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.
of the network’s energy consumption issues. We consider the machines, which will help with performing a better analysis of the PMs, and the respective VM instances on those in this paper, we normalize the energy consumption cost the PM load, the cooling load also varies [45]. Therefore, heat generated during processing [20]. Hence, depending 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 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>
<thead>
<tr>
<th>SC number</th>
<th>Source</th>
<th>Destination</th>
<th>SC Demand</th>
<th>Virtual Path</th>
<th>Physical Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC1</td>
<td>1</td>
<td>6</td>
<td>B–C–A</td>
<td>1–2–4–7–6</td>
<td>1–2–4–7–6</td>
</tr>
<tr>
<td>SC2</td>
<td>5</td>
<td>1</td>
<td>C–A–D</td>
<td>5–9–7–4–1</td>
<td>5–9–7–4–2</td>
</tr>
<tr>
<td>SC3</td>
<td>6</td>
<td>1</td>
<td>A–D–B</td>
<td>6–7–4–2–1</td>
<td>6–7–4–2–1</td>
</tr>
<tr>
<td>SC4</td>
<td>5</td>
<td>6</td>
<td>B–D–B</td>
<td>5–9–4–2–6</td>
<td>5–9–7–4–2–6</td>
</tr>
</tbody>
</table>

Fig. 1. Illustrative example of service chain placement in the network.
TABLE II
RELATED WORKS ON VNF PLACEMENT

<table>
<thead>
<tr>
<th>Reference</th>
<th>Optimal Placement</th>
<th>Dynamic Placement</th>
<th>Energy-Saving Model</th>
<th>Normalized PM &amp; VM Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>[33,34,35,36]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[37]</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>[16,38]</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>This paper</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

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 embedding 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].

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→IDLE: 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 (\(M_{\text{power}}\)) consumed by the machine and the ratio between default capacity (\(D_{\text{cap}}\)) of the machine to the maximum capacity (\(M_{\text{cap}}\)) 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 (\(\sum V_{\text{cap}}\)) 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 works present various articles [26], [39], [41] using the M/M/1 queueing model. However, in practice, real-world applications are not processed by the single-service node. Therefore, we use the M/M/c queueing 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 with rate \(\lambda\), 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 \(i\)th 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 \(S = \{(m, n), m = 0, 1, 0 \leq n \leq \infty\}\) 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 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.

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

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

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

\[
(\lambda + c\mu)P_{0,n} = \lambda \cdot P_{0,n-1} + c\mu \cdot P_{0,n+c} + \theta_2 \cdot P_{1,n},
\]

where \(n = 1, \ldots c - 1,\)

\[
(\lambda + \theta_1)P_{0,0} = \lambda \cdot P_{0,n-1} + c\mu \cdot P_{0,n+c} + \theta_2 \cdot P_{1,n},
\]

where \(n = c, c + 1, \ldots \infty,\)

\[
\theta_1 \cdot P_{0,0} = \lambda \cdot P_{1,0},
\]

\[
\lambda \cdot P_{1,n-1} = \lambda \cdot P_{1,n-1}, \text{ where } n = 1, \ldots c - 1,
\]

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

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

\[
\begin{aligned}
P_{\text{ACTIVE}} &= \sum_{n=1}^{\infty} P_{0,n}, \\
P_{\text{IDLE}} &= P_{0,0}, \\
P_{\text{OFF}} &= \sum_{n=0}^{\infty} P_{1,n}.
\end{aligned}
\]

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

Proof: Please refer to Appendix B.

Assuming, \(\alpha = \frac{(\theta_1 + \lambda) K (k+\alpha)}{2 \sigma}\) for some large integer \(K\), we have,

\[
\sum_{n=1}^{\infty} P_{0,n} = \left[ \frac{\theta_1}{\mu} + \frac{\theta_2}{\mu} \right] P_{0,0}.
\]

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 \(P_{0,0} = \frac{1 + \frac{1}{\mu} + \frac{\theta_1}{\mu} + \frac{\theta_2}{\mu}}{1 + \frac{1}{\mu} + \frac{\theta_1}{\mu} + \frac{\theta_2}{\mu}}\).

Proof: Please refer to Appendix F.

By using this value \(P_{0,0}\) we can find \(P_{\text{OFF}}, P_{\text{IDLE}},\) and \(P_{\text{ACTIVE}}\) 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.
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 \( d_{ij}(u, v) \) is the delay faced by the flow \( k \) at link \((u, v)\). \( q_{df}^s(u) \) 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. \( d_f^s(u) \) represents the demand of function \( f \) of service chain \( sc \) at node \( u \). The last group consists of two binary variables. \( X_f^s(u) \) presents function \( f \) placed on the \( i \)th VM instance of node \( u \). \( Y_s^f(u) \) 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:

\[
e_C = \sum_{u \in N} n(u) * \left( c^L(u) + \sum_{i \in VM(u)} c^f_i(u) \right) * e^N_u,
\]

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

The total energy consumption cost \( t e_C = \sum_{t \in T} e_C(t) \). Here, \( n(u) \) represents the state of the node \( u \). The value is 1 if the node is IDLE or ACTIVE, and 0 otherwise. \( C^N(u) \), \( C^L(u) \), and \( C^f(u) \) represent the maximum capacity, default capacity, and capacity of the machine in the IDLE state, and capacity of the VM instance \( i \) (on \( u \)) of the node \( u \), respectively. \( e^N_u \) 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 the time \( t \). The total cost of energy consumption \( t e_C \) 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 \( t e_C \). 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.

\[
F_i(u, v) \geq 0, \quad \forall i, \quad F_i \in K, \quad \forall (u, v) \in L, \quad \text{(7)}
\]

\[
\sum_{i=1}^{k} F_i(u, v) \leq C^L(u, v) * l(u, v), \quad \forall (u, v) \in L, \quad \text{(8)}
\]

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

\[
0 \leq \frac{\lambda}{e^L u} \leq 1. \quad \text{(10)}
\]

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

<table>
<thead>
<tr>
<th>Variables</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( N )</td>
<td>Set of PMs/nodes</td>
</tr>
<tr>
<td>( L )</td>
<td>Set of links</td>
</tr>
<tr>
<td>( vF )</td>
<td>Set of VNFs</td>
</tr>
<tr>
<td>( sC )</td>
<td>Set of service chains</td>
</tr>
<tr>
<td>( T )</td>
<td>Set of iterations</td>
</tr>
<tr>
<td>( vM(i) )</td>
<td>Set of VM instances on a node ( i )</td>
</tr>
<tr>
<td>( K )</td>
<td>Set of commodities</td>
</tr>
</tbody>
</table>

**List of Commonly Used Variables and Notations**

\[ X_f^s(u) \] Instance \( f \) of service chain \( sc \) mapped to node \( u \)

\[ Y_s^f(u) \] 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 \( k \). \( e_C^N \) shows the energy consumed by the node \( u \). \( t e_C \) and \( e_C \) are the energy consumption cost and the total energy consumption cost, respectively. The third group refers to the delay, demand, and capacity parameters. \( C^N(u) \), \( C^L(u) \), and \( C^f_i(u) \)
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 maximum delay at node $u$. Equation (15) 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 part 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.

$$\sum_{i \in \mathcal{V}} C_i^u(u) \leq C_i^N(u) - C_i^I(u), \forall u \in N,$$

$$Y_{f}^{sC}(u) = 1, \forall u \in N, f \in vF, sC \in sC,$$

$$\sum_{sC \in sC} d_{f}^{sC}(u) \leq C_i^N(u) - \sum_{i \in \mathcal{V}} C_i^I(u) - C_i^I(u), \forall u \in N, f \in vF, sC \in sC,$$

$$d_{f}^{sC}(u) \leq C_i^I(u), \forall u \in N, f \in vF, sC \in sC.$$

3) Placement Constraints: The binary variable in Equation (15) shows the function $f$ placed on the $i$th 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 part 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.

$$X_i^f(u) = 1, \forall u \in N,$$

$$q_d_{f}^{sC}(u) \leq d_{\text{MAX}}(u), \forall u \in N, f \in vF, sC \in sC,$$

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

$$\left[\sum X_i^f(u)\right] \left[\left[\sum_{sC} \left(\frac{C_i^I(u) * e_i^N(u)}{C_i^N(u)}\right)\right] \left[\left(E_i^N(u) = 0\right) \& \left(\sum_{sC} d_{i}^{sC}(u) \geq \text{threshold}\right)\right] = 1, \forall u \in N, \forall f \in vF, sC \in sC, \text{threshold is a constant}.\right.$$

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 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 SC3) 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) 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 $(u_i, u_j) \in E_P$ 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 $C_P(u_j) \geq C_V(w_l)$,
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.

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.
**Algorithm 1 DPVC Algorithm**

1. **Input:** A, B, ST, VM\_cap, Total\_cost, Idl\_max, U\_max, U\_idl, vNF, pt, Idl\_time, min\_cap, Ec, Num\_Flow, boot.
2. **Algorithm:**
   3. \( VM = \) struct(VM\_flg, VM\_fun, VM\_exp, VM\_wait, VM\_flow)  
   4. Service\_Chain = struct(source, chain, destination, FLOW\_len, FLOW\_num)  
   5. Chain\_Time = struct(startr\_time, p\_time, pre\_source, chain\_Dest, hop, end\_time)  
   6. for each iteration \( t \)  
     7. Service\_Chain = Service\_Chain\_IP(Service\_Chain, \( t \))  
     8. Chain\_Time = set\_Chain\_Time(Curr\_INP, Service\_Chain, Chain\_Time, \( t \))  
     9. Curr\_INP = Curr\_VNF\_Input(Curr\_INP, Service\_Chain, Chain\_Time, \( t \))  
     10. Placement(Curr\_INP, Chain\_Time, nodes, A, B, ST, VM, \( t \))  
     11. for each node \( i \) in ST  
       12. for each VM \( j \) in node \( i \)  
       13. if \( t > VM\_exp(i, j) \)  
       14. Release(VM, i, j)  
     15. end for  
     16. if \( i \in ST \) & (U\_ass(i) == 0)  
     17. Idl\_time(i) = Idl\_time(i) + 1;  
     18. if Idl\_time(i) \geq Idl\_max  
     19. Delete(ST, i)  
     20. end for  
     21. end for  
     22. end for  
     23. VM = update\_VM(VM)  
     24. cost = \( \sum_{i=1}^{N} \sum_{i\in ST} E\_c(i) = \frac{U\_d(i) + \sum VM\_cap(i, j) \times VM\_flg(i, j)}{U\_max(i)} \)  
     25. end for  
     26. **Output:** Total\_cost = \( \sum \) cost

**Algorithm 2 Placement Algorithm**

1. Placement(Curr\_INP, Chain\_Time, nodes, A, B, ST, VM, \( t \))  
2. for \( i = 1 \rightarrow |\) Curr\_INP\( | \) do  
3. \( s = \) Curr\_INP\_i\_curr\_source;  
4. \( d = \) Curr\_INP\_i\_destination;  
5. \( nf = \) Curr\_INP\_i\_curr\_VNF;  
6. RDFST\_nodes(A, B, ST, VM\_flg, VM\_exp, s, d, nf, \( t \))  
7. Chain\_Time = update\_Chain\_Time(Chain\_Time, Curr\_INP, new\_Node)

8. **Return:** ST, VM, Chain\_Time

*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 \( i \)th node of the network has the \( j \)th 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 \( \text{Delete}(ST, u) = ST - \{u\} \).

Definition 3: [Assign] if \( u' \) is an arbitrary set and \( i \) is the number of elements in \( u \), and \( j \not\in u \) is an arbitrary element, then we define \( \text{Assign}(u', j) = u'^{+1} \).

Definition 4: [Release] if \( u' \) is an arbitrary set and \( i \) is the number of elements in \( u \), and \( j \not\in u \) is an arbitrary element, then we define \( \text{Release}(u', j) = u'^{-1} \).

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, Flowlen, and Flownum 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, \( currFLOWlen \) shows the flow length, \( Flownum \) 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 \( \text{Release} \) 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 one node, sort the nodes based on their priority, retrieve the most suitable node (nN) for the placement of the VNF from the sorted structure (nodeSorted), 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 (minCap) 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 (offime), we turn that node ACTIVE. After successful placement of a VNF, the RDFST function returns the value to the Placement algorithm.

## 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\textsuperscript{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, \emph{i.e.}, \((i\, +\, 1)\text{th VNF of the service chain can be placed only after completion of the } i\text{th VNF of that service chain.}\) If the \(i\text{th VNF is a special one, then the VM will remain active until the flow reaches the destination. The } (i\, +\, 1)\text{th VNF of the service chain can be placed immediately after the VM is available and packets are ready for processing.}\) 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, \emph{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.

\textsuperscript{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, continues 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, \emph{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
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 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
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%.

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 decreases by nearly 50 percent, and average utilization increases by nearly
with increase/decrease the utilization of the active nodes. As shown in Figure 10, the energy consumption cost changes with the change of utilization of active nodes in the network remains unchanged. However, 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 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 of 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.

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,

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

Hence,

\[ P_{1,n} = \frac{\lambda}{\lambda + \theta_2} \cdot P_{1,n-1} = \left( \frac{\lambda}{\lambda + \theta_2} \right)^{n-c+1} \cdot P_{1,0} \]

\[ = \left( \frac{\lambda}{\lambda + \theta_2} \right)^{n-c} \cdot \theta_1 \cdot P_{0,0} \]

\[ = \frac{\theta_1}{\lambda + \theta_2} \cdot \left( \frac{\lambda}{\lambda + \theta_2} \right)^{n-c} \cdot P_{0,0} \]

So,

\[ \sum_{n=c}^{\infty} P_{1,n} = \sum_{n=c}^{\infty} \frac{\theta_1}{\lambda + \theta_2} \cdot \left( \frac{\lambda}{\lambda + \theta_2} \right)^{n-c} \cdot P_{0,0} \]

\[ = \frac{\theta_1}{\lambda + \theta_2} \cdot P_{0,0} \cdot \sum_{n=c}^{\infty} \left( \frac{\lambda}{\lambda + \theta_2} \right)^{n-c} \]

\[ = \frac{\theta_1}{\lambda + \theta_2} \cdot P_{0,0} \cdot \frac{1}{1 - \frac{\lambda}{\lambda + \theta_2}} \]

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

B. Proof of Theorem 1

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

\[ P_{1,0} = \frac{\theta_1}{\lambda} \cdot P_{0,0} \]

\[ P_{1,0} = P_{1,1} = P_{1,2} = \ldots = P_{1,c-1} \]

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,

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

\[ = \left[ c \cdot \frac{\theta_1}{\lambda} + \frac{\theta_1}{\theta_2} \right] \cdot P_{0,0}. \]
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,

\[
c \mu \cdot P_{0,c+1} = (\lambda + c \mu) \cdot P_{0,1} - \lambda \cdot P_{0,0}, \quad n = 1
\]
\[
c \mu \cdot P_{0,c+2} = (\lambda + c \mu) \cdot P_{0,2} - \lambda \cdot P_{0,1}, \quad n = 2
\]
\[
c \mu \cdot P_{0,c+3} = (\lambda + c \mu) \cdot P_{0,3} - \lambda \cdot P_{0,2}, \quad n = 3
\]
\[
\vdots
\]
\[
c \mu \cdot P_{0,c+c-1} = (\lambda + c \mu) \cdot P_{0,c-1} - \lambda \cdot P_{0,c-2}, \quad n = c-1
\]
\[
c \mu \cdot P_{0,c+c} = (\lambda + c \mu) \cdot P_{0,c} - \lambda \cdot P_{0,c-1}, \quad n = c
\]
\[
\vdots
\]
\[
c \mu \cdot P_{0,c+K} = (\lambda + c \mu) \cdot P_{0,K} - \lambda \cdot P_{0,K-1}, \quad n = K
\]

Adding these \( K \) number of equations we have,

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

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

\[
\sum_{n=1}^{\infty} P_{0,n} = \left[ \alpha - \lambda \right] \cdot P_{0,0}
\]

**D. Lemma 3**

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

**Proof:**

\[
\lambda \cdot \sum_{n=1}^{\infty} P_{0,n-1} = \lambda \cdot P_{0,0} + \lambda \cdot \left[ \frac{P_{0,1} + P_{0,2} + \cdots + P_{0,K}}{c \mu} \right], \quad \text{where} \quad K \approx \infty
\]

\[
= \lambda \cdot P_{0,0} + \lambda \cdot \left[ \frac{\theta(1 + \lambda)}{c \mu} \cdot P_{0,0} \right]
\]

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

\[
= \lambda \cdot P_{0,0} + \lambda \cdot \left[ \frac{\theta(1 + \lambda)}{c \mu} \cdot P_{0,0} \right]
\]

Hence,

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

**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,

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

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

Putting the term “\( \lambda \cdot P_{0,K} \)” as the value is quite negligible and beyond our limit, we have,

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

Putting \( \frac{\theta(1 + \lambda)}{2c^2} \cdot K \cdot (K + c) \) = \( \alpha \), we have,

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

Hence,

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

**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 modeling maps, 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. \[ \sum_{f \in F^{V}} C_{f}(u) \leq C^{N}(u), \text{ and } \sum_{f \in F^{V}} C_{f}(v) \leq C^{N}(v), \forall u, v \in N, \text{ where } \sum_{f \in F^{V}} C_{f}(u) \text{ and } \sum_{f \in F^{V}} C_{f}(v) \text{ are the sum of the capacities of the virtual nodes at physical node } u \text{ and } v \text{ respectively.} \]
2. \[ \sum_{d \in d_{a,b}} d_{a,b}(u, v) \leq C^{L}(u, v), \forall (u, v) \in L, \forall (a, b) \in L^{V}, \text{ where } \sum_{d \in d_{a,b}} d_{a,b}(u, v) \text{ is the sum of the demand of the virtual links mapped to the physical link } (u, v). \]

**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,
\[
\begin{align*}
C_{P}(u_i) &\rightarrow C^{N}(u) \\
C_{P}(u_i) &\rightarrow C^{N}(v) \\
C_{V}(w_k) &\rightarrow \sum_{f \in F^{V}} C_{f}(u_i) \\
C_{V}(w_l) &\rightarrow \sum_{f \in F^{V}} C_{f}(v) \\
\delta_{P}(u_i, u_j) &\rightarrow C^{L}(u_i, v) \\
\delta_{V}(w_k, w_l) &\rightarrow \sum_{d \in d_{a,b}} d_{a,b}(u, v)
\end{align*}
\]

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**


