A Hybrid Virtual Network Function Placement Strategy for Maximizing the Profit of Network Service Deployment Over Time-Varying Workloads

CHI-CHEN YANG1, SATYAJIT PADHY2, AND JERRY CHOU1

1Department of Computer Science, National Tsing Hua University, Hsinchu 300044, Taiwan
2Institute of Information System and Applications, National Tsing Hua University, Hsinchu 300044, Taiwan

Corresponding author: Jerry Chou (jchou@lsalab.cs.nthu.edu.tw)

ABSTRACT The emergence of network function virtualization (NFV) has revolutionized the infrastructure and service management of network architecture. It allows network operators to reduce costs and improve the agility of network service deployment. But finding the best VNF placement is a well-known NP-complete problem. Hence, many previous studies either formulate the problem as an Integer Linear Programming (ILP) Problem or propose greedy algorithms. However, solving ILP is time-consuming, while greedy algorithms could be far from optimal solutions. As a result, neither ILP nor greedy can make quick and accurate placement decisions for dynamic traffic workloads. Therefore, we propose a hybrid method that uses less time to maximize the overall profit of network service deployment. Our evaluations based on real backbone network traffic and topology show that our hybrid approach can achieve up to 36% profit improvement compared to a pure greedy approach, while achieving x30 times computation time speedup over a pure ILP approach.

INDEX TERMS Virtual network function, service function chain, time-varying workload, placement optimization, integer linear programming.

I. INTRODUCTION

The emergence of network function virtualization (NFV) has revolutionized the infrastructure and service management of network architecture. NFV is a technique that virtualizes proprietary hardware-based appliances, such as firewall, load balancer, switch etc. into softwarized Virtual Network Function (VNF) that can be containerized and hosted on commodity servers. The VNF can then be used as the building blocks to form a service function chain (SFC) to provide network services to users. Through NFV, network operators not only can reduce their cost on Capital expenditures (CAPEX), operating expenses (OPEX) and power consumption, but also improve the time and flexibility of network service deployment. Therefore, it has been actively studied by industry [1] and research communities [2] in recent years.

One of the key challenges of NFV orchestration is the VNF placement problem, which places the VNFs of a set of service function chain requests (SFCRs) on physical nodes in order to satisfy the resource and traffic demand of SFCRs. The goal of the placement problem is to minimize the deployment cost and maximize the service quality of the SFCRs. In general, the deployment cost includes the resource consumption from computing power and network bandwidth, and service quality is measured by the end-to-end latency delay of the user requests passing through the whole SFC. The VNF placement significantly impacts the deployment cost and service quality because the same type of network functions (NFs) placed on the same physical node can share the same VNF instance, and the location of NFs can affect the routing path of SFCRs. However, finding the optimal VNF placement is difficult, and proven to be a NP-complete problem [3]–[5].

A common approach is to formulate the VNF placement problems as Integer Linear Programming (ILP) or Mixed Integer Linear Programming (MILP) problems [6]–[8], and solve the problem using ILP solvers, like CPLEX [9]. But ILP solvers are time-consuming, especially for larger problem sizes. Considering the workload of SFCRs could change...
rapidly over time, it is infeasible to use ILP solvers to find new placements frequently. Therefore, other studies [10]–[13] proposed greedy algorithms that aim to reconfigure placements in response to large-scale problem size and time-varying workloads. But in the end, neither ILP nor greedy algorithms can make quick and accurate placement decisions for large-scale networks and time-varying workloads in a timely manner.

To address the aforementioned problem, we investigate the idea of combining the two approaches (ILP and greedy algorithms) together, and proposed a hybrid solution as follows. First, by studying the network traffic of backbone networks, we observe that the traffic with greater volume tend to have more stable demand over time, while the traffic with less volume tend to be less stable. So we classify SFCRs according to their workload variations over time. Then we use an ILP solver to find the initial placement for the stable SFCRs, and use a greedy algorithm to place the remaining unstable SFCRs. Finally, we propose another greedy reconfiguration algorithm to migrate existing VNFs or initiate new VNFs when the time-varying workload cause insufficient resource on servers or links. Our evaluations show that our hybrid approach can achieve up to 36% profit improvement compared to a pure greedy approach, while achieving x30 times computation time speedup over a pure ILP approach. Comparing to a state-of-the-art algorithm proposed in [14], Hybrid increases its profit by 18%, while taking hundreds of times shorter computation time.

The rest of the paper is structured as follows. Section II defines our VNF placement problem. Section III describes our proposed hybrid solution including the ILP formulation and greedy algorithms. Our experiment setup and result are presented in Section IV and Section V, respectively. Section VI describes related work, and Section VII concludes the paper.

II. VNF PLACEMENT PROBLEM

In this section, we first define our VNF placement problem, and illustrate it with an example. Then the formal formulation of our problem objective is given.

A. PROBLEM DEFINITION

We consider the problem from a network operator point of view with several requests of network service (NS) that must be served. A network service is composed of network functions (NF), such as Firewall, IDS, Load Balancer, etc. The chain of NFs is called a Service Function Chain (SFC), and the NFs of a SFC must be executed in order. A request of network service is called a Service Function Chain Request (SFCR). To service SFCRs, network operators have to initiate virtual network functions (VNF) on physical machines (PM). We consider VNFs have multi-tenancy, so the traffic of the same type of NF from different SFCRs can be served by a single VNF. The traffic of SFCRs also needs to be routed through the VNFs, and the profit of serving a SFCR will depend on its end-to-end routing latency.

Since the traffic workload can change over time intervals, the placement decision needs to be reconfigured when the resource consumption of SFCRs exceeds the resource capacity of physical machines or links. The reconfiguration can be realized by either redirecting traffic to other VNF instances or initiating new VNF instances. These reconfiguration operations have associated costs because they may cause service interruptions and consume additional time delay or resource consumption. Therefore, the objective of our problem is to decide the VNF placement at each time interval, so that the net profit (considering service revenue, deployment cost, and redirection cost) can be maximized under the constraints of nodes/links resource capacity, and SFCR service latency requirement.

Figure 1 illustrates our VNF placement problem with 2 SFCRs and 4 PMs. Each SFCR is denoted by a service function chain and a pair of ingress node and egress node. For instance, SFCR1 is from PM4 to PM2, and it is composed of NF a, NF b and NF d; SFCR2 is from PM1 to PM4, and it is composed of NF c, NF a and NF b. We assume the latency delay for each link is 1, and the latency threshold for both SFCRs is 1. As defined in Eq 8, the resource consumption on a node can be divided into base and workload-dependent. Here, we assume the base consumption of a VNF is 2 units of resource, and the workload-dependent consumption of each NF is indicated by the numbers in the brackets. Each PM has 15 units of node capacity. For simplicity, we do not consider the link usage and capacity in this example yet.

Placement (a) & (b) in Figure 1 shows two feasible solutions to the above VNF placement problem.

In placement (a), we only deploy one VNF instance for each type of NFs. VNF a and VNF b are on PM4, VNF c is on PM1, and VNF d is on PM2. The pink line indicates the routing path for SFCR1 from PM4 to PM2, while the blue line indicates the routing path for SFCR2 from PM1 to PM4. The VNF a and VNF b are shared by both SFCR1 and SFCR2. Both routing paths pass through the requested NFs of their SFCRs, and the path lengths are within their latency thresholds (i.e., 1). The resource consumption on a node is the sum of the base consumption of its hosted VNFs, and the workload-dependent consumption of all the SFCRs passing...
through. As shown in the figure, the resource consumption on all the nodes are less than their node capacity (i.e., 15). In comparison, placement (b) deploys one VNF instance for each NF of SFCR1 and SFCR2. The requirements of resource constraint and latency threshold are still satisfied under the placement. But the total number of VNFs and the amount of resource consumption from placement (b) are clearly higher than placement (a). So placement (a) is a better solution. Also, we can observe traffic changes can easily trigger reconfiguration. For instance, if the traffic of SFCR2 is double at time 13 to 16, and the total resource consumption on PM4 in placement (a) will increase from 13 to 16, and the total resource consumption on PM1 in placement (b) will increase from 12 to 18. Therefore, loading must be reduced on the violation nodes by rerouting the traffic to VNFs deployed on other nodes.

B. OPTIMIZATION OBJECTIVE

Here, we formally define the problem inputs/outputs and optimization objective. The problem inputs are a physical network, $G$, and a set of SFCRs, $S$, consisted of different types of network function, $F$. $G = (N, L)$ is a graph, where $N$ indicates the set of physical machines and $L$ indicates the set of network links. Each $n \in N$ has a node capacity denoted as $C_n$. Each $l \in L$, between two nodes $n$ and $m$, has a bandwidth capacity and latency delay denoted by $B_{n,m}, E_{n,m}$, respectively.

For each $s \in S$, its service latency threshold is denoted as $\theta_s$ and its traffic is denoted as $D_s$, routing from $i$ to $s$ to $d$. The network functions of a SFCR is denoted by binary variable $V_{s,i,f}$, so that $V_{s,i,f} = 1$ means the type of $f$ of a SFCR $s$ is $f \in F$. We use two dummy network functions to represent the ingress and egress nodes. We assume a SFCR does not have the same type of NFs, so the length of SFCR is bounded by $|F| + 2$. We also use $F_{X,i}$ to denote the node resource consumption of the $i$-th NF of SFC s, and use $F_{L_{i,j}}$ to denote the link resource consumption for routing the traffic between the $i$-th and $i + 1$-th NFs of SFC $s$.

The problem output is a placement decision denoted by three binary variables: $X, Y, Z, x_{s,i,n}^f, y_{s,i,n,m}^f, z_{f,n}$, $\in X$ indicates whether $i$-th NF of SFCR $s$ is hosted on physical node $n$ at time $t$. $y_{s,i,n,m}^f \in Y$ indicates whether $i$-th virtual link in SFCR $s$ is mapped on the physical link from the node $n$ to $m$, at time $t$. $z_{f,n} \in Z$ indicates whether the $f$ type VNF is deployed on the physical node $n$ at time $t$.

Servicing SFCRs receive revenue, while deploying SFCRs results in costs, including resource consumption cost (CC), service deployment cost (DC), and redirection cost (RC). Our objective is to maximize the net profit of the network operator for deploying and providing network services to users. From a network operator point of view, the revenue is from the SFCRs it services, while the cost can consist of resource consumption cost (CC), service deployment cost (DC), and redirection cost (RC). In this work, we formally define the net profit as the equation below, and explain each of the terms in the rest of the section with the notations in Table 1.

$$netprofit = Revenue - \alpha * CC - \beta * DC - \gamma * RC$$ (1)

where $\alpha, \beta, \gamma$ are the weights to balance the three costs.

- Revenue: Most of the network services are latency-sensitive, hence the revenue of a SFCR is strongly correlated to its end-to-end delay [15]. Therefore, we use a utility function, $Utility(d_s)$, to represent the revenue from servicing a unit of demand of SFCR $s$ with delay $d_s$ as:

$$d_s = \sum_{i=0}^{F} \sum_{n=0}^{N} \sum_{m=0}^{N-1} y_{s,i,n,m}^f * E_{n,m}$$ (2)

$$Utility(d_s) = \begin{cases} 1, & \text{if } d_s \leq \theta_3 \\ ((-1 * \exp(\epsilon * (d_s) + \eta)) / \zeta), & \text{otherwise} \end{cases}$$ (3)

$$Revenue = \sum_{s=0}^{S} D_s * Utility(d_s)$$ (4)
Eq. (2) is the delay of a SFCR $s$ which is calculated by the aggregated network delay of all the links along the routing path of $s$. Eq. (3) is the utility function defined by a set of given parameters: $\epsilon$, $\eta$, $\zeta$. The function value must be between 0 and 1, so if the $d_s^t$ is less than a delay threshold $\theta_s$, the utility value is set to 1. Finally, Eq. (4) is the total revenue from serving all the SFCRs.

- **CC**: The main operating cost for a network operator is often proportional to the resource consumption from servers and network equipment. Resource consumption is related to energy consumption. The energy consumption model for a PM is based on its CPU utilization [16], [17], which can be calculated as shown in Eq. (5). where $U_n$ represents the CPU utilization of the PM $n$, $A_n$ indicates whether the PM $n$ is active and $E_{\text{min}}$, $E_{\text{max}}$ are the energy consumption when the utilization of $n$ is 0%, 100% respectively. The CPU utilization can be calculated as the ratio between the total CPU resources consumed and the available CPU resources in the server, as shown in Eq. (7). The resource consumption on servers is caused by the VNF hosted on the servers. As shown in Eq. (8), the computing resource consumption of a VNF can be further divided into basic [18] and workload-dependent resource consumption. The basic resource consumption is the minimum resource consumption to host a VNF on a server, while the workload-dependent resource consumption will be proportional to the traffic demand served by the VNF. Since multiple SFCs can share the same VNF if they require the same type of NF, the workload of a VNF is the aggregated demand from the SFCRs routed through the VNF.

$$E_n = (E_{\text{max}} - E_{\text{min}}) * U_n + E_{\text{min}} * A^t_n, n \in N$$

$$A^t_n = \begin{cases} 1, & \text{if } \sum_{i=0}^{m-1} c_{i,n}^t \geq 1 \text{ and } n \in N, t \in T \\ 0, & \text{otherwise,} \end{cases}$$

$$U_n = CPU_n/(\sum C_n), n \in N$$

$$CPU_n = \sum_{s=0}^{N-1} \sum_{i=1}^{N-1} FC_{s,i} * x_{s,i,n}^t + \sum_{n=0}^{N-1} \sum_{f=0}^{N-1} \Phi_f * z_{f,n}$$

On the other hand, the network resource consumption of a link can be measured by its bandwidth usage, so it can be computed by the aggregated demand from the SFCRs routed through the link as shown in Eq. (9).

$$SBW = \sum_{s=0}^{N-1} \sum_{i=0}^{N-1} \sum_{n=0}^{N-1} D_s * y_{s,i,n,m}^t, n \neq m$$

In order to calculate total resource consumption cost, we first normalize server power consumption and network resource consumption. We divide the total power consumption by the total maximum power consumption and divide the total network resource consumption by the total link bandwidth respectively as shown in Eq. (10) and Eq. (11).

$$CPU = \frac{\sum_{n=0}^{N-1} E_n}{\sum_{n=0}^{N-1} E_{\text{max}}}, n \in P$$

$$BW = \frac{SBW}{\sum_{m=0}^{N-1} B_{n,m}}, n \neq m$$

Therefore, we formulate the total cost of resource consumption cost by Eq. (12).

$$CC = \omega * CPU + (1 - \omega) * BW$$

where $\omega$ is a weighted ratio between the two resource consumption costs.

- **DC & RC**: A new VNF placement means we have to move the location of existing VNFs and create new VNFs. All these operations can cause service interruption and delay to the SFCR, so there should be a penalty cost associated with these operations in our profit formulation.

$$DC = \sum_{n=0}^{N-1} \sum_{f=0}^{N-1} \sum_{s=0}^{N-1} D_s * x_{s,i,n}^t * V_{s,i,f} * (c_{i,n}^t - c_{i,n}^{t-1})$$

$$RC = \sum_{s=0}^{N-1} D_s * r_s^t$$

$$r_s^t = \begin{cases} 1, & \text{if } \sum_{i=0}^{m-1} |y_{s,i,n}^t| > 0 \\ \sum_{i=0}^{m-1} |y_{s,i,n}^t| - \sum_{i=0}^{m-1} |y_{s,i,n}^{t-1}| > 0, & \text{otherwise,} \end{cases}$$

Specifically, we define deployment cost as the service delay penalty from creating a new VNF instance, and this penalty is proportional to the amount of traffic demand as shown in Eq. (13). We also define the redirection cost as the service delay penalty from traffic redirection between VNFs, and this penalty is proportional to the amount of redirected traffic demand as shown in Eq. (14). Eq. (15) indicates whether a reconfiguration has occurred by comparing the mapping with the mapping of the previous time interval.
III. HYBRID VNF PLACEMENT METHODS

A. APPROACH OVERVIEW

It is possible to solve the placement problem using an ILP solver. However, ILP solver is time-consuming, especially for larger problem scales. Specially with the time-varying traffic demand in the real world, it is infeasible to recompute the configuration for every time interval using an ILP solver. Hence, many researchers have proposed greedy algorithms that aim to get a solution quickly [3], [11] and reconfigure placements in response to the violation caused by time-varying traffic [19], [20]. However, the optimality of greedy algorithms can still be far from the ILP solvers. As a result, neither of these approaches can make quick and accurate placement decisions to deal with the time-varying workloads in practice.

To overcome the drawbacks of ILP and greedy, we propose a hybrid approach that combines the two approaches together. Our approach is motivated by the traffic characteristics observed from several backbone networks in the US and Germany, including Abilene, Nobel-Germany, Germany50, and Brain. The traffic measurements of these networks are made available by [21], [22], which reports the traffic matrix of every pair of ingress and egress routers in the network at periodic time intervals over an extended time period. For instance, Abilene is collected every 5 mins for 6 months, and Nobel-Germany is collected every 5 mins for 1 day. From our study, we found that most connections between ingress and egress routers have small traffic volumes with high traffic variations, but only a few connections with high volumes tend to be stable with little traffic variation. As shown in Figure 2, we plot the correlation between the mean and relative standard deviation (RSD) of the traffic for each pair of ingress and egress routers. Clearly, the distribution of the mean is unbalanced. Most connections have small mean values, while only a few connections have large mean values. Furthermore, the RSD of connections with large mean values are really small which means their traffic variation is stable. In contrast, all the connections with unstable traffic and large RSD values have small traffic volumes. This traffic behavior can somehow be expected and explained because connections or services with higher traffic volume often contributed by more users, and the multiplexing of traffic from more users can lead to less traffic variation.

Based on the above observation, we propose to combine ILP and greedy methods in the following way. We classify SFCRs into stable ones and unstable ones according to their workload variations over time. At the initial time interval, we only use an ILP solver to allocate the stable SFCRs, and then use a greedy algorithm to determine the placement of the remaining SFCRs with the residual node and link capacity. The intuition is that stable SFCRs should be scheduled optimally first so that network operators can always obtain the maximum profit from this part of SFCRs. Since the number of stable SFCRs should also be relatively small compared to the total number of SFCRs, so the problem size of our ILP can also be reduced greatly. On the other hand, unstable SFCRs can only obtain limited benefits from ILP because they are likely to be migrated over time due to traffic variations. Therefore, we use a greedy algorithm to allocate the remaining unstable SFCRs.

After the above initial placement, we then propose another greedy reconfiguration algorithm to refine the placement at each of the following time intervals for two objectives. One is to resolve resource violations caused by the time-varying traffic demand, so the number of SFCR rejections can be minimized. The other one is to consolidate VNFs onto a smaller number of servers, so the resource and energy consumption can be minimized.

In the rest of the section, we first introduce the ILP formulation for finding the optimal placement for stable traffic. Then we explain the greedy placement algorithm for unstable traffic. Finally, we propose the greedy reconfiguration algorithm for refining the placement of every following time interval. Noted, the placement algorithms can also be applied periodically when the traffic demand has more dramatic changes.

B. ILP FORMULATION

The inputs and outputs of our problem have been mentioned in Section II-B, and the objective function is defined in Eq. (1). Hence, an ILP solver can be used to find an optimal placement with subject to the following constraints. All the variables used in the formulation are summarized in Table 1 as well.

\[
\sum_{n=0}^{N-1} x_{s,t,n}^f \leq 1, \quad s \in S, \ i \neq 0, \ |F| + 1, \ t \in T \quad (16)
\]

Eq. (16) means each NF in SFCRs can be either mapped only on one physical node or not mapped at all. But, a SFCR is accepted if and only if all its NFs are mapped.

\[
\sum_{n=0}^{N-1} x_{s,0,n}^f = 0, \quad s \in S, \ n \neq s.s, \ t \in T \quad (17)
\]
\[ \sum_{n=0}^{N-1} x^f_{s,n} = 0, \quad s \in S, n \neq s, d, \quad t \in T \quad (18) \]

\[ x^f_{s,0,n} \leq 1, \quad s \in S, \quad t \in T \quad (19) \]

\[ x^f_{s,[F]+1,n} \leq 1, \quad s \in S, \quad t \in T \quad (20) \]

Eq. (17) to Eq. (20) mean the source and the destination of \( s \) can be only mapped at the specific node.

\[ \sum_{i=0}^{S-1} \sum_{f=0}^{F-1} FC_{s,i} \times x^f_{s,i,n} + \sum_{i=0}^{F-1} \Phi_f \times z^f_{s,n} \leq C_n \times (1 - \Phi), \quad n \in N, \quad t \in T \quad (21) \]

Eq. (21) ensures the total CPU consumption cannot exceed the available CPU usage on each physical node. We introduce a ratio \( \Phi \) to reserve some node capacity for two purposes. One is for handling the traffic variation among time intervals. The other is that sharing VNF can reduce the basic resource consumption and the deployment cost of VNF, so we reserve some capacity for other SFCRs to use in the greedy algorithm.

\[ z^f_{s,n} = \begin{cases} 1, & \text{if } \sum_{i=0}^{S-1} \sum_{f=0}^{F-1} x^f_{s,i,n} \geq 1, \quad f \in F, \quad n \in N, \quad t \in T \quad (22) \\ 0, & \text{otherwise} \end{cases} \]

Eq. (22) means that a type \( f \) VNF is deployed on the physical node \( n \) when there is at least one \( f \) type NF mapped to the node \( n \).

\[ \sum_{i=0}^{S-1} \sum_{m=0}^{N} FB_{s,i} \times y^f_{s,i,n,m} \leq B_{n,m} \times (1 - \Psi), \quad n \in N, \quad m \in N, \quad n \neq m, \quad t \in T \quad (23) \]

Eq. (23) means, on each physical link, the total link consumption cannot exceed the available link usage. \( \Psi \) is the reserve ratio of link capacity.

\[ \sum_{n=0}^{N-1} y^f_{s,i,n,m,n} \leq 1, \quad s \in S, \quad n \in N, \quad m \in N, \quad n \neq m, \quad t \in T \quad (24) \]

\[ \sum_{n=0}^{N-1} y^f_{s,i,n,m,n} \leq 1, \quad s \in S, \quad n \in N, \quad m \in N, \quad n \neq m, \quad t \in T \quad (25) \]

Eq. (24) means if there is one SFCR traffic flow into physical node \( n \), the traffic is only just from one physical node \( m \). Eq. (25) means if there is one SFCR traffic flow out physical node \( n \), the traffic only flows into one physical node \( m \). Both of them describe a virtual link that cannot be mapped on more than one physical link. In short, traffic cannot be split.

\[ \sum_{n=0}^{N-1} y^f_{s,i,n,m,n} - \sum_{n=0}^{N-1} y^f_{s,i,n,m,n} = x^f_{s,i,n} - x^f_{s,i+[F]+1,n}, \quad s \in S, \quad n \in N, \quad m \in S, \quad n \neq m, \quad t \in T \quad (26) \]

Eq. (26) ensures traffic flow is consecutive.

**Algorithm 1: Greedy Placement Algorithm**

**Input:** G_SFCRs: a SFCR list.

**Output:** M: the mapping result of all SFCRs.

1. for \( s \in G \_SFCRs \) do
2. \( \text{Prevm} = M; \)
3. \( \text{mapped} = \text{false}; \)
4. \( f_{\text{prev}} = s, s; \)
5. for \( f \in s, SFC \) do
6. \( \text{Candidate} = \text{FindExistingNodesWithVNF}(f); \)
7. if \( \text{Candidate} \) then
8. \( \text{mapped} = \text{false}; \)
9. \( M = \text{FindShortestPathToMap}(f_{\text{prev}}, f, \text{Candidate}); \)
10. end
11. if \( \text{mapped} \) then
12. \( M = \text{BFSTryToMap}(f_{\text{prev}}, s, d); \)
13. end
14. \( f_{\text{prev}} = f; \)
15. \( \text{mapped} = \text{false}; \)
16. end
17. Reject(s);
18. goto line 1;
19. end
20. end
21. Reject(s);
22. goto line 1;
23. end

**C. Greedy Placement Algorithm**

After ILP processing stable SFCRs, we use a greedy algorithm to map remaining SFCRs based on the following scheduling principles. First, we make the placement decision of SFCRs with higher traffic demand with higher priority because the SFCRs with higher traffic demand can contribute more revenue, and they are more likely to be rejected when residual capacity becomes scarce. Second, we reuse the deployed VNFs if the remaining capacity on the server is sufficient in order to reduce the basic resource consumption and the deployment cost. Finally, we try to place the NFs of a SFC as close to each other as possible to minimize the end-to-end delay and maximize the revenue of a SFC.

Algorithm 1 is our greedy algorithm. Our algorithm contains the following steps. Step1: find the SFCR \( s \in S \) with the highest traffic demand based on the first principle. In line 5 to 14, we iteratively find placement locations for each NF based on the placement location of its predecessor NF (\( f_{\text{prev}} \)) according to the policy below. Step2: If the VNF of \( f \) has been deployed on some nodes (Candidate), place the NF on the VNF with the closest distance with sufficient link capacity to the placement location of \( f_{\text{prev}} \), and route the traffic from \( f_{\text{prev}} \) to \( f \) through the shortest path in line 8. Otherwise, deploy a
D. RECONFIGURATION ALGORITHMS

Due to the time-varying workload, we apply placement reconfiguration at each time interval to resolve resource violations and reduce resource consumption. Our reconfiguration strategy consists of three algorithms: (1) NF migrations algorithm aims to resolve node capacity violation by migrating NFs on the node to the same type VNFs on another node. (2) Traffic re-routing algorithm aims to resolve link capacity violation by re-routing the traffic away from the links without sufficient capacity. (3) VNF Consolidation algorithm aims to shut down the node with less loading by migrating its VNFs to other nodes.

The NF migrations algorithm is shown in Algorithm 2. For each time interval, we first find out all the nodes with insufficient capacity in line 1. Then, in the for-loop we iterate \( n \_violation \) to process each violation node \( n \). We iteratively migrate the NFs on node \( n \) with the following steps until the violation on node \( n \) is resolved. Step 1: Find an unstable SFCR \( s \) routing through the node \( n \) with the least traffic demand and the NF \( f_i \) with the highest resource consumption in \( s \) that maps on the node \( n \) in line 4. Step 2: If an unstable SFCRs exists, find the shortest path connecting the last NF with the destination of \( s \) in line 15. Step 4: compare the mapping result with the one without mapping the \( s \) to ensure our profit increase in line 16. If a SFCR cannot find a valid placement, the SFCR is rejected and the \( M \) is not changed.

The Traffic Re-routing algorithm is shown Algorithm 3. When resource violation occurs on a link, we first find out all the violation links in line 1. Then, in the for-loop we iterate \( l \_violation \) to process each violating link \( l \). We iteratively redirect the traffic of these mapped SFCRs from the link \( l \) with the following steps until the traffic on link \( l \) is less than its link capacity. Step 1: Find a unstable SFCR \( s \) from the link \( l \) with the least traffic demand and find the NF \( f_i \) in \( s \) that uses the link \( l \) to route the traffic to its next NF \( f_{i+1} \) in line 4. Step 2: Re-route the traffic from \( f_i \) to \( f_{i+1} \) through the shortest path between their node locations without using link \( l \) in line 8. If we cannot find any unstable SFCRs, we select a stable SFCR with the smallest traffic and then reject it in line 5 to 7. Step 3: If a set of VNFs that can serve \( f_i \) already exists along the path, migrate \( f_i \) to the node along the path \( p \) with the highest node residual capacity. Otherwise, deploy a new VNF instance on the node with the highest residual capacity along the path \( p \), and migrate \( f_i \) to the VNF.

Traffic re-routing algorithm is shown Algorithm 3. When resource violation occurs on a link, we first find out all the violation links in line 1. Then, in the for-loop we iterate \( l \_violation \) to process each violating link \( l \). We iteratively redirect the traffic of these mapped SFCRs from the link \( l \) with the following steps until the traffic on link \( l \) is less than its link capacity. Step 1: Find a unstable SFCR \( s \) from the link \( l \) with the least traffic demand and find the NF \( f_i \) in \( s \) that uses the link \( l \) to route the traffic to its next NF \( f_{i+1} \) in line 4. Step 2: Re-route the traffic from \( f_i \) to \( f_{i+1} \) through the shortest path between their node locations without using link \( l \) in line 8. If we cannot find any unstable SFCRs, we select a stable SFCR with the smallest traffic and then reject it in line 5 to 7. Step 3: If a set of VNFs that can serve \( f_i \) already exists along the path, migrate \( f_i \) to the node along the path \( p \) with the highest node residual capacity. Otherwise, deploy a new VNF instance on the node with the highest residual capacity along the path \( p \), and migrate \( f_i \) to the VNF.

Traffic re-routing algorithm is shown Algorithm 3. When resource violation occurs on a link, we first find out all the violation links in line 1. Then, in the for-loop we iterate \( l \_violation \) to process each violating link \( l \). We iteratively redirect the traffic of these mapped SFCRs from the link \( l \) with the following steps until the traffic on link \( l \) is less than its link capacity. Step 1: Find a unstable SFCR \( s \) from the link \( l \) with the least traffic demand and find the NF \( f_i \) in \( s \) that uses the link \( l \) to route the traffic to its next NF \( f_{i+1} \) in line 4. Step 2: Re-route the traffic from \( f_i \) to \( f_{i+1} \) through the shortest path between their node locations without using link \( l \) in line 8. If we cannot find any unstable SFCRs, we select a stable SFCR with the smallest traffic and then reject it in line 5 to 7. Step 3: If a set of VNFs that can serve \( f_i \) already exists along the path, migrate \( f_i \) to the node along the path \( p \) with the highest node residual capacity. Otherwise, deploy a new VNF instance on the node with the highest residual capacity along the path \( p \), and migrate \( f_i \) to the VNF.

Finally, VNF Consolidation algorithm is shown in Algorithm 4. The algorithm attempt to shut down nodes for saving resource and energy with the following steps. Step 1: Select the node \( n \) with the lowest utilization as the shutdown candidate. Step 2: For each NF mapped on the node \( n \), we try to move it to a node with the same VNF type. The best candidate node is the one that can result in the highest net profit according to our definition. Step 3: Since migrating NF could result in longer routing paths and lower revenue, we will not shut down the candidate node if the consolidation done in Step 2 cannot improve the overall net profit.

| Algorithm 2: NF Migrations Algorithm |
|--------------------------------------|
| **Input**: G: the topology graph.    |
|                                      |
| **Output**: M: the mapping result of all of SFCRs. |
|                                      |
| 1  \( n \_violation = \text{FindViolatingNodes}(G, M) \); |
| 2  for \( n \in n \_violation \) do |
| 3      while \( n \) violates do |
| 4          \( s_f_i = \text{FindUnstableSFCRWithLowestTraffic}(n) \); |
| 5              if \( s == \text{null} \) then |
| 6                  \( s = \text{FindStableSFCRWithLowestTraffic}(n) \); |
| 7                      Reject(s); |
| 8              else |
| 9                 reroute, \( p = \text{FindShortestPathToRoute}(s, \) |
| 10                \( f_{i-1}, f_{i+1} \); |
| 11                   if reroute == false then |
| 12                         Reject(s); |
| 13                   else |
| 14                      migrated = \text{FindNodeToMigrateNF}(s, \) |
| 15                        \( f_i, p \); |
| 16                   if migrated == false then Reject(s); |
| 17       end |
| 18       end |
| 19       end |

| Algorithm 3: Traffic Re-Routing Algorithm |
|--------------------------------------|
| **Input**: G: the topology graph.    |
|                                      |
| **Output**: M: the mapping result of all of SFCRs. |
|                                      |
| 1  \( l \_violation = \text{FindViolatingLinks}(G, M) \); |
| 2  for \( l \in l \_violation \) do |
| 3      while \( l \) violates do |
| 4        \( s_f_i = \text{FindUnstableSFCRWithLowestTraffic}(l) \); |
| 5            if \( s == \text{null} \) then |
| 6                \( s = \text{FindStableSFCRWithLowestTraffic}(l) \); |
| 7                    Reject(s); |
| 8            else |
| 9               reroute = \text{FindShortestPathToRoute}(s, \) |
| 10                \( f_i, f_{i+1} \); |
| 11                   if reroute == false then Reject(s); |
| 12       end |
| 13       end |
| 14       end |

The NF migrations algorithm is shown in Algorithm 2. For each time interval, we first find out all the nodes with insufficient capacity in line 1. Then, in the for-loop we iterate \( n \_violation \) to process each violation node \( n \). We iteratively migrate the NFs on node \( n \) with the following steps until the violation on node \( n \) is resolved. Step 1: Find an unstable SFCR \( s \) routing through the node \( n \) with the least traffic demand and the NF \( f_i \) with the highest resource consumption in \( s \) that maps on the node \( n \) in line 4. Step 2: If an unstable SFCRs exists, find the shortest path connecting the last NF with the destination of \( s \) in line 15. Step 4: compare the mapping result with the one without mapping the \( s \) to ensure our profit increase in line 16. If a SFCR cannot find a valid placement, the SFCR is rejected and the \( M \) is not changed.

new VNF instance on the node which is the closest distance to \( f_{prev} \) and has sufficient node capacity, and route the traffic from \( f_{prev} \) to \( f \) through the shortest path in line 10. Step 3: find the shortest path connecting the last NF with the destination of \( s \) in line 15. Step 4: compare the mapping result with the one without mapping the \( s \) to ensure our profit increase in line 16. If a SFCR cannot find a valid placement, the SFCR is rejected and the \( M \) is not changed.
Algorithm 4: VNF Consolidation Algorithm

| Line | Description |
|------|-------------|
| 1    | `has_improvement = true;` |
| 2    | `while has_improvement do` |
| 3    | `Prev_M = M;` |
| 4    | `has_improvement = false;` |
| 5    | `for Node n ∈ G: from lowest utility node to high do` |
| 6    | `for VNF v: v on n do` |
| 7    | `if SFC s: s uses v do` |
| 8    | `Candidate = FindExistingNodesWithVNF(v);` |
| 9    | `M = FindBestCandidateToMove(Candidate, s);` |
| 10   | `end` |
| 11   | `end` |
| 12   | `if Profit(Prev_M) < Profit(M) then` |
| 13   | `has_improvement = true;` |
| 14   | `break;` |
| 15   | `else` |
| 16   | `M = Prev_M;` |
| 17   | `end` |
| 18   | `end` |
| 19   | `end` |

IV. EXPERIMENT SETUP

A. NETWORK TOPOLOGY

Our experiments are conducted based on the network and topology information of two backbone networks, Abilene and GÉANT, from SNDlib [21], [22]. Abilene (i.e., also known as Internet2 Network) is a high-performance backbone network connecting several universities and their affiliate institutions in the US. The Abilene network consists of 12 nodes and 15 10 Gbit/s links. GÉANT is a European data network for the research and education community connecting over 50 million users in over 10,000 institutions. Its backbone network consists of 22 nodes and 36 links. Both networks have been widely used for performance evaluation in the research literature. In the rest of the section, we also refer to Abilene as the small-scale network and GÉANT as the large-scale network.

For our evaluation, we consider each node in the network topology as a server with computing capacity to host VNFs. The computing capacity of these servers is considered to be different with resource units randomly generated within a range. The bandwidth of each link in the topology is identical because 10 Gbit/s links are used by both networks. We also generate synthetic networks with a given number of nodes using the k-means clustering algorithm to merge nodes within a neighborhood in the real networks, and aggregate the link bandwidth of merged nodes.

B. SFCR WORKLOAD

We use the method described in the previous VNF placement study [23] to generate the workload of SFCR in our experiment according to the traffic characteristics observed from the real traffic matrix collected in SNDlib [21], [22]. The traffic matrix is generated for every hour over 24 hours. The traffic demand of a SFCR over time intervals is generated using a sinusoidal signal. The demand of each time interval is added with a noise value generated by a Poisson distribution function. To distinguish stable and unstable SFCRs, the variance of noise and sinusoidal signal are much larger for the unstable SFCRs than the stable SFCRs. According to the 80-20 rule, we let only 20% of the SRFCs have stable traffic, but their aggregated traffic demand is counted for roughly 80% of the total network traffic.

Four types of NFs are considered in our experiment, and the number of NFs in each SFCRs is between 2 and 4. Every VNF type has the same basic resource consumption unit for simplicity, but every NFs has a different workload-dependent resource consumption ratio. The delay threshold $\theta$ of each SFCR is given randomly between 1 and the network topology diameter. The ingress and egress nodes of a SFCR are randomly assigned according to the probability of traffic demand ratio between each pair of nodes in the original traffic matrix collected from real systems.

C. COMPARED METHODS

Following four VNF placement algorithms are compared in our evaluation.

1) GREEDY

It simply uses the greedy placement algorithm described in Algorithm 1 to decide the placement decision at the initial time interval. Then apply the NF migrations algorithm, traffic re-routing algorithm, and the VNF consolidation algorithm for the rest of the following time intervals.

2) ILP

It represents an optimal solution that maximizes the net profit including the reconfiguration cost across time intervals. In order to include the reconfiguration cost in ILP, we extend our ILP formulation in Section III-B by giving the placement of the previous time interval as a problem input and adding the cost equation for deployment and redirection in the formulation. The CPLEX ILP solver is used to find the new placement for the current time interval. To restrict the computation time of the ILP solver, the solver is stopped when it reaches a certain percentage of the optimal solution or the result is converged.

3) VITERBI

It represents a state-of-the-art reconfiguration-aware VNF placement solution. The algorithm is proposed by [14], and we briefly describe its algorithm as follows. First, it finds the placement without considering reconfiguration cost at each time interval using the ILP solver. Then it constructs a state transition graph between the placement found from the ILP solvers, where each state of the graph represents the profit from applying a placement decision on the traffic of...
a time interval, and the weight on a state transition edge is the reconfiguration cost from one placement to another placement. Finally, it aims to maximize the net profit across all the time intervals by applying the Viterbi algorithm to find a path from the first time interval to the last time interval in the state transition graph with the maximum aggregated values. As shown in our evaluation results, this approach can achieve better net profit than Greedy, but it still needs to solve the ILP problem at each time interval, and assume the traffic demands of all the time intervals are known in prior.

4) HYBRID
It is our proposed approach, which determines the placement at the initial time interval using ILP solver for stable SFCRs and using the greedy placement algorithm for unstable SFCRs. Then, apply the NF migrations algorithm, traffic re-routing algorithm, and the VNF consolidation algorithm for the rest of the following time intervals.

V. EXPERIMENT RESULTS
Our evaluation consists of three parts. The first is to analyze the benefits from only considering stable traffic in ILP formulation. The second is to compare the results of different algorithms using the small-scale network, Abilene. The third is to exam the scalability and the benefit of consolidation strategy using the large-scale network, GÉANT.

A. ILP SOLUTION ANALYSIS
This set of experiments aims to analyze the results of the ILP solver from our formulation given in Section III-B. The experiments are conducted with 20 SFCRs (4 stable SFCRs, 16 unstable SFCRs) on a 8 nodes network which is generated through k-means clustering based on the GÉANT topology. First of all, we know the strength of ILP is that the ILP solver can always optimize the result according to any given objective function, while greedy algorithms normally are ad-hoc strategies designed for a single objective or intuition. That is important to the network operators because the objective functions can be affected by many factors, such as system environment, user experience, etc. For instance, the value of \( \omega \) in our objective function represents the cost ratio between computing resource and network bandwidth. According to Eq. (12), if \( \omega \) increases, computation cost becomes more important than network cost and vice versa. Hence, the network operator can easily adjust the objective function of ILP to adapt to these changes on finding the placement decision. In Figure 3, we present the placement results from the ILP solver then the settings of \( \omega \) is varied from 0 to 1. The results include the bandwidth consumption (BW), CPU consumption (CPU), number of VNFs (VNF) and, number of active PMs (Active PM). As observed, under larger value of \( \omega \), computation cost becomes more expensive, so the ILP solver will try to reduce the CPU consumption by reducing the number of VNF and active PM. On the other hand, under smaller value of \( \omega \), the placement decision will try to reduce the bandwidth consumption by deploying more VNFs in a near neighborhood, so the routing path and bandwidth consumption can be reduced. Most greedy algorithms, including our proposed algorithm, are oblivious or insensitive to the changes of objective function. Therefore, ILP approach is still necessary to obtain better and more adaptive results.

However, the biggest weakness of ILP approach is its computation complexity. Figure 4 shows the computation time of ILP solver as we increase the number of SFCRs in ILP formulation. The y-axis of the plot is in log scale, so the computation time actually increases exponentially to the problem size (i.e., the number of SFCRs). Therefore, the computation time can be significantly decreased if we can reduce the problem size. Our Hybrid approach did that by only considering the stable SFCRs in ILP. The reason we only consider stable SFCRs for three reasons. (1) The traffic demand of stable SFCRs changes little over time, so their VNF placements don’t need to be reconfigured. Hence, the solution from ILP solver can be reused with good results over multiple time intervals, and computation cost can be amortized over time intervals. (2) Traffic demand of stable SFCRs is counted for the majority of the overall network traffic demand, so maximizing the net profit of stable SFCRs can lead to a close-to-optimal result. (3) Only a small percentage of SFCRs are stable, so considering only the stable SFCRs in ILP can greatly reduce problem size and computation time.

As shown in Figure 5, we plot the profit from ILP solver as we gradually increase the number of SFCRs in ILP formulation according to three different SFCR selection strategies.
Stable-first is our proposed strategy that chooses more stable SFCRs with higher priority. In contrast, Unstable-first chooses the least stable SFCRs first. Lastly, Random chooses the SFCRs to be considered in ILP randomly. The rest of SFCRs not chosen in ILP will be placed with the greedy placement algorithm. As observed from the plot, more SFCRs in ILP can improve the overall profits for all three strategies. But Stable-first strategies only needs 4 SFCRs to reach the best result when all the stable SFCRs are selected. On the other hand, other strategies need to include a lot of SFCRs to get improvements. Interestingly, we also found that including unstable SFCRs in ILP can cause worse results in some cases. This is because the initial placement of unstable SFCRs decided by ILP is not guaranteed to be the best decision for the following time intervals once traffic demand changes. Even worse, stable SFCRs are sacrificed if unstable SFCRs are chosen first which also likely leads to lower overall profits. Therefore, our proposed strategy to only consider stable SFCRs in ILP can generate more profit while lower computation time.

B. SMALL-SCALE NETWORK EVALUATION

Next, we compare the profits and computation time of four different VNF algorithms (Greedy, ILP, Viterbi, Hybrid) introduced in Section IV-C. Because ILP algorithm takes too much time for large-scale networks, we use small-scale networks generated based on Abilene network for this set of experiment. Figure 6 plots the profit under different scale networks. As expected, Greedy has the lowest profits, while ILP has the highest profits. Specifically, the profits of ILP is 30% ∼ 38% more than Greedy. In contrast, our Hybrid algorithm achieves very close results to ILP with less than 1% differences for 4-node and 6-node networks, and less than 6.8% differences for 5-node network. Comparing to Viterbi, Hybrid algorithm also increases the profit by 5% ∼ 18%. Therefore, Hybrid approach clearly can achieve much better results than existing greedy and state-of-the-art ILP approaches.

More importantly, Hybrid approach can achieve near-optimal results with much less computation time. As shown in Figure 6b, as the scale of networks increases, the profit results are not affected too much, but the computation time of all the algorithms, except Greedy, increase greatly. This can be expected, because ILP, Vertibi, and Hybrid are all ILP-based algorithms, so computation time grows exponentially to the problem size, including the number of SFCRs and network nodes. But the computation time of Hybrid is significantly smaller than ILP and Viterbi over any network scale. Furthermore, the growing rate of Hybrid is also much lower than ILP and Viterbi. The y-axis in Figure 6b is in log scale, so the computation time reduction of Hybrid is significant. Specifically, Hybrid is faster than ILP and Viterbi up to a factor of x30 and x415, respectively. Hybrid is much faster because it reduces the problem size by considering a smaller number of SFCRs in its ILP solver. Interestingly, we found that Viterbi is slowest among all, and slower than ILP by a factor of x6 ∼ x14. As explained in Section IV-C, ILP introduces more equations to capture the reconfiguration cost, and the previous placement decision is given as input. We believe the additional equations and input variables actually help to limit the search space of the optimization problem. Furthermore, Viterbi needs additional time to run the Viterbi algorithm after solving the ILP problem at every time interval, so Viterbi becomes the slowest among all. Although Hybrid is still slower than Greedy, especially with a larger network scale, it can still be computed in a few seconds with the given network scales.

Overall, our Hybrid approach can achieve near-optimal results with significantly less computation time. Its computation time is x30 times faster than the ILP, and its profit result is 36% higher than the Greedy. Comparing to the state-of-the-art Viterbi algorithm, Hybrid increases the profit by 18%, while taking hundreds of times shorter computation time.

C. LARGE-SCALE NETWORK EVALUATION

Finally, we evaluate the results on large-scale networks with 12 to 16 nodes generated from the GÉANT network. Over 400 time intervals were simulated for considering 20 SFCRs. We are not able to obtain the results of Viterbi and ILP algorithms due to their high computation complexity, so only the results of Hybrid and Greedy are compared in this set of experiments. For both approaches, we also include their results with and without applying our proposed consolidation algorithm. Again, we compare the profit and
computation time in Figure 7a and Figure 7b, respectively. Similar to the results for small-scale network, Hybrid can improve the profit of Greedy by 20% ∼ 35% while taking longer computation time. But even with a network as large as 16 nodes, Hybrid only took 7.5 minutes to compute the result. Especially, Hybrid only needs to run the ILP solver once in a few hours or days. So the computation overhead should be acceptable in practice. In addition, we observe the consolidation algorithm indeed can further increase the profit up to 7% for both Hybrid and Greedy approaches, and the computation time of the consolidation algorithm is much shorter than the greedy placement algorithm. So our proposed consolidation algorithm is quite efficient.

To analyze the impact of our consolidation algorithm and Hybrid strategy, we summarize several measurement statics from the experiment in Figure 8. We can find the following observations. Hybrid has higher profits than Greedy because its resource consumption (both CPU and BW) is lower, and its revenue from accepted SFCRs (Accept Rate) is also higher. The deployment cost of Hybrid is higher than Greedy mainly because it accepts more SFCRs and creates more VNFs to minimize the latency delay of SFCRs to obtain higher revenue. Finally, consolidation algorithm clearly can reduce the number of active PMs, but also has to pay the price for redirection. So the effectiveness of our consolidation strategy depends on the cost ratio between reconfiguration and power consumption. If the power consumption is more expensive or the redirection cost is cheaper, consolidation is more likely to occur. Because our consolidation algorithm only applies its changes if the overall profit can be improved, the results with consolidation are guaranteed to be better than without it.

VI. RELATED WORK

Many different methods are proposed to solve the VNF placement problem either by ILP formulations [6]–[8] or greedy algorithms [10]–[13]. To the best of our knowledge, none of works proposes the approach of combining ILP and greedy algorithms. In addition, our work consider not only the VNF placement or the SFC deployment but also the reconfiguration. In comparison, [18] aims to minimize the number of used physical nodes and consider two factors, which are the time-varying workload and basic resource consumption when running VNFs on physical nodes. They propose ILP first and then propose the two-stage greedy algorithm. However, unlike us, they do not consider the end-to-end delay of SFCRs. Reference [24] formulated a problem on SFC deployment to reduce energy consumption and maximize the provider’s profit. They offer an effective profit-driven service chain provisioning scheme for large-scale infrastructures spanning different geographically-distributed sites. However, they do not consider the time-varying workload.

Some of them only focus on the reconfiguration to improve service performance because of dynamic changes in resource requirements [19], [20], [23], [25]. There are three methods to solve violations, such as VNF migration, VNF replication, and the flow redirection of SFC. Reference [23] propose a real-time VNF migration algorithm based on the deep belief network to predict future resource requirements. Reference [24] aims to minimize network latency in rapidly changing network environments. They design a model to migrate VNFs for low latency. Reference [19] propose an idea of replication VNF to reduce the number of migrations and discuss the trade-off between replication and migration by using Linear Programming (LP) model. Reference [20] propose a priority-aware flow migrating scheme and minimize the impact of flow-migration using a greedy algorithm.
VII. CONCLUSION
This work proposed a hybrid VNF placement approach to maximize the net profit of a network service provider. To overcome the time complexity problem of ILP solutions under the time-varying workload and large scale, our hybrid approach only applies ILP on stable traffic, and uses the greedy algorithm for placing the unstable traffic on residual network capacity. To resolve resource violations caused by workload, a greedy reconfiguration strategy is designed to re-route traffic. To reduce the resource consumption cost, a greedy consolidation strategy is designed. Our evaluations based on real backbone network traffic and topology show that our hybrid approach can achieve up to 36% profit improvement compared to a pure greedy approach, while achieving x30 times computation time speedup over a pure ILP approach. Comparing to the state-of-the-art Viterbi algorithm, Hybrid increases the profit by 18%, while taking hundreds of times shorter computation time. Therefore, we prove that a hybrid approach combining ILP and greedy is capable to make quick and accurate placement decisions to deal with the time-varying workloads in real system scaled networks.

REFERENCES
[1] M. Chiosi, D. Clarke, P. Willis, A. Reid, J. Feger, M. Bugenhagen, W. Khan, M. Fargano, C. Cui, H. Deng, J. Benitez, U. Michel, H. Daniker, K. Ogaki, and T. Matsuzaki, “Network functions virtualisation: An introduction, benefits, enablers, challenges and call for action,” in Proc. SDN OpenFlow World Cone., 2012, pp. 22–24.
[2] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, “Network function virtualization: Challenges and opportunities for innovations,” IEEE Commun. Mag., vol. 53, no. 2, pp. 90–97, Feb. 2015.
[3] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, “Near optimal placement of virtual network functions,” in Proc. IEEE Conf. Comput. Commun. (INFOCOM), Apr. 2015, pp. 1346–1354.
[4] S. Mehraghdam, M. Keller, and H. Karl, “Specifying and placing chains of virtual network functions,” in Proc. IEEE 3rd Int. Conf. Cloud Netw. (CloudNet), Oct. 2014, pp. 7–13.
[5] P. Quinn and T. Nadeau. (2015). Problem Statement for Service Function Chaining (RFC7498). [Online]. Available: https://rfc-editor.org/rfc/rfc7498.txt
[6] A. Gupta, M. F. Habib, P. Chowdhury, M. Tornatore, and B. Mukherjee, “Joint virtual network function placement and routing of traffic in operator networks,” UC Davis, Davis, CA, USA, Tech. Rep., 2015.
[7] A. Gupta, M. F. Habib, U. Mandal, P. Chowdhury, M. Tornatore, and B. Mukherjee, “On service-chaining strategies using virtual network functions in operator networks,” Comput. Netw., vol. 133, pp. 1–16, Mar. 2018.
[8] H. Moens and F. D. Turck, “VNF-P: A model for efficient placement of virtualized network functions,” in Proc. 10th Int. Conf. Netw. Service Manage. (CNSM) Workshop, Nov. 2014, pp. 418–423.
[9] V12. 1: User’s Manual for CPLEX, IBM ILOG CPLEX, Int. Bus. Mach. Corp., Armonk, NY, USA, 2009, p. 157.
[10] Q. Sun, P. Lu, W. Lu, and Z. Zhu, “Forecast-assisted NFV service chain deployment based on affiliation-aware vNF placement,” in Proc. IEEE Global Commun. Conf. (GLOBECOM), Dec. 2016, pp. 1–6.
[11] M. F. Barl, S. R. Chowdhury, R. Ahmed, and R. Boutaba, “On orchestrating virtual network functions,” in Proc. 11th Int. Conf. Netw. Service Manage. (CNSM), Nov. 2015, pp. 50–56.
[12] O. Houidi, O. Soualah, W. Louati, M. Mechtri, D. Zeghlache, and F. Kamoun, “An efficient algorithm for virtual network function scaling,” in Proc. IEEE Global Commun. Conf. (GLOBECOM), Dec. 2017, pp. 1–7.
[13] A. Alleg, R. Kouah, S. Moussaoui, and T. Ahmed, “Virtual network functions placement and chaining for real-time applications,” in Proc. IEEE 22nd Int. Workshop Comput. Aided Model. Design Commun. Links Netw. (CAMAD), Jun. 2017, pp. 1–6.
[14] V. Eramo, M. Ammar, and F. G. Lavacca, “Migration energy aware reconfigurations of virtual network function instances in NFV architectures,” IEEE Access, vol. 5, pp. 4927–4938, 2017.
[15] D. Strom and J. F. van der Zwaet, “Truth and lies about latency in the cloud,” Interxiontm, Amsterdam, The Netherlands, White Paper, 2012. [Online]. Available: https://www.interxion.com/globalassets/documents/whitepapers-and-pdfs/cloud/WP_TRUTHANDLIES_en_0715.pdf
[16] Y. C. Lee and A. Y. Zomaya, “Energy efficient utilization of resources in cloud computing systems,” J. Supercomput., vol. 60, no. 2, pp. 268–280, 2012.
[17] A. Varasteh, M. De Andrade, C. M. Machuca, L. Wosinska, and W. Kellerer, “Power-aware virtual network function placement and routing using an abstraction technique,” in Proc. IEEE Global Commun. Conf. (GLOBECOM), Dec. 2018, pp. 1–7.
[18] D. Li, P. Hong, K. Xue, and J. Pei, “Virtual network function placement considering resource optimization and SFC requests in cloud datacenter,” IEEE Trans. Parallel Distrib. Syst., vol. 29, no. 7, pp. 1646–1677, Jul. 2018.
[19] F. Carpio, A. Jukan, and R. Pries, “Balancing the migration of virtual network functions with replications in data centers,” in Proc. IEEE/IFIP Netw. Oper. Manage. Symp., Apr. 2018, pp. 1–8.
[20] Z. Meng, J. Bi, C. Sun, A. Xu, and H. Hu, “PRAM: Priority-aware flow migration scheme in NFV networks,” in Proc. Symp. SDN Res., Apr. 2017, pp. 183–184.
[21] S. Orłowski, M. Pióró, A. Tomaszewski, and R. Wessyła, “SNDlib I.0—Survivable network design library,” in Proc. 3rd Int. Netw. Optim. Conf., Apr. 2007, pp. 1–6.
[22] S. Orłowski, R. Wessyła, M. Pióró, and A. Tomaszewski, “SNDlib I.0—Survivable network design library,” Networks, vol. 55, no. 3, pp. 276–286, 2010.
[23] L. Tang, X. He, P. Zhao, G. Zhao, Y. Zhou, and Q. Chen, “Virtual network function migration based on dynamic resource requirements prediction,” IEEE Access, vol. 7, pp. 112348–112362, 2019.
[24] W. Racheg, N. Gharaa, and M. F. Znani, “Profit-driven resource provisioning in NFV-based environments,” in Proc. IEEE Int. Conf. Commun. (ICC), May 2017, pp. 1–7.
[25] D. Cho, J. Taheri, A. Y. Zomaya, and P. Bouvry, “Real-time virtual network function (VNF) migration toward low network latency in cloud environments,” in Proc. IEEE 10th Int. Conf. Cloud Comput., Jun. 2017, pp. 798–801.

CHI-CHEN YANG is currently pursuing the M.S. degree with the Department of Computer Science, National Tsing Hua University, Taiwan. She is working on the topic of virtual network function placement. She has participated in several student cluster competitions. Her research interests include cloud computing, distributed systems, and high-performance computing.

SATYAJIT PADHY received the master’s degree in computer science from Amity University, in 2014. He is currently pursuing the Ph.D. degree with the Department of Computer Science, National Tsing Hua University, Taiwan. He has published several conferences and journals, including IEEE Access, Journal of Parallel and Distributed Computing, and Journal of Supercomputing. His research interests include cloud computing, scheduling algorithms, and resource management optimization.

JERRY CHOU received the Ph.D. degree from the Department of Computer Science and Engineering, University of San Diego (UCSD), in 2009. In 2011, he joined the Department of Computer Science, National Tsing Hua University, as an Assistant Professor, and was promoted to an Associate Professor, in 2016. His research interests include distributed systems, including high performance computing, cloud/edge computing, big data, storage systems, and resource or data management.

C.-C. Yang et al.: Hybrid VNF Placement Strategy for Maximizing Profit of Network Service Deployment