IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received December 6, 2020, accepted December 24, 2020, date of publication December 31, 2020,
date of current version January 12, 2021.

Digital Object Identifier 10.1109/ACCESS.2020.3048421

Flow-Aware Service Function Embedding Algorithm in Programmable Data Plane

JAEWOOK LEE[1], HANEUL KO[2], (Member, IEEE), HOCHAN LEE[1], (Graduate Student Member, IEEE), AND SANGHEON PACK[1], (Senior Member, IEEE)

[1]School of Electrical Engineering, Korea University, Seoul 02841, South Korea
[2]Department of Computer Convergence Software, Korea University, Sejong 30019, South Korea

Corresponding author: Sangheon Pack (shpack@korea.ac.kr)

This work was supported in part by the National Research Foundation (NRF) of Korea through the Korean Government under Grant 2020R1A2C3006786, and in part by the Ministry of Science and ICT (MSIT), Korea, through the Information Technology Research Center (ITRC) support program supervised by the Institute for Information and Communications Technology Planning and Evaluation (IITP) under Grant IITP-2020-2017-0-01633.

ABSTRACT Service function chaining (SFC) is an indispensable technique for Internet service providers to efficiently manage their networks. However, SFC poses requirements of additional processing time of service functions (SFs) and increased routing time owing to detoured paths. In this paper, we introduce the use of a programmable data plane (PDP) to reduce the additional processing and routing times in SFC. We first classify the existing PDP-empowered SFC schemes and analyze their pros and cons. An optimization problem, to find the optimal SF embedding strategy minimizing the SFC completion time while efficiently utilizing the PDP switch resources, is formulated and a flow-aware SF embedding (FASE) algorithm that complementarily combines the redundant SF and re-circulation approaches is devised. FASE is implemented over a commercial PDP switch and experimental results demonstrate that FASE can reduce the SFC completion time by up to 33% compared with conventional approaches while utilizing the switch resources efficiently.

INDEX TERMS Service function chaining, programmable data plane, in-network computing.

I. INTRODUCTION

Service function chaining (SFC) [1] is a prevalent technique for connecting a sequence of service functions (SFs) (e.g., load balancer (LB), firewall (FW), and deep packet inspection (DPI)) and for steering packets to different SFs [2], [3]. In SFC, softwarized SFs (i.e., virtualized network function (VNF)) can be flexibly provisioned on general-purpose servers through network function virtualization (NFV), and software-defined networking (SDN) efficiently constructs a routing path according to the provisioned locations of the softwarized SFs [4]. Therefore, SDN/NFV-based SFC is perceived as an indispensable technique for network management and operation of Internet service providers (ISPs).

However, despite the flexible provision and efficient path management, detoured paths in SFC are inevitable because softwarized SFs should be provisioned on off-path servers. Such detoured paths prolong the SFC completion time and can be an obstacle for supporting latency-sensitive services. Moreover, the processing performance of softwarized SFs is one or two orders of magnitude lower than that of hardwarized SFs [5] owing to the huge performance gap between virtualized processors (of softwarized SFs) and dedicated processors (of hardwarized SFs). Consequently, it is critical to accelerate the processing of softwarized SFs and to reduce the SFC completion time and to realize the low-latency service.

Recently, programmable data planes (PDPs) have attracted considerable attention because of their easily reconfigurable data planes and line-rate packet processing performance. Moreover, PDP has also been instrumental in enabling a new computing paradigm in which parts of an application's logic run within the network core (i.e., in-network computing) [15]. This salient feature can mitigate the above-mentioned limitations of softwarized SFs owing to its high performance and reconfiguration. For example, several types of SFs, such as layer-4 LBs [5] and network cache [7], can be implemented on PDP switches and the implemented SFs shows the higher performance than that of softwarized SFs. In addition, several approaches for the implementation of SFC in the PDP switch

The associate editor coordinating the review of this manuscript and approving it for publication was Guangjie Han.

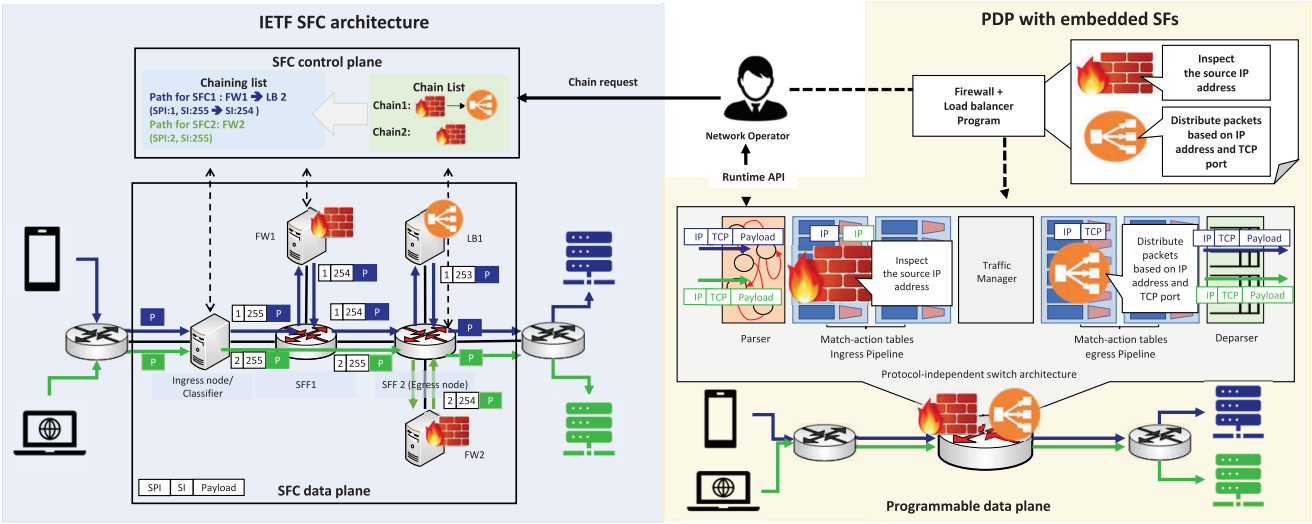


FIGURE 1. The IETF SFC and PDP architectures.

that can satisfy the requested SF processing orders have been studied, which can be categorized into: 1) redundant SF approach [9] and 2) re-circulation approach [3], [10]. Chen *et al.* [9] allowed redundant SF embedding to a PDP switch, and thus, the SF processing order of the requested SFCs could be easily satisfied at the line rate. However, as more than one SF is redundantly embedded, the resources of the PDP switch can be abused and only a few SFCs can be supported at a time. In contrast, Lee *et al.* [10] and Wu *et al.* [3] employed a packet re-circulation function to satisfy the requested SF processing order. Specifically, if the embedded SF order was different from the requested SF processing order, the packet was recirculated from the egress port to the ingress port. Thus, the requested SF processing order could be satisfied; however, the line-rate performance could not be guaranteed owing to the re-circulation. In summary, the redundant SF and re-circulation approaches have contrasting advantages and disadvantages.

By achieving a balance between these approaches, we attempt to find the optimal SF embedding strategy that minimizes the SFC completion time while utilizing the PDP switch resource efficiently. A flow-aware SF embedding (FASE) algorithm that complementary combines the redundant SF and re-circulation approaches is devised to solve the formulated optimization problem with low-complexity. However, not all requested SFCs can be supported without re-circulation owing to the limited resources of the PDP switch. Thus, in FASE, SFC flows with higher incoming rates are first processed by the redundantly embedded SFs. Subsequently, SFC flows with lower incoming rates are processed by employing re-circulation. Consequently, SFC flows with higher incoming rates can be supported without re-circulation, which contributes to the reduction of the overall SFC completion time. FASE has been implemented on a commercial PDP switch (i.e., a Tofino switch supporting a processing performance of 3.2 Tbps with 32 ports).[1]

[1]The implemented code of FASE and its test code on the Tofino switch can be found at `https://github.com/jaewook2/FASE`.

Experimental results demonstrate that FASE can reduce the SFC completion time by up to 33% compared with the redundant SF and re-circulation approaches while utilizing the given switch resource efficiently.

The remainder of this paper is organized as follows. In Section II, we provide the background knowledge on SFC and PDP. After that, we review the conventional PDP-empowered SFC approaches in Section III and propose the FASE algorithm in Section IV. We then describe the experimental results in Section V and present open research topics in Section VI. Finally, Section VII concludes this paper.

II. BACKGROUND

In this section, we first describe the IETF SFC architecture [1] and introduce a representative PDP architecture (i.e., protocol-independent switch architecture (PISA)).

A. SERVICE FUNCTION CHAINING

Figure 1 compares the IETF SFC architecture and PDP with embedded SFs. As shown in Figure 1, the IETF SFC architecture [4] consists of 1) an SFC control plane and 2) an SFC data plane. The SFC data plane has four SFC entities: an ingress node (or classifier), service function forwarder (SFF), SF, and egress node. The ingress node performs classification to apply an appropriate SFC for an incoming packet according to the classification rules. Then, the incoming packet is encapsulated by adding a network service header (NSH) [2], which includes two identifiers: 1) a service path identifier (SPI) and 2) a service index (SI). The SPI and SI represent the path ID of an instantiated SFC and the number of SFs that processed the packets, respectively. Generally, the initial value of the SI is set to 255, and the value is decremented by one when the packet is processed by an SF. According to the SPI and SI, the SFF creates a forwarding table and forwards the incoming packets to satisfy the SF processing order. For instance, in Figure 1, the SF processing orders of SFC1 and SFC2 are defined to $[FW \rightarrow LB]$ and $[FW]$, respectively. The SPIs of SFC1 and SFC2 are assumed as 1 and 2, respectively.

A packet of SFC1 is classified as an NSH of SPI= 1 and SI= 255 at the ingress node. The packet is forwarded to FW1 by SFF1, and the SI is decremented by one after processing at FW1. Subsequently, SFF2 forwards the packet to LB1, and the SI of the packet will be set to 253 after LB1 processes the packet. In contrast, a packet of SFC2 is forwarded to FW2 by SFF2. Finally, the egress node decapsulates the received packets by removing the NSH if they are processed by all the SFs defined in SFC.

Moreover, the control plane manages the data plane and constructs paths for the requested SFCs. To construct a path, the control plane selects the SFs that are involved in the path. To this end, it determines the SPI to deliver the packets according to the selected SFs and constructs the paths by building the packet-forwarding table in the SFFs. As shown in Figure 1, in the conventional SFC framework, the packets need to be detoured to be processed by the SFs provisioned on off-path servers, which incurs an inevitable delay in SFC.

B. PROGRAMMABLE DATA PLANE

The aforementioned detoured path can be avoided by embedding SFs on on-path switches. Moreover, the processing time of SFs can be significantly reduced because the switch can provide line-rate processing performance. Owing to its programming capability for packet processing operations through high-level and domain-specific languages (e.g., P4 [11]), SFs can be easily embedded in the PDP switch with high performance. Moreover, PDP is considered a promising approach to enable the in-network computing [6]. A representative PDP switch model (i.e., PISA) that generalizes reconfigurable match-action tables (RMT) [12], has several programmable components (i.e., parser, ingress/egress pipeline, and de-parser). The header fields that the user wants to process can be extracted and assembled by programming the parser and de-parser. In contrast, the extracted header field can be processed using programmed match-action tables in the pipelines.

Several SFs (e.g., LB [5], cache [7], and FW [8]) have been implemented based on the PDP switch. The right part in Figure 1 shows a PDP switch in which an LB and FW are embedded. Before embedding the SFs, the network operator writes programs that define how programmable components process the packets to provide the LB and FW functions. For instance, to define an FW, a parser is programmed to extract the header fields that are inspected (e.g., IP address), and match-action tables are defined to inspect the extracted header fields and the appropriate actions (e.g., drop or pass). In contrast, a de-parser is programmed to assemble the extracted headers. Based on the written program, the LB and FW are embedded, and the packets can be processed by the embedded LB and FW while passing through the PDP switch. Compared with the conventional SFC, as shown in Figure 1, PDP-based SFs can significantly reduce the SFC completion time because of the high performance of the PDP switch and the absence of the detoured path.

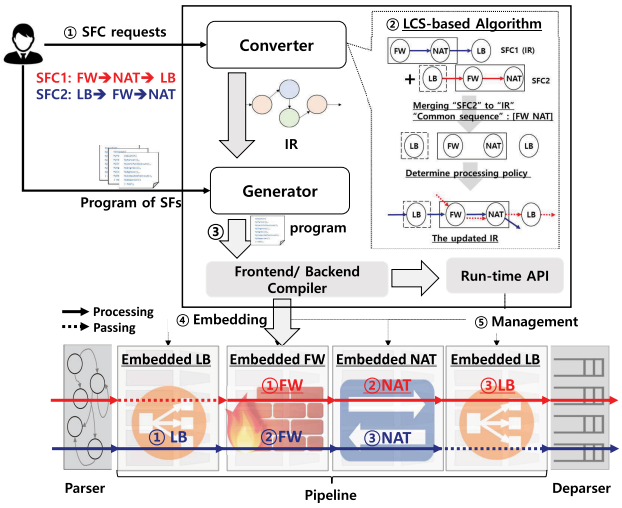


FIGURE 2. P4SC architecture.

III. PDP-EMPOWERED SFC APPROACHES

In this section, we analyze recent PDP-empowered SFC approaches, which can be classified into the redundant SF approach [9] and re-circulation approach [3], [10].

A. REDUNDANT SF APPROACH

In the redundant SF approach, SF tables are redundantly embedded in the PDP switch. The processing order of the requested SFCs can be easily satisfied at the line rate because of the redundant embedding. Chen *et al.* [9] proposed a P4 service chaining (P4SC) framework that allows redundant SF embedding. Figure 2 shows the P4SC framework consisting of control and data planes. The control plane is responsible for determining the SFs that will be redundantly embedded and mapping the requested SFC to the embedded SFs. To this end, P4SC employs the longest common sub-sequence (LCS) based algorithm. In the proposed algorithm, intermediate representation (IR) is generated as an SF sequence that represents the embedding order and mapping result. To generate IR, the LCS-based algorithm first extracts the common SF processing order in the SFC requests and accordingly determines the embedding order. At this step, the SFs that do not exist in the common processing order can be redundantly embedded to satisfy the processing order. After determining the embedding order, the mapping result is determined by the processing order of the SFC requests.

The overall procedure for embedding SFs in P4SC is depicted in Figure 2. First, the control plane receives SFC requests (i.e., SFC1 and SFC2) from the network operator (step 1). Then, the converter merges the SFC requests and generates IR (step 2). In this step, the IR is initially set to SFC1, that is, the IR is represented as $[FW \rightarrow NAT \rightarrow LB]$. To insert SFC2 (i.e., $[LB \rightarrow FW \rightarrow NAT]$) to the IR, the algorithm determines the longest common processing order as $[FW \rightarrow NAT]$. Based on the common processing order, the first SF (i.e., LB) of SFC2 is inserted at the front of the IR, which is then updated as $[LB \rightarrow FW \rightarrow NAT \rightarrow LB]$. After determining the embedding order, the embedded

SFs are mapped to the SFC requests. Thus, the former and the latter LBs are mapped to SFC1 and SFC2, respectively. In contrast, *FW* and *NAT* are mapped to both SFC1 and SFC2. Subsequently, the generator embeds the SFs according to the determined embedding order through the compiler (steps 3-4). Using the run-time API, the control plane manages the packet processing according to the mapped SFs (step 5).

P4SC can significantly reduce the SFC completion time compared with the conventional software-based SFC, as the packets can be processed according to the requested processing orders while passing through the PDP switch [9]. However, inefficient resource usage of the PDP switch cannot be avoided because of the redundant SF embedding. In addition, as all the packets should be processed by the embedding order, the accommodation of a new SFC request cannot be guaranteed even if the SFs in the requested SFC have already been embedded. For instance, we assume that SFC3 having $[NAT \rightarrow FW]$ is requested at P4SC in Figure 2. P4SC cannot support SFC3 although NAT and FW have already been embedded because none of the embedding orders can be matched to $[NAT \rightarrow FW]$. In addition, if all the resources of the PDP switch are fully used, no more SFs can be embedded to support SFC3. Therefore, supporting diverse SFC requests and improving the resource usage of the PDP switch are the remaining challenges of the redundant SF approach.

B. RE-CIRCULATION APPROACH

Most switch ASICs support the packet re-circulation function to repeat the ingress processing on a packet after completing the egress processing [11]. Several studies have used the re-circulation function [3], [10]. As their ideas are almost the same, we describe the re-circulation approach of the Dejavu framework proposed in [3].

Figure 3 shows the Dejavu framework with a single pipeline PDP switch. The main role of the control plane is to determine the embedding order and the SFCs that need re-circulation to satisfy the processing order. The overall procedure is described as follows. First, the control plane receives SFC requests (i.e., SFC1 and SFC2) from the network operator and maintains the incoming rates of each requested SFC (step 1). Then, it determines the embedding order as $[LB \rightarrow FW \rightarrow NAT]$ by considering the incoming rates. In this case, as SFC2 has a higher incoming rate than SFC1, the control plane prefers an embedding order that can support SFC2 without any re-circulation. However, such an embedding order does not match with the SF processing order of SFC1; therefore, re-circulations need to be employed to support SFC1 (step 2). Subsequently, the control plane embeds SFs to the data plane according to the embedding order and creates the run-time API to manage the embedded SFs (step 3). Finally, it manages the embedded SFs and the re-circulation through the run-time API (step 4).

In the re-circulation approach, arbitrary SF processing orders can be supported if the requested SFs can be embedded. However, as re-circulation causes a prolonged SFC completion time, an embedding order with a lower number of re-circulations should be determined. For instance, as SFC1 and SFC2 in Figure 3 have reverse processing orders, re-circulation is inevitable for either SFC1 or SFC2. In this case, as SFC2 has a higher incoming rate than SFC1, the overall SFC completion time can be further reduced when the packets of SFC2 are processed without re-circulation. To summarize, an embedding order that minimizes the SFC completion time needs to be determined, and the under-utilization of PDF switch resources should be avoided.

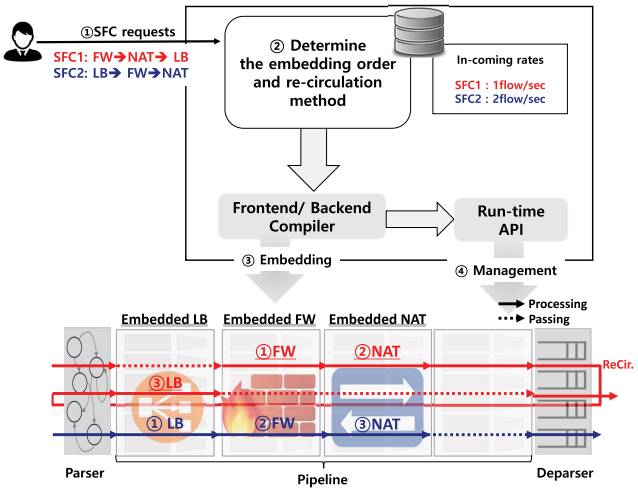


FIGURE 3. Dejavu architecture.

IV. FLOW-AWARE SERVICE FUNCTION EMBEDDING ALGORITHM

In this section, we propose the FASE algorithm to minimize the SFC completion time while providing efficient resource usage. We first present the underlying system architecture and formulate an optimization problem to find the optimal embedding strategy. Subsequently, we provide the details and an illustrative example of FASE.

A. ARCHITECTURE OF FASE

Figure 4 shows the architecture of FASE consisting of a control plane and a data plane. In the control plane, the SFC manager (SFCM) and PDP manager (PDPM) collaboratively operate to support all the requested SFCs. In detail, the SFCM maintains the incoming rates of the requested SFCs and requests appropriate SF embedding from the PDPM. In addition, it manages the embedded SFs to process packets according to the SF processing order. To manage the embedded SFs, the SFCM determines identifiers (i.e., SPI and SI) for SFCs and inserts the determined identifiers as the match key in the embedded SF tables. Moreover, it manages the re-circulation operation for each SFC by interacting with the SFF table in the PDP switch via the run-time API. On the other hand, the PDPM is responsible for embedding the SFs and mapping the requested SFCs to the embedded SFs. To this end, it solves an optimization problem using

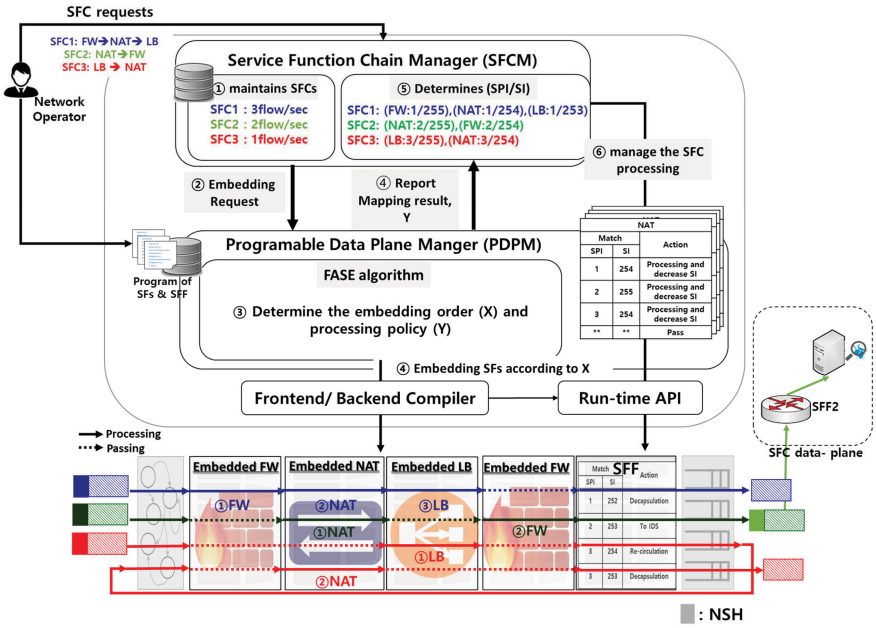


FIGURE 4. FASE architecture.

the FASE algorithm.[2] Then, it embeds the SFs according to the determined embedding order and notifies the determined mapping result to the SFCM.

B. DESCRIPTION OF FASE

In FASE, the SFC completion time is considerably affected by the number of re-circulations, which is determined by the embedding order of SFs and the mapping result of SFC requests. Therefore, we formulate an optimization problem on the embedding order and the mapping result to minimize the number of re-circulations while satisfying the SF processing order. Specially, we define the objective function as

$$\min \sum_{c=1}^{C} I_c \sum_{k=2}^{f_c} \sum_{n=1}^{N} \sum_{n'=n}^{N} y_{k-1,c,n'} y_{k,c,n} \tag{1}$$

where C and I_c represent the number of required SFCs and the incoming rate of SFC c, respectively. On the other hand, f_c and N denote the number of SFs in SFC c and the number of stages in the PDP switch, respectively.[3] In contrast, $y_{k,c,n}$ is a binary decision variable to indicate whether the kth SF in SFC c is mapped to the embedded SF in the nth stage. Thus, $\sum_{n=1}^{N} \sum_{n'=n}^{N} y_{k-1,c,n'} y_{k,c,n}$ denotes whether a re-circulation is required to process the kth SF in SFC c for the mapping results (i.e., $y_{k,c,n}$ and $y_{k-1,c,n'}$).

Because of differently requested processing orders and the limited resource of a PDP switch, we need to consider several constraints on the embedding order and mapping results. First, since all SF types in the requested SFCs should be embedded at least once, we define the corresponding constraint as

$$\sum_{n=1}^{N} x_{s,n} \geq 1, \ \forall s \tag{2}$$

where $x_{s,n}$ is a binary decision variable to indicate whether the nth stage is the first one where the SF type s is embedded. Meanwhile, each stage cannot be simultaneously used by more than one SF and thus the corresponding constraint is denoted by

$$\sum_{s=1}^{S} a_{s,n} \leq 1, \ \forall n \tag{3}$$

where $a_{s,n}$ is an auxiliary binary variable to represent whether SF s is embedded at the nth stage.

Since each SF requires a different number of stages, SF should be consecutively embedded over the required number of stages. The corresponding constraints are defined as

$$\sum_{n=1}^{N} a_{s,n} = r_s \sum_{n=1}^{N} x_{s,n}, \ \forall s, \tag{4}$$

and

$$x_{s,n} \leq a_{s,n+1} \leq \ldots \leq a_{s,n+r_s-1}, \ \forall s, n \leq N - r_s + 1 \tag{5}$$

where r_s is the required number of stages to embed SF s. On the other hand, due to the limited stages of a PDP switch, the number of used stages to embed SFs cannot be bigger than the total number of available stages, and thus we have the corresponding constraint of

$$\sum_{s=1}^{S} \sum_{n=1}^{N} a_{s,n} \leq N. \tag{6}$$

[2]The details of the optimization problem and FASE algorithm will be elaborated in the next sub-section.

[3]The stage is the resource of the PDP switch for embedding the match-action table.

Finally, each SF in the requested SFCs should be mapped to one of the stages where the corresponding SF type is embedded and therefore the corresponding constraints are represented by

$$y_{k,c,n} \leq x_{s,n}, \ \forall s = t_{k,c}, k, c, n, \tag{7}$$

and

$$\sum_{n=1}^{N} y_{k,c,n} = 1, \ \forall k, c \tag{8}$$

where $t_{k,c}$ represents the type of the kth SF in SFC c.

Since the formulated optimization problem is an integer non-linear programming (INLP) with high complexity, we devise the FASE algorithm to find out a practical solution to the formulated problem. The detailed FASE algorithm can be described as follows. The FASE algorithm first determines the embedding order and mapping result without any redundant SF and re-circulation to minimize the SFC completion time under the available resources. At this step, all the SFC requests are sorted according to their incoming rates. Subsequently, the SFs in the sorted SFCs are included in the embedding order, and the SFCs are first mapped to the included SFs. Note that an SF can be included only when it does not exist in the embedding order. As all the SFs are embedded at once, all the requested SFCs can be supported. However, if several SFs are redundantly embedded, the SFC completion time can be further reduced. Thus, if some resources remain, several SFs are redundantly embedded in the FASE. To reduce the SFC completion time, it is intuitive to allow redundant SFs for the SFC requests with high incoming rates. Therefore, after sorting all the SFC requests according to their incoming rates, redundant SFs are embedded one-by-one until no resources remain in the PDP switch. When no more SFs can be embedded owing to limited resources, the embedding order is determined. In contrast, if not all SFs in the requested SFCs are not mapped to the embedded SFs, the remaining SFCs are supported by re-circulation.

Figure 4 shows an example of FASE with three SFCs: 1) SFC1 of $[FW \rightarrow NAT \rightarrow LB]$, 2) SFC2 of $[NAT \rightarrow FW]$, and 3) SFC3 of $[LB \rightarrow NAT]$. SFC1 and SFC3 have the highest and lowest incoming rates, respectively. As SFC1 has the highest incoming rate, the embedding order is first determined as $[FW \rightarrow NAT \rightarrow LB]$, and the required SFs of SFC1 are mapped to the embedded FW, NAT, and LB. At this step, all types of SFs are now embedded in the PDP switch, and thus FASE needs to determine the SFs that will be redundantly embedded to the remaining resources. In Figure 4, as SFC2 has a higher incoming rate than SFC3 and it requires $NAT \rightarrow FW$, an FW is additionally embedded, and the embedding order is updated as $[FW \rightarrow NAT \rightarrow LB \rightarrow FW]$. As no more resources exist in the PDP switch to embed the other SFs, $[FW \rightarrow NAT \rightarrow LB \rightarrow FW]$ is finally determined as the embedding order. As this embedding order cannot support SFC3, re-circulation from the egress port to the ingress port is allowed for SFC3. To summarize, the PDP switch can support SFC1 and SFC2 without any re-circulation while supporting SFC3 only with one re-circulation.

V. EXPERIMENTAL RESULTS

For performance evaluation, we implemented FASE, P4SC [9], and Dejavu [3] over a representative PDP switch (i.e., Tofino switch) and compared their average SFC completion times, T, and the required resource ratio to the total resource of the switch, R. Note that as Dejavu does not specify a detailed algorithm for the embedding order, it is assumed that its embedding order is determined without redundant SF embedding after sorting all the SFC requests according to their incoming rates.

We consider four types of SFCs that have been widely used in mobile networks and data center networks [9], [14]. Specifically, SFC1 and SFC2 require $[FW \rightarrow NAT \rightarrow L3\ routing\ (L3) \rightarrow LB]$ and $[LB \rightarrow FW]$, respectively. Whereas, SFC3 and SFC4 require $[NAT \rightarrow L2\ switching\ (L2) \rightarrow LB \rightarrow L3 \rightarrow FW]$ and $[LB \rightarrow NAT]$, respectively. In addition, we randomly set the incoming rates of SFC1, SFC2, SFC3, and SFC4 to a value in [1, 2, 3, 4] mega packets per second (Mpps).

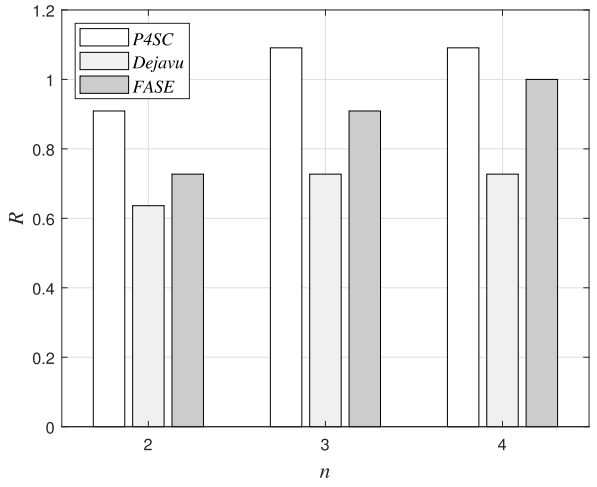
A. REQUIRED RESOURCE RATIO

Figure 5 shows the required resource ratio, R, to support all the requested SFCs under different numbers of requested SFCs, n. $R > 1$ indicates that the switch cannot embed all the required SFs and only some of the requested SFCs can be supported. As Dejavu does not allow any redundant SF embedding, it has the lowest R regardless of n, as shown in Figure 5. In contrast, the R values of P4SC and FASE increase apparently as n increases. This is because P4SC and FASE can redundantly embed SFs. In particular, P4SC requires excessive resources to support all requested SFCs, and thus, its R exceeds 1 when $n \geq 3$. That is, P4SC cannot support all the requested SFCs when $n \geq 3$ owing to excessive resource usage. In contrast, FASE utilizes re-circulations adequately, and therefore, the R of FASE can be bounded to a value less than 1, regardless of n. In other words, if all the SF types are embedded at least once, FASE can support all the requested SFCs by re-circulations.

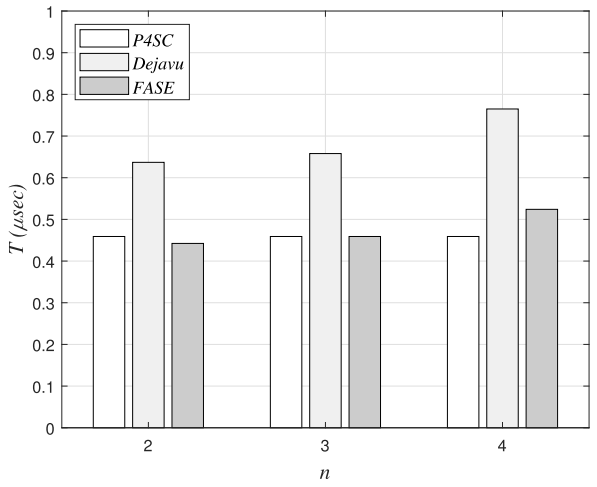
B. SFC COMPLETION TIME

Figure 5(b) shows the average SFC completion time T as a function of the number of requested SFCs, n. The average SFC completion time T is calculated based on the packets that can be correctly processed according to the SF processing order.

Regardless of n, it can be observed that the T value of Dejavu is much higher than those of P4SC and FASE. From Figure 5(b), it can also be observed that the T values of Dejavu and FASE increase with the increase in n. In particular, the increase in T in Dejavu is significant. This is because Dejavu employs re-circulations to process the requested SFCs. In this experiment, more SFs need to be embedded as n increases, and more re-circulations



(a) Required resources ratio, R.



(b) SFC completion time, T.

FIGURE 5. Experimental results.

are triggered in Dejavu accordingly. Although FASE also employs re-circulations, the impact of n is not as remarkable as that in Dejavu because FASE also allows redundant SFs. Moreover, as FASE employs re-circulations for the SFC requests with a lower incoming rate, the increment of T due to re-circulations can be limited.

As shown in Figure 5(b), when $n = 2$, FASE has a slightly lower T than that of P4SC. This is because P4SC uses the switch resources (for redundant SFs) more aggressively than FASE, which contributes to an increase in the internal processing time at the switch. Moreover, for $n \geq 3$, FASE shows an equal or higher T compared with that of P4SC because FASE cannot avoid several re-circulations under this situation. However, this result does not indicate that P4SC outperforms FASE. As demonstrated in Figure 5, P4SC cannot accommodate more than three SFC requests when $n \geq 3$ because R exceeds 1. Therefore, the T of P4SC for $n \geq 3$ in Figure 5(b) is the averaged SFC completion time of the requested SFCs except for SFC3. In summary, the reduced T of P4SC can be achieved only at the expense of increased resource usage, whereas FASE can balance the reduction of T and efficient resource usage.

VI. OPEN RESEARCH TOPICS

In this section, we present open research topics for better embedding of SFs in the programmable data plane.

A. SUPPORT OF COMPLEX OPERATIONS

Since the packets are processed by match-action tables in PDP instead of the software module on the CPU, the PDP switches can conduct only limited actions (e.g., count action, drop action, and modify header field action) [11]. Even though these actions are sufficient for simple networking operations (e.g., rewrite a packet header and select an output port) [15], these actions are not suited for more complex network operations of the stateful SFs [15], [16]. To address this problem, a new machine model called Banzai has been introduced [16]. In Banzai, an atomic unit for packet processing can persistently modify states on the switch. In so doing, the stateful operations can be implemented. However, no implementation results of Banzai have been reported yet. Therefore, further study to develop a practical solution for complex and stateful operations is required.

B. OVERCOME OF RESOURCE CONSTRAINT

Due to its limited resources, it is difficult for a single PDP switch to accommodate a number of SFs [3]. To overcome this limitation, novel PDP switch architectures, called disaggregated reconfigurable match-action table (dRMT) [17] and table extension architecture (TEA) [18], have been proposed in the literature.

In the conventional RMT used for commercial PDP switches (e.g., a Tofino switch), each pipeline stage can use only its local memory and computation resources, which leads to poor resource utilization [17]. To improve the resource utilization, dRMT disaggregates all memory and computation resources of a PDP switch and make them accessible by any pipelines via a crossbar. Even with its attractive feature, dRMT has not been yet implemented over commercial PDP switches. On the other hand, in TEA [18], DRAMs on commodity servers are leveraged to mitigate the resource constraint issue. In particular, TEA employs remote direct memory access (RDMA) that achieves low access latency to external DRAMs without involving any interrupts of the operating system. Even though dRMT and TEA are promising solutions to overcome the resource constraint issue, advanced SFs require more computation/resource-intensive operations, and therefore how to overcome the resource constraints needs to be further investigated.

C. PDP VIRTUALIZATION

When an SF is newly embedded in the PDP switch, the existing service needs to be interrupted to re-configure match-action tables in the PDP switch. To mitigate the service interruption time, PDP virtualization (or multi-tenancy) techniques should be supported [19]. Several PDP virtualization

techniques have been introduced, which can be classified into 1) hypervisor-based approach and 2) compiler-based approach. In the hypervisor-based approach, a special program called P4-hypervisor is first embedded to allow multiple SFs in a PDP switch. Although the hypervisor can provide flexibility for re-configuration, high processing performance cannot be guaranteed due to its frequent re-circulation operations. On the other hand, in the compiler-based approach, a compiler merges several SF programs to a single configuration file while preventing interference (e.g., shared resource usage and functionality between the SF programs). In so doing, the several SFs can be simultaneously embedded without any special P4 program (e.g., hypervisor). This approach can provide high processing performance; however, it cannot provide high flexibility since a new SF can be added only at the compile time [19]. To sum up, current PDP virtualization techniques cannot provide high performance and flexibility simultaneously, and therefore further study on the PDP virtualization is required.

VII. CONCLUSION

In this paper, we proposed the FASE algorithm that combines the redundant SF and re-circulation approaches to use their advantages. Experimental results demonstrated that FASE can reduce the SFC completion time by up to 2% and 33% compared with the redundant SF and re-circulation approaches, respectively. Furthermore, FASE utilizes the given switch resource efficiently, and thus, more SFC requests can be accommodated compared with the redundant SF approach. Therefore, we believe that FASE will be a promising algorithm for PDP-empowered SFC and it will contribute to achieving ultra-low latency services in future networks. In our future work, we will extend FASE to network-wide multiple PDP switches while supporting diverse SFCs.

REFERENCES

[1] J. Halpern and C. Pignataro, *Service Function Chaining (SFC) Architecture*, document IETF RFC 7665, Oct. 2015.
[2] P. Quinn and J. Guichard, "Service function chaining: Creating a service plane via network service headers," *Computer*, vol. 47, no. 11, pp. 44–238, Nov. 2014.
[3] D. Wu, A. Chen, T. Ng, G. Wang, and H. Wang, "Accelerated service chaining on a single switch ASIC," in *Proc. ACM HotNets*, Nov. 2019, pp. 141–149.
[4] A. M. Medhat, T. Taleb, A. Elmangoush, G. A. Carella, S. Covaci, and T. Magedanz, "Service function chaining in next generation networks: State of the art and research challenges," *IEEE Commun. Mag.*, vol. 55, no. 2, pp. 216–223, Feb. 2017.
[5] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, "SilkRoad: Making stateful layer-4 load balancing fast and cheap using switching ASICs," in *Proc. ACM SIGCOMM*, Aug. 2017, pp. 15–28.
[6] Y. Tokusashi, H. Dang, F. Pedone, R. E. Soul, and N. Zilberman, "The case for in-network computing on demand," in *Proc. EuroSys*, Mar. 2019, pp. 1–16.
[7] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica, "NetCache: Balancing key-value stores with fast in-network caching," in *Proc. ACM SOSP*, Oct. 2017, pp. 121–136.
[8] R. Datta, S. Choi, A. Chowdhary, and Y. Park, "P4Guard: Designing P4 based firewall," in *Proc. IEEE MILCOM*, Oct. 2018, pp. 1–6.
[9] X. Chen, D. Zhang, X. Wang, K. Zhu, and H. Zhou, "P4SC: Towards high-performance service function chain implementation on the P4-capable device," in *Proc. IFIP/IEEE IM*, Apr. 2019, pp. 1–9.
[10] H. Lee, J. Lee, H. Ko, and S. Pack, "Resource-efficient service function chaining in programmable data plane," in *Proc. EuroP4*, Sep. 2019. [Online]. Available: https://p4.org/events/2019-09-23-euro-p4-workshop/
[11] The P4 Architecture Working Group. (May 2020). *P416 Portable Switch Architecture (PSA)*. [Online]. Available: https://p4.org/p4-spec/docs/P4-16-working-spec.html
[12] P. Bosshart, G. Gibb, H. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDNs," in *Proc. SIGCOMM*, Aug. 2013, pp. 99–110.
[13] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "MoonGen: A scriptable high-speed packet generator," in *Proc. ACM IMC*, Oct. 2015, pp. 275–287.
[14] W. Haeffner, J. Napper, M. Stiemerling, D. Lopez, and C. Pignataro, "Service function chaining use cases in mobile networks," IETF, Fremont, CA, USA, Tech. Rep. draft-ietf-sfc-use-case-mobility-09, 2019.
[15] N. Gebara, A. Lerner, M. Yang, M. Yu, P. Costa, and M. Ghobadi, "Challenging the stateless quo of programmable switches," in *Proc. ACM HotNets*, Nov. 2020, pp. 153–159.
[16] A. Sivaraman, A. Cheung, M. Budiu, C. Kim, M. Alizadeh, and H. Balakrishnan, "Packet transactions: High-level programming for line-rate switches," in *Proc. ACM SIGCOMM*, Aug. 2016, pp. 15–28.
[17] S. Chole, A. Fingerhut, S. Ma, A. Sivaraman, S. Vargaftik, A. Berger, G. Mendelson, M. Alizadeh, S. Chuang, I. Keslassy, A. Orda, and T. Edsall, Eds., "dRMT: Disaggregated programmable switching," in *Proc. ACM SIGCOMM*, Aug. 2017, pp. 1–14.
[18] D. Kim, Z. Liu, Y. Zhu, C. Kim, J. Lee, V. Sekar, and S. Seshan, "TEA: Enabling state-intensive network functions on programmable switches," in *Proc. ACM SIGCOMM*, Aug. 2020, pp. 90–106.
[19] S. Han, S. Jang, H. Choi, H. Lee, and S. Pack, "Virtualization in programmable data plane: A survey and open challenges," *IEEE Open J. Commun. Soc.*, vol. 1, pp. 527–534, Apr. 2020.

JAEWOOK LEE received the B.S. degree from the School of Electrical Engineering, Korea University, Seoul, South Korea, in 2014, where he is currently pursuing the M.S. and Ph.D. degrees (integrated course). His research interests include 6G mobile networks, distributed computing, federated learning, network automation, programmable data planes, service function chaining (SFC), and in-network computing.

HANEUL KO (Member, IEEE) received the B.S. and Ph.D. degrees from the School of Electrical Engineering, Korea University, Seoul, South Korea, in 2011 and 2016, respectively. He is currently an Assistant Professor with the Department of Computer Convergence Software, Korea University, Sejong, South Korea. From 2016 to 2017, he was a Postdoctoral Fellow of Mobile Network and Communications with Korea University. From 2017 to 2018, he was with the Smart Quantum Communication Research Center, Korea University, and a Visiting Postdoctoral Fellow with The University of British Columbia, Vancouver, BC, Canada. His research interests include 5G networks, network automation, and mobile cloud computing.

HOCHAN LEE (Graduate Student Member, IEEE) received the B.S. degree from Korea University, Seoul, South Korea, in 2018, where he is currently pursuing the M.S. and Ph.D. degrees (integrated course). His research interests include federated learning, programmable data planes, service function chaining (SFC), and in-network computing.

SANGHEON PACK (Senior Member, IEEE) received the B.S. and Ph.D. degrees in computer engineering from Seoul National University, Seoul, South Korea, in 2000 and 2005, respectively. He joined the Faculty of Korea University, Seoul, in 2007, where he is currently a Professor with the School of Electrical Engineering. His research interests include softwarized networking (SDN/NFV), 5G/6G mobile core networks, mobile edge computing/programmable data plane, and vehicular networking. He was a recipient of the IEEE ComSoc APB Outstanding Young Researcher Award in 2009, the Haedong Young Scholar Award from the Korean Institute of Communications and Information Sciences (KICS) in 2013, the Joint Award for IT Young Engineers Award from the IEEE/Institute of Electronics and Information Engineers (IEIE) in 2017, and the Young Information Scientist Award from the Korean Institute of Information Scientists and Engineers (KIISE) in 2017. He served as the Publicity Co-Chair for the IEEE SECON 2012, the Publication Co-Chair for the IEEE INFOCOM 2014 and ACM MobiHoc 2015, the Track Chair for the IEEE VTC 2020-Fall/2010-Fall and the IEEE CCNC 2019, and the TPC Vice-Chair of the Information Systems for the IEEE WCNC 2020. He is an Editor of the IEEE Internet of Things Journal (IoT), the *Journal of Communications Networks* (JCN), and *IET Communications*. He is a Guest Editor of the IEEE Transactions on Emerging Topics in Computing (TETC) and the IEEE Transactions on Network Science and Engineering (TNSE).