Holu: Power-Aware and Delay-Constrained VNF Placement and Chaining

Amir Varasteh, Basavaraj Madiwalar, Amaury Van Bemten, Wolfgang Kellerer, and Carmen Mas-Machuca
Chair of Communication Networks, Department of Electrical and Computer Engineering, Technical University of Munich, Germany
Email: {amir.varasteh, basavaraj.madiwalar, amaury.van-bemten, wolfgang.kellerer, cmas}@tum.de

Abstract—Service function chains (SFCs) are an ordered set of virtual network functions (VNFs) which can realize a specific network service. Enabled by virtualization technologies, these VNFs are hosted on physical machines (PMs), and interconnected by network switches. In today networks, these resources are usually under-utilized and/or over-provisioned, resulting in power-inefficient deployments. To improve power-efficiency, SFCs should be deployed utilizing the minimum number of PMs and network equipment, which are not concomitant. Considering the existing PM and switch power consumption models and their resource constraints, we formulate the power-aware and delay-constrained joint VNF placement and routing (PD-VPR) problem as an Integer Linear Program (ILP). Due to the NP-completeness of the problem, we propose Holu, a fast heuristic framework that efficiently solves the PD-VPR problem in an online manner. Specifically, Holu decomposes the PD-VPR into two sub-problems and solve them sequentially: i) a VNF placement problem that consists of mapping the VNFs to PMs using a centrality-based ranking method, and ii) a routing problem that efficiently splits the delay budget between consecutive VNFs of the SFC, and finds a Delay-Constrained Least-Cost (DCLC) shortest-path through the selected PMs (hosting VNFs) using the Lagrange Relaxation based Aggregated Cost (LARAC) algorithm. Our simulation results indicate that Holu outperforms the state-of-the-art algorithms in terms of total power consumption and acceptance rate by 24.7% and 31%, respectively.

Index Terms—power optimization, power efficiency, energy efficiency, VNF placement, service function chaining

I. INTRODUCTION

In the modern telecommunication world, network providers have been deploying their network services using Virtual Network Functions (VNFs). A service is usually composed of an ordered list of VNFs, deployed on commercial off-the-shelf (COTS) equipment in different parts of the network. This ordered sequence of VNFs is referred to as a Service Function Chain (SFC) [1]. Upon receiving user requests with a specific SFC to traverse, network providers have to find the best server(s) to configure the required VNFs while taking the Network Function Virtualization (NFV) architecture and its available resources into the account. The NFV architecture consists of several interconnected nodes which consist of network switches and hosting Physical Machines (PMs). The former forward the traffic through the network, whereas the latter host VNFs, in form of Virtual Machine (VMs) or containers. Since PMs can host several VNFs, it enables the physical consolidation of networks. To serve the user requests, VNFs must be mapped to the PMs and the traffic should be routed through these VNFs that form the requested SFC.

Despite of its importance, the power-efficiency of the NFV-enabled networks has been only slightly considered. Today, around 7-12% of the total power consumption is required for Internet technologies [2]–[4]. Network providers are challenged to increase their connectivity and services while being sustainable. Bolla et al. [5] have shown the aggressive increase of power consumption in the networks operated by the major Telecom operators worldwide (e.g., AT&T, Verizon). Moreover, the global annual Internet traffic is expected to reach 4.2 ZB, i.e., 4.2×10^{12} GB, in 2022 (with an increase of almost 400% with respect to 2017 [4]). As a result, more and more operators work on the reduction of their carbon footprint and greenhouse gas emission by aiming at decreasing their power consumption [6], [7].

In order to reduce the power consumption, an analysis and modeling of the power consumed by the main NFV components needs to be done, particularly PMs and network equipment. According to Fig. 1, three power states can be considered for PMs and switches: i) standby: the server is in low-power (sleep) mode and consumes a negligible amount of power, ii) idle: the device is powered-on, however its utilization is almost 0% (no traffic load), iii) online: the device is powered-on and its utilization is higher than 0% (processing the traffic load). Starting with the PM power consumption, it has been shown that PMs can consume almost 50% of their maximum power when they are in idle state [8]–[10]. Also, they consume a negligible amount of power when are in standby mode [11] (also referred to as the offline state in some works). Similar to PMs, network resources are usually over-provisioned to support the maximum traffic. However, their utilization rarely reaches the peak network capacity [12], [13]. It has been observed that the online network components such as switch chips and fans consume a significant amount of power even with low workload [12], [14], [15]. Consequently,
idle networking devices are not power-efficient, since an idle network switch can consume up to 90% of the peak power consumption [13].

Accordingly, the relation between power consumption and the device utilization is depicted in Fig. 1 for PMs and switches, assuming a linear power profile [10], [16]. It can be observed that PMs and network devices are not power-proportional, i.e., they do not consume power proportional to their utilization, which differs significantly with the ideal power proportionality case [9], [10], [17]. This difference causes a waste of power consumed by under-utilized devices. Thus, minimizing the number of online PMs and switches can improve the power-proportionality (See Fig. 1) and hence, improving the power-efficiency of the service provider.

In this work, we study the power-aware and delay-constrained joint VNF placement and routing (PD-VPR) problem. Considering the capacitated network resources, the main goal is to minimize the number of online PMs and network switches required to allocate the requested SFCs, while meeting the end-to-end delay (i.e., sum of propagation and VNF processing delay) requirements. We first formulate this problem as an Integer Linear Program (ILP) based on our previous work [18]. The problem formulation has been improved by considering also the VNF processing time, which dynamically changes depending on the traffic. Considering the NP-completeness of the problem, the ILP is not usable for solving real-world problem sizes. Therefore, we propose Holu, an efficient heuristic that solves the PD-VPR problem in an online manner. In more detail, Holu decomposes the PD-VPR problem into two sub-problems which are solved in sequence: i) VNF placement, ii) routing. In the first subproblem, we rank the PMs in the network according to their centrality and the requested VNF types in the SFC. Thereafter, we employ a Delay-Constrained Least-Cost (DCLC) shortest-path algorithm to find the path between the selected VNFs in the previous step using a routing heuristic proposed in our previous work [19]. As an important feature, our routing algorithm is able to efficiently split the end-to-end delay budget between subsequent VNFs in an SFC. This can significantly increase the acceptance rate of requests, even with very strict end-to-end delay requirement.

Therefore, the main contributions of this paper can be summarized as:

- Presenting the PD-VPR problem as an ILP optimization model to i) determine the optimal number of the VNFs and their mapping to the PMs, ii) allocate user requests to the VNF instances, iii) find a path to route the traffic through the allocated VNFs to form the SFC, iv) meet the resource capacities and end-to-end delay constraints including the link propagation and traffic-aware VNF processing delays, and v) minimize the total power consumption.
- Proposing Holu, an online heuristic framework to tackle the PD-VPR problem by dividing it into two sub-problems: VNF placement, and routing.
- Implementation and performance evaluation of the proposed heuristic and comparing with the state-of-the-art CPVNF and BCSP algorithms [18], [20] in terms of total power consumption, acceptance ratio, runtime, etc.

The rest of the paper is organized as follows: The related work is reviewed in Section II. Then, we present the system model and problem definition in Section III followed by the ILP formulation in Section IV. Thereafter, in Section V, we introduce the Holu framework. Finally, we present the performance evaluation of the work in Section VI, and conclude the paper in Section VII.

II. RELATED WORK

Although power-efficient VM placement is a well-studied field in the cloud computing environment [37]–[39], VM placement and VNF placement in NFV/SFC paradigm problems differ in many ways. The former case is a problem that focuses on placing/packing a set of VMs on different PMs, while the latter, considers a specific ordered set of VNFs. In addition, the solution should contain routing and path allocation through these ordered VNFs, which makes it a fundamentally difficult problem to solve [40]. Therefore, the VM placement can be considered as a special case of the latter problem. There has been a large body of work that have investigated the VNF placement and routing problem with different objectives, such as minimizing total deployment cost [41]–[44], minimizing total end-to-end delay [45]–[47], minimizing network resources [48], [49] and routing costs [31], [32], maximizing reliability [50]–[53].

Let us summarize the most recent and relevant works addressing the power-aware VNF placement and routing problem [11], [21]–[29], [33]–[36]. These works have been grouped into three categories: i) VNF placement: place a set of VNFs in order to meet an objective, ii) SFC routing: these works assume the VNFs are already deployed in the network. Thus, they focus on finding the path for the traffic traversing through these VNFs, and iii) joint VNF placement and routing: in addition to VNF placement, the traffic path through these VNFs must be determined. In the first category, not concerned with the routing decisions, authors in [21], [22], [26] have tackled the VNF placement problem. In particular, Pham et. al. [21] aimed at deploying VNFs by using as fewer number of online PMs, such that the communication cost between them is optimized. They proposed a fast solution by using a sampling-based Markov approximation method combined with matching theory. Further, Yang et. al. [22] studied VNF chain placement in data centers. They provided an algorithm to save power in servers and network switches. A step further was taken by authors in [26] and proposed a dynamic server consolidation approach using VM live migration to achieve power-efficiency by maximizing the number of PMs in standby state.

The second category belongs to the works which tackle the challenges brought by the routing problem. There are several dimensions to consider in this category. There are some works that have focused on the routing problem and have considered the power consumption of ternary content-addressable memory (TCAM) of network switches into the account [54]–[58]. However, in our work, we consider the network power consumption based on the online network switches and the number of active ports and their utilization.
Assuming that the VNFs are already placed in the network, some works have focused on finding a path going through the required VNFs (i.e., SFC) considering some constraints, e.g., delay, capacity. For example, the authors in [25] formulated a problem to allocate and schedule traffic flows with deadlines to VNFs while minimizing the total PM power consumption. Recently, the graph layering technique is proposed as an efficient way to find a path through an already placed set of VNFs [19], [30]–[32]. These works transform the network into a layered graph in which each VNF in the SFC is represented by a layer. The user traffic can be routed layer by layer from the top to the bottom layer. For instance, KARIZ, a local search heuristic proposed by [30], finds the path between two layers by solving the minimum cost flow problem. Disregarding the end-to-end delay constraint, the objective of KARIZ is minimizing the network resource costs. As another work, after the graph layering transformation, authors in [31] uses conventional shortest-path algorithms e.g., Dijkstra to calculate the path between the source and destination nodes. To reduce the computational time when using a shortest-path algorithm, Sallam et. al. in [32] propose a pruning algorithm to simplify the constructed layered graph. However, similar to [30] and [31], they did not consider the end-to-end delay constraint in their problem. Nevertheless, in our previous work [19], we proposed a heuristic to find a delay-constrained path, passing through a selected set of VNF nodes in a layered graph, achieving near-optimal performance.

Finally, as the third category, some works extend the problem to consider the routing jointly with the VNF placement decision [11], [23], [24], [27]–[29], [33]–[36]. In more detail, their main goal is to place the VNFs on PMs, allocate them to the user requests, and route the traffic through these VNFs. These decisions can be constrained to capacity and/or delay requirements, optimizing PM and/or network power consumption. Specifically, the authors in [35] focused on determining the required number and placement of VNFs to optimize the network utilization and operational costs (in terms of PMs power consumption), without violating service level agreements. They presented an efficient heuristic based on dynamic programming to solve this problem. Furthermore, Jang et. al. [29] formulated a multi-objective optimization model which maximizes the acceptance ratio and minimizes the power cost for multiple service chains. After transforming the model into a single-objective mixed integer linear programming (MILP) problem, they proved that the problem is NP-hard and proposed an algorithm based on linear relaxation and rounding to approximate the solution of the MILP in polynomial time. In addition to optimizing the VNF placement and routing, the authors in [11] presented online algorithms to reconfigure the network based on traffic changes.

However, in these three categories, there are a number of missing considerations that are addressed in this paper. For example, in the first and third categories, the necessity of coordination between VNF placement and routing decisions is disregarded. Thus, in this work, we tackle the PD-VPR problem. Moreover, opposed to this work, there are some approaches that do not consider both PM and network power consumption into account, which can increase the OPEX of service providers. Moreover, unlike some reviewed related works, we take the QoS constraints in terms of end-to-end delay into account. Also, some works [11], [26], [33] considered VNF migration and reconfiguration which we do not focus on it in this work. A comparison of different state-of-the-art solutions with respect to this work is summarized in Table I. Further, comprehensive surveys on VNF chain placement are available for interested readers [59]–[62]. In this work, we present Holu, an online heuristic framework that presents an efficient VNF placement approach coupled with a fast delay-constrained routing algorithm to solve the PD-VPR problem.

### III. System Model and Problem Definition

Before presenting the mathematical modeling of the work, we note that the used notations through this paper and their definition are presented in Table II.

#### A. Network Model

In this paper, we focus on a Wide Area Network (WAN) scenario, where we represent the network as a unidirectional graph \( G = (N, L) \), being \( N \) the set of nodes, and \( L \) the set of unidirectional links between pair of nodes. Each physical link \((i, j) \in L \) is characterized by the data rate

<table>
<thead>
<tr>
<th>References</th>
<th>Decisions</th>
<th>Power Consumption</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>[19]</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[18]</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[20]</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[21]</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[22]</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[23]</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[24]</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[25]</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[26]</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[11]</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[27]</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[28]</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[29]</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[30]</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[31], [32]</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[33], [34]</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[35], [36]</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TABLE I:** Comparison of related work and the proposed solution.
TABLE II: Notation definition

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( G = (N, L) )</td>
<td>Physical network graph</td>
</tr>
<tr>
<td>( K )</td>
<td>Number of requests</td>
</tr>
<tr>
<td>( G_r = (N_r, L_r) )</td>
<td>Virtual network graph of the request ( r )</td>
</tr>
<tr>
<td>( P )</td>
<td>Set of VNF types</td>
</tr>
<tr>
<td>( V_r^s )</td>
<td>Source node of request ( r )</td>
</tr>
<tr>
<td>( V_r^d )</td>
<td>Destination node of request ( r )</td>
</tr>
<tr>
<td>( B_r )</td>
<td>Data rate of request ( r )</td>
</tr>
<tr>
<td>( D_r )</td>
<td>Maximum delay of request ( r )</td>
</tr>
<tr>
<td>( \alpha_r )</td>
<td>FSC of request ( r )</td>
</tr>
<tr>
<td>( H_{c(i,j)} )</td>
<td>Data rate capacity of physical link ( (i,j) )</td>
</tr>
<tr>
<td>( d_{c(i,j)} )</td>
<td>Propagation delay of physical link ( (i,j) )</td>
</tr>
<tr>
<td>( \psi_f )</td>
<td>Processing delay at VNF type ( f )</td>
</tr>
<tr>
<td>( \Phi_f )</td>
<td>Processing capacity of VNF type ( f )</td>
</tr>
<tr>
<td>( U )</td>
<td>Set of VNFs/PMs resource types</td>
</tr>
<tr>
<td>( \lambda_{u,f} )</td>
<td>Required resource type ( u ) with VNF type ( f )</td>
</tr>
<tr>
<td>( C_{max} )</td>
<td>Maximum capacity of resource type ( u ) in PM ( r )</td>
</tr>
<tr>
<td>( \theta^{CPU} )</td>
<td>CPU utilization of PM ( r )</td>
</tr>
<tr>
<td>( P_{pm} )</td>
<td>PM and network switch idle power consumption</td>
</tr>
<tr>
<td>( P_{port} )</td>
<td>Network switch port power consumption</td>
</tr>
<tr>
<td>( P_{idle} )</td>
<td>Total PM and network power consumption</td>
</tr>
<tr>
<td>( \gamma )</td>
<td>Large positive integer</td>
</tr>
<tr>
<td>( \gamma_f )</td>
<td>The ranking value of PM ( n ) with respect to VNF ( f ) in ( C_n )</td>
</tr>
<tr>
<td>( \alpha_{c(i,j)} )</td>
<td>The centrality impact of PM ( n ) with respect to VNF ( f ) in ( C_n )</td>
</tr>
<tr>
<td>( \beta_{c(i,j)} )</td>
<td>The power consumption impact of PM ( n ) with respect to VNF ( f ) in ( C_n )</td>
</tr>
<tr>
<td>( \psi_{c(i,j)} )</td>
<td>Routing cost function assigned to link ( (i,j) )</td>
</tr>
<tr>
<td>( \Phi_{c(i,j)} )</td>
<td>Routing power impact of using link ( (i,j) )</td>
</tr>
<tr>
<td>( \Theta_f )</td>
<td>Betweenness centrality of node ( j )</td>
</tr>
<tr>
<td>( S_r )</td>
<td>Set of candidate PMs for request ( r )</td>
</tr>
</tbody>
</table>

Decision Variables

- \( x_i \in \{0,1\} \): \( +1 \) if PM \( i \) is online
- \( y_i \in \{0,1\} \): \( +1 \) if switch \( i \) is online
- \( q_{c(i,j)} \in \{0,1\} \): \( +1 \) if link \( (i,j) \) is online
- \( w_{c(i,j)} \in \{0,1\} \): \( +1 \) if virtual link \( (i,j) \) is mapped to \( (i',j') \) in \( L \)
- \( n_{i,j} \in \mathbb{N} \): Number of instances of VNF type \( f \) on PM \( i,j \)

B. User Request: We define a user request \( r \in \mathbb{R} \) as following 5-tuple:

\[
\begin{align*}
    r = (V_r^s, V_r^d, D_r, B_r, C_r) & \quad (1)
\end{align*}
\]

where \( V_r^s \) and \( V_r^d \) are the source and the destination nodes, \( D_r \) is the maximum allowed end-to-end delay, \( B_r \) is the requested data rate, and \( C_r = \{f_1, f_2, ..., f_{|V_r^d|}\} \) is the requested SFC, an ordered set of VNFs that the traffic should be routed through.

SFCs can be modeled as a virtual network represented as a graph \( \tilde{G}_r = (\tilde{N}_r, \tilde{L}_r) \), where for request \( r \), \( \tilde{N}_r \) is the set of virtual nodes which are \( \tilde{C}_r = \{f_1, f_2, ..., f_{|V_r^d|}\} \); and \( \tilde{L}_r \) is the set of links where the virtual link \( (i, i+1) \) interconnects \( f_i \) with \( f_{i+1} \). In particular, for each request \( r \), the virtual nodes \( \tilde{N}_r \) (i.e., the VNFs) and their interconnecting links \( \tilde{L}_r \) should be mapped to the physical network \( G \). Hence, the link \( (i,j) \in \tilde{L}_r \) between two consecutive VNFs must be assigned to a path connecting physical links in \( L \). Also, the ingress and egress virtual nodes match the physical source and destination nodes in the substrate physical network. In this way, \( \tilde{G}_r \) has to be mapped over \( G \) such that the requirements of request \( r \) are met in terms of delay and capacity while the consumed total power is minimized.

C. Power Consumption Models: Let us introduce the power consumption models considered in this problem.

1) Network Power Consumption Model: The network power consumption refers to the amount of power consumed for the transmission, which includes the power consumed by the switches at the network nodes as well as the active interconnecting links. In particular, when a network switch is powered on, it consumes a base power \( P_{idle} \) Watts which is independent of the traffic load [15], [16], [63], [64]. Similarly, the network ports consume \( P_{port} \) Watts if the port is powered on (otherwise, 0 Watts). Therefore, the power of a network switch can be computed as [15], [64]:

\[
P_{switch} = \begin{cases} 
\text{\( P_{idle} \)}, & \text{if it is online} \\
0, & \text{otherwise} 
\end{cases} \quad (2)
\]

2) PM Power Consumption Model: The most power-consuming factor of a PM has been shown to be the CPU [65]–[68]. Hence, the power consumption model for the PM is based on its CPU utilization, denoted by \( \theta^{CPU} \). The power consumption of a PM can be calculated as below [69]–[71]:

\[
P_{pm} = \begin{cases} 
\text{\( P_{idle} \)} + (\text{\( P_{max} \)} - \text{\( P_{idle} \)}) \theta^{CPU}, & \text{if it is online} \\
0, & \text{otherwise} 
\end{cases} \quad (3)
\]

where \( P_{idle} \) and \( P_{max} \) is the consumed power when the CPU utilization is 0% and 100%, respectively. Also, \( \theta^{CPU} \) is the CPU utilization of the PM.

D. Problem Statement: Given a network and a set of requested SFCs: i) determine the optimal number of the VNF instances to be deployed, ii) Mapping of these VNF instances to the PMs, iii) allocating user requests to the deployed VNFs, iv) find a path to route the user traffic through the allocated VNFs (i.e., forming the SFC), v) guarantee the end-to-end delay required by the user requests, which should not exceed the sum of the network propagation delay and the processing delay of VNFs, vi) meet the PM and network capacity constraints, and finally vii) minimize the total PM and network power consumption.

IV. Optimization Formulation

In this section, we mathematically formulate the PD-VPR problem as an Integer Linear Program (ILP) optimization model. **Total Power Consumption:** As mentioned before, the objective function is to minimize the total power consumption which is defined as the sum of the network and PM consumed power. According to the network power consumption model in Eq. 2, the total network power consumption denoted by \( P_{net}^{T} \) can be calculated as:

\[
P_{net}^{T} = \sum_{i \in N} \gamma_i + 2P_{port} \sum_{(i,j) \in L} q_{(i,j)} \quad (4)
\]

where \( \gamma_i \in \{0,1\} \) is a variable indicating if switch \( i \) is online. Also, \( q_{(i,j)} \in \{0,1\} \) is defined as a variable that is equal to
Moreover, the VNFs in SFC require two online ports (one per source/destination node).

Moreover, according to the PM power consumption model Eq. 3, the total PM power consumption $P_{pm}^T$ can be calculated as:

$$P_{pm}^T = \sum_{i \in N} x_i \left( p_{pm}^{idle} + (P_{pm}^{max} - p_{pm}^{idle}) \theta_i^{CPU} \right)$$

where $x_i$ is a binary variable indicating if the PM $i$ is powered on. $\theta_i^{CPU}$ is the CPU utilization of PM $i$, which is calculated as the ratio between all required CPU resources by the hosted VNFs and the available CPU resources on PM $i$, i.e.,

$$\theta_i^{CPU} = \sum_{f \in F} \Delta_{f,u} n_{i,u} \leq C_{i,u}, \forall i \in N, \forall u \in U,$$

where $\Delta_{f,u}$ is the amount of resource type $u$ used by the placed VNF $f$, $n_{i,u}$ is an integer variable indicating the maximum resource type $u$ in PM $i$.

Secondly, each VNF $f$ has a limited processing capacity, which is denoted by $\Phi_f$. Therefore, the sum of the rates of all requests served by function $f$ in the PM must not exceed the processing capacity of VNF $f$, i.e.,

$$\sum_{r \in R} a_{i,f,r} b_r \leq n_{i,f} \Phi_f, \forall i \in N, \forall f \in F,$$

where $a_{i,f,r} \in \{0, 1\}$ is a variable which equals to 1 if VNF $f$ of request $r$ is assigned to the PM $i$, and $B_r$ is the requested data rate of $r$. The last set of capacity constraints belong to the physical link capacities. In fact, the sum of the data rate required by all requests served by link $(i,j)$ should not be larger than the capacity of the link $(i,j)$:

$$\sum_{r \in R} \sum_{(k,l) \in L_r} w_{(i,j)(k,l)} b_{r} \leq B_{(i,j)}, \forall i \in N, \forall j \in N,$$

where $w_{(i,j)(k,l)} \in \{0, 1\}$ is a variable that equals to 1 if the physical link $(k,l)$ is used by the virtual link $(i,j)$ of $r$.

As introduced in Section III, the nodes of the virtual network graph $G_v = (N_v, L_v)$ demanded by the request $r$ must be embedded in the physical graph $G = (N, L)$. Firstly, the source and destination nodes of $G_v$ must be mapped to the physical source and destination nodes in the $G$. Hence, the source and destination nodes of request $r$ should be mapped to the same node on the substrate physical node:

$$a_{i,f,r} = 1, \text{ if } i = f = \Psi^s_r, \forall r \in R, \ (9)$$

$$a_{i,f,r} = 1, \text{ if } i = f = \Psi^d_r, \forall r \in R. \ (10)$$

Moreover, the VNFs in SFC $G_v$ must be mapped to a node $n \in N$ that actually host an instance of VNF $f, \forall f \in G_v$:

$$a_{i,f,r} \leq n_{i,f}, \forall i \in N, \forall f \in F, \forall r \in R. \ (11)$$

The flow conservation law is expressed in flow states such that for each network switch $i \in N$, the difference of all outgoing and incoming physical links that are used for the virtual link between virtual nodes $k$ and $l$, and request $r$ must be equal to:

$$\sum_{(i,j) \in L} w_{(i,j)(k,l)} - \sum_{(j,i) \in L} w_{(j,i)(k,l)} = a_{i,k,r} - a_{i,l,r},$$

$$\forall i \in N, \forall k \in N_r, \forall l \in N_r, \forall r \in R. \ (12)$$

The total delay of the embedded request $r$ is modeled as the sum of the VNF processing time of each VNF $f \in C_r$, denoted by $q_{f,r}$, and the propagation delay $d_{(i,j)}$ in the physical links. This summation must be limited to the required end-to-end delay $D_r$ of each request $r \in R$, thus:

$$\sum_{i \in N} \sum_{f \in F} q_{f,r} a_{i,f,r} - \sum_{(i,j) \in L} d_{(i,j)} w_{(i,j)(k,l)} \leq D_r, \forall r \in R, \ (13)$$

where $q_{f,r}$ denotes the processing delay of the VNF $f$ for request $r$, which is considered proportional to the data rate of request $r$ (i.e., $q_{f,r} = \Phi_f/B_r$).

Finally, we let us introduce three indicator variables to control the operation status (i.e., online or standby) of PMs, physical links, and network switches:

$$\sum_{r \in R} \sum_{(k,l) \in L_r} w_{(i,j)(k,l)} \leq q_{(i,j)} \Psi, \forall i \in N, \forall j \in N, \ (14)$$

$$\sum_{r \in R} \sum_{(k,l) \in L_r} (q_{(i,j)} + q_{(j,i)}) \leq \gamma r \Psi, \forall i \in N, \ (15)$$

$$\sum_{f \in F} n_{i,f} \leq x_i \Psi, \forall i \in N \ (16)$$

where $\Psi$ is a large positive number. Considering the aforementioned definitions and constraints, we can formulate the ILP optimization model as below:

$$\text{min} \{ P_{pm}^T + P_{par}^T \}, \ (17)$$

s.t. constraints (6) – (16).

In this formulation, we jointly map a set of virtual nodes $\hat{N}_r, \forall r \in R$ to PMs together with the mapping of virtual links $\hat{L}_r, \forall r \in R$ onto paths in the underlying network $G$ connecting the respective servers. Also, this embedding must meet capacity and path length (i.e., delay) constraints. This problem can be reduced to the graph embedding problem which is generally NP-hard and inapproximable [40], [72]. Thus, the presented ILP model (17) does not apply to practical-sized problems as it cannot be solved in a reasonable period. Therefore, in the following sections, we propose an online heuristic algorithm to solve the aforementioned problem in polynomial time.

V. HOLU: THE ONLINE HEURISTIC FRAMEWORK

In this section, we propose an efficient online heuristic framework named HOLU to solve the PD-VPR problem with much lower and more manageable complexity. Many of the heuristics presented in the state-of-the-art divide the problem based on each VNF, they solve the resulting sub-problems...
Holu acceptance rate. In contrast, the quality of found solutions dramatically as well as the SFC (budget-division problem). This can potentially reduce the cost/constraint budgets between different sections of a network. Therefore, they cannot efficiently distribute the cost/constraint budgets within different portions of a SFC (budget-division problem). This can potentially reduce the quality of found solutions dramatically as well as the acceptance rate. In contrast, Holu employs an end-to-end decision-making strategy to solve the PD-VPR problem. This strategy consists of placing all the VNFs at once, and then find the best route, both considering the power consumption. In this way, we can solve the budget-division problem, which exists in the state-of-the-art sequential solutions. Moreover, a fallback mechanism improves the quality of solution in an iterative way.

To do so, we divide the PD-VPR in two sub-problems, regardless of the length of the requested SFC: i) Sub-problem 1: VNF placement to minimize $P_{pm}$, and ii) Sub-problem 2: routing aiming at minimizing $P_{net}$, while meeting the delay constraints. As depicted in the flowchart in Fig. 2, having a request $r$ in hand, the VNF placement sub-problem finds a set of PMs to host the VNFs $f \in C_r$. Particularly, Holu uses a ranking mechanism to assign a score to the candidate PMs for hosting the VNFs requested in the $C_r$. Thereafter, it builds a matrix (referred as exploration matrix) to represent the candidate PMs that can host VNFs $f \in C_r$ and sorts them based on the calculated ranking values. According to this sorting, a potential VNF placement solution (the PMs to host VNFs $f \in F$) is selected and passed as an input to Sub-problem 2, which attempts to find a routing through these PMs, considering the delay constraint $D_r$. To do so, we employ our previous work [19] to find a delay-constrained path to route the traffic through the candidate PMs in a power-aware manner. If the routing is not successful (no path is found with respect to delay/capacity constraints), another set of candidate PMs is selected by Sub-problem 1 and the routing is recomputed.

Note that since it is not straightforward to select a good set of PMs at once (as it would imply solving the whole problem at once), there is an iteration in order to gradually improve the solution. This loop is repeated either until a successful routing is found, which leads to accepting the request $r$, or a stopping condition (e.g., VNF placement returns no result), in which case the request $r$ is rejected. An illustrative example is presented in Fig. 3.

As a result, by identifying the VNF hosting PMs for the SFC (Sub-problem 1) and finding a path passing through these PMs (Sub-problem 2), we are able to distribute the cost (power) and the delay budget between different segments of the SFC. In below, the details of these two sub-problems and their respective solutions are presented.

**A. Sub-problem 1: VNF Placement**

As mentioned before, the VNF placement process identifies to map each of VNFs in the requested $C_r$ on a PM in network with the goal of minimizing the first element of the objective function, $P_{pm}$ (See the ILP model (17)). In order to improve power-efficiency, we focus on increasing the reusability of VNFs by sharing them between more SFCs, thus, increasing their utilization and minimizing the number of online PMs (i.e., improving power-proportionality, See Fig. 1). Therefore, it is critical for the VNF placement process to identify the PMs that maximize the VNF reusability factor. We achieve this goal by introducing a PM ranking strategy, as explained below.

1) PM Ranking Mechanism: Given the request $r$, we determine a set of PMs that are the best candidates for hosting each VNF $f \in C_r$. To do so, we employ a mechanism that assigns a ranking value to each PM, according to the requested SFC $C_r$. We define the score of PM $n \in N$ to host VNF $f \in C_r$ as

$$
\gamma_{n,f} = \alpha_{n,f} + \beta_{n,f}, \quad f \in C_r, n \in N
$$

In following, we define $\alpha_{n,f}$ and $\beta_{n,f}$ in detail.

**Node Centrality Impact ($\alpha_{n,f}$):** Given a network topology, a wide spectrum of centrality metrics can be defined in order to find the most critical/important node(s) in a network [76], [77]. Metrics relying only on the node degree have been shown to not improve the resource reusability as they incur higher network power consumption or too long delays [77]. Hence, in this work, three centrality metrics relying on the length of shortest-paths between all node pairs are used to determine the value of $\alpha_{n,f}$: Betweenness (BC), Closeness (CC), and Katz Centrality (KC) [76], [77]. Using one of the three chosen metrics, we calculate the centrality value for all the nodes...
n ∈ N and then normalize the obtained values between 0 and 1 to obtain $\gamma_{n}^{f} \forall n \in N$.

**Power Consumption Impact ($\beta_{n}^{f}$):** This metric aims at prioritizing PMs which causes the minimum increase in the power consumption to process a new SFC request $r$ (either by creating a new VNF instance or using an existing one). The $\beta_{n}^{f}$ can take three different values based on the state of a PM $n \in N$ with respect to VNF $f \in C_r$ (the power consumption caused by hosting $f$ is increasing from first to the third case):

- $\beta_{n}^{f} = 1$: The PM $n$ is *online* and already hosts an instance of VNF $f$ with enough available capacity to process the data rate $B_r$.
- $\beta_{n}^{f} = 0.1$: The PM $n$ is *online* and has sufficient resources to launch a new instance of VNF $f$.
- $\beta_{n}^{f} = 0$: The PM $n$ is *standby* and must be turned on to launch a new instance of VNF $f$.

The $\beta_{n}^{f}$ values can be tuned by the operator e.g., based on the power-proportionality of PMs and/or the size (impact) of specific VNF types that is being used. After determining the values of $\alpha_{n}^{f}$ and $\beta_{n}^{f}$, upon receiving a SFC request $r$, for each $f \in C_r$, the values of $\gamma_{n}^{f}$ can be computed. Note that the weights of $\alpha_{n}^{f}$ and $\beta_{n}^{f}$ metrics can be changed according to the operator preferences/priorities.

2) **PM Exploration Matrix:** As mentioned before, given a request $r$, we would prefer to place the VNF $f \in C_r$ on a PM with the highest $\gamma_{n}^{f}$ value. Here, the challenge is how to select this set of candidate PMs, which is denoted by $S_r$, where $|S_r| = |C_r|$. To do so, we define a PM exploration matrix $M^{f}_{|C_r| \times |N|}$ which represents the search space of the candidate PMs: each row represents a set of candidate PMs to host a VNF $f \in C_r$, which are sorted in decreasing order according to their $\gamma_{n}^{f}$ value. Potentially, since all the PMs in the network are candidate nodes, there are $N$ columns in the matrix $M^{f}$. Fig. 3b shows an example of a PM exploration matrix $M^{f}$ for a user request $r$. In this exemplary matrix, there are three (i.e., $|C_r|$) rows, one per VNF in $C_r$, and four (i.e., $|N|$) columns.

3) **PM Selection Mechanism:** The VNF placement problem consists in selecting one PM from each row in the matrix $M^{f}$. Since the PMs in each row are sorted based on their $\gamma_{n}^{f}$ value, the first (and the *best*) choice of PM candidates is the first PM from each row in $M^{f}$. For example, in Fig. 3b, the best (first) candidate PMs to host VNFs $\{f_1, f_2, f_3\}$ are supposed to be $n_1, n_2$, and $n_5$, respectively.

Having the set of candidate PMs $S_r$, a path from $V_r^{s}$ to $V_r^{d}$ passing through the sequence of PM nodes $n \in S_r$ should be calculated. However, such a path might not be found because of the routing algorithm incompleteness or capacity/delay constraint violations. If this happens, we retry to find a path for an updated set of candidate PMs. In our approach, we select a PM from the current $S_r$ and replace it by the next PM with the highest ranking value from the respective row in matrix $M^{f}$. For example, in Fig. 3d, after no solution is found for the first candidate PM set $n_1, n_2, n_5$ (Fig. 3b), the PM $n_5$ is chosen to be replaced by the next PM $n_3$ to form the new candidate PM set $n_1, n_2, n_3$.

The question here is which PM from the current candidate PM sets $S_r$ should be replaced. Since different candidate PM sets can lead to different solutions with different qualities, it is important to determine how to select the next set of candidate PMs, which is expected to reduce the power consumption while allowing a path through them guaranteeing the constraints. Having the PM exploration matrix in hand, we propose three different candidate PM selection mechanisms to determine the next candidate PM set:

- **The Highest Node Ranking (HNR):** The first approach uses the $\gamma_{n}^{f}$ metric to select and replace a PM from the candidate PM sets $S_r$. HNR selects the candidate PM with the highest $\gamma_{n}^{f}$ value among the current set of PMs. Then the selected candidate PM is replaced by the next PM in the corresponding row of $M^{f}$. Thereby, this approach relaxes the power savings to meet the end-to-end delay and in-turn, increases the acceptance ratio.
- **The Highest Remaining Capacity (HRC):** HRC selects the PM with the highest remaining capacity and replaces it by the next PM from its corresponding row in matrix $M^{f}$. The idea is to use up PMs fewer remaining compute resources over PM with more compute resources so that the power-efficiency is higher (See Fig. 1).
- **The Largest Sub-path Delay (LSD):** The third approach uses the sub-path delay metric to identify and prune a
PM from the candidate PM set. To do so, we first divide the shortest-path from \( V_r^s \) to \( V_r^d \) through candidate PMs of \( S_r \) into \(|C_r|\) sub-paths: \( V_r^f \) to \( f_2 \) (here \( f_1 \) is the intermediate VNF), \( f_1 \) to \( f_3 \), and \( f_2 \) to \( V_r^d \). Each sub-path contains a single PM hosting a VNF as an intermediate PM. Then, for each sub-path, we calculate the shortest-path from the start to the end node of the sub-path with the delay as the cost function. Afterward, we prune the PM which is an intermediate PM of the sub-path with the largest sub-path delay from the current PM set. As a result, the candidate PM with a high contribution to the end-to-end delay of the original path is replaced with the next PM from the respective row in the matrix \( M' \). Fig. 4 shows an example where a path and its delay from \( V_r^s \) to \( V_r^d \) through \( f_1 \), \( f_2 \), and \( f_3 \) VNFs is shown.

The heuristic continues to try different candidate PM sets \( S_r \) until a solution meeting the delay and capacity constraints is found or a stopping condition is reached (See Fig 2). This condition can be chosen by the operator based on the network size and the service provider’s decision time constraints.

### B. Sub-problem 2: Routing

In this section, we present the solution to the routing sub-problem which aims at minimizing the second element of the objective function, \( P_{net}^T \) (Eq. 17). At this stage, the user request \( r \) and an ordered set of candidate PMs \( S_r \) are given by Sub-problem 1. The problem now, is to find a path from source \( V_r^s \) to destination \( V_r^d \), that traverses the candidate PMs in \( S_r \), and that satisfies the delay constraint \( D_r \). Given a cost function that models the power consumption of a path, the problem is to find a Delay-Constrained Least-Cost (DCLC) path from a given source \( src \) to destination \( dst \), which can be modeled as:

\[
z^*(src, dst) = \min_{p \in P_{src,dst}} \sum_{(i,j) \in p} c_{i,j} \tag{19}
\]

\[
\text{s.t.} \sum_{(i,j) \in p} d_{i,j} \leq D_r
\]

where \( P_{src,dst} \) is the set of paths from node \( src \) to \( dst \), and \( c_{i,j} \) and \( d_{i,j} \) are the cost (i.e., network power consumption) and delay functions of link \( (i,j) \) respectively.

In the following, we show how to solve Sub-problem 2 by using the Lagrange Relaxation based Aggregated Cost (LARAC) algorithm [78] and an extension of it, LARAC-SN [19]. Let us first shortly describe these two algorithms.

1) **LARAC Algorithm:** The DCLC problem is NP-hard [79]. Accordingly, numerous heuristics have been proposed to quickly find close to optimal solutions [80]. The LARAC algorithm finds a DCLC path in a graph by running several times the Dijkstra shortest-path algorithm. It is commonly considered as one of the best performing heuristics for the DCLC problem [19], [80]. The algorithm is based on the Lagrange relaxation, a mathematical optimization technique for solving constrained optimization problems. Using the Lagrange relaxation principle, a heuristic solution \( z^R(s,r) \) to the DCLC problem (Eq. 19) can be found by optimally solving [80]:

\[
z^R(V_r^s, V_r^d) = \max_{A \in \mathbb{R}^+} \left\{ \min_{p \in P_{r_r}^{V_r^s \rightarrow V_r^d}} \sum_{(i,j) \in p} c_{i,j} + \lambda \left( \sum_{(i,j) \in p} d_{i,j} - D_r \right) \right\}
\]

Solving this maximization problem requires to solve several times the inner minimization problem (called relaxed problem) by varying the \( \lambda \) parameter. Interestingly, for this problem, the relaxed problem corresponds to a shortest-path problem with a modified cost function \( c^\lambda = c_{i,j} + \lambda \times d_{i,j} \) [78]. As a result, LARAC finds a heuristic solution to the DCLC problem by successively running Dijkstra with the modified cost function \( c^\lambda \) and by adapting the \( \lambda \) at each iteration to converge to the maximum of Eq. 21.

LARAC starts by finding the least-delay and least-cost paths in the network from a given source node \( src \) to destination \( dst \). If the least-delay path does not exist, it returns null since there no path exists that meets the delay constraint. If the least-cost path (\( \lambda = 0 \) does not violate the delay constraint, the path is returned as the solution (which is the optimal solution for sub-problem 2). Otherwise, if the delay of the least-cost path is higher than \( D_r \), it performs subsequent Dijkstra runs with the modified cost function \( c^\lambda \). At each iteration, \( \lambda \) is adapted in order to give more or less importance to the cost and/or the delay. More details on the rationale behind the convergence strategy are available in the original reference [78] and the survey [80]. The algorithm is complete, i.e., it always finds a solution if it exists, but not optimal. However, it was shown that its optimality gap is quite low [80] while keeping a relatively low runtime. Thus, it becomes a perfect candidate to be used as part of the solution to Sub-problem 2 that we solve repeatedly to obtain a solution to our original problem.

2) **LARAC-SN Algorithm:** LARAC-SN builds on top of LARAC to force the traversal of an ordered set of nodes [19]. To do so, the algorithm adapts the underlying Dijkstra procedure used by LARAC. Instead of running Dijkstra from the source to the destination node, LARAC-SN splits the Dijkstra run from the source node to the first VNF, from the second VNF to the next one, and so on until the destination. By doing so, the algorithm ensures that LARAC only considers paths that traverse the set of candidates PMs and hence that the final path satisfies this constraint. As we ensure that \( p_c \) and \( p_d \) always traverse the VNFs of \( C_r \) (i.e., the PM nodes in...
S_r), the path returned by LARAC-SN will also traverse the VNFS. Splitting the underlying Dijkstra run rather than the LARAC run removes the need for splitting the delay constraint between the different VNFS. That is automatically handled by the LARAC procedure by adapting the weights of the aggregated cost function, which can significantly increase the acceptance rate of the requests. Inheriting the properties of LARAC, LARAC-SN is complete and close-to-optimal [19].

3) Routing Metrics: Delay, Cost, and Capacity: The first metric is the delay. It is a global constraint metric. The delay of a link corresponds to its propagation delay (the VNF processing delay can be pre-computed and subtracted from the delay constraint budget D_r), thus, its value for a link is static for a given routing request. The second metric is the cost which is a global optimization metric. Our cost function essentially considers the power consumption impact of taking a given path. The cost of a link (i, j) is defined as

\[ c_{i,j} = \hat{p}_{i,j} + \hat{Q}_j, \]

where the notation represents the fact that both values are normalized to a value between 0 and 1. As it can be seen, \( c_{i,j} \) consists of two cost elements. The first one is the impact of power consumption \( \hat{p}_{i,j} \) for using a given link \((i, j)\) and it is computed as

\[ \hat{p}_{i,j} = \frac{1}{2} p_{\text{idle}} \times (2 - y_i - y_j) + 2 p_{\text{port}} \times (1 - q_{i,j}), \]

where \( y_i \) and \( q_{i,j} \) are binary variables reporting whether the switch \( i \) or link \((i, j)\) are already powered on, respectively. In fact, variables \( y \) and \( q \) allow to penalize the usage of a switch or link that is not yet used (i.e., that is in the standby state). The second term forming the \( c_{i,j} \) is the centrality metric \( \hat{Q}_j \) of the destination node \( j \) of the link. The reason for adding this term is that the power consumption of links can often be equal, as only six different values are possible (based on the \( y \) and \( q \) variables). As a result, the routing algorithm will often face equal-cost paths. Instead of letting the algorithm pseudo-randomly choose among these paths, the centrality term acts as a tie breaker to direct the algorithm towards more central nodes that tend to be used more often and hence reduce the power consumption of subsequent requests. We note that we use the BC value of the node \( j \) for the \( \hat{Q}_j \). Computing the value of our cost function \( c_{i,j} \) requires to know all the links previously visited by the routing algorithm. Indeed, since the state \( s_{i,k,l} \) of a link depends on whether it is used already (i.e., whether we already visited it), knowing all the previously visited links is necessary to compute \( \hat{p}_{i,j} \) and hence \( c_{i,j} \). In [81], it is shown that such a metric as global optimization metric increases the optimality gap of an algorithm. Yet, the algorithm stays complete.

The third and last metric we use is the capacity of links. It is a local constraint metric. A link can only be used if it still has sufficient capacity to host the new flow. Similar to the cost function, checking if the new flow can be accommodated by the remaining capacity at a link requires to know whether we visited this link already. In traditional routing, it is not the case, because we know the remaining capacity of all links in advance and we do not route into loops. However, when routing through a set of VNFS, we can potentially route through loops. Unfortunately, such a metric as a local constraint metric makes the routing algorithm incomplete [81]. Recovering the completeness of algorithms in such situation leads to intractable runtime [81]. In fact, this is one of the reasons for which we have to iterate back and forth between sub-problems 1 and 2. Indeed, once \( S_r \) has been selected in Sub-problem 1, the algorithm used in Sub-problem 2 cannot guarantee (because it is not complete) to find a solution for the \( S_r \), even if it exists. We iterate between the two sub-problems until the routing algorithm finds a solution. In the evaluation section, we show that the incompleteness of the algorithm is not very high, and hence solutions can be found in relatively few iterations and hence, short runtime.

VI. PERFORMANCE EVALUATION

A. Simulation Setup

This section provides details about the network topologies and the simulation parameters considered for the evaluation. The results presented in this section correspond to: NobelEU [82], and Internet2 [83]. Similar results have been obtained for Geant topology [84] but not included due to space limitations. Each network node is equipped with a switch and a PM for hosting VNFS. The PM resources are characterized by the number of CPU cores, which has been set to 16. The PM power consumption is computed in accordance to Eq. 3. The idle and maximum power consumption of the PM is chosen as \( p_{\text{idle}} = 299 \) Watts and \( p_{\text{max}} = 521 \) Watts, respectively [85]. On the other hand, the network switch power consumption consists of a static hardware power and network interface power (See Eq. 2). The static hardware power of the switch \( p_{\text{switch}} \) is set to 315 Watts [86]. The network interface power consumption of switch port varies based on the data rate configuration, which is modeled to operate at three different configurations: 100 Mbps, 1 Gbps, or 10 Gbps with 26 Watts, 30 Watts, and 55 Watts of power consumption, respectively [64], [87]. The physical link data rate capacity is set to a randomly generated value between 6 to 10 Gbps. To generate the user requests, the source and destination pairs are generated randomly, such that \( V_r \neq V_r^d \). We have considered commonly used service types: video streaming, web service, voice-over-IP (VoIP), and online gaming [18], [24]. As it is presented in Table III, each of these service types requires a specific SFC, data rate, maximum end-to-end delay, and the share of the service type in the set of user requests. Additionally, the VNFS forming the SFCs have considered with different resource requirements and processing capacities as listed in Table IV [35], [88].

<table>
<thead>
<tr>
<th>Type</th>
<th>SFC Set</th>
<th>Data Rate</th>
<th>Delay</th>
<th>Share</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web</td>
<td>NAT-FW-TM-VOC-IDPS</td>
<td>0.1[6-1] Mbps</td>
<td>500 ms</td>
<td>18.2%</td>
</tr>
<tr>
<td>VCI</td>
<td>NAT-FW-TM-VOC-IDPS</td>
<td>0.1[6-1] Mbps</td>
<td>500 ms</td>
<td>18.2%</td>
</tr>
<tr>
<td>Streaming</td>
<td>NAT-FW-TM-VOC-IDPS</td>
<td>0.1[6-1] Mbps</td>
<td>500 ms</td>
<td>18.2%</td>
</tr>
<tr>
<td>Gaming</td>
<td>NAT-FW-VOC-IDPS</td>
<td>0.1[6-1] Mbps</td>
<td>500 ms</td>
<td>18.2%</td>
</tr>
</tbody>
</table>

TABLE III: Overview of service types and their SFC set, data rate, end-to-end delay, and share of the total requests. We have considered request aggregation by considering a range of data rate to mimic the traffic coming from multiple users at the same time from a single network node. (NAT: network address translator, FW: Firewall, TM: traffic monitor, WOC: WAN optimization controller, IDPS: intrusion detection prevention system, VOC: video optimization controller).
<table>
<thead>
<tr>
<th>VNF Type</th>
<th>Num. CPU Cores</th>
<th>Processing Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAT</td>
<td>2</td>
<td>500 Mbps</td>
</tr>
<tr>
<td>FW</td>
<td>8</td>
<td>400 Mbps</td>
</tr>
<tr>
<td>TM</td>
<td>1</td>
<td>200 Mbps</td>
</tr>
<tr>
<td>VOC</td>
<td>2</td>
<td>580 Mbps</td>
</tr>
<tr>
<td>WOC</td>
<td>2</td>
<td>300 Mbps</td>
</tr>
<tr>
<td>IDPS</td>
<td>8</td>
<td>600 Mbps</td>
</tr>
</tbody>
</table>

TABLE IV: The number of CPU cores and their processing capacity (maximum throughput) of each VNF type.

To choose the best centrality metric and PM selection mechanism for Holu, we have extensively evaluated them in terms of power consumption and acceptance rate. Based on the results (omitted due to the lack of space), we have selected the best-performing settings to evaluate Holu. In particular, for each node \( n \in N \), we use CC as centrality metric \( \alpha_n^f \) and the LSD as the PM candidate exploration strategy (supportive evaluations are presented in the results section). The proposed ILP optimization model (17) has been implemented using Gurobi solver [89] in Python. Besides, Holu framework and the state-of-the-art approaches are simulated using Java. The simulations have been performed on a machine equipped with Intel Core i7-6700HQ @2.60 GHz, 16 GB of RAM, running Ubuntu 18.04.

B. Algorithms to Compare

We compare the proposed ILP model and the Holu heuristic with two state-of-the-art algorithms: i) A state-of-the-art approach named CPVNF [20], and ii) The BCSP algorithm, which was presented in our previous work [18] (referred as the BC-Based algorithm). These two algorithms are briefly explained in below:

i) CPVNF: The CPVNF algorithm [20] addresses VNF placement and routing problem with the objective of minimizing the number of online PMs and communication costs, while meeting end-to-end delay and resource capacities. CPVNF acts in a sequential way (solving VNF per VNF). To select high-quality PMs in the network to host each VNF, it uses a ranking algorithm based on the Google PageRank method. PageRank is a variant of the Eigen Vector centrality measure [90] and assigns ranks to a web page based on both the quality and quantity of the web pages linked to it. Comparatively, the authors of CPVNF algorithms assign ranks to PMs based on the remaining capacity of PM itself, aggregated remaining bandwidth of its outgoing links, and rank of its directly connected nodes [20]. Additionally, nodes are assigned personalized PageRank to bias towards PMs with VNF instance under question. For every request, the CPVNF algorithm computes the rank metric along with a delay function to select the VNF hosting PMs. CPVNF uses the k-shortest-path algorithm to compute the routing at each stage, and selects the path with the lowest delay value. If the end-to-end delay is less than the \( D_r \), the user request is accepted in the network. Otherwise, a new set of VNF hosting PMs is selected by increasing the importance of aggregate outgoing link bandwidth over the remaining CPU resource in node rank computation. The search continues until the iteration reaches the predefined search limit.

ii) BCSP: This algorithm represents the BC-based algorithm in our previous work [18]. This algorithm performs the VNF placement and routing sequentially for each VNF in the requested SFC. BCSP works based on the BC centrality metric for VNF placement and shortest-path for routing. It first calculates the BC metric for all the nodes in the graph. Then, for each request \( r \), it calculates the shortest-path between source \( V_r^s \) and destination \( V_r^t \) nodes using Dijkstra with delay as the cost function. Then, the BCSP algorithm tries to place the required VNFs of the \( C_r \) on the nodes with higher BC value on the shortest-path in a greedy manner. According to the BC definition, a higher BC value means a higher number of shortest-paths that are passing through a node \( n \) and hence, a higher probability of VNF re-usability for future requests, which can lead to power-efficiency.

C. Simulation Results

This subsection summarizes the performance evaluation on two topologies. Before presenting the results, we note that due to the scalability issues, we were able to run the ILP model for up to 25 user requests (more than 48 hours of computation time for a single problem instance with 50 requests).

1) Power Consumption: Let us start comparing the total, PM, and network power consumption per accepted user request for the optimal case and the three heuristics for different number of user requests. Fig. 5 shows the mean power consumption per accepted request value. Among all the heuristics, it can be observed that the Holu has the lowest total power consumption per request values in all the topologies. In more detail, Holu is performing on average 19.3% worst than the optimal solution, and outperforming CPVNF and the BCSP algorithms by 19% and 24.7%, respectively. However, for higher number of requests, the network gets saturated and hence, the gap between the different heuristics decreases.

All the three heuristics utilize centrality measures to determine the PMs to place VNFs. However, it is evident from the results that the effectiveness of all the centrality measures is not the same. The performance of the CPVNF algorithm decreases when the network contains densely connected regions away from the graph center (e.g., more connected at the edges of the network). That is, unlike the ranking method of Holu, the PMs with higher ranks have more distance from other high rank nodes, making it difficult to meet the required SFC delay constraint. For instance, Internet2 is from this group of graphs, which the power consumption of CPVNF is even worse than the BCSP algorithm, because BCSP is always using the shortest-path (cost is considered as delay) between source and destination (See Fig. 5a). On the bright side, the Holu algorithm first ranks the PMs by using the CC and their power consumption impacts (i.e., \( \alpha_n^f \) and \( \beta_n^f \), respectively) The CC value of a node depends on its position in the network for all other nodes. As a result, Holu can identify the PMs that increase the reusability of resources, leading to lower power consumption. The average number of online PMs per request have been summarized in Table V for 25 user requests. The lower number of online PMs per user request for Holu compared to the other heuristics reflects the effectiveness of the proposed PM ranking and selection approach. Concerning BCSP, the disadvantage when solving the VNF placement
problem, is that the search space is limited to the nodes along the shortest-path between request’s source and destination. As a result, the resource reusability decreases, which leads to an increase of online PMs in both topologies.

2) Acceptance Rate: In our next study, we run Holu, CPVNF, and BCSP for up to 500 user requests to compare the acceptance rate for the two topologies. As we were able to run the ILP for a maximum of 25 requests, the feasibility of the problem for the larger inputs is unknown to us and requires further investigations. Also, we note that the ILP model either accepts all the requests or nothing, therefore it is omitted from the plots. As it can be seen in Fig. 6, Holu can accept 31% and 20% more requests for Internet2, and 16.3% and 14.1% for NobelEU topology compared to the CPVNF and BCSP, respectively. An important parameter in successfully accepting a request is meeting its delay requirement. The results in Fig. 6 indicate that opposed to the other algorithms, by employing LARAC, Holu is able to divide the delay budget between the VNFs in the chain. In this way, it can increase its delay-awareness which leads to a significant improve in acceptance ratio.

The CPVNF algorithm sorts the PMs of the network based on their ranking (i.e., according to the PageRank metric), ignoring the delay role. Thus, it is possible that the node with a high ranking is located far from the request’s source and/or destination. As a result, the delay requirement of the user request may not be guaranteed and hence, the request is rejected. Besides, the BCSP algorithm uses the BC metric for the PMs along the shortest-path to select the candidate nodes for VNFs. Therefore, a node with a high BC value may be located closer to the user request’s destination node. As a result, the BCSP algorithm needs to make a loop along the shortest-path to realize the SFC. Under such circumstances, the actual SFC path computed by the BCSP is longer than the shortest-path which can lead to missing the delay requirement, and rejecting the request. Also, we can see that the acceptance rate for BCSP is higher than CPVNF for the Internet2 topology and almost similar for the NobelEU. The Internet2 topology contains high ranked nodes (according to PageRank metric) far away from each other and edges with large lengths (i.e., higher delay). As a result, the CPVNF algorithm performs poorly as compared to the BCSP algorithm in meeting the delay requirement, hence reducing the user request acceptance rate. Therefore, it can be seen that the coordination of an efficient VNF placement and a power and delay-aware routing can play an important role in the acceptance delay. Thereby, Holu is able to balance the power and delay requirement, achieving a higher acceptance rate.

3) Path Stretch: The path stretch metric represents the end-to-end delay difference between the path computed by each algorithm and the actual shortest-path (based uniquely on delay) between the source and destination request. This metric is important because the lower path stretch can be translated to lower network resource consumption, i.e., lower OPEX costs. Fig. 7 shows the CDF of path stretch values for ILP, Holu, CPVNF, and BCSP approaches for 25 user requests. We omit to show these results for other numbers of user requests, since their behavior is similar. It can be seen that in both topologies, the proposed Holu heuristic achieves the minimum-maximum path stretch compared to all the approaches, followed by CPVNF, BCSP, and the ILP. The
reason is that the delay-aware routing algorithm can efficiently balance the importance of the power vs. the delay metrics. That is, the ILP shows an extreme high path stretch, due to focusing on power minimization, while Holu causes a reasonable path stretch, while having a lower power consumption compared to the other heuristics. Also, the plots indicate that the ILP solution has many instances with a high path stretch values. The reason is that the ILP, in an effort to reduce the total power, makes the requests to traverse through VNF hosting nodes which are not necessarily close to the source and/or destination of the request.

4) Delay Tolerance: This study aims to compare the acceptance rate of Holu, CPVNF, and BCSP algorithms for user requests with varying delay requirements. The goal is to explore how different algorithms can handle user requests with tight or loose delay requirements. For this purpose, we run the simulation for different sets of user requests, ranging from 25 to 500. Each of these user requests are generated with random source and destination pairs, 4 Mbps data rate, and a specific SFC type (See Table III). Moreover, we select the delay values of user requests from 80 to 150 ms in steps of 10 ms. The goal is to check which algorithm is more sensitive to the delay variations in terms of acceptance rate. Fig. 8 shows the heatmap of the acceptance rate for a different number of user requests with varying delay requirements for both topologies. These plots exhibit that Holu has a generally higher acceptance rate for all the delay ranges and for different number of requests. This is because Holu puts emphasis on the delay parameter during the selection of VNF hosting nodes and routing process. Thereby, it succeeds in achieving a greater acceptance rate even under strict delay conditions. The CPVNF has the second-best performance after Holu and BCSP has the worst performance. The reason is CPVNF applies a more sophisticated ranking method and also explores a larger solution space compared to the BCSP approach.

5) Runtime: In this subsection, we compare the runtime of the algorithms. To do this, we record the computation time of each algorithm at the end of processing of every 10% of the user requests. Due to space limitations, we only show the comparison of the runtime of different approaches for 100 user requests. We note that we omitted the ILP results from the plot, because it runtime is in order of hours. Fig. 9 shows that Holu can do the admission and solve the PD-VPR for a given user request in 10 ms for NobelEU, and 15 ms for Internet2 network topology. We note that the implementation of Holu can be optimized e.g., by using more efficient data structures to achieve even lower runtime for practical use-cases. Besides, the results in Fig. 9 indicate that the BCSP algorithm is the fastest approach. The main reason behind it is the lower number of times that requires to run Dijkstra (only once per user request compared with several times for the other heuristics). For instance, Holu needs to compute the shortest-path several times iterative for certain requests (depending on the PM selection mechanism).

On the other hand, the CPVNF and Holu approaches require to calculate PM ranking values before processing a user request resulting in higher runtime. After processing some of the initial 40-50% of the user requests, the CPVNF algorithm requires more time to process the remaining user requests compared to the other approaches. This is because it saturates the the high ranked PMs for serving the initial requests. As a result, it requires more iterations to find the VNF hosting PMs which meets the delay requirements for remaining user requests. Moreover, unlike our approach, the CPVNF needs to recompute the node ranks at every iteration, which leads to a higher runtime.
Motivating by low power-proportionality of both Physical Machines (PMs) and network switches, we formulate the power-aware and delay-constrained joint VNF placement and service function chain (SFC) routing problem (PD-VPR). Since the resource are usually under-utilized and/or over-provisioned, minimizing the number of online devices can lead to improved resource utilization and more power-proportionality. To achieve this goal, we first proposed an Integer Linear Program (ILP) model. Due to its scalability issues, we presented an online heuristic framework named Holu which tackles the problem by breaking it into two sub-problems which are solved sequentially: i) Virtual Network Function (VNF) placement, and ii) routing. The VNF placement suggests a set of candidate PMs to host the requested SFC. It uses a PM ranking mechanism considering the centrality of the PM and the power consumption impact by hosting a VNF. We show that centrality is an important metric to consider for PM ranking, since it can improve the resource utilization and power-efficiency in long-term. Also, to find a path traversing through the set of suggested candidate PMs (returned by the VNF placement sub-problem), we employed a complete algorithm based on Lagrange Relaxation based Aggregated Cost (LARAC) heuristic to solve a Delay-Constrained Least-Cost (DCLC) shortest-path routing problem. We showed that our algorithm is able to efficiently split the delay budget between two consecutive VNFs in an SFC, leading to a high acceptance ratio compared to state-of-the-art algorithms. Our simulation results indicated that compared to existing approaches, the end-to-end SFC placement and routing approach, employed by Holu, is able to determine the VNF to PM mapping and also accurately split the delay budget among the routing paths between the consecutive VNFs in the SFC, improving its power consumption and acceptance rate up to 24.7% and 31%, respectively.

Future work. We believe there are many interesting open challenges to be tackled, especially in improving the coordination of the VNF placement and routing sub-problems. For example, one can introduce a distance metric from the source and/or destination to the PM ranking calculation (See Eq. 18). Using that, similar to the A* algorithm [91], this information can be used to prune the candidate PMs that are far from source and/or destinations. This can improve the completeness of Holu, hence, its performance, especially in terms of acceptance rate. In addition, PM selection methods can be improved to generate better candidate PM sets. For example, one can replace multiple PMs based on more complex PM selection methods at each iteration to improve the search space exploration. These improvements can also lead to reducing the average number of VNF placement iterations, thus, reducing the runtime. Finally, a resource (PM and network) consolidation module can be added to Holu, which can be triggered periodically or in case of resource over/under-load. For example, in case of resource overload, it can use live Virtual Machine (VM) migration to migrate the workload away from the hot spots, thus, potentially improving the acceptance rate. Also, in the case of under-load, it can move the load away from an under-utilized region to further improve the total power-efficiency.

ACKNOWLEDGMENT

The authors would like to thank Nemanja Deric and the anonymous reviewers for their fruitful comments. This work is funded by Germany Federal Ministry of Education and Research under project OptiCON (grant IDs #16KIS0989K and #16KIS0991).

REFERENCES


