MrgRecmp4: Efficient Stateful SFC Recompilation in PDP Switches with Flexible Table Merging

Tingyu Li, Zhen Wei, Xiaoliang Chen, and Zuqing Zhu†
School of Information Science and Technology, University of Science and Technology of China, Hefei, China
Email: {ltywd119, zhenwei2429}@mail.ustc.edu.cn, xlichen@ieee.org, zqzhu@ieee.org†

***Abstract*—The fast development of P4-based programmable data plane (PDP) has motivated the offloading of service function chains (SFCs) to switches. To adapt to dynamic service demands, PDP switches will need to be reprogrammed occasionally to update their packet processing pipelines, especially for stateful SFCs. In this work, we propose a novel stateful SFC recompilation system, namely, MrgRecmp4. It jointly considers the existing and new SFCs to optimize the P4 program recompilation for deploying new SFCs as reconstructed pipelines, such that the complexity of SFC reconfiguration can be minimized together with the hardware resource utilization in PDP switches. We first lay out the system design of MrgRecmp4, and explain how to leverage a flexible and fine-grained table merging scheme to optimize the P4 programs of stateful SFCs. Then, we formulate a bi-level integer linear programming (BILP) model to accomplish network-wide optimization for MrgRecmp4, which jointly considers the existing and new SFCs in a PDP network to determine the PDP switches that need to be reprogrammed and to generate reconstructed pipelines for them with table merging. We also propose a sliding-window-based heuristic to solve the network-wide optimization quickly. Both simulations and real-world experiments confirm that MrgRecmp4 outperforms existing benchmarks.**

***Index Terms*—Service function chains, Recompilation, Programmable data plane, Table merging, Bi-level optimization.**

I. Introduction

Nowadays, network innovations are developing fast in the Internet, especially for the metro and backbone segments [1–7]. Meanwhile, it is popular for service providers to instantiate virtual network functions (vNFs) on general-purpose software and hardware platforms and arrange them as service function chains (SFCs) for various network services [8]. This motivates researchers to offload SFCs to P4-based programmable data plane (PDP) switches (*e.g.*, those based on Tofino ASICs [9]), to explore their line-speed packet processing at 100 Gbps or beyond and hop latency at the sub-microsecond level [10]. However, certain hardware restrictions make it difficult for PDP switches to accommodate dynamic service demands. Specifically, the challenges of supporting reconfigurable SFCs with hardware PDP switches lie in the limitations of hardware resources, the support of stateful packet processing, and the complexity and overhead of pipeline reconfiguration.

A packet processing pipeline of Tofino-based PDP switches includes a limited number of stages, each of which contains restricted sizes of SRAM, TCAM, hash bits, *etc*. These limitations restrict the number of flow entries and the complexity of logic that can be implemented for SFCs. Also, if there are dependencies between two vNFs, they can only be realized across multiple stages, further degrading resource efficiency. Although people have proposed a few table merging techniques to offload SFCs with improved resource utilization [11, 12], they only tried to merge identical match-action tables (MATs), which have very limited application scope given the diversity of services and user flows across SFCs. Moreover, the SFC deployment with simple table merging can hardly cope with stateful SFCs, which becomes increasingly popular recently [13–17]. For instance, as Tofino ASIC only supports no more than two conditional statements in the register action, it cannot fully realize stateful SFCs like TCP firewall and thus has to rely on the control plane for state updates [18, 19], while since stateful vNFs such as heavy hitter and stateful load-balancer consume a lot of memory resources, offloading them to a PDP switch can only process a limited number of flows. Therefore, it is challenging to offload stateful SFCs to PDP switches with high resource efficiency and good performance.

To adapt to dynamic service demands, we need to reconfigure SFCs. This brings another major challenge to offloading SFCs to PDP switches, because their resources restrict the number of vNF types that can be pre-deployed. Hence, reprogramming PDP switches becomes inevitable when new types of vNFs are requested or SFCs need to be redeployed to re-optimize network resource usage and quality-of-service (QoS) [20]. However, to the best of our knowledge, existing studies have not tried to optimize the P4 program recompilation for deploying new stateful SFCs as reconstructed pipelines. Such an optimization is relevant, because without it, we cannot deploy sufficient stateful vNFs after each recompilation, leading to reprogramming PDP switches more frequently and thus more service interruptions. Meanwhile, the optimization is challenging for the following reasons. First, we need to develop a way to compile the P4 programs of existing and new stateful SFCs. Here, a fine-grained and accurate abstraction of the hardware resources in PDP switches is necessary to analyze P4 programs of stateful SFCs to achieve resource-efficient table merging in advance. Second, we need to jointly optimize the QoS of existing and new stateful SFCs.

In this work, we propose a novel stateful SFC recompilation system, namely, MrgRecmp4. It jointly considers existing and new SFCs to optimize the P4 program recompilation for deploying new SFCs as reconstructed pipelines, such that the complexity of SFC reconfiguration can be minimized together with the hardware resource utilization in PDP switches. We first design the system of MrgRecmp4, and explain how to

leverage flexible and fine-grained table merging to optimize the P4 programs of stateful SFCs. Then, we formulate a bi-level integer linear programming (BILP) model to accomplish network-wide[1] optimization for MrgRecmp4. A sliding-window-based heuristic is also proposed to solve the optimization quickly. Both simulations and real-world experiments confirm that MrgRecmp4 outperforms existing benchmarks.

The rest of the paper is organized as follows. In Section II, we describe the fine-grained table merging strategy and dynamic deployment mechanism for stateful SFCs, which is the basis of MrgRecmp4. Section III explains the network-wide optimization for MrgRecmp4, formulates it as a BILP model, and designs a sliding-window-based heuristic to solve the problem quickly. The performance evaluations with numerical simulations and real-world experiments are discussed in Section IV. Finally, Sections V and VI summarize recent related work and the paper, respectively.

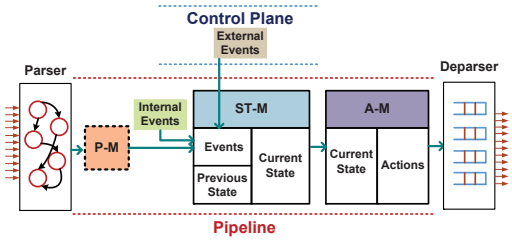


Fig. 1. General processing paradigm for stateful vNFs in PDP switch.

II. Operation Principle

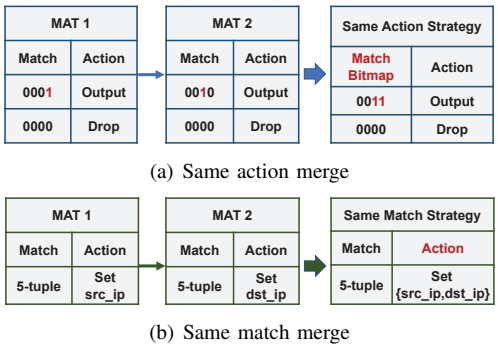
In this section, we explain the operation principle that MrgRecomp4 uses to optimize the P4 programs of stateful SFCs for recompilation, which is basis of MrgRecomp4.

A. Fine-Grained Table Merging for Stateful vNFs

We first introduce the state machine abstraction and fine-grained table merging for stateful vNFs. As there can be many types of stateful vNFs, we need to find a common paradigm to abstract their core parts. As shown in Fig. 1, a stateful vNF can be abstracted to a state machine composed of two MATs, *i.e.*, a state transition module (ST-M) and an action module (A-M). In certain stateful vNFs, there can also be preprocessing modules (P-M) (*e.g.*, hash operations), which convert packet fields into that required by subsequent ST-M and A-M. When a packet enters a stateful vNF, it first passes through the ST-M, which obtains the current state (can be per-flow state, per-packet state and link state) based on perceived internal events (determined by the pipeline) or external events (instructed by the control plane) and store the state information in registers. Next, the packet is processed by the A-M, matching to its fields and current state for corresponding actions.

Fig. 2 explains how we merge state machines in a fine-grained manner after abstracting stateful vNFs, with three table merging strategies [21, 22]. The first one is the exact merge for

[1]Here, "network-wide" means that our optimization spans the entire network, addressing all the PDP switches in it rather than only focusing on one.



(a) Same action merge

(b) Same match merge

Fig. 2. Examples on fine-grained table merging.

merging two identical MATs. The second one, namely, same action merge, merges two A-Ms in different stateful vNFs if they contain same actions. Specifically, the match items can be merged by setting the A-Ms to use a match bitmap that covers all the match items, as shown in Fig. 2(a). Finally, the third one addresses identical match items in two MATs (*i.e.*, same match merge), as shown in Fig. 2(b), and it can be used to save the hardware resources spent on frequently-used match items (such as five-tuples). These three table merging strategies are the basic operations used in our stateful SFC recompilation.

B. SFC Isolation and Runtime Dynamic Deployment

As stateful SFC recompilation will bring PDP switch(es) offline and cause service interruptions, it is desired to have certain flexibility for dynamic SFC deployment in runtime without SFC recompilation. This can be achieved by concatenating vNFs to form a superset SFC on each PDP switch (*i.e.*, different stateful SFCs can be realized in runtime by letting packets select vNFs in the superset SFC to pass through). Meanwhile, the superset SFC should have the ability to isolate the traffic processing of different SFCs. Fig. 3 provides an example on this, where we need to provision three SFCs in runtime. The SFCs contain four types of vNFs, *i.e.*, the stateful network address translation (NAT), Layer-4 load balancing (LB), stateful firewall (FW), and heavy hitter detection (HH). Then, by adding a match item of *client flag* in each merged MAT, we can let the superset SFC distinguish the traffic of the three SFCs. Therefore, a stateful SFC can be deployed in runtime as long as all of its vNFs can be found in the superset stateful SFCs pre-deployed in the PDP network.

Nevertheless, even with such flexibility in runtime, PDP switches might still be reprogrammed occasionally to adapt to dynamic service demands. Since the QoS demands of existing and new SFCs should be satisfied after recompilation, we need to perform network-wide optimization to determine the PDP switches that need to be reprogrammed and to generate reconstructed pipelines for them with proper table merging, such that the complexity of the SFC recompilation is minimized together with resource utilization in PDP switches.

III. Network-Wide Optimization for MrgRecomp4

In this section, we discuss the algorithm design for the network-wide optimization in MrgRecomp4.

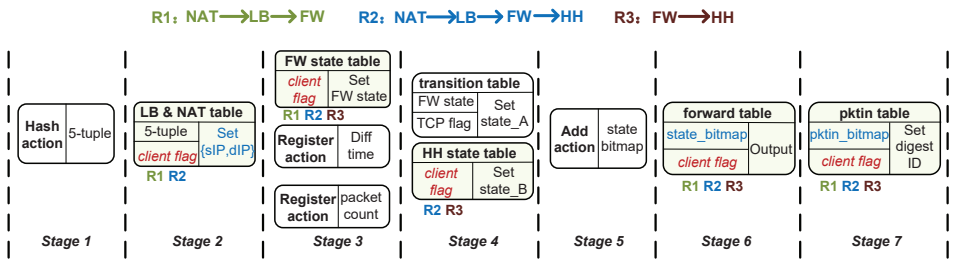


Fig. 3. Example on realizing different SFCs with a superset stateful SFC in the pipeline of a PDP switch.

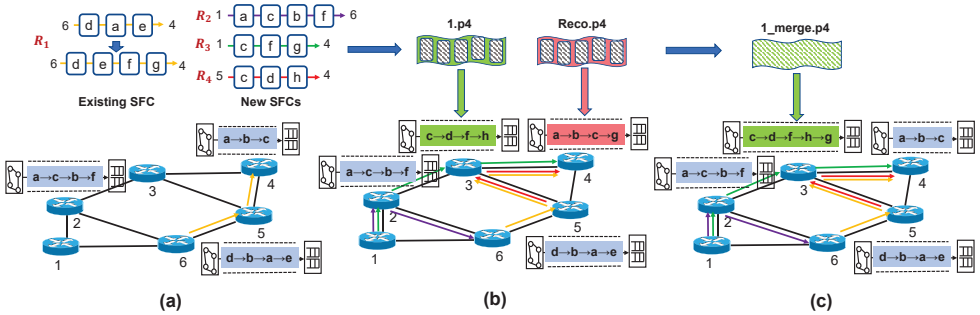


Fig. 4. Examples on SFC recompilation, (a) original network, (b) simple SFC recompilation, and (c) SFC recompilation from network-wide optimization.

A. Problem Description and Network Model

We assume that the substrate network (SNT) is built with PDP switches. Fig. 4 explains the stateful SFC recompilation considered in this work and why network-wide optimization is needed. The network status before the SFC recompilation is shown in Fig. 4(a), where there is only one in-service SFC R_1 running over 6→5→4 and three superset SFCs deployed on *Nodes* 2, 4, and 5, respectively. R_1 uses the superset SFC on *Node* 5 to realize its SFC, but we need to change its SFC to a longer one. Meanwhile, there are three new SFC requests (*i.e.*, R_2-R_4) to be deployed in the network. As R_1, R_3 and R_4 ask for the vNFs that are not included in the deployed superset SFCs, an SFC recompilation is inevitable.

Fig. 4(b) shows a simple SFC recompilation scheme without proper network-wide optimization. Here, we assume that the fine-grained table merging is not used, and thus the hardware resources on each PDP switch can only accommodate 4 vNFs at most. R_2 can be provisioned with the superset SFC on *Node* 2 (no SFC recompilation is required). To serve the SFCs of R_1, R_3 and R_4, we decide to deploy a new superset SFC on *Node* 3 and recompile the existing superset SFC on *Node* 4. Then, two switches need to be reprogrammed. On the other hand, Fig. 4(c) shows the SFC recompilation scheme obtained by network-wide optimization. Here, with the fine-grained table merging and optimization concerning the existing and new SFCs, we can deploy a longer superset SFC on *Node* 3 to satisfy all the missing vNFs in R_1, R_3 and R_4, and only need to reprogram one PDP switch. Therefore, the network-wide optimization is necessary to recompile stateful SFCs efficiently as reconstructed pipelines in the PDP network.

We model the topology of the SNT as $G(V, E)$, where V and E are the sets of PDP switches and links in it, respectively. There are $\mathcal{S}$ stages in each PDP switch, and based on the setting of Tofino 1 [9], we set $\mathcal{S} = 12$. As for the memory in PDP switches, this work only considers SRAM and TCAM, as they are the bottleneck resources in each stage. The capacity of SRAM and TCAM in each stage is U_S and U_T, respectively, and the bandwidth capacity of each link $e \in E$ is B. We denote the set of SFC requests as $\mathcal{R}$, in which each request R_i corresponds to a set of flows using a same SFC and can be expressed as $R_i(s_i, d_i, C_i, b_i, f_i)$, where s_i and d_i are the source and destination, C_i is the required SFC with vNFs $\{c_{i,1}, \cdots, c_{i,J}\}$, b_i is the total bandwidth demand, and f_i is the number of flows, which determines the table size of MATs.

B. BILP Optimization Model

The network-wide optimization in MrgRecomp4 needs to consider the existing and new SFCs to select PDP switches to reprogram, generate reconstructed pipelines for them with table merging, and provision all the SFCs that depend on or are impacted by SFC recompilation. As it is a fairly complex problem, we divide it into two steps for being modeled with a BILP. Specifically, the upper-level model selects proper PDP switches to reprogram to minimize the complexity of the SFC recompilation, while the lower-level model accepts feasible solutions from the upper-level model to determine the provisioning schemes of all the SFCs such that the resource usage and total end-to-end (E2E) latency of SFCs are minimized. Table I lists the parameters used in the BILP.

1) Upper-level Optimization: Table II shows the variables used in the upper-level optimization. Note that, SFC recompilation can move a vNF to a node v that originally carries the vNF or a node u that only carries the vNF after recompilation. In the former case, the vNF does not use any additional

TABLE I
PARAMETERS USED IN THE BILP MODEL

$\mathcal{M}/\mathcal{A}$	Set of match and action types that can be used in MATs.
$\mathcal{T}$	Set of vNF types that can be supported in the SNT.
h_{v_1,v_2}	Hop-count of the shortest path between v_1 and v_2.
U_S/U_T	Capacities of SRAM/TCAM resources in each stage.
u_m^S/u_m^T	Sizes of SRAM/TCAM used by matches of type m.
u_a^S/u_a^T	Sizes of SRAM/TCAM used by actions of type a.
$u^S_{a,\text{match}}$	Size of preset match bitmap for same action merging.
$k_{i,j,l}^{m,a}$	Boolean that equals 1 if the l-th MAT in the j-th vNF of SFC R_i is MAT $(m;a)$, and 0 otherwise.
$d_{l_1,l_2}^{i,j}$	Boolean that equals 1 if the l_2-th MAT in the j-th vNF of SFC R_i depends on the l_1-th MAT, and 0 otherwise.
M	A large positive integer introduced for linearization.
$\mu_{v,t}^S/\mu_{v,t}^T$	Sizes of SRAM/TCAM used by a vNF of type t, which has been deployed on node $v \in V$.
u_t^S/u_t^T	SRAM/TCAM usages per flow for a vNF of type t, if table merging is not adopted.
$W_v^t/\tilde{W}_v^t$	Number of flows that a vNF of type t deployed on node v can still supported/currently carries.
w_i^t/w_v^t	Number of flow entries to be used if deploying a vNF of type t in request R_i/originally on node v.

resources on node v, but needs to use the flow entries there. In the latter case, the vNF needs to use additional resources on node u. Table merging is not considered in the upper-level optimization and we only output the values of variables $\{o_v\}$ to the lower-level optimization, because the actual SFC recompilation scheme and service provisioning scheme of SFC requests are determined in the lower-level optimization.

TABLE II
VARIABLES USED IN THE UPPER-LEVEL MODEL

o_v	Boolean variable that equals 1 if node v needs to be recompiled, and 0 otherwise.
$p_{i,t,v}^{\text{old/new}}$	Boolean variable that equals 1 if the type-t vNF of a new request R_i is placed on a node v that *originally carries the vNF/has been recompiled*, and 0 otherwise.
$q_{v_1,t,v_2}^{\text{old/new}}$	Boolean variable that equals 1 if a type-t vNF is moved from node v_1 to node v_2 (node v_2 originally *carries/does not carry* such a vNF) by the SFC recompilation, and 0 otherwise.

Objective:

The objective of the upper-level optimization is to minimize the number of flows impacted by the SFC recompilation (*i.e.*, minimizing the complexity due to flow migration).

$$\text{Minimize} \qquad \sum_{v,t} \tilde{W}_v^t \cdot o_v. \tag{1}$$

Constraints:

$$\begin{cases} \sum_{i,t} p_{i,t,v}^{\text{new}} \cdot u_t^S \cdot f_i + \sum_{v_1,t} q_{v_1,t,v}^{\text{new}} \cdot \mu_{v_1,t}^S \le o_v \cdot U_S \cdot \mathcal{S}, \\ \sum_{i,t} p_{i,t,v}^{\text{new}} \cdot u_t^T \cdot f_i + \sum_{v_1,t} q_{v_1,t,v}^{\text{new}} \cdot \mu_{v_1,t}^T \le o_v \cdot U_T \cdot \mathcal{S}, \\ \qquad\qquad \forall v,\ \{v_1 : v_1 \in V, v_1 \ne v\}. \end{cases} \tag{2}$$

Eq. (2) ensures that SRAM/TCAM usages on each recompiled node v are within their capacities after the SFC recompilation.

$$\sum_i p_{i,t,v}^{\text{old}} \cdot w_i^t + \sum_{v_1} q_{v_1,t,v}^{\text{old}} \cdot w_{v_1}^t \le (1-o_v) \cdot W_v^t,\ \forall v,t,v_1 \ne v. \tag{3}$$

Eq. (3) ensures that used flow entries in each original vNF on node v are within their capacity after the SFC recompilation.

$$\sum_v p_{i,t,v}^{\text{old}} + p_{i,t,v}^{\text{new}} = 1, \quad \forall i,t. \tag{4}$$

Eq. (4) ensures that each vNF in an SFC request is deployed on one and only one node.

$$\sum_{v_1} q_{v,t,v_1}^{\text{old}} + q_{v,t,v_1}^{\text{new}} = o_v,\ \forall v,t,\{v_1 : v_1 \in V, v_1 \ne v\}. \tag{5}$$

Eq. (5) ensures that each original vNF on a recompiled node is moved to one and only one node.

2) Lower-level Optimization: After getting the nodes that need to be recompiled, the lower-level model records the SFC deployment in the remaining nodes as parameters and proceeds to optimize the actual SFC recompilation and provisioning schemes of SFC requests. Tables III and IV respectively list the parameters and variables used in the lower-level optimization.

TABLE III
PARAMETERS USED IN THE LOWER-LEVEL MODEL

Ω	Set of nodes that need to be recompiled.
$y_{m,a}^{\text{pre},v,s}$	the Boolean parameter that equals 1 if the MAT $(m;a)$ has been deployed on the s-th stage of node v, and 0 otherwise.
$k_t^{i,j}$	Boolean that equals 1 if the j-th vNF in SFC R_i is of type t, and 0 otherwise.
$v^{v,t}$	Boolean that equals 1 if a vNF of type t has been deployed on node v, and 0 otherwise.
$\lambda_{m,a}^{v,t}$	Number of remaining flow entries in an MAT $(m;a)$ in the type-t vNF deployed on node v.

Objective:

The objective of the lower-level optimization is to minimize the used hardware resources in PDP switches together with the total hop-count of provisioned SFC requests as

$$\text{Minimize} \qquad \alpha \cdot \sum_{v,s} (\mathcal{C}_S^{v,s} + \mathcal{C}_T^{v,s}) + (1-\alpha) \cdot \mathcal{L}. \tag{6}$$

where α is the weight to adjust the importance of the two terms, $\mathcal{C}_S^{v,s}$ and $\mathcal{C}_T^{v,s}$ are the total sizes of used SRAM and TCAM in the s-th stage of node v, respectively, and $\mathcal{L}$ is the total hop-count of SFC requests, as

$$\begin{cases} \mathcal{C}_S^{v,s} = \sum_m \left(\sum_a \eta_{m,a}^{v,s} \cdot u_a^S + u_m^S \right) \cdot \sum_i \tau_{i,m}^{v,s} + \\ \sum_a \sum_i \tau_{i,a}^{v,s} \cdot (u_a^S + u^S_{a,\text{match}}) + \sum_{m,a} \sum_i \tau_{i,m,a}^{v,s,4} \cdot \left(u_m^S + u_a^S \right), \\ \mathcal{C}_T^{v,s} = \sum_{m,a} \sum_i \tau_{i,m,a}^{v,s,4} \cdot \left(u_m^T + u_a^T \right), \\ \mathcal{L} = \sum_{v_1,v_2,i,j_1,j_2} x_{i,j_1}^{v_1} \cdot x_{i,j_2}^{v_2} \cdot h_{v_1,v_2} \cdot f_i. \end{cases} \tag{7}$$

Note that, we define table merging types as $\chi = \{1,2,3,4\}$ to denote the exact merging, same match merging, same action merging, and no merging, respectively.

Constraints:

$$\begin{cases} \varepsilon_{i,j,l}^{v,\text{out}} - \varepsilon_{i,j,l}^{v,\text{in}} \ge 0, \\ \varepsilon_{i,j,l}^{v,\text{in}} \ge M \cdot x_{i,j,l}^{v,\text{range}}, \\ \varepsilon_{i,j,l}^{v,\text{out}} \ge M \cdot x_{i,j,l}^{v,\text{range}}, \end{cases} \quad \forall v,i,j,l. \tag{8}$$

TABLE IV
VARIABLES USED IN THE LOWER-LEVEL MODEL

$x_{i,j,l}^{v,s}$	Boolean variable that equals 1 if the l-th MAT of j-th vNF in R_i is on the s-th stage of node v, and 0 otherwise.
$\varepsilon_{i,j,l}^{v,\text{in/out}}$	Integer variables that indicate the stage of node v where the deployment of the l-th MAT of j-th vNF in R_i starts and ends.
z_i^e	Boolean variable that equals 1 if SFC R_i uses link $e \in E$, and 0 otherwise.
$x_{i,j,l}^{v,\text{range}}$	Boolean variable that equals 1 if the l-th MAT of j-th vNF of R_i is deployed on node v, and 0 otherwise.
$x_{i,j}^{v,\text{vNF}}$	Boolean variable that equals 1 if the j-th vNF of R_i is deployed on node v, and 0 otherwise.
$y_{m,a}^{v,s}$	Boolean variable that equals 1 if an MAT $(m;a)$ is deployed on the s-th stage of node v, and 0 otherwise.
$\zeta_{m,a}^{v,s,\text{count}}$	Integer variable that indicates the times an MAT $(m;a)$ being reused on the s-th stage of node v.
$\zeta/\eta/\theta/\delta_{m,a}^{v,s}$	Boolean variables that equal 1 if an MAT $(m;a)$ with *exact/same match/same action/without any merging* is on s-th stage of node v, and 0 otherwise.
$\beta_{m,a_1,a_2}^{v,s}$	Boolean variable that equals 1 if MATs $(m;a_1)$ and $(m;a_2)$ are on the s-th stage of node v, and 0 otherwise.
$\gamma_{m_1,m_2,a}^{v,s}$	Boolean variable that equals 1 if MATs $(m_1;a)$ and $(m_2;a)$ are on the s-th stage of node v, and 0 otherwise.
$\tau_{i,m,a}^{v,s,\chi}$	Number flows of R_i using MAT $(m;a)$ with type-χ merging, which is deployed on s-th stage of node v.
$\tau_{i,m/a}^{v,s}$	Numbers of flows of R_i using match item m/action item a, which is deployed on s-th stage of node v after same match/same action merging.
$\rho_{m,a,\chi}^{i}$	Boolean variable that equals 1 if MAT $(m;a)$ of R_i uses type-χ merging strategy, and 0 otherwise.
$\kappa_{i,j,l}^{v}$	Auxiliary variables for linearization.

Eq. (8) ensures the correct mapping relation to deploy each MAT on a stage of one node.

$$\begin{cases} \kappa_{i,j,l}^{v} \le \varepsilon_{i,j,l}^{v,\text{out}} - \varepsilon_{i,j,l}^{v,\text{in}} + 1, \\ \kappa_{i,j,l}^{v} \le \mathcal{S} \cdot x_{i,j,l}^{v,\text{range}}, \\ \kappa_{i,j,l}^{v} \ge \left(\varepsilon_{i,j,l}^{v,\text{out}} - \varepsilon_{i,j,l}^{v,\text{in}} + 1\right) - \mathcal{S}\left(1 - x_{i,j,l}^{v,\text{range}}\right), \\ \sum_{s} x_{i,j,l}^{v,s} = \kappa_{i,j,l}^{v}, \end{cases} \forall v, i, j, l. \quad (9)$$

Eq. (9) ensures the correct relation among the variables related to deploying MATs.

$$\begin{cases} \zeta_{m,a}^{v,s,\text{count}} = \sum_{i,j,l} x_{i,j,l}^{v,s} \cdot k_{m,a}^{i,j,l}, \\ \zeta_{m,a}^{v,s} \le \frac{1}{2} \cdot \zeta_{m,a}^{v,s,\text{count}}, \\ \zeta_{m,a}^{v,s} \ge \frac{1}{M} \cdot \left(\zeta_{m,a}^{v,s,\text{count}} - 1\right), \end{cases} \forall v, s, m, a. \quad (10)$$

Eq. (10) ensures that the MATs using exact merging are correctly determined.

$$\begin{cases} \beta_{m,a_1,a_2}^{v,s} \le \sum_{i,j,l} x_{i,j,l}^{v,s} \cdot k_{m,a_1}^{i,j,l}, \\ \beta_{m,a_1,a_2}^{v,s} \le \sum_{i,j,l} x_{i,j,l}^{v,s} \cdot k_{m,a_2}^{i,j,l}, \\ \beta_{m,a_1,a_2}^{v,s} \ge y_{m,a_1}^{v,s} + y_{m,a_2}^{v,s} - 1, \\ \eta_{m,a_1}^{v,s} \ge \beta_{m,a_1,a_2}^{v,s}, \end{cases} \forall v, s, m, a_1, a_2, \quad (11)$$

$$\eta_{m,a_1}^{v,s} \le \sum_{a_2} \beta_{m,a_1,a_2}^{v,s}, \quad \forall v, s, m, a_1. \quad (12)$$

Eqs. (11) and (12) ensure that the MATs using same match merging are correctly determined.

$$\begin{cases} \gamma_{m_1,m_2,a}^{v,s} \le \sum_{i,j,l} x_{i,j,l}^{v,s} \cdot k_{m_1,a}^{i,j,l}, \\ \gamma_{m_1,m_2,a}^{v,s} \le \sum_{i,j,l} x_{i,j,l}^{v,s} \cdot k_{m_2,a}^{i,j,l}, \\ \gamma_{m_1,m_2,a}^{v,s} \ge y_{m_1,a}^{v,s} + y_{m_2,a}^{v,s} - 1, \\ \theta_{m_1,a}^{v,s} \ge \gamma_{m_1,m_2,a}^{v,s}, \end{cases} \forall v, s, m_1, m_2, a. \quad (13)$$

$$\theta_{m_1,a}^{v,s} \le \sum_{m_2} \gamma_{m_1,m_2,a}^{v,s}, \quad \forall v, s, m_1, a. \quad (14)$$

Eqs. (13) and (14) ensure that the MATs using same action merging are correctly determined.

$$y_{m,a}^{v,s} \ge x_{i,j,l}^{v,s} \cdot k_{m,a}^{i,j,l}, \quad \forall v, s, m, a, i, j, l. \quad (15)$$

$$\begin{cases} \sum_{i,j,l} x_{i,j,l}^{v,s} \cdot k_{m,a}^{i,j,l} \le y_{m,a}^{v,s} \cdot \sum_{i,j,l} k_{m,a}^{i,j,l}, \\ \sum_{i,j,l} x_{i,j,l}^{v,s} \cdot k_{m,a}^{i,j,l} \ge \left(y_{m,a}^{v,s} - 1\right) \cdot \left(1 + \sum_{i,j,l} k_{m,a}^{i,j,l}\right), \end{cases} \forall v, s, m, a. \quad (16)$$

Eqs. (15) and (16) ensure the correct relation between the deployment of each MAT and the deployment of its vNF.

$$y_{m,a}^{v,s} = \eta_{m,a}^{v,s} + \zeta_{m,a}^{v,s} + \theta_{m,a}^{v,s} + \delta_{m,a}^{v,s}, \quad \forall v, s, m, a. \quad (17)$$

Eq. (17) ensures that an MAT can only use a merging strategy.

$$\begin{cases} \tau_{i,m,a}^{v,s,1} \le f_i \cdot \zeta_{m,a}^{v,s}, \tau_{i,m,a}^{v,s,2} \le f_i \cdot \eta_{m,a}^{v,s}, \\ \tau_{i,m,a}^{v,s,3} \le f_i \cdot \theta_{m,a}^{v,s}, \tau_{i,m,a}^{v,s,4} \le f_i \cdot \delta_{m,a}^{v,s}, \end{cases} \forall v, s, i, m, a, \quad (18)$$

$$\sum_{j,l} x_{i,j,l}^{v,s} \cdot k_{m,a}^{i,j,l} \le \sum_{\chi} \tau_{i,m,a}^{v,s,\chi}, \quad \forall v, s, m, a, \quad (19)$$

$$\begin{cases} \tau_{i,m,a}^{v,s,2} = \tau_{i,m}^{v,s}, \\ \tau_{i,m,a}^{v,s,3} = \tau_{i,a}^{v,s}, \end{cases} \forall v, s, i, m, a, \quad (20)$$

$$\begin{cases} \sum_{v,s} \rho_{m,a,\chi}^{i} \cdot \tau_{i,m,a}^{v,s,\chi} = \rho_{m,a,\chi}^{i} \cdot f_i, \\ \sum_{\chi} \rho_{m,a,\chi}^{i} = 1, \end{cases} \forall i, m, a. \quad (21)$$

Eqs. (18)-(21) ensure that for each MAT in SFC R_i, no matter which merging strategy is used, the number of flows using it is within the total number of flows f_i.

$$\mathcal{C}_S^{v,s} \le U_S, \mathcal{C}_T^{v,s} \le U_T, \quad \forall v, s, \quad (22)$$

Eqs. (22) ensures that the SRAM/TCAM used in each stage on a node v do not exceed their capacities in the stage.

$$\sum_{i} z_i^e \cdot b_i \le B, \quad \forall e. \quad (23)$$

Eq. (23) ensures the bandwidth constraint of each link.

$$\varepsilon_{i,j,l_1}^{v,\text{out}} \le \varepsilon_{i,j,l_2}^{v,\text{in}} - d_{l_1,l_2}^{i,j} \cdot \left(x_{i,j,l_1}^{v,\text{range}} + x_{i,j,l_2}^{v,\text{range}} - 1\right), \quad \forall v, i, j, l_1, l_2, l_1 \le l_2. \quad (24)$$

Eq. (24) ensures that the deployment of MATs on each node is correctly determined.

$$\begin{cases} y^{\text{pre},v,s}_{m,a} \geq x^{v,s}_{i,j,l} \cdot k^{i,j,l}_{m,a} - M \cdot (1 - x^{v,\text{vNF}}_{i,j}), \\ y^{\text{pre},v,s}_{m,a} \leq x^{v,s}_{i,j,l} \cdot k^{i,j,l}_{m,a} + M \cdot (1 - x^{v,\text{vNF}}_{i,j}), \end{cases} \forall v \in V \backslash \Omega, s, i, j, l, m, a. \tag{25}$$

Eq. (25) ensures that for each node v, which is not recompiled, the value of variable $x^{v,s}_{i,j,l}$ is correctly determined based on the corresponding variable $x^{v,\text{vNF}}_{i,j}$ and parameter $y^{\text{pre},v,s}_{m,a}$.

$$\sum_{i,j} x^{v,\text{vNF}}_{i,j} \cdot k^{i,j}_t \cdot f_i \leq \lambda^{v,t}_{m,a}, \quad \forall v \in V \setminus \Omega, t, m, a. \tag{26}$$

Eq. (26) ensures that the number of flows using each MAT on node v, which is not recompiled, is within the MAT's capacity.

$$x^{v,\text{vNF}}_{i,j} \cdot k^{i,j}_t \leq \upsilon^{v,t}, \quad \forall v \in V \setminus \Omega, i, j, t. \tag{27}$$

Eq. (27) ensures that if the j-th vNF of R_i is on node v, which is not recompiled, the value of variable $x^{v,\text{vNF}}_{i,j}$ is correctly set.

C. Heuristic Algorithm Design

The time complexity of solving the BILP may challenge its adaptability to dynamic network operations. Meanwhile, how to offload SFCs to PDP switches resource-efficiently is known to be $\mathcal{NP}$-hard [23]. Hence, we design a polynomial-time heuristic to solve the network-wide optimization quickly. Similar to the BILP, the heuristic also tackle the problem in two steps. First, we process the existing and new SFC requests to find the PDP switches to reprogram, and obtain the SFCs that need to be reprovisioned or provisioned accordingly. Next, we design a sliding-window-based algorithm to generate reconstructed pipelines for the selected PDP switches with fine-grained table merging, and to serve related SFCs accordingly.

1) Find PDP Switches to Recompile: This step is accomplished with two phases. First, we find the SFCs that can be directly served in the current PDP network by updating table entries. Second, we update the set of pending SFCs and find the PDP switches that need to be recompiled to serve them.

The first phase is handled by *Algorithm* 1, which takes the set of pending SFC requests $\mathcal{R}'$ as input. *Line* 1 sorts the SFCs in $\mathcal{R}'$ to make sure that those with larger resource demands are served earlier. Then, the *Lines* 2-11 try to serve each SFC $R_i \in \mathcal{R}'$ in the sorted order with K-shortest-path routing. If an SFC can be served, that is, all of its vNFs, bandwidth and number of table entries can be satisfied, we remove it from $\mathcal{R}'$ and update the network status accordingly (*Lines* 5-9).

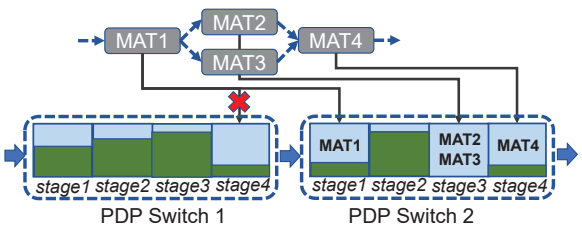


Fig. 5. Example on deploying MATs in a vNF in PDP switches.

Algorithm 2 takes care of the second phase. *Line* 1 is for the initialization, where we introduce V^r to record the PDP switches that will be selected for SFC recompilation, a dummy SNT $\tilde{G}(\tilde{V}, \tilde{E})$ that is identical to $G(V, E)$ to track hypothetical SFC deployment, and an SFC set $\tilde{\mathcal{R}}'$ to track the SFCs that still need to be processed. Then, the for-loop of *Lines* 2-6 try to hypothetically deploy the unprocessed vNFs in each SFC on the PDP switches that currently do not carry any superset SFCs. Specifically, if all the MATs in a vNF can be placed in PDP switch v (*i.e.*, we do not allow a vNF being deployed across multiple PDP switches to avoid QoS degradation), we deploy it hypothetically, update the network status of $\tilde{G}(\tilde{V}, \tilde{E})$ accordingly, and insert v in V^r. Fig. 5 explains the procedure. Here, we assume that the vNF consists of 4 MATs and each PDP switch only contains 4 stages. The dependencies among the MATs indicate that they will take at least 3 stages. Then, we traverse the stages of PDP Switch. Due to the resource limitation of *PDP Switch* 1 and that vNF cannot be placed across switches, we have to deploy MATs in *PDP Switch* 2.

Algorithm 1: Processing Pending SFC Requests

```
Input: Set of pending SFC requests R', SNT G(V, E).
Output: Updated set of pending SFC requests R'.
1  sort SFCs in R' first in descending order of number of vNFs
   and then in descending order of number of flows;
2  for each SFC R_i ∈ R' in sorted order do
3      calculate K shortest paths from s_i to d_i for R_i;
4      for each path from short to long do
5          if R_i can be provisioned with the path then
6              serve R_i with the path and remove it from R';
7              update network status accordingly;
8              break;
9          end
10     end
11 end
12 return R';
```

Next, we remove each SFC whose vNFs are all processed from $\tilde{\mathcal{R}}'$, and use *Lines* 7-19 to process the remaining SFCs in $\tilde{\mathcal{R}}'$. This time, we need to select the PDP switches that currently carry SFCs to recompile, and thus try to first select those serving less active flows to reduce the complexity of flow migration. Therefore, the weight ϖ_v of each in-service PDP switch v can be calculated as

$$\varpi_v = \sum_t \tilde{W}^t_v, \quad \forall v \in V^s, \tag{28}$$

where $\tilde{W}^t_v$ is the number of flows that a vNF of type t deployed on v currently carries (*Line* 8). Then, we try to empty the SFCs deployed on an in-service PDP switch and deploy SFCs in $\tilde{\mathcal{R}}'$ on it hypothetically (*Lines* 9-19). Finally, we put all the PDP switches selected to be recompiled in V^r, including both unused and in-service ones, and insert the SFCs currently using nodes in $V^r \cap V^s$ in set of pending SFC requests $\mathcal{R}'$, since reprogramming nodes in $V^r \cap V^s$ will impact these SFCs.

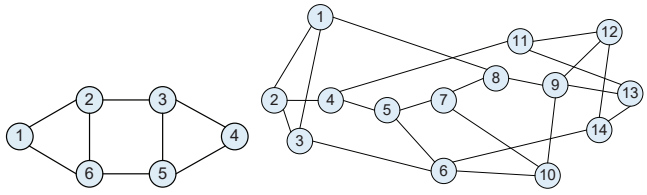
2) Generated Reconstructed Pipelines and Provision Pending SFCs: Algorithm 3 solves the second step to generate reconstructed pipelines for the selected PDP switches with fine-grained table merging, and to provision all the pending SFCs

Algorithm 2: Selecting PDP Switches to Recompile

Input: Set of pending SFC requests $\mathcal{R}'$, SNT $G(V, E)$, set of unused nodes V^a, and set of in-service nodes V^s.
Output: Updated set of pending SFC requests $\mathcal{R}'$, and set of nodes that need to be recompiled V^r.

1 $V^r = \emptyset$, $\tilde{G}(\tilde{V}, \tilde{E}) = G(V, E)$, $\tilde{\mathcal{R}}' = \mathcal{R}'$;
2 **for** *each SFC* $R_i \in \mathcal{R}'$ *in sorted order* **do**
3 **for** *each PDP switch* $v \in V^a$ *from close to far to* s_i **do**
4 hypothetically deploy c_{ij} on v and update status of $\tilde{G}(\tilde{V}, \tilde{E})$ accordingly and update $\tilde{\mathcal{R}}'$;
5 **end**
6 **end**
7 sort SFCs in $\tilde{\mathcal{R}}'$ first in descending order of unprocessed vNFs and then in descending order of number of flows;
8 sort nodes in V^s in ascending order of weights with Eq. (28) and reset their pipelines hypothetically in $\tilde{G}(\tilde{V}, \tilde{E})$;
9 **for** *each SFC* $R_i \in \tilde{\mathcal{R}}'$ *in sorted order* **do**
10 **for** *each node* $v \in V^s$ *in sorted order* **do**
11 try to deploy as many unprocessed vNFs in R_i in v hypothetically as possible;
12 update status of $\tilde{G}(\tilde{V}, \tilde{E})$ accordingly;
13 insert v in V^r if it is not already there;
14 **if** *all the vNFs in* R_i *are processed* **then**
15 **break**;
16 **end**
17 **end**
18 **end**
19 insert SFCs currently using nodes in $V^r \cap V^s$ in $\mathcal{R}'$;
20 **return** $\mathcal{R}'$ and V^r;

accordingly. *Algorithm* 3 uses two sub-procedures (*DeploySingleMAT*$(\cdot)$ and *DeploySFC*$(\cdot)$), while its main procedure is in *Lines* 22-26. Here, the basic idea is to 1) concatenate all the stages in the nodes in V^r to get a super PDP switch $\tilde{v}$ with $|V^r| \cdot \mathcal{S}$ stages (*Line* 22), 2) deploy each pending SFC R_i on $\tilde{v}$ with a sliding-window-based approach (*Lines* 23-25), and 3) partition stages in $\tilde{v}$ to $|V^r|$ parts, each of which corresponds to the reconstructed pipeline of a PDP switch, and map the pipelines to PDP switches to finally provision the pending SFCs with the smallest total hop-count (*Line* 26).



(a) 6-node topology (b) 14-node NSFNET topology

Fig. 6. Network topologies used in simulations.

DeploySFC$(\cdot)$ explains how to deploy a pending SFC in the super PDP switch (*Lines* 11-21). Specifically, for each vNF in each pending SFC in sequence, we first find the MAT in it that uses the most hardware resources (*Line* 13), then determine the sliding window of stages in $\tilde{v}$, which can be used to deploy the MAT, based on its dependencies with other MATs and the resource usage of the stages in $\tilde{v}$ (*Line* 14), and finally deploy the MAT by calling *DeploySingleMAT*$(\cdot)$ (*Line* 15). Next, after determining the deployment scheme of the MAT that uses the most hardware resources, we repeat the similar procedure to deploy each MAT in the vNF (*Lines* 16-19). *DeploySingleMAT*$(\cdot)$ deploys an MAT in a stage within the current sliding window (*Lines* 1-10). Specifically, it checks each stage within the current window, tries to merge the MAT with deployed MATs in the stage, and deploys the MAT in the stage where the smallest resource usage can be achieved.

Algorithm 3: Recompiling Pipelines for Pending SFCs

Input: Set of pending SFCs $\mathcal{R}'$, nodes to be recompiled V^r.

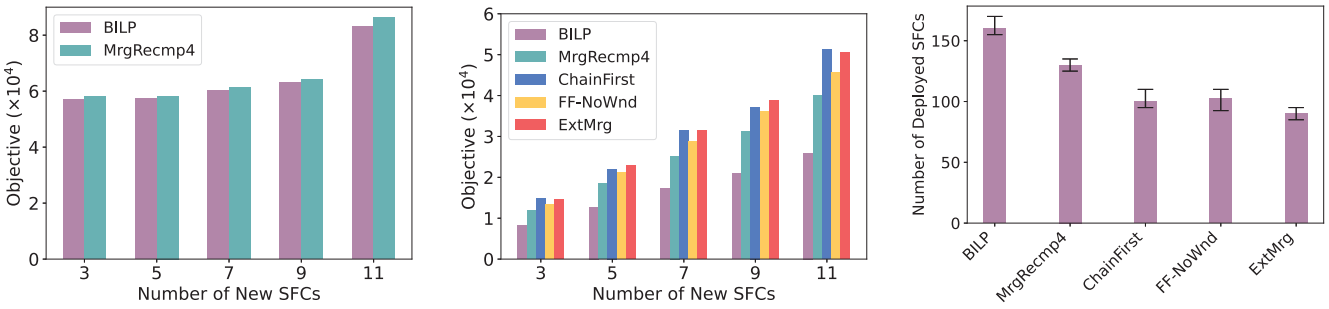
1 *DeploySingleMAT*(MAT $(m; a)$, a sliding window ω):
2 **for** *each stage* s *in sliding window* ω **do**
3 **for** *each deployed MAT* $(m'; a')$ *in* s **do**
4 **if** *MATs* $(m; a)$ *and* $(m'; a')$ *can be merged* **then**
5 calculate resource usage after merging;
6 **end**
7 **end**
8 **end**
9 select the merging scheme leading to smallest resource usage and deploy $(m; a)$ in the corresponding stage s^*;
10 **return** s^*;
11 *DeploySFC*(SFC R_i, super node $\tilde{v}$):
12 **for** *each vNF* c_{ij} *in* R_i **do**
13 find MAT $(m_{ij\tilde{l}}; a_{ij\tilde{l}})$ that uses the most SRAM/TCAM;
14 determine sliding window of stages ω in super PDP switch $\tilde{v}$ that can carry $(m_{ij\tilde{l}}; a_{ij\tilde{l}})$;
15 $s^*_{\tilde{l}} =$ *DeploySingleMAT*$((m_{ij\tilde{l}}; a_{ij\tilde{l}}), \omega)$;
16 **for** *each MAT* $(m_{ijl}; a_{ijl})$ *in* c_{ij} *from close to far to MAT* $(m_{ij\tilde{l}}; a_{ij\tilde{l}})$ **do**
17 update sliding window ω for deploying $(m_{ijl}; a_{ijl})$;
18 $s^*_l =$ *DeploySingleMAT*$((m_{ijl}; a_{ijl}), \omega)$;
19 **end**
20 **end**
21 **return** $\tilde{v}$ with updated status;
22 initialize $|V^r| \cdot \mathcal{S}$ stages for a super PDP switch $\tilde{v}$;
23 **for** *each SFC* $R_i \in \mathcal{R}'$ **do**
24 $\tilde{v} =$ *DeploySFC*$(R_i, \tilde{v})$;
25 **end**
26 partition stages in super PDP switch $\tilde{v}$ into $|V^r|$ parts, each of which contains $\mathcal{S}$ stages, and map each part to a PDP switch in V^r to minimize the total hop-count of SFCs in $\mathcal{R}'$;

IV. Performance Evaluations

This section discusses the performance evaluations of our MrgRecmp4 with numerical simulations and experiments with real-world PDP switches based on Tofino ASICs.

A. Numerical Simulations

The simulations consider PDP switches based on Tofino ASICs, and use the P4 programs of discrete stateful vNFs as inputs. We assume that MrgRecmp4 can support 10 types of stateful vNFs, which are the stateful firewall, DNS reflection attack protector, Layer-4 load balancer, stateful NAT, heavy hitter detection, SYN flood detection, DNS request analyzer, flow size monitor, and super spreader identification. According to the operation principle in Section II, MrgRecmp4 does not have restrictions on how many or which types of vNFs that it can support, and if a new type of vNFs need to be addressed,



(a) Upper-level objective (one-time)

(b) Lower-level objective (one-time)

(c) Dynamic operation (multiple-times)

Fig. 7. Results of small-scale simulations.

we only need to add in the P4 program of such a vNF and then incremental support can be achieved. Our simulations average the results from 5 independent runs to get each data point.

The simulations emulate the pipeline compilation of PDP switches based on P4 Insight [24], and consider two networks using the 6-node and NSFNET topologies in Fig. 6, respectively. The 6-node network is used for the small-scale simulations where the BILP is solvable. In addition to our BILP and heuristic with *Algorithms* 1-3, there are three benchmarks:

- *First-fit without Window (FF-NoWnd)*: Instead of using the sliding-window-based scheme in *Algorithm* 3, it selects the first stage that is available to deploy each MAT and applies fine-grained merging in the stage if possible.
- *Exact Merging Only (ExtMrg)*: Fine-grained merging is not used during SFC recompilation, and only identical MATs can be merged (*i.e.*, the exact merging scheme addressed in SPEED [11] and pSFC [12]).
- *Chaining First (ChainFirst)*: Instead of using *Algorithm* 3, it first merges all the pending SFCs to superset SFCs according to resource constraints of PDP switches, and then applies fine-grained merging to adjacent MATs.

We first discuss the small-scale simulations with the 6-node topology. Each simulation randomly selects PDP switches (with an average of 3) as in-service nodes, deploys a superset SFC with $[5, 8]$ vNFs in random types on each in-service node, and generates $[3, 11]$ new SFCs, each of which contains $[3, 8]$ vNFs in random types. The numbers of flow entries allocated for each existing superset SFC and each new SFC are set within $[10000, 40000]$ and $[5000, 15000]$, respectively. The other parameters are set according to the practical values used in [11, 12]. For Eq. (6) (low-level objective), we set $\alpha = 0.5$.

TABLE V
RUNNING TIME OF SMALL-SCALE SIMULATIONS (SECONDS)

Number of New SFCs	BILP	MrgRecmp4	FF-NoWnd	ExtMrg	Chain First
3	352.2	0.039	0.040	0.028	0.029
5	504.8	0.065	0.057	0.050	0.052
7	1209	0.076	0.088	0.060	0.062
9	1221	0.103	0.111	0.092	0.095
11	1638	0.126	0.143	0.131	0.135

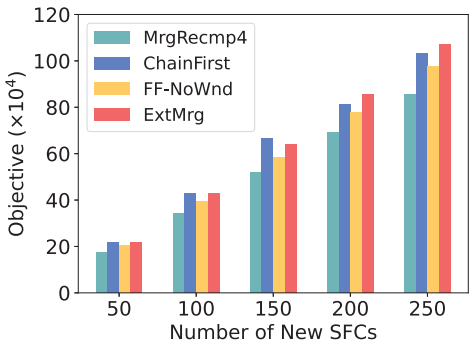
Figs. 7(a) and 7(b) show the results of one-time operation, where each simulation only considers a batch of new SFCs for SFC recompilation. Fig. 7(a) indicates that the heuristic (MrgRecmp4) approximates the solutions from the BILP well when solving PDP switch selection in the upper-level model. It is interesting to observe that when the number of SFCs is 11, the objective increases significantly. This is because due to the resource limitations of each switch pipeline, more PDP switches need to be recompiled in this case.

The objectives of the lower-level model, which are obtained based on the used resources in PDP switches and total hop-count of provisioned SFCs after SFC recompilation, are plotted in Fig. 7(b). Our heuristic outperforms all the benchmarks and its gaps to the solutions from the BILP are the smallest. Table V lists the running time of the algorithms, which indicates that the heuristic runs much faster than solving the BILP directly and its running time is similar as that of the benchmarks. Fig. 7(c) shows the results of dynamic operations, where each simulation deploys SFCs in batches (an SFC recompilation for each batch) from an empty network, and get the maximum number of SFCs that each algorithm can eventually put in the network. Each batch contains 5 new SFCs whose setting is the same as that for new SFCs in one-time operation. The BILP accommodates the most SFCs, and our heuristic still outperforms all the benchmarks, verifying its effectiveness on achieving efficient SFC recompilation.

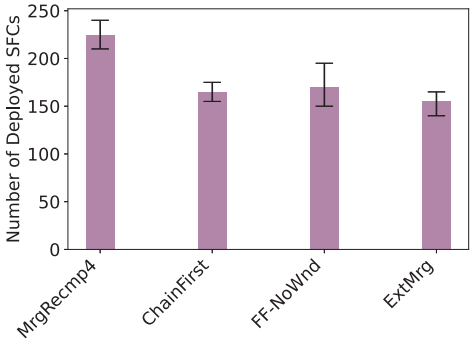
Fig. 8 shows the results of large-scale simulations using NSFNET, where the parameter settings stay the same. The lower-level objectives in Fig. 8(a) still indicate that our heuristic uses the least resources in PDP switches and serves SFCs with the smallest total hop-count, after SFC recompilation, while Fig. 8(b) suggests that our proposal accommodates much more SFCs than the benchmarks in dynamic operations (with an improvement of $[32.3\%, 45.1\%]$).

B. Experimental Evaluations

Finally, we evaluate the performance of our proposed MrgRecmp4 in a real-world PDP switch based on Tofino ASICs. Specifically, we generate 10 batches of random SFCs, where each batch contains 5 SFCs, each of which includes $[2, 6]$ vNFs in random types, use the algorithms to obtain the arrangement



(a) Lower-level objective (one-time)



(b) Dynamic operation

Fig. 8. Results of large-scale simulations.

of MATs in each superset SFC that combines the SFCs in each batch, and implement the SFC compilation results as P4 programs. Similar as the setting of simulations, the flow entries of each SFC are between $[5000, 15000]$. Then, we use P4 Insight to check the resource usages after compiling the P4 programs in the PDP switch. Table VI shows the results on average resource usages, and compared with the three benchmarks, MrgRecmp4 (with our heuristic) consumes same or less resources in all the types. It is interesting to notice that MrgRecmp4 can even use fewer stages than the BILP. This is because the BILP only tries to find solutions with the optimal resource saving, while our sliding-window-based heuristic for MrgRecmp4 can optimize the MAT placement on used stages to increase the probability of table merging. Note that, as the existing studies in [11, 12, 25] have already verified that SFC compilation with table merging will not affect the processing latency and throughput of SFCs noticeably, we do not show the related experimental results here to save space.

V. Related Work

Stateful vNF deployment on PDP switches and related resource optimization. The works in [18, 19, 26–30] studied the offloading of one or a few specific vNFs to PDP switches, and thus their solutions are not generic enough. On the other hand, the proposals of OPP [13], FlowBlaze [15], and SDPA [31] designed general state machine paradigms for offloading stateful vNFs to PDP switches, but they did not address how to optimize the deployment of a stateful SFC with multiple vNFs. The studies in [32–35] tried to merge P4 programs, but their merging strategies were not fine-grained enough for stateful vNFs. P4SC [36] addressed the merging problem at the vNF level, while SPEED [11], pSFC [12], and Dapper [37] went one step further and considered table merging at the MAT level to optimize the offloading of SFCs to PDP switches. However, they only tried to merge identical MATs, and their ILP models handled table merging and SFC deployment separately, which might only provide suboptimal results.

TABLE VI
Hardware Resource Usages of SFC Recompilation

Resource	BILP	MrgRecmp4	FF-NoWnd	ExtMrg	Chain First
SRAM	23.3%	24.1%	27.2%	30.8%	29.8%
Exact Match Input Xbar	9.4%	8.4%	9.4%	10.7%	9.3%
Gateway	8.3%	7.8%	8.3%	8.3%	7.8%
Hash Bit	13.1%	11.2%	13.2%	15.5%	12.7%
TCAM	2.7%	2.7%	2.7%	2.7%	2.7%
Stash	6.8%	5.2%	7.3%	8.9%	6.8%
Logical Table ID	19.3%	18.2%	19.3%	19.8%	18.2%
Stages	14	11	11	11	11

Dynamic SFC deployment on PDP switches. P4SC [36], p4NFV [38] and DPPx [39] have developed the system architectures and interfaces to enable dynamic SFC deployment using P4. The studies of Flexmash [40], P4SFC [23] and FlexNF [41] focused on how to independently serve SFCs of multiple types and QoS demands with separate PDP pipelines. ILP models were formulated in [20, 42] to optimize the vNF deployment and traffic rerouting during reprogramming PDP switches. Sirius [43] was one of the latest studies on the topic, which designed a system to automate SFC composition on PDP switches and leveraged packet recirculation to realize different SFCs. However, all these aforementioned studies did not consider the fine-grained table merging addressed in this work to optimize the SFC recompilation on PDP switches.

VI. Conclusion

We proposed a stateful SFC recompilation system, namely, MrgRecmp4, which considers the existing and new SFCs in a PDP network to optimize the P4 program recompilation for deploying new SFCs as reconstructed pipelines. A BILP model was first formulated for the network-wide optimization of MrgRecmp4, and then a sliding-window-based heuristic was proposed to solve the problem quickly. Both simulations and real-world experiments confirmed that MrgRecmp4 outperforms existing benchmarks in terms of hardware resource usages in PDP switches and performance of SFC provisioning.

Acknowledgments

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

References

[1] 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.

[2] L. Gong *et al.*, "Efficient resource allocation for all-optical multicasting over spectrum-sliced elastic optical networks," *J. Opt. Commun. Netw.*, vol. 5, pp. 836–847, Aug. 2013.

[3] Y. Yin *et al.*, "Spectral and spatial 2D fragmentation-aware routing and spectrum assignment algorithms in elastic optical networks," *J. Opt. Commun. Netw.*, vol. 5, pp. A100–A106, Oct. 2013.

[4] L. Gong and Z. Zhu, "Virtual optical network embedding (VONE) over elastic optical networks," *J. Lightw. Technol.*, vol. 32, pp. 450–460, Feb. 2014.

[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] W. Lu, Z. Zhu, and B. Mukherjee, "On hybrid IR and AR service provisioning in elastic optical networks," *J. Lightw. Technol.*, vol. 33, pp. 4659–4669, Nov. 2015.

[7] P. Lu and Z. Zhu, "Data-oriented task scheduling in fixed- and flexible-grid multilayer inter-DC optical networks: A comparison study," *J. Lightw. Technol.*, vol. 35, pp. 5335–5346, Dec. 2017.

[8] J. Liu *et al.*, "On dynamic service function chain deployment and readjustment," *IEEE Trans. Netw. Serv. Manag.*, vol. 14, pp. 543–553, Sept. 2017.

[9] Tofino switch. [Online]. Available: https://www.barefootnetworks.com/products/brief-tofino/.

[10] B. Niu *et al.*, "Visualize your IP-over-optical network in realtime: A P4-based flexible multilayer in-band network telemetry (ML-INT) system," *IEEE Access*, vol. 7, pp. 82 413–82 423, 2019.

[11] 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.

[12] 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.*, pp. 1–13, Feb. 2023.

[13] 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. [Online]. Available: https://arxiv.org/abs/1605.01977.

[14] A. Sivaraman *et al.*, "Packet transactions: High-level programming for line-rate switches," in *Proc. of ACM SIGCOMM 2016*, pp. 15–28, Aug. 2016.

[15] S. Pontarelli *et al.*, "Flowblaze: Stateful packet processing in hardware," in *Proc. of NSDI 2019*, pp. 531–547, Feb. 2019.

[16] N. Gebara *et al.*, "Challenging the stateless quo of programmable switches," in *Proc. of HotNets 2020*, pp. 153–159, Nov. 2020.

[17] V. Shrivastav, "Stateful multi-pipelined programmable switches," in *Proc. of ACM SIGCOMM 2022*, pp. 663–676, Aug. 2022.

[18] L. Teng, C. Hung, and C. Wen, "P4SF: A high-performance stateful firewall on commodity P4-programmable switch," in *Proc. of NOMS 2022*, pp. 1–5, Apr. 2022.

[19] 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.

[20] D. Moro, G. Verticale, and A. Capone, "Network function decomposition and offloading on heterogeneous networks with programmable data planes," *IEEE Open J. Com. Soc.*, vol. 2, pp. 1874–1885, Aug. 2021.

[21] 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.*, vol. 21, pp. 6658–6669, Dec. 2024.

[22] Z. Wei, T. Li, Z. Xu, and Z. Zhu, "Optimizing stateful service function chaining on PDP switches with table merging," in *Proc. of GLOBECOM 2024*, pp. 1–6, Dec. 2024.

[23] J. Ma, S. Xie, and J. Zhao, "Flexible offloading of service function chains to programmable switches," *IEEE Trans. Serv. Comput.*, vol. 16, pp. 1198–1211, Mar. 2022.

[24] Intel p4 insight. [Online]. Available: https://www.intel.cn/content/www/cn/zh/products/details/network-io/intelligent-fabric-processors/p4-insight.html.

[25] Z. Ma *et al.*, "SFCache: Hybrid NF synthesization in runtime with rule-caching in programmable switches," *IEEE Trans. Netw. Serv. Manag.*, vol. 21, pp. 4613–4624, Aug. 2024.

[26] R. Miao *et al.*, "Silkroad: Making stateful Layer-4 load balancing fast and cheap using switching ASICs," in *Proc. of ACM SIGCOMM 2017*, pp. 15–28, Aug. 2017.

[27] C. Zeng *et al.*, "Tiara: A scalable and efficient hardware acceleration architecture for stateful Layer-4 load balancing," in *Proc. of NSDI 22*, pp. 1345–1358, Apr. 2022.

[28] M. Scazzariello *et al.*, "A High-Speed stateful packet processing approach for Tbps programmable switches," in *Proc. of NSDI 23*, pp. 1237–1255, Apr. 2023.

[29] M. Chiesa *et al.*, "Fast ReRoute on programmable switches," *IEEE/ACM Trans. Netw.*, vol. 29, pp. 637–650, Apr. 2021.

[30] V. Sivaraman *et al.*, "Heavy-hitter detection entirely in the data plane," in *Proc. of SOSR 2017*, pp. 165–176, Apr. 2017.

[31] C. Sun *et al.*, "Toward a stateful data plane in software-defined networking," *IEEE/ACM Trans. Netw.*, vol. 25, pp. 3294–3308, Dec. 2017.

[32] D. Hancock and J. Van der Merwe, "Hyper4: Using P4 to virtualize the programmable data plane," in *Proc. of CoNEXT 2016*, pp. 35–49, Dec. 2016.

[33] C. Zhang *et al.*, "Hyperv: A high performance hypervisor for virtualization of the programmable data plane," in *Proc. of ICCCN 2017*, pp. 1–9, Jul. 2017.

[34] 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.

[35] H. Soni, T. Turletti, and W. Dabbous, "P4bricks: Enabling multiprocessing using linker-based network data plane architecture," Feb. 2018. [Online]. Available: https://inria.hal.science/hal-01632431

[36] 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.

[37] X. Zhang *et al.*, "Dapper: Deploying service function chains in the programmable data plane via deep reinforcement learning," *IEEE Trans. Serv. Comput.*, pp. 2532–2544, Jan. 2023.

[38] M. He *et al.*, "P4NFV: An NFV architecture with flexible data plane reconfiguration," in *Proc. of CNSM 2018*, pp. 90–98, Nov. 2018.

[39] T. Osinski, H. Tarasiuk, L. Rajewski, and E. Kowalczyk, "DPPx: A P4-based data plane programmability and exposure framework to enhance NFV services," in *Proc. of NetSoft 2019*, pp. 296–300, Jun. 2019.

[40] Y. Zhou *et al.*, "Flexmesh: Flexibly chaining network functions on programmable data planes at runtime," in *Proc. of Networking 2020*, pp. 73–81, Jun. 2020.

[41] J. Xiao *et al.*, "FlexNF: Flexible network function orchestration for scalable on-path service chain serving," *IEEE/ACM Trans. Netw.*, pp. 1–16, Nov. 2023.

[42] L. Jose, L. Yan, G. Varghese, and N. McKeown, "Compiling packet programs to reconfigurable switches," in *Proc. of NSDI 15*, pp. 103–115, May 2015.

[43] J. Gao *et al.*, "Sirius: Composing network function chains into P4-capable edge gateways," in *Proc. of NSDI 2024*, pp. 477–490, Apr. 2024.