to address the energy issue in the DC. First, they use a static VM

TABLE II
RELATED WORKS ON VNF PLACEMENT

Reference	Optimal Placement	Dynamic Placement	Energy-Saving Model	Normalized PM & VM Cost
[33,34,35,36]	✓	×	×	×
[37]	×	✓	×	×
[16,38]	✓	✓	×	×
This paper	✓	✓	✓	✓

placement scheme to minimize the active PMs and second they propose a dynamic VM migration scheme to minimize the maximum link utilization to improve the network performance. A multi-dimensional space partition model was proposed in [9] to characterize the resource uses in the PMs. They proposed a VM placement algorithm to maximize the PM utilization and minimize the energy consumption.

B. Virtualized Network Function Placement

The concept of NFV is creating the greatest buzz in the telecommunication industry, given a large number of several complex network functions in telecom networks.

Several optimization frameworks and planning tools for NFV are available. Ghaznavi *et al.* [27] introduced the elastic virtual network placement problem and presented a model for minimizing the operational costs and providing VNF services. Addis *et al.* [29] provided a framework to evaluate the relative benefits of NFV in various scenarios and solved the problem of VNF service chain placement using mixed integer linear programming (MILP) to minimize the total provisioning cost. Moens and De Turck presented a model for resource allocation in NFV in [31]. They consider a NFV-enabled hybrid environment by giving insights into trade-offs between legacy and NFV networks. D'Oro *et al.* discussed the service chain composition problem in NFV networks [28]. They used the non-cooperative game theory to propose a distributed and privacy-preserving algorithm in polynomial time. D'Oro *et al.* [30] focus on the distributed resource allocation and orchestration of softwarized network. They used the game theory to model the interaction between a user's demand and a server's availability and response. A dynamic function composition optimization problem was proposed in [32], where they used the Markov chain approximation method to dynamically decide the appropriate service function instances at run time. Sahhaf *et al.* [46] discussed the optimal decomposition and embedment of network services. They minimize the mapping cost of the network service chains and address the scalability issue heuristically. A service chain instantiation framework was discussed in [48] to combine the network function optimally.

Table II lists some papers based on VNF placement and how our contribution differs from them. Cohen *et al.* [33] discuss algorithms for near-optimal placement of VNFs. They presented a linear program (LP) relaxation-based approach for finding the inter-data center VNF chain placement. In this VNF placement problem, each demand considers a VNF set. Their goal is to minimize the overall operational cost. A context-free language-based VNF placement model is proposed by Mehraghdam *et al.* [34]. They used mixed integer quadratically constrained program (MIQCP)-based mapping to find the PM for VNF placement. Luizelli *et al.* [35] proposed an integer linear program (ILP) model to embed VNF chains on a network infrastructure. The proposed model targets a minimum number of VNFs to be mapped on the substrate. The article [36] presented an Eigen-decomposition-based approach for the placement of network function chains. The previously discussed articles in Table II focused on the optimal placement of VNF service chains, but they are not dynamic. However, Clayman *et al.* [37] described an architecture based on an orchestrator, which ensures that the placement of the virtual nodes, and the allocation of network services on them, is automatic. In this architecture, they used a monitoring system that collects and reports on the behavior of the resources. However, their method is not optimal. Bari *et al.* [38] solve the problem of determining the number of VNFs required, and their placement to optimize operational expenses dynamically while adhering to service level agreements using an ILP. Optimal deployment of new service function chains and readjustment of the in-service chains dynamically was discussed in [16].

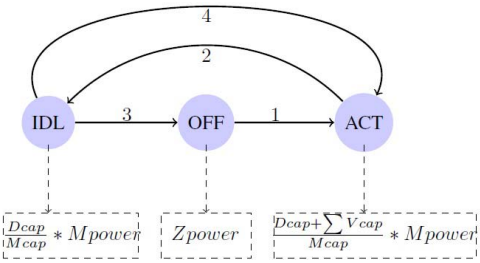

Fig. 2. OIA state transition diagram of PM.

In this paper, we focus on the optimal dynamic placement of service chains and propose an energy-saving model using an M/M/c queuing model. As the PMs are not homogeneous in terms of performance, the amount of each machine's energy consumption is affected by the number of VM instances on it and their capacities. So during the evaluation of energy consumption cost, we normalized the PM and VM cost together, which has not been done before.

III. DESIGN AND MODELING

A. Off-Idle-Active State Transition

Figure 2 illustrates the *Off-Idle-Active* (OIA) state transition diagram of the PM. Each machine has three states named "OFF," "IDLE," and "ACTIVE." A machine can transit from one state to another with the following four rules. The power consumption of the machine can be evaluated in each state as well.

(1) OFF→ACT: Initially, the machine is in an OFF state, and it consumes zero power (*Zpower*). The machine will turn ACTIVE when the sum of the capacities of the VNFs in the queue exceeds the minimum capacity, or when the waiting time of any VNF in the queue exceeds the maximum waiting time. We adopted this method to maximize utilization and to avoid the machine from engaging in the switching state too

often. This method also minimizes the waiting time of the VNFs, which were waiting in the queue for a longer time.

(2) ACT→IDL: When all VNFs in the ACTIVE machine finish or migrate to other machines, the machine will go to the IDLE state. In the IDLE state, a machine will consume the basic amount of energy, which can be evaluated as the product of maximum power (*Mpower*) consumed by the machine and the ratio between default capacity (*Dcap*) of the machine to the maximum capacity (*Mcap*) of that machine.

(3) IDL→OFF: If no new VNFs are assigned to the machine in the IDLE state within a predefined time, the machine will turn OFF.

(4) IDL→ACT: If new VNFs are assigned, or if the VNFs migrate from other ACTIVE machines, the machine will turn ACTIVE from the IDLE state. Evaluation of the machine's energy consumption in the ACTIVE state is similar to the IDLE state, only we need to add the summation of the capacities of the deployed VNFs ($\sum Vcap$) in the machine to the default capacity of the machine. Here, the maximum power of the machine represents the maximum computing and cooling power of the machine.

B. M/M/c Queuing Network Model

Many analytical models have been presented by various articles [26], [39], [41] using the M/M/1 queuing model. However, in practice, real-world applications are not processed by the single-service node. Therefore, we use the M/M/c queuing network model [40], [42], where each service chain request can be processed through multiple service nodes, and each service node can process multiple network functions. Our energy-saving model adheres to the following assumptions. The VNFs of the service chain arrivals follow a Poisson process λ and are served in the order of their arrivals, *i.e.*, the $(i+1)$th VNF of a service chain can start only after completion of the ith VNF of that service chain. In our model, a service node can process a maximum c number of VNFs of different service chains together. We assume all service chains are independent. All service times are independent and exponentially distributed with mean $1/\mu$. The idle time follows the exponential distribution with mean: $1/\theta_1$, and the off time follows exponential distribution with mean: $1/\theta_2$. Both aforementioned variables are independent of each other. Here, the state space is settled by $S = \{(m, n),\ m = \{0, 1\},\ 0 \le n \le \infty\}$ where m denotes the machine is ON or OFF, and n denotes the number of VNFs in the machine. The state-transition-rate diagram for a queuing system is shown in Figure 3. State $(0, 0)$ denotes that the machine is ON, but with no VNF, *i.e.*, the IDLE state, and $(0, n)$ denotes that the machine is ACTIVE with n number of VNFs. State $(1, n)$ shows the OFF state with n number of VNFs in waiting.

Let $P_{m,n}$ denote the steady-state probabilities at state (m, n), then the following notations are used:

$P_{0,n}$ = Probability that n VNFs exist in the PM in the ACTIVE state.

$P_{0,0}$ = Probability that no VNFs exist in the PM, and it is IDLE.

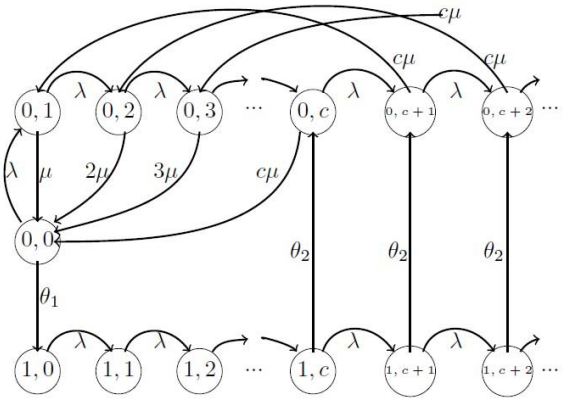

Fig. 3. M/M/c queuing model state transition diagram for a service node.

$P_{1,n}$ = Probability that n VNFs exist in the PM in the OFF state.

Based on Figure 3, the following balanced equations can be given:

$$(\lambda + \theta_1)P_{0,0} = \mu \sum_{n=1}^{c} n \cdot P_{0,n}, \tag{1}$$

$$(\lambda + c\mu)P_{0,n} = \lambda \cdot P_{0,n-1} + c\mu \cdot P_{0,n+c}, \tag{2}$$

where $n = 1, \ldots c-1$,

$$(\lambda + c\mu)P_{0,n} = \lambda \cdot P_{0,n-1} + c\mu \cdot P_{0,n+c} + \theta_2 \cdot P_{1,n}, \tag{3}$$

where $n = c, c+1, \ldots \infty$,

$$\theta_1 \cdot P_{0,0} = \lambda \cdot P_{1,0}, \tag{4}$$

$$\lambda \cdot P_{1,n} = \lambda \cdot P_{1,n-1}, \text{ where } n = 1, \ldots c-1, \tag{5}$$

$$(\lambda + \theta_2)P_{1,n} = \lambda \cdot P_{1,n-1}, \text{ where } n = c, c+1, \ldots \infty. \tag{6}$$

Let P_{ACTIVE}, P_{IDLE}, and P_{OFF} denote the probabilities that a PM is in the ACTIVE, IDLE, and OFF states respectively. With the normalizing equation $\sum_{n=1}^{\infty} P_{0,n} + P_{0,0} + \sum_{n=0}^{\infty} P_{1,n} = 1$, the solutions of these equations can be obtained as:

$$\begin{cases} P_{ACTIVE} = \sum_{n=1}^{\infty} P_{0,n}, \\ P_{IDLE} = P_{0,0}, \\ P_{OFF} = \sum_{n=0}^{\infty} P_{1,n}. \end{cases}$$

Theorem 1: $\sum_{n=0}^{\infty} P_{1,n} = [c \cdot \frac{\theta_1}{\lambda} + \frac{\theta_1}{\theta_2}]P_{0,0}$.

Proof: Please refer to Appendix B. ■

Assuming, $\alpha = \frac{(\theta_1+\lambda)\cdot K\cdot(K+c)}{2c^2}$ for some large integer K, we have,

Theorem 2: $\sum_{n=1}^{\infty} P_{0,n} = [\frac{\alpha}{c\mu} + \frac{\theta_1}{\lambda+c\mu}]P_{0,0}$.

Proof: Please refer to Appendix E. ■

Theorem 3: If $\sum_{n=0}^{\infty} P_{0,n} + \sum_{n=0}^{\infty} P_{1,n} = 1$, *then* $P_{0,0} = \frac{1}{1+\frac{\theta_1}{\theta_2}+\frac{c\theta_1}{\lambda}+\frac{\alpha}{c\mu}+\frac{\theta_1}{\lambda+c\mu}}$.

Proof: Please refer to Appendix F. ■

By using this value $P_{0,0}$ we can find P_{OFF}, P_{IDLE}, and P_{ACTIVE} of each PM. That is, we can determine what is the state of the machine, and how many VNFs are on the machine. Then, as given in Figure 2, the amount of the power consumed by the machine in different states can be evaluated.

TABLE III
LIST OF COMMONLY USED VARIABLES AND NOTATIONS

Variables	Descriptions
Sets	
N	Set of PMs/nodes
L	Set of links
vF	Set of VNFs
sC	Set of service chains
T	Set of iterations
$vM(i)$	Set of VM instances on a node i
K	Set of commodities
Variables and Network Parameters	
u	Node u
$l(u,v)$	Link (u,v)
s_k, t_k	Source and destination of commodity $f_k \in K$
$F_k(u,v)$	Flow of commodity k along link (u,v)
P_k	Path of flow k
sc	Service chain sc
$n(u)$	Decision variable of physical node u, showing node is OFF/IDLE/ACTIVE
e_u^N	Energy consumed by physical node u
e_C	Energy consumption cost
te_C	Total energy consumption cost
Delay, Demand and Capacity Parameters	
$C^N(u)$	Maximum capacity of the node u
$C^L(u,v)$	Capacity of the link (u,v)
$C^I(u)$	Default load of node u at IDLE state
$C_i^V(u)$	Capacity of ith VM of node u
$d_k(u,v)$	Delay faced by the flow k at link (u,v)
$d_f^{sc}(u)$	Demand of function f of service chain sc at node u
$qd_f^{sc}(u)$	Queuing delay of function f of service chain sc at node u
$d_{MAX}(u)$	Maximum delay at node u
D_k	Maximum delay that the flow can tolerate
Binary Variables	
$X_i^f(u)$	Instance i of function f mapped to node u
$Y_f^{sc}(u)$	Function f of service chain sc placed on node u

IV. PROBLEM FORMULATION

As described in the previous section, we can get the state of a PM at a particular time and the number of VM instances on the PM at that time. Considering this, we propose an optimization problem to minimize the energy consumption cost in this section.

A. Variable Declaration

In Table III, we declared the variables used to formulate the optimization problems. We classify all variables in four groups. The first group represents the different sets we will use. N and L are the set of nodes and links, respectively. Here, a node means a PM. sC represents the set of all requested service chains and vF is the set of VNFs we have in our network. T is the set of iterations. vM is the set of VM instances on a particular node and K is the set of commodities. The second group represents the variables and different network parameters we will use. $n(u)$ is the decision variable of the physical machine u, which shows the state of the machine. s_k and t_k are the source and destination of commodity k, respectively. F_k is the flow of commodity k, and P_k is the path of flow k. e_u^N shows the energy consumed by the node u. e_C and te_C are the energy consumption cost and the total energy consumption cost, respectively. The third group refers to the delay, demand, and capacity parameters. $C^N(u)$, $C^I(u)$, and $C_i^V(u)$ represent the maximum capacity, default capacity, and capacity of the VM instance i of the node u, respectively. $C^L(u,v)$ is the capacity of link (u,v) and $d_k(u,v)$ is the delay faced by the flow k at link (u,v). $qd_f^{sc}(u)$ is the queuing delay of the function f of service chain sc at node u. $d_{MAX}(u)$ is the maximum delay at node u, and D_k is the maximum delay that the flow can tolerate. $d_f^{sc}(u)$ represents the demand of function f of service chain sc at node u. The last group consists of two binary variables. $X_i^f(u)$ presents function f placed on the ith VM instance of node u. $Y_f^{sc}(u)$ shows the function f of service chain sc placed on node u.

B. Objective Function and Constraints

By using the notations given in Table III, we state the energy consumption cost is:

$$e_C = \sum_{u \in N} n(u) * \left(\frac{C^I(u) + \sum_{i \in vM(u)} C_i^V(u)}{C^N(u)} \right) * e_u^N,$$

$$\text{Where} \quad n(u) = \begin{cases} 1, & \text{if node } (u) \text{ is IDLE or ACTIVE}, \\ 0, & \text{Otherwise}. \end{cases}$$

The total energy consumption cost $te_C = \sum_{t \in T} e_C(t)$. Here, $n(u)$ represents the state of the node u. The value is 1 if the node is ACTIVE or IDLE, and 0 otherwise. $C^N(u)$, $C^I(u)$, and $C_i^V(u)$ represent the maximum capacity, default capacity of the machine in the IDLE state, and capacity of the VM instance i (on u) of the node u, respectively. e_u^N is the cost of energy consumed by node u at a utilization of 100%. $e_C(t)$ is the cost of energy consumption by the network at time t. The total cost of energy consumption te_C of the network is the sum of energy consumption of each individual node in various states over a period of time. Our objective is to *Minimize* te_C. The set of operational constraints to be noticed are.

1) Flow Constraints: The inequality in Equation (7) ensures that the flow from node u to node v must be positive. Equation (8), ensures that the total flow along each link should not exceed the total capacity of that link. The flow conservation constraint is shown in Equation (9), where r_k unit of traffic is created in its source, and is destroyed in its destination. For a stable system, the limit of the utilization of each node and links lies between [0, 1]. Equation (10) ensures the utilization limit of each PM.

$$F_i(u,v) \geq 0, \ \forall i, \ F_i \in K, \ \forall (u,v) \in L, \tag{7}$$

$$\sum_{i=1}^{k} F_i(u,v) \leq C^L(u, \ v) * l(u, \ v), \ \forall (u,v) \in L, \tag{8}$$

$$\sum_{(u,v) \in L} F_k(u, \ v) - \sum_{(v,u) \in L} F_k(v,u) = \begin{cases} r_k, & \text{if } u = s_k, \\ -r_k, & \text{if } u = t_k, \\ 0, & \text{otherwise}, \end{cases} \tag{9}$$

$$0 \leq \frac{\lambda}{c\mu} \leq 1. \tag{10}$$

2) Capacity Constraints: The inequality in Equation (11) ensures that the total sum of the capacities of VNFs on node u must be less than or equal to 'utility capacity' of node u, i.e., the difference between maximum capacity and default capacity of u. The variable

in Equation (12) shows the function f of service chain sc is placed on node u. The Equation (13) inequality ensures that the demand of function f of service chain sc at node u must be less than or equal to the available capacity of node u. The next inequality in Equation (14) presents the demand of function f of service chain sc at node u less than or equal to the capacity of the VM instance i of the node u.

$$\sum_{i \in vM(u)} C_i^V(u) \leq C^N(u) - C^I(u), \ \forall u \in N, \tag{11}$$

$$Y_f^{sc}(u) = 1, \ \forall u \in N, \ f \in vF, \ sc \in sC, \tag{12}$$

$$\sum_{sc \in sC} d_f^{sc}(u) \leq C^N(u) - \sum_{i \in vM(u)} C_i^V(u) - C^I(u), \quad \forall u \in N, \ f \in vF, \ sc \in sC, \tag{13}$$

$$d_f^{sc}(u) \leq C_i^V(u), \ \forall u \in N, \ f \in vF, \ sc \in sC. \tag{14}$$

3) Placement Constraints: The binary variable in Equation (15) shows the function f placed on the ith VM instance of node u. The inequality in Equation (16) shows the queuing delay of function f of service chain sc must be less than or equal to the maximum delay at node u. Equation (17) ensures that the delay faced by a flow along its path must be less than or equal to the maximum delay the flow can tolerate. Equation (18) checks the status of the node for the placement of the function. It consists of three parts. The first inequality checks whether the node is ACTIVE or not, and the second part checks the node is IDLE or not. The last part checks if the node is OFF or not, and whether the demand on the node exceeds the threshold value.

$$X_i^f(u) = 1, \ \forall u \in N, \tag{15}$$

$$qd_f^{sc}(u) \leq d_{MAX}(u), \ \forall u \in N, \ f \in vF, \ sc \in sC, \tag{16}$$

$$\sum_{(u,v) \in P_k} d_{F_k}(u,v) \leq D_k, \ \forall k, \ F_k \in K, \tag{17}$$

$$\left[\left[Y_f^{sc}(u) \leq \sum X_i^f(u)\right] \middle\| \left[\left[E^N(u) = \frac{C^I(u) * e^N(u)}{C^N(u)}\right] \middle\| \left[\left(E^N(u) = 0\right) \ \& \left(\sum_{sc \in sC} d_i^{sc}(u) \geq thresold\right)\right]\right]\right] = 1, \quad \forall u \in N, \ \forall f \in vF, sc \in sC, \ thresold \text{ is a constant}. \tag{18}$$

C. Problem Analysis

The optimization problem we formulated in this paper can be shown to be NP-hard, by reducing the standard Virtual Network Embedding (VNE) problem [43], which is known to be NP-hard, to our problem in polynomial time.

In the first step, we describe the mapping of virtual networks to a physical network with an example, and then we state the VNE problem, which is an existing NP-hard problem. In the second step, we redefine our optimization problem to a decision problem and later demonstrate that the VNE problem could be reduced to our problem. Figure 4 depicts a scenario of virtual and physical network mapping. It consists of two virtual networks and one physical network. The capacity of each node and links (both physical and virtual) are given in Figure 4. Two virtual networks are mapped to the physical network in such a way that the sum of the capacities of the virtual nodes/links on a physical node/links must be less than or equal to the capacity of that physical node/links.

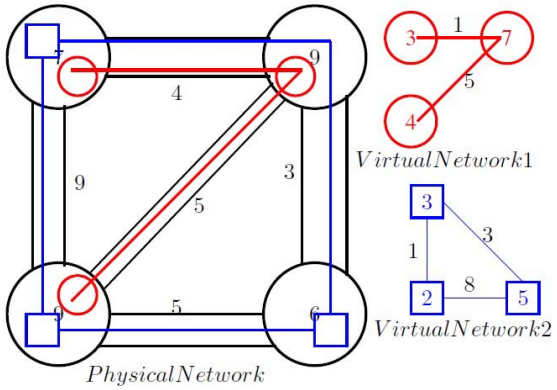

Fig. 4. Two virtual networks mapped onto one physical network [43].

1) Virtual Network Embedding (VNE) Problem: Given an undirected graph $G_P = (U_P, E_P)$, where U_P is the set of vertices and E_P is the set of edges. Each vertex $u_i \in U_P$ is assigned a capacity $C_P(u_i)$ and each edge $(u_i, u_j) \in E_P$, $u_i, u_j \in U_P$ has a bandwidth $b_P(u_i, u_j)$. Given another undirected graph $G_V = (W_V, E_V)$, where W_V is the set of vertices and E_V is the set of edges. Each vertex $w_k \in W_V$ is assigned a capacity $C_V(w_k)$ and each edge $(w_k, w_l) \in E_V$, $w_k, w_l \in W_V$ has a bandwidth $b_V(w_k, w_l)$.

The problem is to determine whether or not we can find a set of valid mapping from E_V to E_P. In each mapping from edge $(w_k, w_l) \in E_V$ to $(u_i, u_j) \in E_P$, two conditions are required to be satisfied:

1. $C_P(u_i) \geq C_V(w_k)$, and $C_P(u_j) \geq C_V(w_j)$,
2. $b_P(u_i, u_j) \geq b_V(w_k, w_l)$.

Theorem 4: Our optimization problem is NP-hard.

Proof: Please refer to Appendix G. ■

V. Solution Approach

In this section, we will propose the dynamic placement of the VNF chains heuristic algorithm. This placement method reduces the number of active[1] nodes in the network. We use a restricted spanning tree mechanism for the placement of the VNF. To reduce the energy cost, we select the path for the flow, which has more active nodes, and fewer hop counts from the source to destination. For example, in the network (Figure 1), we have a new flow (*say SC5*) from source 1 to destination 6, with a service chain demand (B-D-A). After placement of the first two functions, B and D on node 2, and 4 respectively, we have two options for the placement of A. If we place A on node 3, we have to turn it ACTIVE, which will increase energy consumption. We can minimize energy consumption by placing A on node 7, which is in the ACTIVE state, and redirect the flow to the destination via node 9. This algorithm consists of three stages. The first stage is the DPVC algorithm presented in Algorithm 1. It takes the input in each iteration of the loop and calls the *Placement* function (given in

[1]Node is either in the ACTIVE or IDLE state.

Algorithm 1 DPVC Algorithm

Input: A, B, ST, VM_{cap}, $Total_{cost}, Idl_{max}, U_{max}$, U_{idl}, vNF, pt, Idl_{time}, min_{cap}, E_c, $NumFlow$, $boot$.
Algorithm:
$VM = struct(VM_{flg}, VM_{fun}, VM_{exp}, VM_{wait}, VM_{flow})$
$ServiceChain = struct(source, chain, destination, FLOW_{len}, FLOW_{num})$
$ChainTime = struct(startTime, pTime, preSource, chainDest, hop, endTime)$
$CurrInp = struct(chainNum, currSource, currVNF, currDest, currFLOW_{len}, FLOW_{no}, ETime)$
for *each iteration* t
  $ServiceChain = ServiceChainIP(ServiceChain, t)$
  $ChainTime = setChainTime(CurrInp, ServiceChain, ChainTime, t)$
  $CurrInp = CurrVNFinput(CurrInp, ServiceChain, ChainTime, t)$
  $Placement(CurrInp, ChainTime, nodes, A, B, ST, VM, t)$
  for *each node* i *in ST*
    for *each VM* j *in node* i
      if $t > VM_{exp}(i,j)$
        Release(VM, i, j)
      end
    end
    if $(i \in ST)$ & $(U_{ass}(i) == 0)$
      $Idl_{time}(i) = Idl_{time}(i) + 1$;
      if $Idl_{time}(i) \geq Idl_{max}$
        Delete(ST, i)
      end
    end
  end for
  $VM = updateVM(VM)$
  $cost = \sum_{i=1, i\in ST}^{N} E_c(i) * \frac{U_{idl}(i) + \sum VM_{cap}(i,j) * VM_{flg}(i,j)}{U_{max}(i)}$
end for
Output: $Total_{cost} = \sum cost$

Algorithm 2 Placement Algorithm

$Placement(CurrInp, ChainTime, nodes, A, B, ST, VM, t)$
for $i = 1 \rightarrow |CurrInp|$ **do**
  $s = CurrInp(i).currSource$;
  $d = CurrInp(i).destination$;
  $nf = CurrInp(i).currVNF$;
  $RDFST(nodes, A, B, ST, VM_{flg}, VM_{exp}, s, d, nf, t)$
  $ChainTime = updateChainTime(ChainTime, CurrInp, newNode)$
end for
Return: ST, VM, $ChainTime$

Algorithm 3 RDFST Algorithm

$RDFST(nodes, A, B, ST, VM, s, d, fun, t)$
$Index = nodeWithfun(B, fun)$
$nodes = struct(num, CapAct, sNode, spd)$
$nodes = assignPrioritytoNodes(nodes, VM, s, d)$
$x = totalVMInstances(VM)$
$y = x - 1$;
While $y < x$ **do**
  $nodes_{sorted} = nestedSortStructure(nodes, \{CapAct, sNode, spd\})$
  for $i = 1 \rightarrow |Index|$ **do**
    if $nodes_{sorted}(i).num \leq 0$ **then**
      Display ('*Error!*');
    else
      $nN = nodes_{sorted}(i).num$;
      Assign$(ST, nN, VM, fun, NumFlow, boot)$
      if $nN \in ST$
        exit; ;
      else if $U_{ass}(nN) \geq min_{cap}$
        Add(ST, nN)
        exit; ;
      else if $max(VM_{wait}(nN, 1), VM_{wait}(nN, 2), \ldots) \geq off_{time}$
        Add(ST, nN)
        exit;
      else
        Display('*Wait!*');
        exit;
      end
    end
  end
  end if
end for
$y = totalVMInstances(VM)$
end while
Return: ST, VM, nN

Algorithm 2). The *Placement* function will take a set of VNFs as input and call the *Restricted Depth First Spanning Tree* (RDFST) function (given in Algorithm 3) for each individual VNF, and find the appropriate location for placement.

We randomly generated a connected graph (matrix A) consisting of a set of nodes and links, of equal weight. We assign different types of VNFs to each node randomly presented by matrix B, *i.e.*, the rows of the matrix present the nodes and columns existing in the network functions. $B(i, j) = 1$, *if* the ith node of the network has the jth function, *else* 0. ST is the *spanning tree*. U_{max} and U_{idl} are the arrays presented as the maximum capacity and default capacity of each node in the graph, respectively. vNF is the set of functions, and pt is the processing time of each function. Idl_{time} is the amount of time the node can stay IDLE. If it does not receive any function, during this time limit it will turn OFF. off_{time} is the maximum amount of time a VNF can wait in an OFF PM. If the amount of time is exceeded the limit, the PM will turn ON. VM_{cap} is the capacity of each VM instance. We are using the *spanning tree* concept in our algorithm. Here, if a machine turns ACTIVE, we will add it to the spanning tree, and if an active machine turns OFF, we will remove it from the spanning tree. We are using two sets of operations (*Add* and *Delete*) in our algorithm to handle this. When a machine turns ACTIVE, we use the *Add* operation to add that machine to the spanning tree, and when a machine turns OFF, we use the *Delete* operation to remove it from the spanning tree. We are using two more operations such as *Assign* and *Release* for the placement of a VNF. When a new VNF is placed on the machine, by the *Assign* operation, we provide resources to that VM instance. If a running VNF terminates by the *Release* operation, we release the assigned resources of that VM instance, which can be assigned to a new VNF. The definitions of these operations are as follows.

Definition 1: [Add] if ST is an arbitrary set, $u \notin ST$ is an arbitrary element, where $ST = \{u_i : i \in I\}$, I is an Index set, then we define $Add(ST, u) = ST \cup \{u\}$.

Definition 2: [Delete] if ST is an arbitrary set, $u \in ST$ is an arbitrary element, where $ST = \{u_i : i \in I\}$, I is an Index set, then we define $Delete(ST, u) = ST - \{u\}$.

Definition 3: [Assign] if u^i is an arbitrary set and i is the number of elements in u, and $j \notin u$ is an arbitrary element, then we define $Assign(u^i, j) = u^{i+1}$.

Definition 4: [Release] if u^i is an arbitrary set and i is the number of elements in u, and $j \in u$ is an arbitrary element, then we define $Release(u^i, j) = u^{i-1}$.

The DPVC algorithm works as follows: First, we generate four structures named, *ServiceChain*, *ChainTime*, and *CurrInp*. The structure *VM* consists of five fields. VM_{flg} shows whether the VM is ON or OFF, VM_{exp} presents the termination time of the VM, and VM_{fun} presents the network function running in the VM. VM_{wait} shows the waiting time, and VM_{flow} shows a number of flows are sharing that VNF. The structure *ServiceChain* consists of five fields, *i.e.*, the *chain* presents the service chain. The *source*, *destination*, $Flow_{len}$, and $Flow_{num}$ represent the source, destination, length, and number of the flows, respectively. The structure *ChainTime* consists of six fields. The first field *startTime* holds the start time of each VNF of the service chain and the second field *pTime* shows the processing time of each VNF of the service chain, and the third one is the *preSource*, *i.e.*, the node where the previous VNF of the service chain was placed. Initially, *preSource* is the chain source. *chainDest* shows the destination of the flow. *hop* and *endTime* present the end-to-end number of hop and termination time of the flow, respectively. *CurrInp* is the structure, which holds a set of VNFs for the current iteration for placement. After placement, the structure will discard all values of the structure. This structure consists of seven fields, *i.e.*, *currVNF* shows the VNF name, *currSource* shows its source, *currDest* shows its destination, *chainNum* shows which service chain the VNF belongs to, $currFLOW_{len}$ shows the flow length, $Flow_{no}$ shows the flow number, and *ETime* shows the termination time of the flow. After creation of the structure for each iteration, we do the following: We take as a maximum one flow and its service chain as an input and set its service time by *setChainTime* function. By *CurrVNFinput*(), we select the VNFs from different existing service chains for placement. Then, we call the placement function for the *Placement* of the selected VNFs. We check the termination time of all the VM instances of each active node. If any VNF terminates, we *Release* them. We also check the *idle-time* of each IDLE node, if the *idle-time* exceeds the *maximum idle-time*, we turn that node OFF. We calculate the energy consumption cost of the system for each iteration by considering the status (ACTIVE, IDLE, OFF) of each node and the number of VNFs on them. After each loop iteration, we update the structure *VM*.

In the *Placement* algorithm, we retrieve each VNF (*nf*) and their current source node (*s*), *i.e.*, where the previous function of that service chain has been placed and their destination node (*d*). Then, we call the RDFST function for the placement of each VNF. After placement of the VNF, the chain time of the service chain gets updated. After placement of all VNFs, the Placement function returns the values to the DPVC algorithm.

The *RDFST* algorithm works as follow. First, we retrieve the nodes that contain the required service function (*fun*) using *nodeWithfun*(). We assign priority to these nodes by the function, *assignPrioritytoNodes*. Here, if the same node has availability for the new function, then it will be given the highest priority. Second priority will be given to the other active nodes with availability. Third priority will be given to the non-empty OFF nodes, and fourth priority will be allocated to empty OFF nodes. If two nodes have the same priority, then preference will be given to the node with the minimum shortest path distance (*spd*). Here *spd* is calculated by adding the shortest path from the current source (*s*) to the node and from the node to the destination (*d*). By using *structural sorting,* we sort the nodes based on their priority, retrieve the most suitable node (*nN*) for the placement of the VNF from the sorted structure ($nodes_{sorted}$), and *Assign* the VNF (*fun*) to that node and add the boot time if the VM is OFF. If the node is not ACTIVE, we check to see if the assigned capacity of that node exceeds the minimum capacity (min_{cap}) or not. If the minimum capacity has been exceeded, then we turn that node ACTIVE. Then, by the *Add* operation, we add the node to the spanning tree (*ST*). Otherwise, we check the waiting time of all the VMs. If the waiting time of any VM exceeds the maximum waiting time (off_{time}), we turn that node ACTIVE. After successful placement of a VNF, the *RDFST* function returns the value to the *Placement* algorithm.

TABLE IV
EXPERIMENT PARAMETERS

Item	Description/Value
Graph type	*Partially mesh*
Service chain demand	*Random*
Flow length	*10-100 packets*
Packets length	*Equal*
Type of VNFs	*Minimum 3 and Maximum 10*
No. of VNF in a Service Chain	*Minimum 5 VNFs and maximum 14 VNFs*
Capacity of nodes	*Equal*
Maximum idle-time of node	*3*
off-time (off_{time})	*3*
boot-time ($boot$)	*5*
No of flow request per unit time	*Maximum 1*
Energy consumption by nodes in the IDLE state	*30% , 40%, and 50%*

VI. PERFORMANCE EVALUATION

In this section, we will discuss the experimental setup, which is used in this paper to evaluate our proposed algorithms. In this experiment, we considered multiple partially meshed networks where the network does not have a predefined structure for service chain placement. Through this experiment, we demonstrate the performance of our algorithm. As our design and objective are different from the existing VNF placement papers, we compare our DPVC algorithm with random [19] and first-fit [21] placement algorithms. In the random placement algorithm (RND), we randomly select a node with sufficient capacity for the placement of the function. In the first-fit placement algorithm (FF), we select the first node with available capacity for the placement of the function.

A. Experiment Setup

We used MATLAB to compare the performance of the algorithms, Table IV shows the details of the experimental parameter used in the simulated scenario for this work. For this

simulation, we considered the randomly generated partially meshed networks. Randomly generated flows[2] are given as the input from a set of source nodes to a set of destination nodes, where for each flow, the source and destination nodes are not equal. The length of the flows is 10–100 packets, and all packets are of equal size. For each flow, the service chains are randomly generated of lengths consisting of 5 to 14 VNFs. We considered 10 different types of network functions out of which 9 are the general functions (VNF *remains active until all packets of the flow get processed*) and one is a special function (VNF *remains active until flow reaches the destination node*). Different general VNFs have different processing times and can appear one or more times in a single service chain. If a VNF has a processing time of 20 packets/sec, then it will take 4 sec to process a flow of 80 packets. The special VNF can appear a maximum of one time in a service chain and remain active until the flow reaches the destination node. Each service chain contains a minimum of 3 different types of functions. Placement of a service chain's VNFs is sequential, *i.e.*, $(i+1)$th VNF of the service chain can be placed only after completion of the ith VNF of that service chain. If the ith VNF is a special one, then the VM will remain active until the flow reaches the destination. The $(i+1)$th VNF of the service chain can be placed immediately after the VM is available and packets are ready for processing. After the placement of a special VNF of a service chain, the next VNFs of that chain can be placed on the same node along with the special VNF, if that function is available on that node and the node has available capacity for placement. For example, in Figure 1, we consider 'C' as a special function, and we have a flow from node 5 to 6 with service chain demand C-B-A. The first VNF 'C' will be placed on node 9 and will remain active until the flow reaches the destination node 6. The second VNF 'B' can be placed on node 9 if the node has available capacity. This is a case where multiple VNFs of the same chain run on the same node. However, 'B' will remain active until all packets of the flow get processed. Without loss of generality, we assume a service chain demand of a flow at the system will terminate only after all the packets of the flow get processed by the respective VNFs, and the flow reaches the destination nodes, whereby all flows are not able to split. All nodes are of equal capacity. After releasing all the VNF instances, the nodes can stay IDLE for duration of *maximum idle-time*, within this period, if new VNF instances are assigned to the IDLE machine, it will turn ACTIVE or else it will turn OFF. Because energy consumption in the IDLE state is a big issue, in our evaluation, we have considered three different cases, *i.e.*, the IDLE node consumes 30%, 40%, or 50% of the energy of the maximum energy consumption of the node during full utilization. When new VNFs are placed on an OFF PM and within *off-time* duration after placement of the first VNF, if the PM is unable to get the *minCap* value, it will turn ACTIVE, which will minimize the waiting time of the VNFs already in the queue.

[2]In this paper, we assume short flows (generated by user tasks that have a short duration [49]).

B. Results Analysis

In this section, we will demonstrate the performance of the algorithms under multiple topologies. In this evaluation, we have considered all three cases of energy consumption of the nodes in the IDLE state.

1) Energy Consumption Cost Analysis: Figure 5(a) presents the total energy consumption cost of the networks. Total energy consumption cost is nothing but the sum of the energy consumption cost of the network after each iteration. We check the status of the nodes and amount of VM instances on them after each iteration. As the result, in Figure 5(a), shows, our DPVC saves nearly 45% and 65% more on costs than the *FF* and *RND*, respectively. As the input of the number of flows increases, the total energy consumption cost difference between the algorithms, continue to increase. Figure 5(b) shows the variation of energy consumption cost in each iteration. The result shows the DVCP consumes less energy compared to other algorithms, because it always gives priority to select active PMs for the placement of VMs instead of OFF PMs.

Figure 5(c) shows the average energy consumption cost by the network per flow. The result in Figure 5(c) clearly shows that the average energy consumption cost in the DPVC is relatively less than in other algorithms due to its node selection process, which gives priority to select the active nodes for the placement of the VNF. This process minimizes the number of active nodes in the network, increases the utilization, and, as a result, the cost decreases. Figure 5(d) shows the average number of end-to-end hops per flow. The number of hops in the FF is less than the number in the DVCP algorithm, as it selects the shortest available node for the placement of function. However, its energy consumption is very high compared to the DVCP, as shown in Figure 5(c). In the DVCP algorithm we select the path which contains a greater number of active nodes for the placement of VNFs, instead of the shortest path. Hence, in the DVCP, the hop count is more, but energy consumption is less.

2) Utilization of Active Nodes: Figure 6(a) shows the average utilization of active PMs, that are not in the OFF state, *i.e.*, we assume the IDLE machines are also active here, as they consume default amount of energy. Average utilization refers to the mean utilization of all nodes in the ACTIVE or IDLE state. For example, a network consists of five nodes, if three nodes are in the ACTIVE state with a utilization of 40%, 60%, and 80%, one node is in the OFF state and consumes no energy, and one node is in the IDLE state with 0% utilization of energy. Then the average utilization of the active nodes of the network can be $(40\% + 60\% + 80\% + 0\%)/4 = 45\%$. As the result shows, in Figure 6(a), the utilization of the active nodes is relatively 45% more than other algorithms. This is because, in the DPVC, the percentage of active nodes in the network is relatively less, as presented in Figure 6(b). The percentage of active nodes of a network means that in a network with 50 nodes, if 15 nodes are either in the ACTIVE or IDLE state at the time t_1, then we consider the percentage of active nodes to be 30% at t_1. The percentage of the active nodes in the DVCP is less because by the RDFST method, it primarily selects the ACTIVE or IDLE nodes for

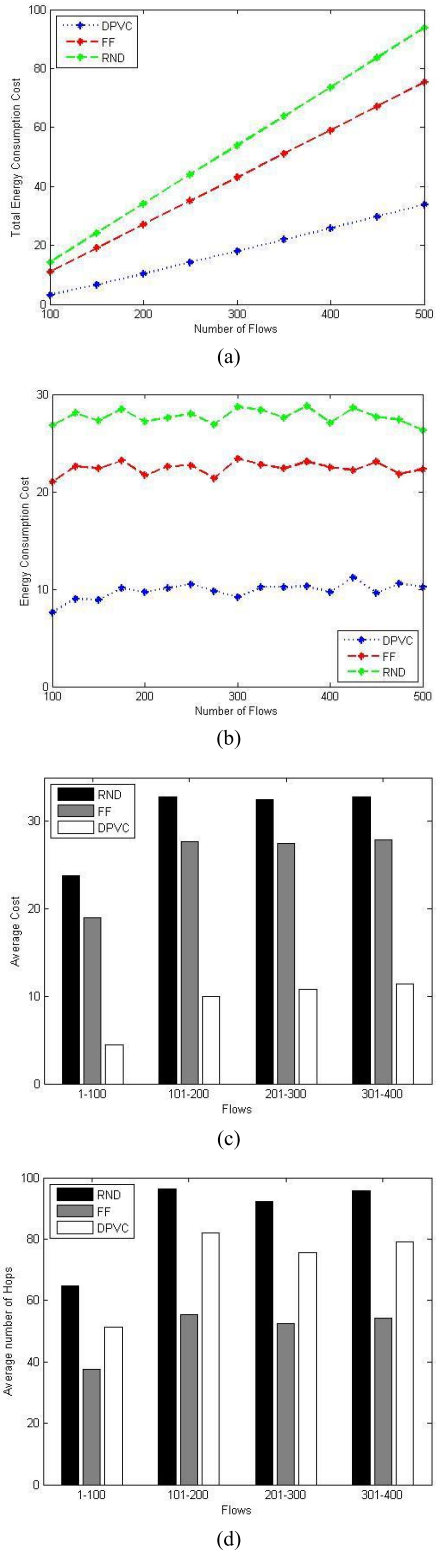

Fig. 5. (a) Total energy consumption cost of the network per number of flows. The DPVC algorithm saves more on cost than other algorithms. (b) Energy consumption cost variation per flows. (c) Average energy consumption cost per flows. The DPVC saves more on cost than others. (d) Percentage of the average number of hops from the ingress node to the egress node per flows.

the placement of the VNFs rather the OFF nodes. This minimizes the number of nodes in the OFF state that turn ACTIVE, whereas, the *RND* method selects the node randomly and

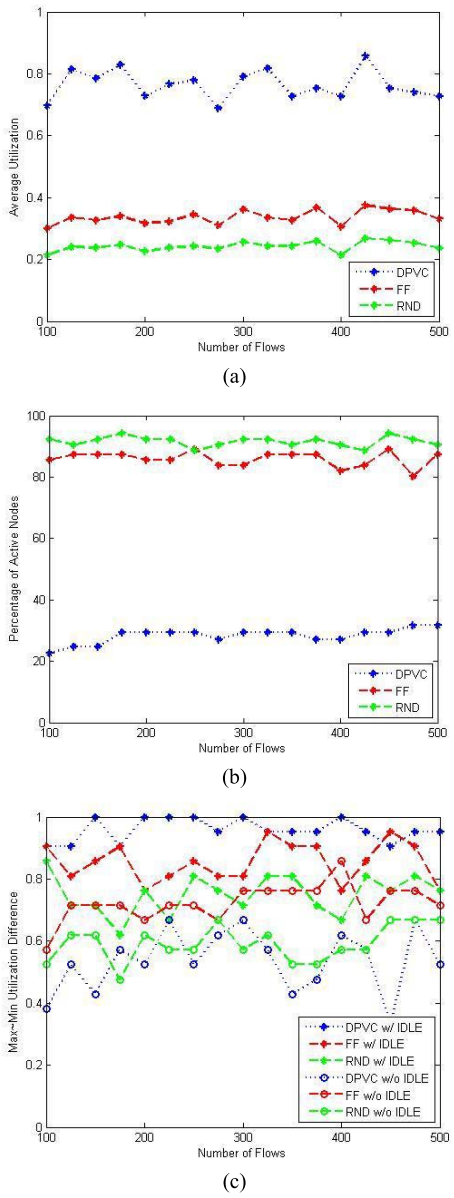

Fig. 6. (a) Average utilization of active nodes per flow. The DPVC utilizes its nodes better than other algorithms. (b) Percentage of active nodes in the networks per flow. The DPVC has fewer active nodes. (c) Average utilization variation of the active nodes 'w/' and 'w/o' IDLE nodes.

the *FF* selects the first available node for the placement of the VNF.

Figure 6(c) shows the difference between the maximum and minimum average utilization of the active node. The network experiences the highest variation when a node is in the IDLE state (0% utilization) and another node is fully utilized (100%). In the DVCP algorithm, the IDLE node remains IDLE for a specific duration before turning OFF if no new VNF is assigned. To switch an OFF PM to an ACTIVE state, the DVCP requires a certain *minCap* value, which increases the utilization. Via the RDFST method, we always try to place a VNF in an active node rather than in an OFF node. Hence, the DVCP experiences more utilization variation than other algorithms with IDLE nodes utilization. However, the variation

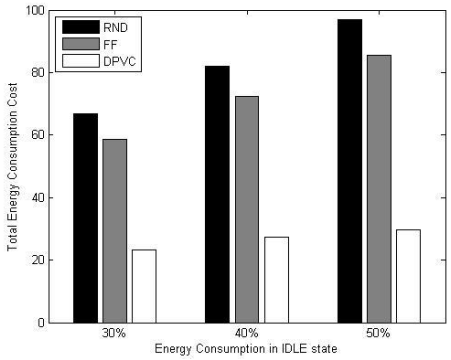

Fig. 7. Total energy consumption cost on different default consumption in the IDLE state. As default consumption increases, cost increases.

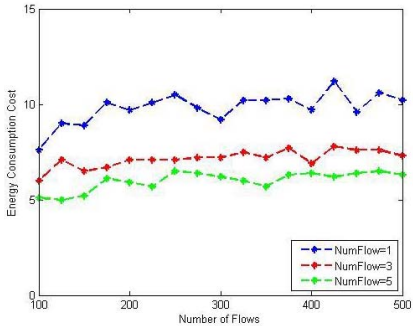

Fig. 8. Energy consumption cost per number of flows in the DVCP algorithm with each VNF shared by multiple flows. As the more flows shared by the VNF, energy consumption cost decreases.

is significantly less when we exclude the utilization of the network's IDLE nodes.

3) Performance Changes With Different Default Energy Consumption in the IDLE State: Energy consumption is one of the biggest concerns in our research. In Figure 7 we presented the results of the total energy consumption cost of the network in different percentages of default energy consumption in the IDLE state. As the results show, as the amount of energy consumption in the IDLE state increases, the total energy consumption also increases. The greater the number of IDLE nodes in the network, the more the unutilized energy consumption exists. As the default energy consumption in the IDLE state increases, the total energy consumption increases. At the same time, our DPVC algorithm saves more on cost than other algorithms in all three cases.

4) Performance Changes With Different Flow Sharing Limit: Figure 8 shows the performance of the DVCP algorithm on different flow sharing limits (*how many numbers of flows can share a VNF together?*). For example, if maximum 5 flows can share a VNF at a time, then flow sharing limit is 5. In all the previous results, we considered the maximum sharing limit 1. However, a VNF can be shared among different flows together. The results in Figure 8 show that by increasing the sharing limit of the VNFs, the energy consumption of the network reduces significantly. By increasing the VNF's flow sharing limit from 1 flow to 5 flows, the energy consumption decreases nearly 30%–35%.

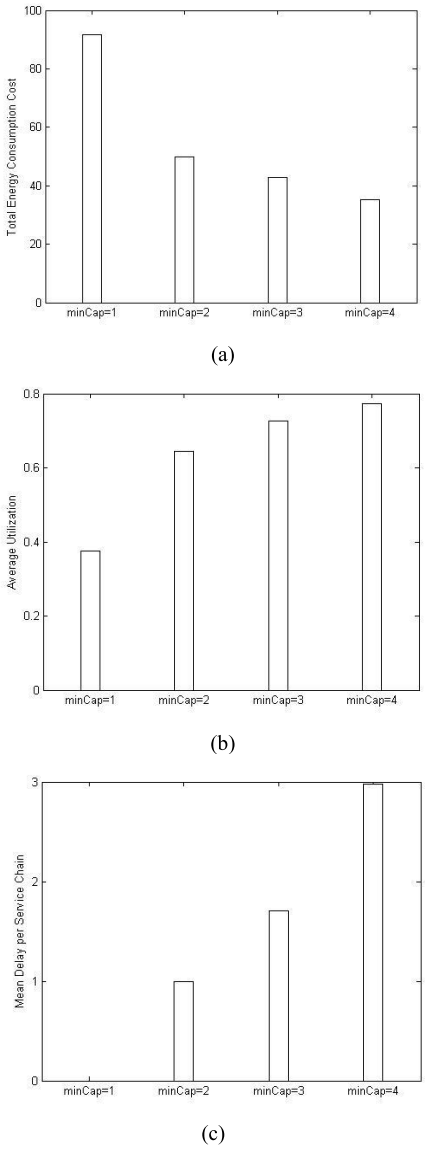

Fig. 9. The DPVC algorithm with different minimum capacity (*minCap*) required to turn ACTIVE, the OFF nodes. With increase in *minCap* (a) total energy consumption cost decreases, (b) utilization increases, and (c) mean delay per service chain increases.

5) Performance Changes With Different minCap Value: Figure 9 shows the performance of the DPVC algorithm on different *minCap* values. The *minCap* value is the minimum capacity required to turn the node in on OFF state to an ACTIVE state. As we have considered the capacity of the VM instances to be equal, so we considered the minimum number of VM instances required to turn a machine in an OFF state to an ACTIVE state. This value significantly affects the performance of the network. It minimizes the number of active nodes and significantly increases the utilization of the network. Figure 9(a) and 9(b) show the total energy consumption cost and average utilization of the network on a different *minCap* value, respectively. By increasing the *minCap* value from 1 to 4, the energy consumption cost decreases by nearly 50 percent, and average utilization increases by nearly

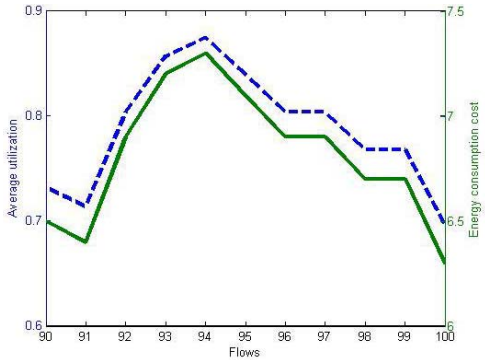

Fig. 10. Energy consumption of network increases/decreases with increase/decrease of the average utilization of the active nodes.

40 percent. Figure 9(c) shows the mean queuing delay of VNFs per service chain. With the increase in *minCap*, the delay increases. However, compared to the good energy saving performance, this delay can be negligible.

6) Intermediate Results Analysis: Figure 10 shows the intermediate results of VNF placement algorithms. We described these results as intermediate results because they are based on a single network; whereas other previous results are on multiple networks of the DPVC algorithm. Here we retrieve the status of the system for ten-iterations when the percentage of active nodes in the network remains unchanged. However, the energy consumption cost changes with the change of utilization of the active nodes. As shown in Figure 10, the energy consumption cost of the network increases/decreases with increase/decrease the utilization of the active nodes.

VII. Conclusion

In this paper, we analyzed the energy consumption issue in the network function virtualization network. We proposed an energy-saving model using an M/M/c queuing network. We formulated an optimization problem to minimize the total energy consumption cost of the network, which proved to be NP-hard. Our proposed algorithm can be used to determine the most suitable PMs for the placement of VNFs to minimize the energy consumption of the network. By normalized PM and VM cost estimation, we found that the energy consumption cost of the network depends on the utilization of the active nodes. We reduced the unutilized nodes of the network by using the minimum capacity policy. Via MATLAB experimentation, we found that our algorithm saves nearly 40% more total energy consumption cost while processing 500 flows. It also minimizes the number of active nodes in the network and maximizes the utilization of the active nodes by 40%–50%.

In this paper, the VNF chains placement is limited to only short flows and single source and single destination pairs. However, we can handle the long flows (generated by applications with long duration [49]), by avoiding sequential processing of flows in general VNFs, as a result, the processing time will not become an issue to process the long flows and a single flow can be processed simultaneously by multiple VNFs. In our future research work, the long flows and flow splitting scenario will be discussed.

Appendix

A. Lemma 1

$$\sum_{n=c}^{\infty} P_{1,n} = \frac{\theta_1}{\theta_2} \cdot P_{0,0}.$$

Proof: From Equation (6), we have,

$$(\theta_2 + \lambda) \cdot P_{1,n} = \lambda \cdot P_{1,n-1}, \ \textit{where } n = c, \ c+1, \ \ldots, \ \infty.$$

Hence,

$$\begin{aligned} P_{1,n} &= \frac{\lambda}{\lambda+\theta_2} \cdot P_{1,n-1} = \left(\frac{\lambda}{\lambda+\theta_2}\right)^{n-c+1} \cdot P_{1,0} \\ &= \left(\frac{\lambda}{\lambda+\theta_2}\right)^{n-c+1} \cdot \frac{\theta_1}{\lambda} \cdot P_{0,0} \\ &= \frac{\theta_1}{\lambda+\theta_2} \cdot \left(\frac{\lambda}{\lambda+\theta_2}\right)^{n-c} \cdot P_{0,0} \end{aligned}$$

So,

$$\begin{aligned} \sum_{n=c}^{\infty} P_{1,n} &= \sum_{n=c}^{\infty} \frac{\theta_1}{\lambda+\theta_2} \cdot \left(\frac{\lambda}{\lambda+\theta_2}\right)^{n-c} \cdot P_{0,0} \\ &= \frac{\theta_1}{\lambda+\theta_2} \cdot P_{0,0} \cdot \sum_{n=c}^{\infty} \left(\frac{\lambda}{\lambda+\theta_2}\right)^{n-c} \\ &= \frac{\theta_1}{\lambda+\theta_2} \cdot P_{0,0} \cdot \left[1 + \frac{\lambda}{\lambda+\theta_2} + \left(\frac{\lambda}{\lambda+\theta_2}\right)^2 \right. \\ &\qquad \left. + \left(\frac{\lambda}{\lambda+\theta_2}\right)^3 + \cdots + \infty\right] \\ &= \frac{\theta_1}{\lambda+\theta_2} \cdot P_{0,0} \cdot \frac{1}{1-\frac{\lambda}{\lambda+\theta_2}} \\ \sum_{n=c}^{\infty} P_{1,n} &= \frac{\theta_1}{\theta_2} \cdot P_{0,0}. \end{aligned}$$

■

B. Proof of Theorem 1

From the derived equations (Equation (4) and Equation (5)), we have,

$$\begin{aligned} P_{1,0} &= \frac{\theta_1}{\lambda} \cdot P_{0,0} \\ P_{1,0} &= P_{1,1} = P_{1,2} = \ldots = P_{1,c-1} \end{aligned}$$

So

$$\sum_{n=0}^{c-1} P_{1,n} = c \cdot \frac{\theta_1}{\lambda} \cdot P_{0,0}. \tag{19}$$

$$\sum_{n=0}^{\infty} P_{1,n} = \sum_{n=0}^{c-1} P_{1,n} + \sum_{n=c}^{\infty} P_{1,n} \tag{20}$$

Putting the values from Equation (19) and Lemma 1 in Equation (20), we have,

$$\begin{aligned} \sum_{n=0}^{\infty} P_{1,n} &= c \cdot \frac{\theta_1}{\lambda} \cdot P_{0,0} + \frac{\theta_1}{\theta_2} \cdot P_{0,0} \\ &= \left[\frac{c \cdot \theta_1}{\lambda} + \frac{\theta_1}{\theta_2}\right] \cdot P_{0,0}. \end{aligned}$$

C. Lemma 2

$$\sum_{n=1}^{\infty} P_{0,n+c} = \frac{P_{0,0}}{c\mu} \cdot [\alpha - \lambda].$$

Proof: From the Equation (2), we have,

$$c\mu \cdot P_{0,n+c} = (\lambda + c\mu) \cdot P_{0,n} - \lambda \cdot P_{0,n-1}$$

Putting $n = 1, 2, \ldots, c-1, c, \ldots K$, where $K \approx \infty$, we will have a series of equations,

$$\begin{aligned}
c\mu \cdot P_{0,c+1} &= (\lambda + c\mu) \cdot P_{0,1} - \lambda \cdot P_{0,0}, \ n = 1 \\
c\mu \cdot P_{0,c+2} &= (\lambda + c\mu) \cdot P_{0,2} - \lambda \cdot P_{0,1}, \ n = 2 \\
c\mu \cdot P_{0,c+3} &= (\lambda + c\mu) \cdot P_{0,3} - \lambda \cdot P_{0,2}, \ n = 3 \\
&\vdots \\
c\mu \cdot P_{0,c+c-1} &= (\lambda + c\mu) \cdot P_{0,c-1} - \lambda \cdot P_{0,c-2}, \ n = c-1 \\
c\mu \cdot P_{0,c+c} &= (\lambda + c\mu) \cdot P_{0,c} - \lambda \cdot P_{0,c-1}, \ n = c \\
&\vdots \\
c\mu \cdot P_{0,c+K} &= (\lambda + c\mu) \cdot P_{0,K} - \lambda \cdot P_{0,K-1}, \ n = K
\end{aligned}$$

Adding these K number of equations we have,

$$\begin{aligned}
&c\mu \cdot \sum_{n=1}^{K} P_{0,n+c} \\
&= c\mu\big[P_{0,1} + P_{0,2} + \cdots + P_{0,c} + P_{0,c+1} + \cdots + P_{0,K}\big] \\
&\quad + \lambda \cdot P_{0,K} - \lambda \cdot P_{0,0} \\
&= c\mu \cdot \big[P_{0,1} + P_{0,2} + \cdots + P_{0,c}\big] \\
&\quad + c\mu \cdot \big[P_{0,c+1} + P_{0,c+2} + \cdots + P_{0,2c}\big] \\
&\quad + c\mu \cdot \big[P_{0,2c+1} + \cdots\big] \ldots + \lambda \cdot P_{0,K} - \lambda \cdot P_{0,0}
\end{aligned}$$

Assuming $c\mu \cdot [P_{0,1} + P_{0,2} + \cdots + P_{0,c}]$

$$\approx \mu\big[P_{0,1} + 2P_{0,2} + \cdots + c \cdot P_{0,c}\big]$$

by Equation (1), we have

$$= \mu \cdot \sum_{n=1}^{c} n \cdot P_{0,n} = (\lambda + \theta_1) \cdot P_{0,0},$$

Hence,

$$\begin{aligned}
&c\mu\big[P_{0,1} + P_{0,2} + \cdots + P_{0,c} + P_{0,c+1} + \cdots + P_{0,K}\big] \\
&= (\theta_1 + \lambda)P_{0,0} + 2 \cdot (\theta_1 + \lambda)P_{0,0} + \cdots + \frac{K}{c} \cdot (\theta_1 + \lambda)P_{0,0} \\
&= (\theta_1 + \lambda) \cdot P_{0,0}\left[1 + 2 + \cdots + \frac{K}{c}\right] \\
&= \frac{(\theta_1 + \lambda) \cdot K \cdot (K + c)}{2c^2} \cdot P_{0,0} \qquad (21)
\end{aligned}$$

So,

$$\begin{aligned}
c\mu \cdot \sum_{n=1}^{K} P_{0,n+c} &= \frac{(\theta_1 + \lambda) \cdot K \cdot (K + c)}{2c^2} \cdot P_{0,0} \\
&\quad + \lambda \cdot P_{0,K} - \lambda \cdot P_{0,0},
\end{aligned}$$

putting the value from Equation (21).

Eliminating the term "$\lambda \cdot P_{0,K}$", as the value is quite negligible and beyond our limit, we have,

$$c\mu \cdot \sum_{n=1}^{K} P_{0,n+c} = \left[\frac{(\theta_1 + \lambda) \cdot K \cdot (K + c)}{2c^2} - \lambda\right] \cdot P_{0,0}.$$

Putting $\frac{(\theta_1+\lambda)\cdot K\cdot(K+c)}{2c^2} = \alpha$, we have,

$$c\mu \cdot \sum_{n=1}^{K} P_{0,n+c} = [\alpha - \lambda] \cdot P_{0,0}$$

Hence, $\sum_{n=1}^{\infty} P_{0,n+c} = \frac{P_{0,0}}{c\mu} \cdot [\alpha - \lambda]$. ■

D. Lemma 3

$$\sum_{n=1}^{\infty} P_{0,n-1} = [\frac{\alpha}{c\mu} + 1]P_{0,0}.$$

Proof:

$$\begin{aligned}
&\lambda \cdot \sum_{n=1}^{\infty} P_{0,n-1} \\
&= \lambda \cdot P_{0,0} + \lambda \cdot \big[P_{0,1} + P_{0,2} + \cdots + P_{0,K}\big], \ \textit{where } K \approx \infty \\
&= \lambda \cdot P_{0,0} + \frac{\lambda}{c\mu} \cdot c\mu\big[P_{0,1} + P_{0,2} + \cdots + P_{0,K}\big]
\end{aligned}$$

putting the value from Equation (21), we have,

$$= \lambda \cdot P_{0,0} + \frac{\lambda}{c\mu} \cdot (\theta_1 + \lambda) \cdot \frac{K \cdot (K + c)}{2c^2} \cdot P_{0,0}$$

putting $(\theta_1 + \lambda)\frac{K\cdot(K+c)}{2c^2} = \alpha$, we have,

$$\begin{aligned}
&= \lambda \cdot P_{0,0} + \frac{\lambda}{c\mu} \cdot \alpha \cdot P_{0,0}, \\
&= \left[\frac{\alpha\lambda}{c\mu} + \lambda\right]P_{0,0}
\end{aligned}$$

Hence,

$$\sum_{n=1}^{\infty} P_{0,n-1} = [\frac{\alpha}{c\mu} + 1]P_{0,0}.$$ ■

E. Proof of Theorem 2

From Equation (3), we have,

$$(\lambda + c\mu)\sum_{n=1}^{\infty} P_{0,n} = \lambda \sum_{n=1}^{\infty} P_{0,n-1} + c\mu \sum_{n=1}^{\infty} P_{0,n-1} + \theta_2 \sum_{n=c}^{\infty} P_{1,n}.$$

Putting the values from Theorem 1, Lemma 2, and Lemma 3, we have,

$$\begin{aligned}
(\lambda + c\mu)\sum_{n=1}^{\infty} P_{0,n} &= \left[\frac{\alpha\lambda}{c\mu} + \lambda\right]P_{0,0} + [\alpha - \lambda] \cdot P_{0,0} + \theta_1 \cdot P_{0,0} \\
&= \left[\frac{\alpha\lambda}{c\mu} + \alpha + \theta_1\right] \cdot P_{0,0} \\
&= \left[\alpha \cdot \left(\frac{\lambda}{c\mu} + 1\right) + \theta_1\right] \cdot P_{0,0}
\end{aligned}$$

Hence,

$$\sum_{n=1}^{\infty} P_{0,n} = [\frac{\alpha}{c\mu} + \frac{\theta_1}{\lambda + c\mu}] \cdot P_{0,0}.$$

F. Proof of Theorem 3

$$\sum_{n=0}^{\infty} P_{0,n} + \sum_{n=0}^{\infty} P_{1,n} = \sum_{n=1}^{\infty} P_{0,n} + P_{0,0} + \sum_{n=0}^{\infty} P_{1,n}$$

Putting the value from theorem 1 and theorem 2,we have,

$$= \left[\frac{\alpha}{c\mu} + \frac{\theta_1}{\lambda + c\mu}\right] \cdot P_{0,0} + P_{0,0} + \left[\frac{c \cdot \theta_1}{\lambda} + \frac{\theta_1}{\theta_2}\right] \cdot P_{0,0},$$
$$= \left[1 + \frac{\theta_1}{\theta_2} + \frac{c\theta_1}{\lambda} + \frac{\alpha}{c\mu} + \frac{\theta_1}{\lambda + c\mu}\right] \cdot P_{0,0}$$

Hence, $P_{0,0} = \frac{1}{1+\frac{\theta_1}{\theta_2}+\frac{c\theta_1}{\lambda}+\frac{\alpha}{c\mu}+\frac{\theta_1}{\lambda+c\mu}}$.

G. Proof of Theorem 4

Given an undirected graph $G(N, L)$ representing the physical network, where N is the set of vertices and L is the set of edges. Each vertex $u \in N$ and edge $(u, v) \in L$ have assigned the capacity $C^N(u)$ and $C^L(u, v)$, respectively. Given another undirected graph $G^V(N^V, L^V)$ representing the virtual network, where N^V is the set of vertices and L^V is the set of edges. Here, we consider that virtual nodes refer to instances of the virtual functions, and virtual links refer to links between two instances of the virtual function in a service chain. Each instance of the functions has been assigned a capacity C_f, to represent the capacity of the instance of function $f \in F_V$ and F^V is the set of virtual functions. Each virtual link has a certain service chain demand $d_{a,b}(u, v)$, which represents the demand of virtual link (a, b), on physical link(u, v).

We see in the last example in Figure 4, in virtual and physical mapping models, multiple virtual nodes are mapped to a single physical node of the network. That is, at a physical node u, the sum of the capacity of all the virtual nodes mapped to u, must be less than or equal to the maximum capacity of u. Again, as multiple virtual links are mapped to single physical links, the total sum of the demand of virtual links mapped to a physical link must be less than or equal to the maximum capacity of that physical link. In a virtual to physical mapping scenario, for all $a \in N^V$ mapped to $u \in N$, and all $b \in N^V$ mapped to $v \in N$, and for all links, $(a, b) \in L^V$ mapped to $(u, v) \in L$ is required to satisfy the following conditions:

1) $\sum_{f \in F_V} C_f(u) \leq C^N(u)$, and $\sum_{f \in F_V} C_f(v) \leq C^N(v)$, $\forall u, v \in N$, where $\sum_{f \in F_V} C_f(u)$ and $\sum_{f \in F_V} C_f(v)$ are the sum of the capacities of the virtual nodes at physical node u and v respectively.
2) $\sum d_{a,b}(u, v) \leq C^L(u, v)$, $\forall (u, v) \in L$, $\forall (a, b) \in L^V$, where $\sum d_{a,b}(u, v)$ is the sum of the demand of the virtual links mapped to the physical link (u, v).

Definition 5: A function $f : \delta_1 \rightarrow \delta_2$ is called a *mapping reduction* from A to B *iff*

a) For any $\beta \in \delta_1$, $\beta \in A$ *iff* $f(\beta) \in B$,

b) f is a computable function.

Intuitively, a mapping reduction from A to B says that a computer can transform any instance of A into an instance of B such that the answer to B is the answer to A. By mapping the variable of the VNE problem to the variable of our problem, we have,

$$\left\{\begin{array}{c} C_P(u_i) \rightarrow C^N(u) \\ C_P(u_j) \rightarrow C^N(v) \\ C_V(w_K) \rightarrow \sum_{f \in F_V} C_f(u) \\ C_V(w_L) \rightarrow \sum_{f \in F_V} C_f(v) \\ b_P(u_i, u_j) \rightarrow C^L(u, v) \\ b_V(w_K, w_L) \rightarrow \sum d_{a,b}(u, v) \end{array}\right\} \quad (22)$$

By Definition 5 and Equation (22), we can map and reduce the VNE NP-hard problem to our optimization problem. Hence, our optimization problem is NP-hard.

ACKNOWLEDGMENT

The authors would like to thank the associate editor and the anonymous reviewers for their valuable comments and suggestions.

REFERENCES

[1] S. Shafiee and E. Topal, "When will fossil fuel reserves be diminished?" *Energy Policy*, vol. 37, no. 1, pp. 181–189, Jan. 2009.
[2] International Energy Agency, *World Energy Outlook*. Paris, France: IEA, 2010.
[3] "Renewable energy policy network for the 21st century," Renew. Glob. Status Rep., Manila, Philippines, Rep., Jun. 2016. [Online]. Available: http://www.ren21.net/wp-content/uploads/2016/05/GSR_2016_Full_Report_lowres.pdf
[4] A. Vaughan, "Greenhouse gas emission report," Manuf. Construct. Energy Divis., Guardian, London, U.K., Rep., Sep, 2015. [Online]. Available: https://www.theguardian.com/environment/2015/sep/25/server-data-centre-emissions-air-travel-web-google-facebook-greenhouse-gas
[5] "Make IT green: Cloud computing and its contribution to climate change," Greenpeace Int., Amsterdam, The Netherlands, Rep., Mar. 2010. [Online]. Available: http://www.greenpeace.org/international/en/publications/reports/make-it-green-cloud-computing/
[6] W. Zhang *et al.*, "OpenNetVM: A platform for high performance network service chains," in *Proc. Workshop Hot Topics Middleboxes NFV*, 2016, pp. 26–31.
[7] D. Jiankang, W. Hongbo, and C. Shiduan, "Energy-performance tradeoffs in IaaS cloud with virtual machine scheduling," *China Commun.*, vol. 12, no. 2, pp. 155–166, Feb. 2015.
[8] Z. Royaee and M. Mohammadi, "Energy aware virtual machine allocation algorithm in cloud network," in *Proc. Smart Grid Conf. (SGC)*, Tehran, Iran, 2013, pp. 259–263.
[9] W. Huang, X. Li, and Z. Qian, "An energy efficient virtual machine placement algorithm with balanced resource utilization," in *Proc. 7th Int. Conf. Innov. Mobile Internet Services Ubiquit. Comput.*, Taichung, Taiwan, 2013, pp. 313–319.
[10] K. S. Rao and P. S. Thilagam, "Heuristics based server consolidation with residual resource defragmentation in cloud data centers," *Future Gener. Comput. Syst.* vol. 50, pp. 87–98, Sep. 2015.
[11] G. Singh and P. Gupta, "A review on migration techniques and challenges in live virtual machine migration," in *Proc. 5th Int. Conf. Rel. Infocom Technol. Optim. (Trends Future Directions) (ICRITO)*, Noida, India, 2016, pp. 542–546.
[12] R. Mijumbi *et al.*, "Network function virtualization: State-of-the-art and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 236–262, 1st Quart., 2016.

[13] W. Dargie, "Estimation of the cost of VM migration," in *Proc. 23rd Int. Conf. Comput. Commun. Netw. (ICCCN)*, Shanghai, China, 2014, pp. 1–8.

[14] T. Nadeau and P. Quinn, "Problem statement for service function chaining," Internet Eng. Task Force," Fremont, CA, USA, RFC 7498, Nov. 2015.

[15] V. Eramo, E. Miucci, M. Ammar, and F. G. Lavacca, "An approach for service function chain routing and virtual function network instance migration in network function virtualization architectures," *IEEE/ACM Trans. Netw.*, vol. 25, no. 4, pp. 2008–2025, Aug. 2017.

[16] J. Liu, W. Lu, F. Zhou, P. Lu, and Z. Zhu, "On dynamic service function chain deployment and readjustment," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 3, pp. 543–553, Sep. 2017.

[17] R. Huang *et al.*, "Data center IT efficiency measures," Nat. Renew. Energy Lab. (NREL), Golden, CO, USA, Rep. NREL/SR–7A40-63181, 2015.

[18] G. Chen *et al.*, "Energy-aware server provisioning and load dispatching for connection-intensive Internet services," in *Proc. 5th USENIX Symp. Netw. Syst. Design Implement.*, San Francisco, CA, USA, 2008, pp. 337–350.

[19] C. Rose and M. Hluchyj, "The performance of random and optimal scheduling in a time-multiplex switch," *IEEE Trans. Commun.*, vol. 35, no. 8, pp. 813–817, Aug. 1987.

[20] Info-Tech, *Top 10 Energy-Saving Tips for a Greener Data Center*, Info-Tech Res. Group, London, ON, Canada, Apr. 2010. [Online]. Available: http://static.infotech.com/downloads/samples/070411_premium_oo_greendc_top_10.pdf

[21] X. Tang, Y. Li, R. Ren, and W. Cai, "On first fit bin packing for online cloud server allocation," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, Chicago, IL, USA, 2016, pp. 323–332.

[22] L. P. Pires and B. Barán, "A virtual machine placement taxonomy," in *Proc. 15th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput. (CCGrid)*, Shenzhen, China, 2015, pp. 159–168.

[23] J. Dong *et al.*, "Energy-saving virtual machine placement in cloud data centers," in *Proc. 13th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput. (CCGrid)*, Delft, The Netherlands, 2013, pp. 618–624.

[24] A. Khosravi, S. K. Garg, and R. Buyya, "Energy and carbon-efficient placement of virtual machines in distributed cloud data centers," in *Proc. Eur. Conf. Parallel Process.*, Santiago de Compostela, Spain, 2013.

[25] Y. Ding, X. Qin, L. Liu, and T. Wang, "Energy efficient scheduling of virtual machines in cloud with deadline constraint," *Future Gener. Comput. Syst.*, vol. 50, pp. 62–74, Sep. 2015.

[26] Y.-J. Chiang, Y.-C. Ouyang, and C.-H. R. Hsu. "An efficient green control algorithm in cloud computing for cost optimization," *IEEE Trans. Cloud Comput.*, vol. 3, no. 2, pp. 145–155, Apr./Jun. 2015.

[27] M. Ghaznavi *et al.*, "Elastic virtual network function placement," in *Proc. IEEE 4th Int. Conf. Cloud Netw. (CloudNet)*, Niagara Falls, ON, Canada, 2015, pp. 255–260.

[28] S. D'Oro, L. Galluccio, S. Palazzo, and G. Schembra, "Exploiting congestion games to achieve distributed service chaining in NFV networks," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 2, pp. 407–420, Feb. 2017.

[29] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," in *Proc. IEEE 4th Int. Conf. Cloud Netw. (CloudNet)*, Niagara Falls, ON, Canada, 2015, pp. 171–177.

[30] S. D'Oro, L. Galluccio, S. Palazzo, and G. Schembra, "A game theoretic approach for distributed resource allocation and orchestration of softwarized networks," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 3, pp. 721–735, Mar. 2017.

[31] H. Moens and F. De Turck, "VNF-P: A model for efficient placement of virtualized network functions," *Proc. 10th Int. Conf. Netw. Service Manag. (CNSM)*, Rio de Janeiro, Brazil, Nov. 2014, pp. 418–423.

[32] P. Wang, J. Lan, X. Zhang, Y. Hu, and S. Chen, "Dynamic function composition for network service chain: Model and optimization," *Comput. Netw.*, vol. 92, pp. 408–418, Dec. 2015.

[33] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Hong Kong, 2015, pp. 1346–1354.

[34] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," in *Proc. IEEE 3rd Int. Conf. Cloud Netw. (CloudNet)*, Oct. 2014, pp. 7–13.

[35] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, "Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manag. (IM)*, Ottawa, ON, Canada, May 2015, pp. 98–106.

[36] M. Mechtri, C. Ghribi, and D. Zeghlache, "A scalable algorithm for the placement of service function chains," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 3, pp. 533–546, Sep. 2016.

[37] S. Clayman, E. Maini, A. Galis, A. Manzalini, and N. Mazzocca, "The dynamic placement of virtual network functions," in *Proc. IEEE Netw. Oper. Manag. Symp. (NOMS)*, Ottawa, ON, Canada, May 2015, pp. 98–106.

[38] F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and O. C. M. B. Duarte, "Orchestrating virtualized network functions," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 4, pp. 725–739, Dec. 2016.

[39] Y. Deng, W. J. Braun, and Y. Q. Zhao, "M/M/1 queueing system with delayed controlled vacation," *OR Trans.*, vol. 4, pp. 17–30, 1999. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.506.8850&rep=rep1&type=pdf

[40] T. Phung-Duc, "Exact solutions for M/M/c setup queues," *Telecommun. Syst.*, vol. 64, no. 2, pp. 309–324, 2017.

[41] R. Basmadjian, F. Niedermeier, and H. de Meer, "Modelling performance and power consumption of utilisation-based DVFS using M/M/1 queues," in *Proc. 7th Int. Conf. Future Energy Syst.*, Waterloo, ON, Canada, 2016, Art. no. 14.

[42] E. Gelenbe and G. Pujolle, *Introduction to Queueing Networks*. New York, NY, USA: Wiley, 1998.

[43] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 4, pp. 1888–1906, 4th Quart., 2013.

[44] M. Yadin and P. Naor, "Queueing systems with a removable service station," *J. Oper. Res. Soc.*, vol. 14, no. 4, pp. 393–405, 1963.

[45] A. Dhesikan, *Data Center Energy Efficiency: Power VS. Performance*, Scribd, 2012. [Online]. Available: https://www.scribd.com/document/228651809/303

[46] S. Sahhaf *et al.*, "Network service chaining with optimized network function embedding supporting service decompositions," *Comput. Netw.*, vol. 93, pp. 492–505, Dec. 2015.

[47] B. Heller *et al.*, "ElasticTree: Saving energy in data center networks," in *Proc. 7th USENIX Symposium Netw. Syst. Design Implement. (NSDI)*, San Jose, CA, USA, Apr. 2010, pp. 249–264.

[48] G. Cheng, H. Chen, H. Hu, Z. Wang, and J. Lan, "Enabling network function combination via service chain instantiation," *Comput. Netw.*, vol. 92, pp. 396–407, Dec. 2015.

[49] F. Carpio, A. Engelmann, and A. Jukan, "DiffFlow: Differentiating short and long flows for load balancing in data center networks," in *Proc. Glob. Commun. Conf. (GLOBECOM)*, Washington, DC, USA, Dec. 2016, pp. 1–6.

Binayak Kar is currently pursuing the Ph.D. degree with the Department of Computer Science and Information Engineering, National Central University, Taiwan. His research interests include network security, cloud computing, software defined networking, and network function virtualization.

Eric Hsiao-Kuang Wu received the B.S. degree in computer science and information engineering from National Taiwan University, Taipei, Taiwan, in 1989 and the M.S. and Ph.D. degrees in computer science from the University of California, Los Angeles, in 1993 and 1997, respectively. He is currently a Professor of computer science and information engineering with the Department of Computer Science and Information Engineering, National Central University, Chung-Li, Taiwan. His research interests include wireless networks, mobile computing, and broadband networks.

Ying-Dar Lin (F'13) received the Ph.D. degree in computer science from the University of California at Los Angeles in 1993. He is a Distinguished Professor of computer science with National Chiao Tung University, Taiwan. He was a Visiting Scholar with Cisco Systems, San Jose, CA, USA, from 2007 to 2008, and the CEO with Telecom Technology Center, Taipei, Taiwan, from 2010 to 2011. Since 2002, he has been the Founder and the Director of Network Benchmarking Laboratory, which reviews network products with real traffic and has been an approved test laboratory of the Open Networking Foundation (ONF) since 2014. He also co-founded L7 Networks Inc. in 2002, which was later acquired by D-Link Corp. His research interests include network security, wireless communications, and network cloudification. His work on multihop cellular was the first along this line, and has been cited over 800 times and standardized into IEEE 802.11s, IEEE 802.15.5, IEEE 802.16j, and 3GPP LTE-Advanced. He published a textbook entitled *Computer Networks: An Open Source Approach*, (McGraw-Hill, 2011), with Ren-Hung Hwang and Fred Baker. He was an IEEE Distinguished Lecturer from 2014 to 2017 and an ONF Research Associate. He currently serves on the editorial boards of several IEEE journals and magazines, and is the Editor-in-Chief of *IEEE Communications Surveys and Tutorials*.