Optimizing Stateful Service Function Chaining on PDP Switches with Table Merging

Zhen Wei, Tingyu Li, Zichen Xu, and Zuqing Zhu[†]
School of Information Science and Technology, University of Science and Technology of China, Hefei, China
[†]Email:{zqzhu}@ieee.org

***Abstract*—The rapid development of network function virtualization (NFV) has notably increased the implementation of virtual network functions (vNFs), especially the stateful ones, on high-performance programmable data plane (PDP) switches (*e.g.*, those based on P4 and Tofino ASICs). This facilitates offloading stateful service function chains (SFCs) to PDP switches. However, the capability of PDP switches to carry stateful SFCs is still constrained by their limited hardware resources. In this work, we propose to address this issue with adaptive table merging, formulate an integer linear programming (ILP) model to optimize the deployment of stateful SFCs on P4-based PDP switches with table merging, and design a time-efficient heuristic to solve the problem quickly. Simulation results validate that our proposals successfully enhance the deployment efficiency of stateful SFCs, outperforming existing benchmarks significantly.**

***Index Terms*—Network function virtualization, Service Function Chaining, Programmable data plane, P4, Table merging.**

I. Introduction

Propelled by the synergistic fusion of network function virtualization (NFV) and software defined networking (SDN) [1, 2], service function chaining (SFC) [3] has advanced the state-of-the-art of service provisioning in the Internet strongly. NFV decomposes network services into the virtual network functions (vNFs) that can be instantiated on general-purpose software and hardware platforms to enhance cost-efficiency and flexibility, while SDN separates the control and data planes of a network to bring in unprecedented programmability. Their confluence facilitates the agile service provisioning with SFC (*i.e.*, steering application traffic through a series of vNFs), successfully mitigating the high cost and long lead time of service provisioning with special-purpose middle-boxes [3–5].

Meanwhile, the advent of programmable data plane (PDP) [6] has catalyzed a paradigm shift of SFC deployment, positioning vNFs for being deployed across software and hardware platforms to explore their advantages simultaneously [7]. In addition to the protocol-agnostic programmability enabled by P4 [6], PDP switches can realize customized packet processing at a line-rate of 100 Gbps and beyond. Such attributes have seen P4-based PDP switches being increasingly leveraged to support the offloading of various vNFs, showcasing superior performance on packet handling throughput and latency [8, 9]. Despite the advantages, offloading SFCs to PDP switches still faces a few challenges. For instance, when realizing an SFC on one PDP switch, the vNFs are instantiated as packet processing pipelines, each of which is formed in stages with static random access memory (SRAM), ternary content addressable memory (TCAM), arithmetic and logic units (ALUs), and registers. However, as the SRAM and TCAM in each stage are limited, the number of flow entries that can be accommodated in a vNF is much smaller than that of running the vNF on a server [10]. This necessitates optimizing memory allocation of SFCs deployed on PDP switches, especially for stateful SFCs [9].

Although people have considered how to optimize memory allocation in PDP switches to deploy stateless SFCs [11, 12], they treated each vNF as an atomic unit, leading to resource-inefficient solutions because different vNFs can contain similar logic units. To address this issue, researchers have delved into consolidating stateless vNFs to mitigate redundant resource usage [13, 14]. However, these studies overlooked the stateful SFCs that retain state information on PDP switches through P4 primitives and registers and thus can effectively reduce the interactions with the control plane during operation [15, 16]. Note that, due to the intrinsic differences between stateless and stateful SFCs, the vNF consolidation approach developed for stateless SFCs can hardly be applied to stateful ones, so does the optimization scheme for SFC deployment on account of vNF consolidation. More recently, pSFC [17] was proposed to compose multiple stateless/stateful SFCs to a compound control flow graph for eliminating redundant logic units among the SFCs, but it only tried to merge the same tables in vNFs. Hence, we can still improve it, considering the possibility of merging tables of different types in stateful vNFs.

In this work, we first explain why stateful vNFs in various types can be consolidated based on logic unit combinations [18]. Then, with the table merging method that is applicable to different vNFs, we study how to optimize the deployment of a set of stateful SFCs on PDP switches, to minimize the total resource usage and average path length of the SFCs. An integer linear programming (ILP) model is formulated, and we also propose a time-efficient heuristic for the optimization. Extensive simulations verify the effectiveness of our proposals.

The rest of the paper is organized as follows. We describe

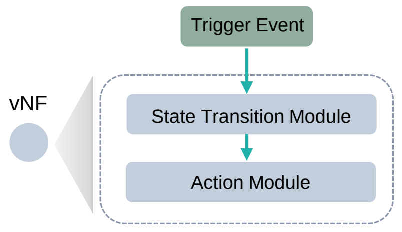


Fig. 1. Modeling a stateful vNF as a state machine.

how to decompose stateful vNFs into fine-grained logic units to enable subsequent table merging in Section II. Section III formulates an ILP model to optimize the SFC deployment on account of the table merging method. The time-efficient heuristic for deploying stateful SFCs across PDP switches is proposed in Section IV. In Section V, we discuss the simulation results. Finally, Section VI summarizes the paper.

II. Network Model and Problem Description

To deploy SFCs including stateful vNFs on PDP switches with high resource-efficiency, we can first model each stateful vNF as one state machine built with a state transition module (STM) and an action module (AM), as illustrated in Fig. 1. Specifically, the STM maintains the state information of the vNF (*e.g.*, state per packet, per flow, or about link(s)) based on the events that are internal or external to a PDP switch, while the AM performs the match-action operation defined by the current state. Then, the STM and AM can be implemented in the pipeline of a PDP switch with match action tables (MATs) and register actions. Hence, table merging can be applied to different vNFs after decomposing each of them into two logic units (STM and AM), to reduce redundant resource usage.

Specifically, in addition to the *Exact Merge* (*i.e.*, merging two exactly same logic units in two vNFs) considered in [17], we can introduce two new principles for table merging: 1) the *Action Merge* that merges two logic units only with the same actions, and 2) the *Match Merge* that merges two logic units only with the same match items. Fig. 2 provides examples on *Action Merge* and *Match Merge*, where the states of stateful vNFs are denoted as bitmaps with significant bits in the states marked in red. For example, the states in *MAT* a are $\{001, 000\}$, where the lowest bit is significant. It can be seen that *MATs* a and b use the same actions but their match items are different. Hence, we can apply *Action Merge* to merge their state bitmaps to $011 = 001 + 010$, as shown in the bottom part of Fig. 2, which will reduce the SRAM demand of MATs a and b from 8 bytes to 4 bytes. Similarly, by applying *Match Merge* to *MATs* c and d, which use identical match items, the SRAM usage of the two MATs is cut from 8 bytes to 5 bytes. Note that, reducing the SRAM and TCAM usages of a vNF will make more of its entries be accommodated in each stage of a PDP switch's pipeline (it may only have 12 stages).

Then, to optimize the SFC deployment on account of the aforementioned table merging method, we model the network topology of the substrate network (SNT) as $G(V, E)$, where V and E are the sets of PDP switches and links between them, respectively. For each PDP switch $v \in V$, we define its SRAM and TCAM capacities as C_{SRAM} and C_{TCAM}. The bandwidth capacity of each link $e \in E$ is C_{BW}. The set of pending SFCs is $\mathbf{S} = \{S_1, \cdots, S_N\}$, where the i-th SFC ($i \in [1, N]$) is denoted as $S_i(s_i, d_i, R_i, b_i, f_i)$. Here, s_i and d_i are the SFC's source and destination nodes, respectively, $R_i = \{r_1, \cdots, r_n\}$ is its vNF sequence, b_i is its total bandwidth demand, and f_i denotes the number of flows that are anticipated to use the SFC. In R_i, each vNF is denoted as r_j ($j \in [1, n]$) and can be decomposed into m_j logic units, *i.e.*, $r_j = \{l_{j,1}, \cdots, l_{j,m_j}\}$.

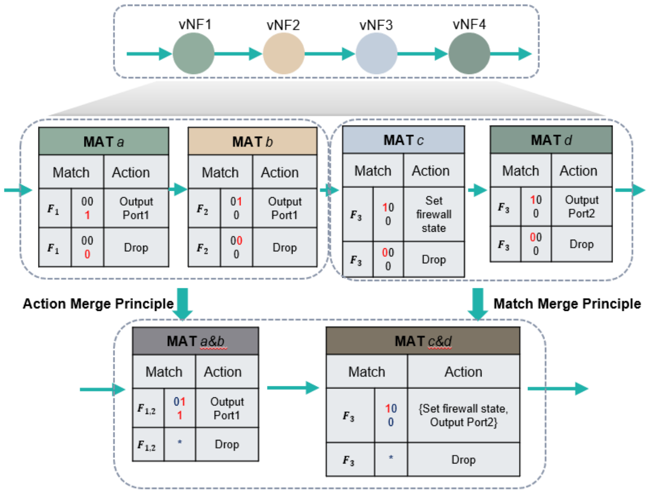


Fig. 2. Examples on table merging principles.

Here, each logic unit $l_{j,k}$ ($k \in [1, m_j]$) can be an MAT, a register action, an independent action, or a conditional branch.

III. ILP Formulation

In addition to those defined in the previous section, our ILP model also uses the following parameters.

- $\mathcal{M}$: the set of match item types considered for MATs.
- $\mathcal{A}$: the set of action types considered for MATs.
- $c_m^{\text{SRAM}}/c_m^{\text{TCAM}}$, $c_a^{\text{SRAM}}/c_a^{\text{TCAM}}$: the SRAM/TCAM usages of match items in type m and actions in type a, respectively.
- $g_{m,a}^{i,j,k}$: the boolean indicator that equals 1 if the k-th logic unit in the j-th vNF of SFC S_i is an MAT (m, a) (*i.e.*, the types of match items and actions of the MAT is in types $m \in \mathcal{M}$ and $a \in \mathcal{A}$, respectively), and 0 otherwise.
- sp^{v_1,v_2}: the smallest hop-count of paths between nodes v_1 and v_2 ($v_1, v_2 \in V$).
- $D_{k_1,k_2}^{i,j}$: the boolean indicator that equals 1 if in the j-th vNF of SFC S_i, the k_2-th logic unit's execution depends on the k_1-th logic unit, and 0 otherwise.
- M: a large positive integer.

Decision Variables:

- $x_{i,j,k}^{v,s}$: the boolean variable that equals 1 if the k-th logic unit in the j-th vNF of SFC S_i is deployed on the s-th stage of PDP switch $v \in V$, and 0 otherwise.
- $x_{i,j,k}^{v,\text{in}}$: the integer variable for the start-stage on v where the k-th logic unit in the j-th vNF of SFC S_i is deployed.
- $x_{i,j,k}^{v,\text{out}}$: the integer variable for the end-stage on v where the k-th logic unit in the j-th vNF of SFC S_i is deployed.
- $x_{i,j,k}^{v,\text{range}}$: the boolean variable that equals 1 if the k-th logic unit in the j-th vNF of SFC S_i is on v, and 0 otherwise.
- z_i^e: the boolean variable that equals 1 if SFC S_i passes through link $e \in E$, and 0 otherwise.
- $y_{m,a}^{v,s}$: the boolean variable that equals 1 if an MAT (m, a) is deployed on the s-th stage of switch v, and 0 otherwise.
- $y^{v,s}$: the boolean variable that equals 1 if s-th stage of switch v is used, and 0 otherwise.
- $\xi_{m,a}^{v,s}$: the integer variable that indicates how many SFCs use an MAT (m, a) deployed on the s-th stage of v (*i.e.*, these SFCs use *Exact Merge* for their MATs in (m, a)).

- $\zeta^{v,s}_{m,a}$/$\eta^{v,s}_{m,a}$/$\theta^{v,s}_{m,a}$/$\delta^{v,s}_{m,a}$: the boolean variable that equals 1 if an MAT (m,a) is deployed on the s-th stage of v and *Exact Merge*/*Match Merge*/*Action Merge*/no merge has been applied to the MAT, and 0 otherwise.
- $\beta^{v,s}_{m,a_1,a_2}$: the boolean variable that equals 1 if MATs (m,a_1) and (m,a_2) are deployed on the s-th stage of PDP switch v, and 0 otherwise.
- $\gamma^{v,s}_{m_1,m_2,a}$: the boolean variable that equals 1 if MATs (m_1,a) and (m_2,a) are deployed on the s-th stage of PDP switch v, and 0 otherwise.
- $\chi^{v,s,1}_{i,m,a}$-$\chi^{v,s,4}_{i,m,a}$: the integer variable that indicates the number of flows, which belong to SFC S_i and will use MAT (m,a) with *Exact Merge*/*Match Merge*/*Action Merge*/no merge deployed on the s-th stage of PDP switch v.
- $\chi^{v,s}_{i,m}$: the integer variable that indicates the number of flows, which belong to SFC S_i and will use match item m with *Match Merge* deployed on the s-th stage of v.
- $\chi^{v,s}_{i,a}$: the integer variable that indicates the number of flows, which belong to SFC S_i and will use action a with *Action Merge* deployed on the s-th stage of v.
- $x^{v}_{i,j}$: the boolean variable that equals 1 if the j-th vNF of SFC S_i is deployed on PDP switch v, and 0 otherwise.

The optimization of SFC deployment should consider both the stage usages on PDP switches and hop-counts of SFCs.

$$\begin{cases} \mathcal{S} = \sum_{v,s} y^{v,s}, \\ \mathcal{L} = \sum_{v_1,v_2,i,j_1,j_2} x^{v_1}_{i,j_1} \cdot x^{v_2}_{i,j_2} \cdot sp^{v_1,v_2}, \end{cases} \tag{1}$$

where $\mathcal{S}$ is the total number of used stages, and $\mathcal{L}$ is the total hop-count of the SFCs' paths[1]. $\mathcal{L}$ impacts the overall latency of SFCs, while $\mathcal{S}$ is proportional to the total resource usage in PDP switches. The objective is defined as

$$\text{Minimize} \qquad \alpha \cdot \mathcal{S} + (1-\alpha) \cdot \mathcal{L}, \tag{2}$$

where α is the weight parameter to balance the two terms.

Constraints:

1) Constraints on deploying logic units:

$$\begin{cases} \sum_{s} x^{v,s}_{i,j,k} = \left(x^{v,\text{out}}_{i,j,k} - x^{v,\text{in}}_{i,j,k} + 1 \right) \cdot x^{v,\text{range}}_{i,j,k}, \\ x^{v,\text{out}}_{i,j,k} - x^{v,\text{in}}_{i,j,k} \geq 0, \\ x^{v,\text{in}}_{i,j,k} \geq M \cdot x^{v,\text{range}}_{i,j,k}, \\ x^{v,\text{out}}_{i,j,k} \geq M \cdot x^{v,\text{range}}_{i,j,k}, \end{cases} \forall v,i,j,k. \tag{3}$$

Eq. (3) ensures that the variables about deployment of logic units ($\{x^{v,s}_{i,j,k}\}$, $\{x^{v,\text{in}}_{i,j,k}\}$, $\{x^{v,\text{out}}_{i,j,k}\}$, $\{x^{v,\text{range}}_{i,j,k}\}$) are correctly set.

2) Constraints on table merging:

$$\begin{cases} \xi^{v,s}_{m,a} = \sum_{i,j,k} x^{v,s}_{i,j,k} \cdot g^{i,j,k}_{m,a}, \\ \zeta^{v,s}_{m,a} \leq \frac{1}{2} \cdot \xi^{v,s}_{m,a}, \\ \zeta^{v,s}_{m,a} \geq \frac{1}{M} \cdot (\xi^{v,s}_{m,a} - 1), \end{cases} \forall v,s,m,a. \tag{4}$$

[1]As Eq. (1) is nonlinear because it contains multiplying of variables, we linearize it when solving the ILP. The linearization is omitted to save space.

Eq. (4) ensures that the MATs, which use *Exact Merge*, are correctly determined.

$$\begin{cases} \beta^{v,s}_{m,a_1,a_2} \leq \sum_{i,j,k} x^{v,s}_{i,j,k} \cdot g^{i,j,k}_{m,a_1}, \\ \beta^{v,s}_{m,a_1,a_2} \leq \sum_{i,j,k} x^{v,s}_{i,j,k} \cdot g^{i,j,k}_{m,a_2}, \\ \beta^{v,s}_{m,a_1,a_2} \geq y^{v,s}_{m,a_1} + y^{v,s}_{m,a_2} - 1, \\ \eta^{v,s}_{m,a_1} \geq \beta^{v,s}_{m,a_1,a_2}, \end{cases} \forall v,s,m,a_1,a_2. \tag{5}$$

$$\eta^{v,s}_{m,a_1} \leq \sum_{a_2} \beta^{v,s}_{m,a_1,a_2}, \quad \forall v,s,m,a_1. \tag{6}$$

Eqs. (5) and (6) ensure that the MATs, which use *Match Merge*, are correctly determined.

$$\begin{cases} \gamma^{v,s}_{m_1,m_2,a} \leq \sum_{i,j,k} x^{v,s}_{i,j,k} \cdot g^{i,j,k}_{m_1,a}, \\ \gamma^{v,s}_{m_1,m_2,a} \leq \sum_{i,j,l} x^{v,s}_{i,j,k} \cdot g^{i,j,k}_{m_2,a}, \\ \gamma^{v,s}_{m_1,m_2,a} \geq y^{v,s}_{m_1,a} + y^{v,s}_{m_2,a} - 1, \\ \theta^{v,s}_{m_1,a} \geq \gamma^{v,s}_{m_1,m_2,a}, \end{cases} \forall v,s,m_1,m_2,a. \tag{7}$$

$$\theta^{v,s}_{m_1,a} \leq \sum_{m_2} \gamma^{v,s}_{m_1,m_2,a}, \quad \forall v,s,m,a_1. \tag{8}$$

Eqs. (7) and (8) ensure that the MATs, which use *Action Merge*, are correctly determined.

$$y^{v,s}_{m,a} \geq x^{v,s}_{i,j,k} \cdot g^{i,j,k}_{m,a}, \quad \forall v,s,m,a,i,j,k, \tag{9}$$

$$\begin{cases} \sum_{i,j,k} x^{v,s}_{i,j,k} \cdot g^{i,j,k}_{m,a} \leq y^{v,s}_{m,a} \cdot \sum_{i,j,k} g^{i,j,k}_{m,a}, \\ \sum_{i,j,k} x^{v,s}_{i,j,k} \cdot g^{i,j,k}_{m,a} \geq (y^{v,s}_{m,a} - 1) \cdot \left(1 + \sum_{i,j,k} g^{i,j,k}_{m,a} \right), \end{cases} \forall v,s,m,a. \tag{10}$$

Eqs. (9) and (10) ensure that the relation between variables for logic unit deployment ($\{x^{v,s}_{i,j,k}\}$) and those for MAT deployment ($\{y^{v,s}_{m,s}\}$) are correctly set.

$$y^{v,s}_{m,a} = \eta^{v,s}_{m,a} + \zeta^{v,s}_{m,a} + \theta^{v,s}_{m,a} + \delta^{v,s}_{m,a}, \quad \forall v,s,m,a. \tag{11}$$

Eq. (11) ensures that each MAT uses one and only one table merging principle (including no table merging).

3) Constraints on SFC deployment:

$$\chi^{v,s,l}_{i,m,a} \leq f_i \cdot \zeta^{v,s}_{m,a}, \quad \forall v,s,i,m,a,l \in [1,4]. \tag{12}$$

$$\sum_{j,k} x^{v,s}_{i,j,k} \cdot g^{i,j,k}_{m,a} \leq \sum_{l=1}^{4} \chi^{v,s,l}_{i,m,a}, \quad \forall v,s,m,a, \tag{13}$$

$$\begin{cases} \chi^{v,s,2}_{i,m,a} = \chi^{v,s}_{i,m}, \\ \chi^{v,s,3}_{i,m,a} = \chi^{v,s}_{i,a}, \end{cases} \forall v,s,i,m,a. \tag{14}$$

Eqs. (12)-(14) ensure that for an MAT of SFC S_i, no matter which table merging principle is used on it, the number of flows that belong to S_i and use it does not exceed f_i.

$$\begin{aligned} &\sum_{i,m} \left(\sum_{a} \eta^{v,s}_{m,a} + c^{\text{SRAM}}_{m} \right) \cdot \sum_{i} \chi^{v,s}_{i,m} + \sum_{a} \sum_{i} \chi^{v,s}_{i,a} \cdot c^{\text{SRAM}}_{a} \\ &+ \sum_{i,v,s,m,a} \chi^{v,s,4}_{i,m,a} \cdot (c^{\text{SRAM}}_{m} + c^{\text{SRAM}}_{a}) \leq C_{\text{SRAM}}, \end{aligned} \quad \forall v,s. \tag{15}$$

$$\sum_{i,m,a} \chi_{i,m,a}^{v,s,4} \cdot \left(c_m^{\text{TCAM}} + c_a^{\text{TCAM}}\right) \le C_{\text{TCAM}}, \quad \forall v, s. \tag{16}$$

Eqs. (15) and (16) ensure that the stages on each node v do not consume more SRAM/TCAM than the corresponding capacity.

$$\sum_i z_i^e \cdot b_i \le C_{\text{BW}}, \quad \forall e. \tag{17}$$

Eq. (17) ensures that the total bandwidth usage of SFCs on each link does not exceed its bandwidth capacity.

$$x_{i,j,k_1}^{v,\text{out}} \le x_{i,j,k_2}^{v,\text{in}} - D_{k_1,k_2}^{i,j} \cdot \left(x_{i,j,k_1}^{v,\text{range}} + x_{i,j,k_2}^{v,\text{range}} - 1\right), \ \forall v, i, j, k_1, k_2. \tag{18}$$

Eq. (18) ensures that the relations between logic units of each SFC are correctly set.

$$\begin{cases} y^{v,s} \geqslant y_{m,a}^{y,s}, \\ y^{v,s} \leqslant \sum\limits_{m,a} y_{m,a}^{y,s}, \end{cases} \quad \forall v, s, m, a. \tag{19}$$

Eq. (19) ensures that each vNF is placed correctly on a stage.

$$x_{i,j}^v \leqslant \sum_k x_{i,j,k}^{v,\text{range}} \quad \forall v, i, j. \tag{20}$$

$$x_{i,j,k}^{v,s} \leqslant x_{i,j}^v, \quad \forall v, s, i, j, k. \tag{21}$$

Eqs. (20) and (21) ensure that the values of variables related to vNF deployment are set correctly.

In the ILP above, Eqs. (3) and (15) nonlinear. We omit their linearization procedures to save space.

IV. Heuristic Algorithm

To accelerate problem-solving, we, in this section, propose a polynomial-time heuristic to solve the optimization above. It has three sub-procedures, *i.e.*, coarse-grained SFC merging, fine-grained table merging, and deployment of SFC superset.

Algorithm 1 shows the proposed sub-procedure for coarse-grained SFC merging, which tries to combine all the SFCs in $\mathbf{S}$ to formed an SFC superset S' that records all the vNFs in the SFCs in the right order. This will promote the reusing of vNFs among SFCs, benefiting the subsequent step of fine-grained table merging. *Line* 1 initializes L_a, which will store vNFs and their feasible locations in the SFC superset S'. Then, we initialize S' as the longest SFC in $\mathbf{S}$ (*Line* 2). Next, the for-loop of *Lines* 3-27 checks all the remaining SFCs in $\mathbf{S}$ to update L_a accordingly. Specifically, *Lines* 4-12 check each vNF r_j in an SFC S_i to record its index in S' in P_i, and mark the index of r_j as -1 if it cannot be found in S'. Then, *Line* 13 finds the longest sequence of increasing indices $\widehat{P}$ in P_i, and the vNFs in S_i with these indices can be considered as included in S' already. Next, *Lines* 14-26 check each vNF whose index is not included in $\widehat{P}$, and insert it and its feasible location in S', which will not make S_i out of order, in L_a. Finally, we check each vNF in L_a to insert it in the SFC superset S' according to the feasible location recorded in L_a.

Algorithm 2 explains the fine-grained table merging based on the output of *Algorithm* 1. *Line* 1 is for the initialization, where R and $\mathcal{M}$ are for the processed vNFs in S' and applicable table merging schemes, respectively, and S'' is for

Algorithm 1: Coarse-grained SFC Merging

```
   Input: Set of pending SFCs $\mathbf{S} = \{S_1, \cdots, S_N\}$.
1  $L_a = \emptyset$;
2  find the longest SFC in $\mathbf{S}$ whose stateful vNFs are shared the
   most by other SFCs, and mark it as $S'$;
3  for each SFC $S_i \in \mathbf{S} \setminus S'$ do
4      $P_i = \emptyset$;
5      for each vNF $r_j \in S_i$ do
6          if $r_j \in S'$ then
7              record the index of $r_j$ in $S'$ as $p_j$;
8          else
9              $p_j = -1$;
10         end
11         insert $p_j$ in $P_i$;
12     end
13     apply dynamic programming to find the longest sequence
       of increasing indices $\widehat{P}$ in $P_i$;
14     record the set of vNFs whose indices are not in $\widehat{P}$ in $L_i$;
15     if $L_i \neq \emptyset$ then
16         for each vNF $r_j \in L_i$ do
17             if $r_j \in L_a$ then
18                 check $L_a$ to see whether feasible location(s)
                   of $r_j$ in $S'$ satisfies the requirement of $S_i$;
19                 if the requirement cannot be satisfied then
20                     insert $r_j$ and its feasible location in $S'$,
                       which satisfies requirement of $S_i$, in $L_a$;
21                 end
22             else
23                 insert $r_j$ and its feasible location in $S'$, which
                   satisfies requirement of $S_i$, in $L_a$;
24             end
25         end
26     end
27 end
28 for each vNF $r_j \in L_a$ do
29     insert $r_j$ in $S'$ according to its feasible location in $L_a$;
30 end
31 return $S'$;
```

the SFC superset after table merging. The for-loop of *Lines* 2-15 processes each vNF $r_j \in S'$ to find a feasible table merging scheme for its logic units. Here, we first check whether r_j has been processed, and if not, we put it in R (*Lines* 3-6). Next, we check each vNF r_k in S' after r_j to see whether *Exact Merge*, *Action Merge* or *Match Merge* can be applied to the logic units in r_j and r_k (*Lines* 7-14). If yes, we store the table merging scheme in $\mathcal{M}$ and mark r_k as processed (*Lines* 8-10). Otherwise, we break the for-loop because only adjacent vNFs can use table merging (*Lines* 11-12). Finally, *Lines* 16-18 apply the table merging schemes in $\mathcal{M}$ to get S''.

Algorithm 3 is for the deployment of merged SFC superset S''. *Line* 1 initializes V_d and P for the selected nodes for SFC deployment and the shortest paths between each node pair in V, respectively. We then calculate the smallest number of adjacent PDP switches (m) that are in a line and can accommodate the merged superset SFC S'' (*Line* 2). *Line* 3 finds the shortest path between each node pair in V, and *Line* 4 finds the node v that has the largest intersection count with the shortest paths of all the SFCs in $\mathbf{S}$ and puts it as the first node in the selected node set V_d. Next, we find m adjacent nodes from v in a line to put in V_d (*Lines* 5-8). We treat the

Algorithm 2: Fine-grained Table Merging

```
**Input**: Merged SFC superset $S'$.
$R = \emptyset$, $\mathcal{M} = \emptyset$, $S'' = S'$;
for each vNF $r_j \in S'$ do
    if $r_j \in R$ then
        continue;
    end
    insert $r_j$ in $R$;
    for each vNF $r_k$ in $S'$ from $r_{j+1}$ do
        if table merging can be applied to $r_j$ and $r_k$ then
            append table merging scheme in $\mathcal{M}$;
            insert $r_k$ in $R$;
        else
            break;
        end
    end
end
for each table merging scheme in $\mathcal{M}$ do
    merge related logic units and update $S''$ accordingly;
end
return $S''$;
```

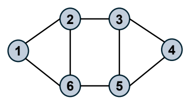
PDP switches in V_d as a big switch [17], deploy the logic units in S'' on the big switch in the greedy manner, and compute the total hop-count of all the SFCs in $\mathbf{S}$ as h_m (*Line* 9). Then, the for-loop of *Lines* 10-20 tries to optimize V_d in iterations, hoping to further reduce h_m. Specifically, in each iteration, we try to replace a node in V_d and update the subsequent nodes accordingly, to reduce h_m (*Lines* 11-19).

The time complexity of *Algorithm* 1 is $O(N\cdot|S'|^2+|L_a|)$, that of *Algorithm* 2 is $O(|S'|^2+|\mathcal{M}|)$, and *Algorithm* 3 runs in $O(|S''|+|V|^2\cdot\log_2|V|+|V|\cdot|E|+N+m+M\cdot|V_d|)$.

V. Performance Evaluation

A. Simulation Setup

The simulations consider two topologies for the SNT, *i.e.*, the 6-node topology and 24-node US Backbone (USB) topology as depicted in Figs. 3(a) and 3(b), respectively. The capacity of each link in the topologies is assumed to be 40 Gbps, and each PDP switch can provide 12 stages to carry vNFs. In order to make the simulations practical, we consider 10 different types of stateful vNFs in each simulation, which are the stateful load balancer, stateful NAT, TCP firewall, heavy hitter detection, DNS reflection attack mitigator, SYN flood detection, DNS request analysis, super spreader identification, and flow size monitor. Given the computational complexity of the ILP, its application is confined to small-scale scenarios, *e.g.*, those in the 6-node topology. In the following discussions, we refer to the three-step heuristic designed in Section IV


(a) 6-node topology

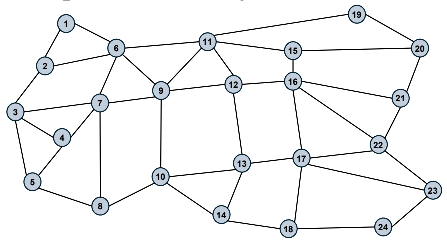

(b) 24-node US Backbone topology

Fig. 3. Topologies used in simulations.

Algorithm 3: Deployment of SFC Superset

```
**Input**: $G(V,E)$, $\mathbf{S}=\{S_1,\cdots,S_N\}$, $S''$, $C_{\text{SRAM}}$, $C_{\text{TCAM}}$, $C_{\text{BW}}$,
        the stages capacity $s$, and upper-limit of iterations $M$.
**Output**: Nodes with SFC deployment $V_d$, total hop-count $h_m$.
$V_d = \emptyset$, $P = \emptyset$;
calculate the smallest number of PDP switches $m$ needed to
  deploy $S''$ based on $C_{\text{SRAM}}$, $C_{\text{TCAM}}$, $C_{\text{BW}}$, $s$ and $\mathbf{S}$;
find the shortest path between each node pair in $V$ to put in $P$;
find node $v$ that has the largest intersection count with the
  shortest paths of all SFCs in $\mathbf{S}$, $V_d \leftarrow v$, and $V' = V \setminus v$;
while $|V_d| < m$ do
    find $u \in V'$ as a node that is adjacent to last node in $V_d$;
    $V_d = V_d \cup \{u\}$, $V' = V' \setminus u$;
end
treat PDP switches in $V_d$ as a big switch, deploy $S''$ on the big
  switch, and compute total hop-count as $h_m$;
for each $i \in [1, M]$ do
    for each node $v \in V_d$ do
        $V'_d = V_d$;
        randomly select an adjacent node $u$ of $v$ that is not in
          $V'_d$, replace the next node of $v$ in $V'_d$ with $u$, and
          update subsequent nodes in $V'_d$ accordingly;
        treat PDP switches in $V_d$ as a big switch, deploy $S''$
          on the big switch, and compute total hop-count as $h'$;
        if $h' < h_m$ then
            $h_m = h'$, $V_d = V'_d$;
            break;
        end
    end
end
```

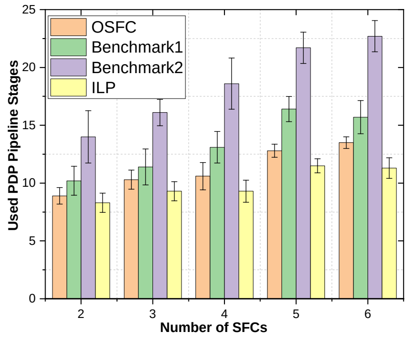
as OSFC. We compare OSFC against two benchmarks: 1) *Benchmark* 1, which replaces *Algorithm* 1 in OSFC with a simple greedy approach that merges all the SFCs in $\mathbf{S}$ to a longest SFC superset, and 2) *Benchmark* 2, which replaces *Algorithm* 2 in OSFC with the table merging scheme in pSFC [17] (*i.e.*, only considering *Exact Merge*). All the heuristics set the upper-limit of iterations in *Algorithm* 3 as $M = 5$. The simulations obtain each data point by averaging the results of 10 independent trials and mark the 95% confidence intervals.

B. Small-Scale Simulations

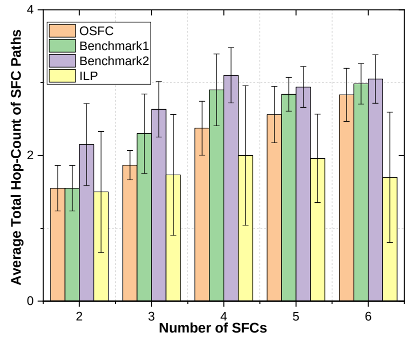
The small-scale simulations employ the 6-node topology, set $|\mathbf{S}| \in [2, 6]$ and $\alpha = 0.6$, and make each SFC contain $[3, 5]$ vNFs to serve $f_i = 100$ flows. We limit the ILP's running time to 10 hours (36,000 seconds), opting for the best solution attained within this duration if the optimal result has not been obtained. Fig. 4 shows the results of small-scale simulations, which indicate that OSFC approximates the ILP better than the two benchmarks, in terms of both the number of used stages and average total hop-count of SFC paths. As depicted in Fig. 4(a), the rise in OSFC's stage usage is more gradual with an increase in SFCs, suggesting that its consolidation strategy can use the memory resources in PDP switches more effectively.

C. Large-Scale Simulations

The large-scale simulations do not consider the ILP due to its complexity, use the 24-node USB topology, set $|\mathbf{S}| \in [10, 30]$, and make each SFC contain $[3, 10]$ vNFs to serve $f_i = 1,000$ flows. The results in Fig. 5 still confirm that OSFC outperforms the two benchmarks. The comparison between

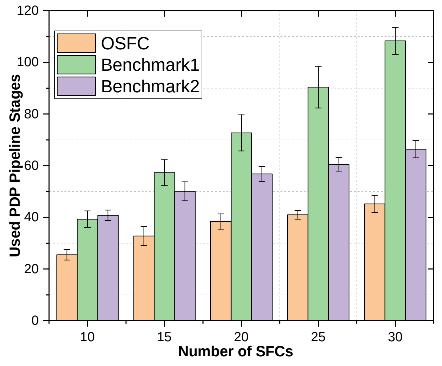


(a) Used PDP pipeline stages

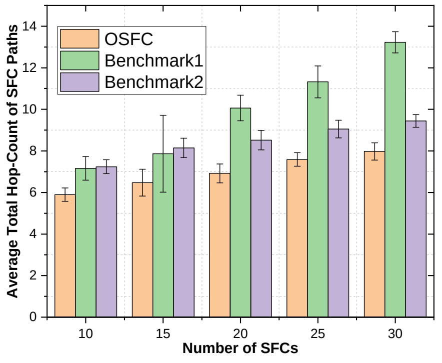


(b) Average total hop-count of SFC paths

Fig. 4. Results of small-scale simulations.



(a) Used PDP pipeline stages



(b) Average total hop-count of SFC paths

Fig. 5. Results of large-scale simulations.

OSFC and *Benchmark* 1 demonstrates that OSFC's approach of coarse-grained SFC merging efficiently reutilizes memory resources for vNFs, thereby reducing the redundancy of logic units in them and minimizing the usage of stages in PDP switches. Meanwhile, contrasting OSFC with *Benchmark* 2 underscores the effectiveness of our proposed fine-grained merging strategy in minimizing resource redundancy.

VI. Conclusion

This paper explored the optimization of initial deployment of stateful SFCs on P4-enabled PDP switches. We commenced by first developing an ILP model and then designing a time-efficient heuristic to solve the problem quickly. Extensive simulations demonstrated that our proposals efficiently accommodate stateful SFC requests. Notably, our heuristic closely approximated the ILP's optimal outcomes and remarkably outperformed two existing benchmarks.

Acknowledgments

This work was supported by the National Key R&D Program of China under Grant 2023YFB2903903.

References

[1] W. Lu *et al.*, "AI-assisted knowledge-defined network orchestration for energy-efficient data center networks," *IEEE Commun. Mag.*, vol. 58, pp. 86–92, Jan. 2020.
[2] L. Gong and Z. Zhu, "Virtual optical network embedding (VONE) over elastic optical networks," *J. Lightw. Technol.*, vol. 32, pp. 450–460, Feb. 2014.
[3] J. Liu *et al.*, "On dynamic service function chain deployment and readjustment," *IEEE Trans. Netw. Serv. Manag.*, vol. 14, pp. 543–553, Sept. 2017.
[4] Z. Zhu, W. Lu, L. Zhang, and N. Ansari, "Dynamic service provisioning in elastic optical networks with hybrid single-/multi-path routing," *J. Lightw. Technol.*, vol. 31, pp. 15–22, Jan. 2013.
[5] P. Lu *et al.*, "Highly-efficient data migration and backup for Big Data applications in elastic optical inter-datacenter networks," *IEEE Netw.*, vol. 29, pp. 36–42, Sept./Oct. 2015.
[6] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 87–95, Jul. 2014.
[7] L. Dong, N. da Fonseca, and Z. Zhu, "Application-driven provisioning of service function chains over heterogeneous nfv platforms," *IEEE Trans. Netw. Serv. Manag.*, vol. 18, pp. 3037–3048, Sept. 2021.
[8] V. Sivaraman *et al.*, "Heavy-hitter detection entirely in the data plane," in *Proc. of SOSR 2017*, pp. 164–176, Apr. 2017.
[9] J. Cao *et al.*, "CoFilter: High-performance switch-accelerated stateful packet filter for bare-metal servers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, pp. 2249–2262, Sept. 2021.
[10] K. Qian *et al.*, "FlexGate: High-performance heterogeneous gateway in data centers," in *Proc. of APNet 2019*, pp. 36–42, Aug. 2019.
[11] P. Zheng, T. Benson, and C. Hu, "P4visor: Lightweight virtualization and composition primitives for building and testing modular programs," in *Proc. of CoNEXT 2018*, pp. 98–111, Dec. 2018.
[12] X. Chen *et al.*, "P4SC: Towards high-performance service function chain implementation on the p4-capable device," in *Proc. of IM 2019*, pp. 1–9, Apr. 2019.
[13] D. Wu *et al.*, "Accelerated service chaining on a single switch ASIC," in *Proc. of HotNets 2019*, pp. 141–149, Nov. 2019.
[14] X. Chen *et al.*, "Speed: Resource-efficient and high-performance deployment for data plane programs," in *Proc. of ICNP 2020*, pp. 1–12, Oct. 2020.
[15] G. Bianchi *et al.*, "Open packet processor: a programmable architecture for wire speed platform-independent stateful in-network processing," *arXiv preprint arXiv:1605.01977*, May 2016.
[16] A. Sivaraman *et al.*, "Packet transactions: High-level programming for line-rate switches," in *Proc. of ACM SIGCOMM 2016*, pp. 15–28, Aug. 2016.
[17] X. Zhang, L. Cui, F. Tso, and W. Jia, "Compiling service function chains via fine-grained composition in the programmable data plane," *IEEE Trans. Serv. Comput.*, vol. 16, pp. 2490–2502, Jul./Aug. 2023.
[18] T. Li, Z. Ma, and Z. Zhu, "st-SFC: Optimizing dynamic deployment of stateful SFCs on P4-based PDP switches," *IEEE Trans. Netw. Serv. Manag., in Press*, 2024.