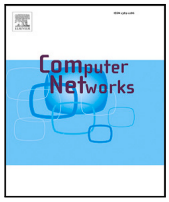
Computer Networks 239 (2024) 110163

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



Programmable device deployment for efficient network function offloading

Huaqing Tu [a,b,c,1], Gongming Zhao [b,c,*,2], Hongli Xu [b,c,2], Chunming Qiao [d,3]

[a] Zhejiang Lab, China
[b] School of Computer Science and Technology, University of Science and Technology of China, China
[c] Suzhou Institute for Advanced Research, University of Science and Technology of China, China
[d] Department of Computer Science and Engineering, University at Buffalo, USA

ARTICLE INFO

Keywords:
Network function offloading
Programmable devices
Approximation algorithm
Network upgrade

ABSTRACT

Network functions (NFs) play an important role in ensuring network security and performance. To improve the NF throughput performance, an emerging method is to offload NFs on programmable devices, bringing orders-of-magnitude improvements. A primary task for efficient NF offloading is how to deploy programmable devices (*e.g.*, programmable switches) for network upgrades. Although programmable switches have powerful computing resources, their memory resources are usually limited, which poses a challenge of offloading stateful NFs (*e.g.*, load balancer, NAT) with a large-size memory requirement. A promising solution to break the memory limitation is using external memory (*e.g.*, on commodity servers) with the help of remote direct memory access (RDMA) supported by SmartNIC. Therefore, this paper studies the problem of upgrading networks by replacing legacy switches with programmable switches and equipping commodity servers with SmartNICs so that NFs can be offloaded on programmable switches with external memory expansion. We prove that this problem is NP-Hard, and there is no polynomial-time algorithm with an approximation ratio of $(1-\epsilon)\cdot\ln h$, where ϵ is an arbitrarily small value, and h is the total number of requests in the network. Then we design an efficient algorithm with an approximation ratio of $2.5\cdot H(m\cdot n)$, where m is the number of NF types, n is the maximum number of requests through a switch, and $H(m\cdot n)$ is the $(m\cdot n)$th harmonic number. The simulation results show that our solution can reduce the upgrade cost by about 70% compared with the state-of-the-art approaches while preserving the same system throughput.

1. Introduction

As an intrinsic and fundamental part of today's networks, network functions (NFs) support a diverse set of functions, such as traffic accelerators, firewalls and IDSes [1,2]. Since network function virtualization (NFV) [3–5] can significantly improve the scalability and flexibility of resource management, NFs implemented in software have been widely deployed in various networking scenarios, such as backbone networks, and data center networks. However, software-based NFs on commodity servers have limited processing capability, resulting in poor throughput performance [6,7]. For example, the maximum throughput of an NF implemented on a server is only about 10 Gbps [8], which makes it hard to deal with ever-growing traffic.

To improve NF performance, recent researches [6,9,10] offload NFs on programmable devices. The line-rate packet processing capacity of novel programmable devices brings significant NF performance improvement in throughput and forwarding latency compared with implementing NFs in software on commodity servers. For example, the system throughput and switch capacity of Edgecore Wedge 100BF-32X are 3.2 Tbps [11] and 6.4 Tbps [12], respectively, which are several hundred times as those of a software-based NF. A single switch can process 5 billion packets per second [12], whereas a software-based load balancer can process approximately 15 million packets per second on a single server [13]. The high throughput of programmable switches makes it more cost-effective than commodity servers for a scenario of high traffic rate, even if commodity servers are cheaper than programmable switches [8]. Due to these advantages of programmable devices, the NF offloading problem has been studied for different targets, such as minimizing end-to-end latency, maximizing service chain acceptance ratio or system throughput [6,9]. However, previous works mainly focus on the provisioning of NFs and assume that a set of programmable devices has been deployed on given positions by default.

* Corresponding author at: School of Computer Science and Technology, University of Science and Technology of China, China.
E-mail address: gmzhao@ustc.edu.cn (G. Zhao).
[1] Student Member, IEEE.
[2] Member, IEEE.
[3] Fellow, IEEE.

https://doi.org/10.1016/j.comnet.2023.110163
Received 8 May 2023; Received in revised form 13 December 2023; Accepted 20 December 2023
Available online 22 December 2023

In fact, *a primary task for NF offloading is how to cost-efficiently deploy programmable devices with the current legacy network.*

Due to the ever-growing amounts of data traffic and stateful NFs, when deploying programmable devices for network upgrades, we must ensure that the deployed programmable devices provide sufficient computing resources for NF processing, as well as memory resources for storing flow state information. For stateful NFs, maintaining flow state is the key to successfully processing and forwarding traffic. For example, load balancers need to maintain per-flow server mapping tables so that all the packets belonging to the same flow are forwarded to the same application server [14].

One may think that deploying only programmable switches (*e.g.*, Edgecore Wedge 100BF-32X [12]) is enough for NF offloading. However, programmable switches' limited memory resources pose a challenge of offloading stateful NFs with a large-size memory requirement. For example, the SRAM resource on a switch chip is only 10–100 MB [8,10,15], which is shared by routing and flow state management. In practice, for stateful NFs, such as network address translation (NAT) and load balancer, the status table size may be up to 525 MB [8], which far exceeds the memory capacity of a programmable switch. Ignoring the memory size limitation, existing work [16] of network upgrades has to deploy more programmable devices so that the memory resource needs of stateful NFs are satisfied, resulting in high network upgrade cost, especially in large-scale networks.

Although evicting infrequently used entries can reuse scarce storage resources, deleting entries is only applicable when the state of active connections requires fewer storage resources than a programmable switch has. Nevertheless, this situation may be rare, since the throughput of programmable switches is very large. A promising way to break the memory limitation of programmable switches is using external memory (*e.g.*, on commodity servers). The traditional method like remote procedure call (RPC) [17] mechanisms realizing the communication between programmable switches and external servers will introduce intolerable access delay (100 μs–500 μs) [17]. To this end, this paper adopts remote direct memory access (RDMA) technique supported by SmartNIC to enable programmable switches to lookup state tables stored in servers' memory within access delay of 1.8 μs–2.2 μs [8]. In this way, programmable switches mainly provide computing resources, and commodity servers provide memory resources. Moreover, since the memory capacity of a server is usually greater than 500 GB, a server can provide memory expansion for multiple programmable switches. Therefore, although configuring SmartNICs incurs a cost, it is still more economical than only using programmable switches to offload network functions. This approach has been adopted to enable programmable switches to retrieve state stored in servers and achieve fault tolerance in case of switch failures [8,18].

Thus, to realize network upgrades and break the memory resource limitation, we study the problem of not only deploying programmable switches but also configuring SmartNICs for commodity servers. We call this as programmable device deployment (PDD) problem, as both smartNICs and programmable switches are programmable devices. It should be noted that the PDD problem needs to determine: (1) which legacy switches should be replaced with programmable switches; (2) which servers should be upgraded with SmartNICs; and (3) which programmable switches should an upgraded server provide memory expansion for. Existing works [16,19] on network upgrades only consider deploying servers or programmable switches to implement NFs (explained in detail in Section 2.2). Thus, *their methods cannot be directly used to solve the PDD problem*. To the best of our knowledge, this is the first work that focuses on deploy both programmable switches and SmartNICs to construct a programmable network with powerful computing capacity on the premise of ensuring sufficient memory resources. Note that the communication mechanism between programmable and external memory has been thoroughly researched by TEA [8]. Thus, the communication mechanism is not the focus of our work. We summarize the contributions of this work as follows:

1. We propose the programmable device deployment (PDD) problem for efficient NF offloading and prove its NP-Hardness. We also show its inapproximation ratio of $(1-\epsilon)\cdot\log h$, where h is the total number of requests in a network, and ϵ belongs to [0, 1].
2. Then we design an algorithm with an approximation ratio of $2.5\cdot H(m\cdot n)$ based on the greedy 0–1 knapsack method, where n is the maximum number of requests passing through a programmable switch, and m is the number of NF's types.
3. We evaluate the performance of the proposed algorithms through extensive simulations. The results show that our solution can reduce the upgrade cost by about 70% compared with the baselines while preserving the same system throughput.

The rest of this paper is organized as follows. Section 2 introduces background and motivation. In Section 3, we formulate the PDD problem. Then we propose an approximate algorithm to solve it in Section 4. Section 5 presents the simulation results and Section 6 reviews the related works. Section 7 concludes the paper.

2. Background and motivation

2.1. Comparison of NF implementation methods

Currently, there are three different NF implementation methods. Schematic diagrams of traffic processing for these methods are shown in Fig. 1 .

- *Implemented on commodity servers* with the virtualization technology. As shown in Fig. 1(a), traffic is forwarded from the switch to the connected server for NF processing. Although computing resources and memory resources are sufficient on commodity servers, the disadvantage of this method is low throughput and high processing delay, since NFs are implemented in software, also called network function virtualization.
- *Offloaded on programmable switches*. Traffic is handled by NFs simultaneously with packet processing on the programmable switch without forwarding traffic to a server. Due to the powerful computing capacity, this method brings orders-of-magnitude improvements in throughput and processing delay. However, programmable switches' limited memory resources present a challenge for offloading stateful NFs, especially in large-scale networks.
- *Offloaded on programmable switches with memory expansion on external servers* [8]. This method is proposed to break the memory resource limitation in the second method. Specifically, programmable switches provide computing resources for NFs, and external servers provide memory extension. Through RDMA technology supported by SmartNICs, programmable switches can access servers' memory without involving servers' CPUs. Traffic is processed by the programmable switch and the state information of NFs is retrieved from an external server. It should be noted that the external server does not process traffic but only provides memory expansion.

In conclusion, compared with the former two NF implementation methods, the third method provides both powerful computing resources and sufficient memory resources. But the key step is how to deploy programmable switches and SmartNICs for efficient NF processing.

2.2. Limitations of prior works

The network upgrade schemes for the first and second NF implementation methods have been extensively studied [16,19]. For the first NF implementation method, Liu et al. [19] study how to deploy commodity servers to build a scalable NFV-enabled network. For the second NF implementation method, Xue et al. [16] study the shift from

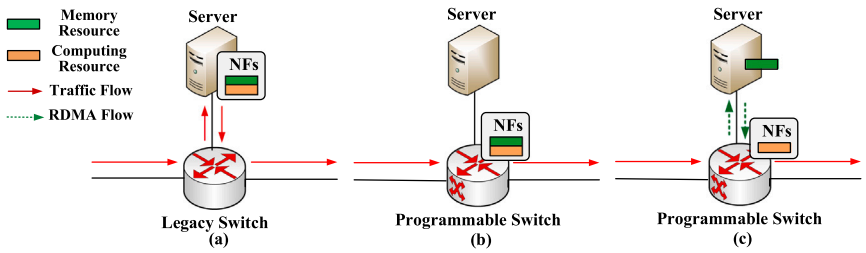


Fig. 1. Traffic processing of different NF implementation methods. Black solid lines are links. Red solid lines denote traffic flows. Green dotted lines indicate RDMA traffic. (a) Commodity servers provide computing and memory resources for NFs. (b) Programmable switches provide computing and memory resources for NFs. (c) Programmable switches provide computing resources and commodity servers provide memory resources. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

homogeneous NFV networks to heterogeneous ones by deploying programmable devices. In the second NF implementation method, the PDD problem needs to answer: (1) which legacy switches should be replaced with programmable switches; (2) which servers should be upgraded with SmartNICs; and (3) which programmable switches should an upgraded server provide memory expansion for. The algorithms proposed by Liu et al. [19] and Xue et al. [16] cannot answer these questions, so their method cannot be used to solve the PDD problem. Although Kim et al. [8] have designed a mechanism for programmable switches to access storage resources of external servers, they do not present a cost-efficiently network upgrade algorithm to address the three questions raised by PDD. To the best of our knowledge, this is the first work on network upgrade algorithm for the third NF implementation method.

3. Preliminaries

3.1. Network model

The original (or traditional) network contains two types of devices, *i.e.*, legacy switches and servers. We use V and S to denote the legacy switch set and server set, respectively. V and S can be obtained according to the current network topology. Servers are interconnected by these legacy switches. We assume that there is a set of NF types, such as Proxy, Firewall and IDS. Let $\mathcal{F}$ denote the NF type set. Note that, some sophisticated network function operations like buffering, payload encryption and loop are difficult to implement on programmable switches, since the existing programmable switches are designed to process packet in a pipeline manner. [6]. As a result, only certain types of NFs can be offloaded on programmable switches. For ease of expression, if an NF can be offloaded on programmable switches, we call it offloadable. We use $\mathcal{F}' \subseteq \mathcal{F}$ to indicate the offloadable NF set.

To transform a traditional network into a programmable network, we collect the long-term statistics of all requests in the traditional network. For security and performance reasons, requests need to be processed orderly by multiple NFs, which is also called as service function chain (SFC) [19]. For clarity, we identify a request by three elements <source, destination, SFC requirement>. Through long-term statistics with traffic measurement methods (*e.g.*, sketch [20]), for each request γ, we obtain the estimated traffic size denoted as θ_γ and the set of NFs (say, $\mathcal{F}_\gamma$) specified in the SFC requirement. For instance, $\mathcal{F}_\gamma = \{\text{Firewall, NAT, Proxy}\}$, if the SFC requirement of request γ is Firewall-NAT-Proxy. Let $\mathcal{F}'_\gamma \subseteq \mathcal{F}_\gamma$ be the set of offloadable NF types in the SFC requirement. Moreover, we denote the request set as Γ. The symbols appearing in Section 3 are summarized in Table 1. Besides, the important abbreviations used in this paper are listed in Table 2

3.2. Problem formulation

This section presents PDD's problem definition. Let the binary variable x_v indicate whether legacy switch $v \in V$ is replaced with a programmable switch or not and the binary variable y_s denote if server $s \in S$ is configured with a SmartNIC or not. In addition, we adopt α to express the price of a programmable switch and β to denote the price of configuring a SmartNIC for a server. The values of α and β are determined by the network administrator when purchasing network equipment. The optimization goal of the PDD problem is to minimize a network's upgrade cost, while satisfying the computing resource capacity constraints on programmable switches, memory resource capacity constraints on servers and the offloaded NFs' processing capacity.

For NF offloading, programmable switches mainly provide computing resources, since NFs' memory needs can be satisfied by external servers. Let the binary variable z_v^f denote whether NF with type $f \in \mathcal{F}'$ will be offloaded on programmable switch v or not and δ_v^f denotes the resource consumptions of NF with type f on programmable switch v. Although it is hard to specify which specific action units (or SRAM/TCAM, etc.) are allocated to NF f, we can set the value of δ_v^f according to compilation results by employing a static method via the compiler for programmable switch [6]. Note that δ_v^f does not specify exactly which action units (or SRAM/TCAM, *etc.*) are allocated to an NF. It represents the amount of resources that an NF with type f will consume on switch v. This is helpful for deploying NFs in an appropriate position without violating the resource constraints of programmable switches. The resource needs (including computation and storage) of offloaded NFs on programmable switch v cannot exceed the switch's resource capacity (denoted as C_v), which can be set according to the parameters of the purchased network equipment. In order to make the upgrade process from traditional networks to programmable networks more practical, this paper adopts an architecture in which a commodity server's DRAM can be an external storage space for programmable switches. This architecture was first proposed by [8]. In this architecture, all states of an NF are held in an external server, and the storage on the programmable switch is mainly used as a cache to improve the speed of accessing states. Although the storage resources consumed by the global state of an NF will change with the number of connections [21,22], its storage space size as a cache on the programmable switch can be fixed. Thus, storage resource requirements can be specified in δ_v^f. The external servers provide memory resources for programmable switches. Though the storage resources consumption of a stateful NF changes over time while the NF is running, when upgrading networks and offloading NFs, we set ϕ_f as a constant in order to better determine the deployment of programmable switches and SmartNICs and the offloading of NFs. Since the storage resources on commodity servers are much more than those on programmable switches, the value of ϕ_f is set to be slightly higher than the actual amount of resources required by network function f. For example, a network administrator observes that over a period of time, a stateful firewall only saved a maximum of eight million connections at any moment. When offloading a stateful firewall on a programmable switch, the network administrator reserves storage resources that can save ten million connections on a commodity server to deal with emergencies. We adopt C_s to indicate the memory resource capacity of server s. The amount of memory resources (say, π_v) required by programmable switch v depends on the offloaded NFs. Thus, we have $\pi_v = \sum_{f \in \mathcal{F}'} z_v^f \cdot \phi_f$, where ϕ_f is the memory needs of NF with type f.

Table 1
Key notations.

Parameters	Description
V	A legacy switch set
S	A server set
α	The price of a programmable switch
β	The price of a SmartNIC
$\mathcal{F}$	An NF type set
$\mathcal{F}'$	An offloadable NF type set
Γ	A request set
$\mathcal{F}_\gamma$	The NF type set required by request γ
$\mathcal{F}'_\gamma$	The offloadable NF type set require by request γ
θ_γ	The traffic size of request γ
C_v	The resource capacity of programmable switch v
C_v^f	The processing capacity of NF with type $f \in F'$ offloaded on programmable switch v
δ_v^f	The resource consumption of NF with type f when offloaded on programmable switch v
x_v	Whether legacy switch $v \in V$ is replaced with a programmable switch or not
y_s	Whether server s is configured with a SmartNIC or not
z_v^f	Whether NF with type $f \in F$ is offloaded on switch $v \in V$ or not
q_v^s	Whether switch $v \in V$ selects server $s \in S$ to expand memory or not
$w_{v,\gamma}^f$	Whether γ is processed by the network function f offloaded on switch v or not

Table 2
Key abbreviations.

Abbreviations	Description
NFV	Network Function Virtualization
NF	Network Function
RDMA	Remote Direct Memory Access
NAT	Network Address Translation
SFC	Service Function Chain
PDD	Programmable Device Deployment
KPBP	Knapsack and Bin-Packing Algorithm for PDD
NOTS	NF Offloading and Traffic Scheduling
RBNS	Rounding-based NF Offloading and Traffic Scheduling Algorithm

In addition, NF f offloaded on programmable switch v has a processing capacity denoted as C_v^f. In order to make the model of the PDD problem more practical, we use the following steps to determine the value of C_v^f. First, for each NF $f \in F'$, where F' is a set of offloadable NFs, we implement it on programmable switch v. Then we test the maximum throughput of NF f and set this to the value of C_v^f. Note that recirculation occurs when a packet traverses through two or more pipelines, which will degrade the throughput of programmable switches. The above method for determining the value of C_v^f has taken into account the possible throughput degradation caused by recirculation. Each request has a candidate switch set V_γ and it will only be processed by NFs offloaded on the switches in V_γ for guaranteeing routing performance. The definition of V_γ is as follows. Let $\tau(v, v')$ be the hop count of the shortest route between two legacy switches v' and v in a network. We term v'' a feasible switch of request γ with source switch s and destination t, if and only if $\tau(s, v'') + \tau(v'', t) \le \sigma \cdot \tau(s, t)$, where $\sigma \ge 1$ is the strech [23]. Then, we use $V_\gamma = \{v'' | \tau(s, v'') + \tau(v'', t) \le \sigma \cdot \tau(s, t)\}$ to denote the feasible switch set of request γ. Let the binary variable $w_{v,\gamma}^f$ denote whether request γ is processed by the network function f offloaded on switch $v \in V_\gamma$ or not. Note that, through efficient routing algorithms [24,25], the NFs' sequence specified in SFC requirements can be meet. Besides, the resource consumption of NF offloading on programmable switches will not be affected by the NF's sequence. Thus, similar to [19], since NF processing order will not impact the upgrade solution, we do not consider it in this paper. With above notations, the PDD problem is formulated as follows:

$$\min \sum_{v \in V} \alpha \cdot x_v + \sum_{s \in S} \beta \cdot y_s$$

$$S.t. \begin{cases} x_v \ge z_v^f, & \forall v \in V, f \in \mathcal{F}' \\ z_v^f \ge w_{v,\gamma}^f, & \forall \gamma \in \Gamma, v \in V_\gamma, f \in \mathcal{F}'_\gamma \\ \sum_{v \in V} w_{v,\gamma}^f \ge 1, & \forall \gamma \in \Gamma, f \in \mathcal{F}'_\gamma \\ \sum_{f \in \mathcal{F}'} z_v^f \cdot \delta_v^f \le C_v, & \forall v \in V \\ \sum_{\gamma \in \Gamma : f \in \mathcal{F}'_\gamma} w_{v,\gamma}^f \cdot \theta_\gamma \le C_v^f, & \forall f \in \mathcal{F}', v \in V \\ y_s \ge q_v^s, & \forall v \in V, s \in S \\ \sum_{s \in S} q_v^s \ge x_v, & \forall v \in V \\ \sum_{v \in V} q_v^s \cdot \pi_v \le C_s, & \forall s \in S \\ \pi_v = \sum_{f \in \mathcal{F}'} z_v^f \cdot \phi_f, & \forall v \in V \\ x_v, y_s, z_v^f, q_v^s, w_{v,\gamma}^f \in \{0, 1\}, & \forall v, s, f, \gamma \end{cases} \quad (1)$$

The first set of inequalities indicates that network function f is offloaded on switch v if and only if the switch v is replaced with a programmable switch. The second inequality set means that request γ can be processed by network function f on switch v if and only if network function f is offloaded to switch v. The third inequality set denotes that request γ should be processed by NF $f \in \mathcal{F}_\gamma$ at least once to satisfy its SFC requirement. The fourth set of inequalities expresses programmable switches computing resource constraints for NF offloading. The fifth inequality set means the NF processing capacity constraints. The sixth set of inequalities means server $s \in S$ can be selected to expand the memory of switch v if and only if server s is configured with a SmartNIC. The seventh set of inequalities indicates each programmable switch will choose at least one server to expand its memory capacity. The eighth set of inequalities expresses servers' memory resource constraints. The ninth set of equalities defines the external memory needs of switch v. Our objective is to minimize the total upgrade cost, *i.e.*, $\sum_{v \in V} \alpha \cdot x_v + \sum_{s \in S} \beta \cdot y_s$.

This paper focuses on efficient network function offloading through the deployment of programmable switches and SmartNICs with minimal costs for network upgrades. Thus, our main focus is on modeling the resource consumption of NF offloading and the deployment cost of programmable switches and SmartNICs. The proposed model in this paper is based on the TEA architecture [8]. However, as mentioned

in [26], TEA faces several challenges such as only supporting address-based memory access, performance degradation caused by frequent updates in the remote memory, potential packet loss between the switch and the remote memory, the complex interaction between general data plane applications and the remote memory. Therefore, the model proposed in this paper based on TEA may also encounter these issues. Despite these challenges, TEA offers cost-effective and flexible solutions by leveraging existing resources in commodity servers. It overcomes the memory resource limitations of programmable switches, enabling the offloading of network functions with large status tables. Hence, the model proposed in this paper is based on TEA.

Theorem 1. *The PDD problem is NP-hard.*

Proof. We will show that the Minimum Set Cover (MSC) problem [27] is a special case of the problem studied here. We first present an instance of MSC: let $E = \{e_1, e_2, \dots, e_n\}$ denote the collection of n elements and $C = \{E_i \subseteq E, i = 1, 2, \dots, m\}$. The MSC problem attempts to construct a minimum set $C' \subseteq C$ so that a element $e \in E$ is included in as least one set in C'. Then, to prove PDD's NP-hardness, we construct a special case of PDD. Assume that: (1) each programmable switch has unlimited computing power; (2) the network has only one type of NF; (3) each server has been equipped with a SmartNIC, so the cost of SmartNIC is zero. In this case, we can think of each request as an element and each set of requests passing through a same switch is abstracted as a set in C. Similar to MSC, PDD try to replace legacy switches with minimum number of programmable switches while covering all requests. Therefore, we can conclude that the PDD problem's special instance described above becomes the standard MSC problem. The PDD problem is also NP-Hard, since MSC is NP-hard. □

Theorem 2. *There does not exist a polynomial time algorithm for the PDD problem with an approximation ratio better than* $(1-\epsilon)\cdot \ln h$*, where* h *is the total number of requests, for any* $\epsilon > 0$*, unless* $P = NP$*.*

Proof. The previous works presented by Feige [28], Raz and Safra [29] have proved that there is no polynomial time algorithm whose approximate ratio is $\ln m \cdot (1-\epsilon)$ for the MSC problem, where m denotes the number of elements in the MSC problem, for any $\epsilon > 0$, unless $P = NP$. Let $h = |\Gamma|$ be the size of request set in a network. Because MSC is a special case of PDD, if the PDD problem has an algorithm with a better approximation ratio than $\ln h \cdot (1-\epsilon)$, this algorithm can also be adopted for the MSC problem. However, it contradicts the previous inapproximation analysis. Therefore, we can conclude that, for any $\epsilon > 0$, the PDD problem has an inapproximation ratio of $\ln h \cdot (1-\epsilon)$. □

4. Algorithm for programmable device deployment

Due to NP-Hardness, the PDD problem cannot be optimally solved in polynomial time. An approximation algorithm called KPBP for the PDD problem is designed for the PDD problem (Section 4.1). We analyze its approximation performance (Section 4.2). Moreover, we present the extension to network function offloading and traffic scheduling when programmable switches and SmartNICs are deployed (Section 4.3).

4.1. Algorithm description

In this section, we present an algorithm, called KPBP, based on knapsack and bin packing to minimize the upgrade cost. The KPBP algorithm consists of two steps formally described in Algorithm 1. The algorithm chooses a set of legacy switches to upgrade in the first step. To satisfy SFC requirements, each request should traverse specific NFs. Let Γ^f be the set of requests, which should be handled by NF with type $f \in \mathcal{F}'$, and the traffic size of Γ^f is denoted as g_f. Besides, we use U_v^f to represent the request set that has not been uncovered by NF with type f on switch v (Line 3–6). The first step goes through a series of iterations. The KP algorithm is performed by the KPBP algorithm in each iteration to construct a chosen NF set (say, $\mathcal{F}_v$) for each switch v (Line 8–9). The detail of the KP algorithm will be presented in the next paragraph. Then, the KPBP algorithm selects a switch v which has the maximum profit to upgrade (Line 10–11). After determining a legacy switch that will be upgraded, KPBP updates the uncovered request set (Line 12–15). Specifically, for each NF with type f offloaded on switch v, KPBP will update the requests that is covered by f (Line 12–13). In addition, for those switches that have not been determined to be upgraded, the request set that has not been uncovered should also be updated, since some of them may be covered by the newly upgraded switch (Line 14–15). Until all requests are covered by the NFs on upgraded switches, the first step will end. Based on the switch set that needs to be upgraded through the first step, the second step selects servers to upgrade. The KPBP algorithm computes the memory consumption (say, b_v) of NFs offloaded on switch v, and rearrange the switch $v \in V'$ in the decreasing order with memory consumption b_v. Using bin packing algorithm, the minimum number of upgraded servers can be determined. At last, KPBP adopts P-Center algorithm [30] to determine the location of upgraded servers so that the maximum distance between upgrade servers and switches is minimized.

Now, we introduce the KP algorithm, which is used to decide the NF set offloaded on switch v while maximizing the request set covered by these NFs with the assumption that the legacy switch v has been upgraded to a programmable one. When NF with type f is offloaded

Algorithm 1 KPBP: Knapsack and Bin-Packing Algorithm for PDD

1: $V' \leftarrow \emptyset$, $S' \leftarrow \emptyset$, $P_v \leftarrow \emptyset$
2: **Step 1: Choose Legacy Switches to Upgrade**
3: **for** each NF type $f \in \mathcal{F}'$ **do**
4: $g_f \leftarrow |\Gamma^f|$
5: **for** each legacy switch $v \in V$ **do**
6: $U_v^f \leftarrow \Gamma_v^f$
7: **end for**
8: **end for**
9: **while** $g_f > 0, \forall f \in \mathcal{F}'$ **do**
10: **for** each legacy switch $v \in V - V'$ **do**
11: Choose an NF set using the KP algorithm, $\mathcal{F}_v \leftarrow KP(v)$
12: **end for**
13: Replace a legacy switch v with a programmable switch which has the maximum profit $\sum_{f \in \mathcal{F}_v} \psi_v^f$
14: $V' \leftarrow V' \cup \{v\}$
15: **for** each NF type $f \in \mathcal{F}_v$ **do**
16: $g_f \leftarrow g_f - |U_v^f|$
17: **for** each legacy switch $v_i \in V - V'$ **do**
18: $U_{v_i}^f \leftarrow U_{v_i}^f - U_v^f$
19: **end for**
20: **end for**
21: **end while**
22: **Step 2: Select Servers to Upgrade based on Bin Packing Method**
23: **for** $v \in V'$ **do**
24: Compute the memory consumption of NFs offloaded on switch v with $b_v \leftarrow \sum_{f \in \mathcal{F}_v} \phi_f$
25: **end for**
26: Rearrange the switch $v \in V'$ in the decreasing order with memory consumption b_v
27: **for** $v \in V'$ **do**
28: Use first-fit bin packing algorithm to determine the minimum number of upgraded servers
29: **end for**
30: Adopts P-Center algorithm to determine the location of upgraded servers

Algorithm 2 KP: 0-1 Knapsack Algorithm on Switch v

1: $\mathcal{F}_v \leftarrow \emptyset$, $A_v^f \leftarrow \emptyset$, $a_v^f \leftarrow 0, \forall f \in \mathcal{F}'$
2: **for** each $f \in \mathcal{F}'$ **do**
3: Sort the descending request $\gamma \in U_v^f$ according to the traffic size θ_γ
4: **for** each request $\gamma \in U_v^f$ **do**
5: **if** $\sum_{\gamma \in A_v^f} \theta_\gamma \le C_v^f$ **then**
6: $A_v^f \leftarrow A_v^f \cup \{\gamma\}$
7: **end if**
8: **end for**
9: $a_v^f \leftarrow \sum_{\gamma \in A_v^f} \theta_\gamma$
10: **end for**
11: Rearrange the NF with type $f \in \mathcal{F}'$ in the decreasing order with the unit profit value $\frac{a_v^f}{\delta_v^f}$
12: **for** $f \in \mathcal{F}'$ **do**
13: **if** $\sum_{f \in \mathcal{F}_v} \delta_f^v \le C_v$ **then**
14: $\mathcal{F}_v \leftarrow \mathcal{F}_v \cup \{f\}$
15: **end if**
16: **end for**
17: return $\mathcal{F}_v$

on switch v, some requests (*i.e.*, a subset of U_v^f) will be covered by this NF while satisfying the switch's and NF's capacity constraints. Since this process is similar to the 0–1 knapsack problem [31,32], this case is regarded as a 0–1 knapsack problem [31,32]. Specifically, the joint consideration of switch's and NF's processing capacity is the knapsack capacity. The traffic size of requests covered by the NF is the item's profit and the offloading cost of the NF is the item size. The KP algorithm's goal is maximizing the total traffic size of requests that are handled by these NFs offloaded on the switch, similar to the knapsack problem.

Algorithm 2 describes the details of the KP algorithm. Let $\mathcal{F}_v$ be the NF set offloaded on switch v. Let A_v^f be request set determined to be handled by NF with type of f on switch v. Moreover, a_v^f is the total traffic size of requests in A_v^f. At the beginning, the KP algorithm first initializes $\mathcal{F}_v$, A_v^f and a_v^f (Line 1). Then, the algorithm sort descending request $\gamma \in U_v^f$ according to the traffic size θ_γ, and puts these requests into the set A_v^f under the constraints of NF processing capacity on switch v (Line 2–7). After the set of requests processed by each type of NF is determined, the KP algorithm sorts the decreasing NF with type $f \in \mathcal{F}'$ according the unit profit value $\frac{a_v^f}{\delta_v^f}$, where δ_v^f is the offloading cost of NF with type f on switch v (Line 8). At last, the KP algorithm greedily chooses NF that has the maximum unit profit while satisfying the switch processing capacity to determine the set of NFs offloaded on the programmable switch v (Line 9–11). The key code of the proposed algorithm is publicly available on GitHub.[4]

4.2. Performance analysis for KPBP

This section presents KPBP's approximation performance and analyze its time complexity.

The first step of the KPBP algorithm is to devise the set of legacy switches that will be upgraded. It consists of multiple iterations (say $\{1, 2, \dots, t\}$), and in each iteration $k \in \{1, 2, \dots, t\}$, a legacy switch $v(k)$ will be selected with the help of the KP algorithm. It is proved that for the 0–1 knapsack problem, KP has an approximation ratio of 2 [33]. Based on [33], we give the following theorem.

[4] https://github.com/joytuhq/Programmable-Device-Deployment

Theorem 3. *The following inequality holds for each iteration* $k \in \{1, 2, \dots, t\}$.

$$2 \cdot d_{v(k)}(k) \ge b_i(k) \tag{2}$$

In the k*th iteration,* $b_i(k)$ *denote the profit on switch* v_i *computed by KPBP and* $d_{v(k)}(k)$ *is the approximation result on the selected switch* v_k *by KP.*

Proof. In each iteration of KPBP, the KP algorithm is applied for those switches which have not been chosen to upgrade. Let $p_i(k)$ be the optimal profit of KPBP on switch v_i through the KP algorithm in the kth iteration. Thus, we have $b_i(k) \le p_i(k)$. It is straightforward that $2 \cdot d_i(k) \ge p_i(k)$ because the optimal solution of KP on switch v_i is $d_i(k)$ and KP has an approximation ratio of 2. In the kth iteration, KPBP will select a switch (say, $v(k)$). We have $d_{v(k)}(k) \ge d_i(k)$, since KPBP always replaces a legacy switch that has the maximum profit with a programmable switch. Therefore, it is concluded that $\forall k \in \{1, 2, \dots, t\}$, $2 \cdot d_{v(k)}(k) \ge b_i(k)$. □

Assume that the optimal result and KPBP will upgrade l and t legacy switches, respectively. After the kth iteration, let $\xi_f(k)$ be the number of requests that will be handled by NF with type f. The total number of requests which will be covered by all kinds of NFs is denoted as $\xi(k)$. That is, $\xi(k) = \sum_{f \in F'} \xi_f(k)$. Let F_i^* be set of NFs implemented on the programmable switch v_i in the optimal solution. In addition, after the kth iteration, the request set that will be processed by NF with type f is denoted as B_i^f. Before iteration, *i.e.* $k = 0$, it follows $\xi(0) = \sum_{i=1}^{l} \sum_{f \in F_i^*} |B_i^f|$. In the following, we demonstrate that the selected switches' overall profit will not more than a specific value. It is easy to find that, if a request set is covered by j types of NFs, it is covered j times. An integer variable is used to defined the cover times of requests, *i.e.*, $u_i \in \{1, 2, \dots, \sum_{f \in F_i^*} |B_i^f|\}$ for $\forall i \in \{1, 2..., l\}$. Therefore, we have $\xi(0) = \sum_{i=1}^{l} \sum_{f \in F_i^*} |B_i^f|$. It may include duplicated values. After that, these integer variables are sorted into a ascending sequence. For simplicity, e_x is used to indicate these values and we set them as $e_1 \le e_2 \le \dots \le e_{\xi(0)}$.

Theorem 4. *We have the following inequality for each iteration* $k \in \{1, 2, \dots, t\}$

$$2 \cdot d_{v(k)}(k) \ge e_{\xi(k)} \tag{3}$$

Proof. The requests will be incrementally covered $\xi(k)$ times after the kth iteration. In the optimal solution, each type of NF's cover ratio can be guaranteed, if the KPBP algorithm covers all rest requests. That is, $\xi(k) \le \sum_{i=1}^{l} b_i(k)$. The definition of $b_i(k)$ and e_x points that $b_i(k)$ is more than e_x. That is, for all $i \in \{1, 2, \dots, l\}$, $e_x \in \{1, 2, \dots, b_i(k)\}$. Combined with Theorem 3, it follows

$$2 \cdot d_{v(k)}(k) \ge e_x \tag{4}$$

There are at least $\xi(k)$ indices x meeting Eq. (4), due to $b_i(k) \le \max_{s \le l} \sum_{f \in F'} |B_s^f|$ and $\sum_{i=1}^{l} b_i(k) \ge \xi(k)$, $\forall i \in \{1, 2, \dots, l\}$. Combining $e_1 \le e_2 \le \dots \le e_{\xi(0)}$, we can derive that, $\forall k \in \{1, 2, \dots, t\}$, $2 \cdot d_{v(k)}(k) \ge e_{\xi(k)}$. □

For simplicity, we use n and m to indicate the maximum number of requests passing through a switch and the size of $\mathcal{F}'$, respectively. In the following, we give the approximation performance for the cost of upgraded switches through the KPBP algorithm.

Theorem 5. *The proposed KPBP algorithm has an approximation of* $2 \cdot H(m \cdot n)$ *for the cost of upgrading the legacy switches.*

Proof. We have the following inequality for each $k \in \{1, 2, \dots, t\}$ according to Theorem 4.

$$2 \cdot d_{v(k)}(k) \ge e_{\xi(k)} \ge e_{\xi(k)-1} \ge \cdots \ge e_{\xi(k+1)+1}$$

$$\Rightarrow \frac{1}{2 \cdot d_{v(k)}(k)} \leq \frac{1}{e_{\xi(k)}} \leq \frac{1}{e_{\xi(k)-1}} \leq \ldots \leq \frac{1}{e_{\xi(k+1)+1}} \tag{5}$$

Due to $d_{v(k)}(k) = \xi(k) - \xi(k+1)$, we have

$$1 \leq 2 \cdot \left(\frac{1}{e_{\xi(k)}} + \frac{1}{e_{\xi(k)-1}} + \cdots + \frac{1}{e_{\xi(k+1)+1}} \right) \tag{6}$$

Combining the above inequalities, we have

$$\begin{aligned} t &\leq 2 \cdot \left(\frac{1}{e_{\xi(1)}} + \cdots + \frac{1}{e_1} \right) \leq 2 \cdot \left(\frac{1}{e_{\xi(0)}} + \cdots + \frac{1}{e_1} \right) \\ &= 2 \cdot \sum_{i=1}^{l} H\Big(\sum_{f \in F_i^*} |B_i^f| \Big) \leq 2 \cdot l \cdot H(m \cdot n) \end{aligned} \tag{7}$$

$H(m \cdot n)$ is the $(m \cdot n)$th harmonic number defined as $H(m \cdot n) = 1 + \frac{1}{2} + \cdots + \frac{1}{m \cdot n} \approx \log(m \cdot n)$. The third equation of Eq. (7) holds due to the definition of e_x and the last inequality of Eq. (7) holds is because $\sum_{f \in F_i}^{*} |B_i^f| \leq \sum_{f \in \mathcal{F}} |\Gamma_{v_i}^f| \leq m \cdot n$. So we have

$$\frac{t}{l} \leq 2 \cdot H(m \cdot n) \tag{8}$$

Thus, we conclude that the KPBP algorithm has an approximation of $2 \cdot H(m \cdot n)$ for the cost of upgrading the legacy switches. □

Theorem 6. *The proposed KPBP algorithm has an approximation of $2.5 \cdot H(m \cdot n)$ for the cost of upgrading switches and servers.*

Proof. Let l_1 and t_1 be the number of servers that will be upgraded by the optimal solution and KPBP. Besides, we use t_2 to indicate the number of servers that will be upgraded through optimal bin packing scheme given t programmable switches. The previous work [34] has shown that the offline bin packing algorithm can achieve an approximation of 1.25. Thus, according to the definition of t_1 and t_2, we have $t_1 \leq 1.25 \cdot t_2$. Then, combining with Theorem 5, it follows

$$t_1 \leq 1.25 \cdot t_2 \leq 1.25 \cdot 2 \cdot H(m \cdot n) \cdot l_1 \tag{9}$$

So we have

$$\frac{t_1}{l_1} \leq 2.5 \cdot H(m \cdot n) \tag{10}$$

It means that for the cost of upgrading servers, the KPBP algorithm can achieve $2.5 \cdot H(m \cdot n)$-approximation. Then, combining with Eqs. (8) and (10), it follows

$$\begin{aligned} \alpha \cdot t + \beta \cdot t_1 &\leq \alpha \cdot 2 \cdot l \cdot H(m \cdot n) + \beta \cdot 2.5 \cdot H(m \cdot n) \\ &\leq 2.5 \cdot H(m \cdot n) \cdot (\alpha \cdot l + \beta \cdot l_1) \end{aligned} \tag{11}$$

So we have

$$\frac{\alpha \cdot t + \beta \cdot t_1}{\alpha \cdot l + \beta \cdot l_1} \leq 2.5 \cdot H(m \cdot n) \tag{12}$$

Therefore, the KPBP algorithm can obtain $2.5 \cdot H(m \cdot n)$-approximation for upgrading switches and servers. □

Theorem 7. *The total time complexity of the KPBP algorithm is $O(|V|^2 \cdot |F'| \cdot |\Gamma| + |S| \cdot |V|)$.*

Proof. Our proposed KPBP has two main steps. The time complexity of Line 3–6 in Algorithm 1 is $O(|V| \cdot |\mathcal{F}'|)$ in the first step. In Line 7–14, KPBP runs at most $|V|$ iterations. Note that, the time complexity of the KP algorithm is $O(|\Gamma| \cdot \log|\Gamma|)$, where $|\Gamma|$ is the number of requests. Thus, in each iteration, it takes KPBP $O(|\Gamma| \cdot \log|\Gamma|)$ to compute profit. After that, KPBP selects a switch with the maximum profit in $O(|V|)$ time complexity. At the end of each iteration in the first step, KPBP takes $O(|V| \cdot |\mathcal{F}'| \cdot |\Gamma|)$ to update the request set. Therefore, the time complexity of the first step is $O(|V|^2 \cdot |\mathcal{F}'| \cdot |\Gamma|)$. In addition, the KPBP algorithm takes $O(|S| \cdot |V|)$ time complexity to determine the set of servers that will be upgraded in the second step. In conclusion, the total time complexity of KPBP is $O(|V|^2 \cdot |\mathcal{F}'| \cdot |\Gamma| + |S| \cdot |V|)$. □

4.3. Extension to network function offloading and traffic scheduling

Once the programmable switches and SmartNICs are deployed, it is hard to change their position. However, the set of NFs offloaded on a programmable and traffic scheduling among these NFs can change according to the current network workload. For example, at time t_1, network address translation (NAT) is offloaded on programmable switch v. Several hours later, load balancer (LB) is also offloaded on this switch, so the set of NFs offloaded on programmable switch v is changed from {NAT} to {NAT, LB}. In addition, traffic will be scheduled to LB on switch v for processing. In the following, we refer to the changes in the set of NFs offloaded on a switch as NF offloading changes for simplicity. NF offloading changes are mainly to cope with traffic dynamics. In real network scenarios, network traffic changes drastically over time and places [22,35], so NF offloading should be dynamically changed to deal with traffic dynamics. To this end, we present the NF offloading and traffic scheduling (NOTS) problem in this section. When changing NF offloading, it is critical that NFs can be added and removed without disrupting the service of programmable switches. Fortunately, Xing et al. [36] have paved the way toward runtime programmable switches by investigating the necessary building blocks and proposing concrete designs for each of them. The core of the NOTS problem is to determine: (1) which NFs will be offloaded on a programmable switch; and (2) which requests will be scheduled to each offloaded NFs. Let V' be the programmable switch set. We use the binary variable z_v^f to indicate if NF with type f is offloaded on programmable switch v or not. Besides, we $w_{v,\gamma}^f$ to denote whether request γ is scheduled to the NF with type f on programmable switch v or not. The other notations used in the NOTS problem have been introduced in Section 3. We formulate the NOTS problem as follows:

$$\max \sum_{\gamma \in \Gamma} \sum_{f \in \mathcal{F}_\gamma} \sum_{v \in V'} w_{v,\gamma}^f \cdot \theta_\gamma$$

$$S.t. \begin{cases} \sum_{v \in V'} w_{v,\gamma}^f \leq 1, & \forall \gamma \in \Gamma, f \in \mathcal{F}_\gamma' \\ z_v^f \geq w_{v,\gamma}^f, & \forall v \in V', \gamma \in \Gamma, f \in \mathcal{F}' \\ \sum_{f \in \mathcal{F}'} z_v^f \cdot \delta_v^f \leq C_v, & \forall v \in V' \\ \sum_{v \in S_v} \sum_{f \in \mathcal{F}'} z_v^f \cdot \phi_f \leq C_s, & \forall s \in S' \\ \sum_{\gamma \in \Gamma : f \in \mathcal{F}_\gamma} \theta_\gamma \cdot w_{v,\gamma}^f \leq C_v^f, & \forall v \in V', f \in \mathcal{F}' \\ z_v^f, w_{v,\gamma}^f \in \{0,1\}, & \forall s, v, \gamma, f \end{cases} \tag{13}$$

The first inequality set denotes that whether request γ will be handled by NF with type f on programmable switch v or not. The second inequality set indicates that request γ can be handled by NF $f \in \mathcal{F}$ on programmable switch $v \in V'$ if and only if network function f is offloaded on programmable switch $v \in V'$. The third inequality set denotes the computing resource constraints on programmable switches. The fourth inequality set means the memory resource capacity constraints on servers. The fifth set of inequalities expresses that the load of each NF on programmable switches does not exceed the processing capacity C_v^f. Our objective is to maximize the overall traffic handled by programmable switches, *i.e.*, $\max \sum_{\gamma \in \Gamma} \sum_{f \in \mathcal{F}_\gamma} \sum_{v \in V'} w_{v,\gamma}^f \cdot \theta_\gamma$.

We propose a rounding-based NF Offloading and traffic scheduling (RBNS) algorithm to solve this problem. This algorithm mainly consists of two steps. The RBNS algorithm relax the variables z_v^f and $w_{v,\gamma}^f$ in Eq. (13) to be fractional in the first step. It means that the NFs can be deployed partially, and the traffic of each request γ can be arbitrarily distributed on multiple NFs. Since the relaxed Eq. (13) is a linear program, a linear program solver, *e.g.*, Gurobi [37] can be adopted to solve it in polynomial time. Assume that the optimal solutions for the relaxed Eq. (13) are denoted by $\{\tilde{z}_v^f\}$ and $\{\tilde{w}_{v,\gamma}^f\}$. We obtain integer

solutions $\{\hat{z}_v^f\}$ through the randomized rounding method [38]. That is, we set $\hat{z}_v^f = 1$ with probability $\tilde{z}_v^f$. If $\hat{z}_v^f = 1$, we offload NF with type f on switch v. In the second step, for each NF type in F', the RBNS algorithm sets $\hat{z}_v^f$ to 1 with probability $\{\tilde{z}_v^f\}$. Then, for NF $f \in F'_\gamma$ required by request γ, the RBNS algorithm chooses a unique NF on programmable switch v with probability $\frac{\tilde{w}_{v,\gamma}^f}{\tilde{z}_v^f}$ to satisfy the SFC requirement of each request. The RBNS algorithm is formally described in Algorithm 3.

Algorithm 3 RBNS: Rounding-based NF Offloading and Traffic Scheduling Algorithm

Step 1: Solving the Relaxed Problem
Construct a linear program by replacing with z_v^f, $w_{v,\gamma}^f \in [0,1]$ in Eq. (13)
Obtain the optimal solutions $\{\tilde{z}_v^f\}$ and $\{\tilde{w}_{v,\gamma}^f\}$
Step 2: Offloading NFs on Programmable Switches
Derive integer solutions $\hat{z}_v^f$ by randomized rounding method
for programmable switch $v \in V'$ **do**
  for each NF type $f \in \mathcal{F}'$ **do**
    Set $\hat{z}_v^f = 1$ with probability $\tilde{z}_v^f$
    if $\hat{z}_v^f = 1$ **then**
      Offload NF with type f on switch v
    end if
  end for
end for
Let V'_f be the set of programmable switches on which NF with type $f \in \mathcal{F}'$ is offloaded
for each request $\gamma \in \Gamma$ **do**
  for each NF type $f \in \mathcal{F}'_\gamma$ **do**
    Set $\hat{w}_{v,\gamma}^f = 1$ with probability $\frac{\tilde{w}_{v,\gamma}^f}{\tilde{z}_v^f}$, where $v \in V'_f$
    if $\hat{w}_{v,\gamma}^f = 1$ **then**
      request γ is processed by NF with type f offloaded on programmable switch v
    end if
  end for
end for

For the convenience of approximate ratio analysis, the minimum resource of servers and programmable switches are denoted as C_s^{min} and C_v^{min}, respectively. Besides, let $C_{s,f}^{min}$ and $C_{v,f}^{min}$ be the minimum capacity of NF with type f placed on server s and programmable switch v, respectively. Then a constant value is defined as follows:

$$\varpi = \min\left\{\frac{C_v^{min}}{\delta_v^f}, \frac{C_{v,f}^{min}}{\theta_\gamma}, \forall f \in \mathcal{F}', v \in V', \gamma \in \Gamma\right\} \tag{14}$$

Similar to the approximation analysis in [39,40], the following approximation factors of the RBNS algorithm can be obtained. When offloading NFs on programmable switches, the resource capacity constraints on programmable switches will hardly be violated by $\frac{2\log|V'|}{\varpi} + 3$. Besides, the processing capacity constraints of NFs offloaded on programmable switches will hardly be violated by a factor of $\frac{2\log|V'|}{\varpi}+3$. It means that the algorithm can achieve the optimal solution, violating the resource capacity of programmable switches and the processing capacity of NFs by at most a factor $\frac{2\log|V'|}{\varpi} + 3$ at most, which is also known as bi-criteria approximation [41]. To avoid the network congestion, the traffic controlling method can be adopted to limit each request's intensity. Moreover, similar to [42–44], the bound of RBNS can be tightened to 2 in practice. Due to limited space, we omit the detail of the proofs here.

5. Performance evaluation

We first introduce simulation settings, benchmarks, and performance metrics in Section 5.1. Then we present the large-scale simulation results in Section 5.2.

5.1. Evaluation methodology

5.1.1. Simulation settings

In the simulations, we select three typical and practical topologies as running examples. The first one is Fat-Tree [45], which consists of 128 servers and 80 switches (including 32 aggregation switches, 32 edge switches, and 16 core switches). The second one is VL2 [46], which consists of 245 servers and 70 switches. The third one is a campus network topology from Monash university [22,47], which consists of 200 servers and 100 switches. These three topologies represent various networks with different characteristics. Specifically, Fat-Tree and VL2 are for data centers and Monash topology is for campus networks. Meanwhile, Fat-Tree is a three-tier architecture design, while VL2 is a two-tier architecture design. For all these topologies, the data traces of Alibaba Cluster [48] are adopted to generate requests. Since there is no SFC information in the data traces, we generate the SFC requirement of each request according to the real SFCs [6,49,50]. There are seven NFs, Load Balancer (LB), Firewall (FW), Traffic Monitor (TM), Router, Intrusion Detection System (IDS), VPN gateway, and Network Address Translation (NAT). According to the previous works [6,49,50], we set five real SFCs (*i.e.*, $FW-LB, IDS-FW-NAT-Router, VPN-TM-FW-LB, IDS-TM-LB, NAT-FW-IDS-VPN$) and each request will be assigned with one of these real SFCs. The network operator deploys programmable switches and SmartNICs to accelerate traffic processing. We set the prices of a programmable switch and a SmartNIC as \$7500 [51] and \$2738 [52], respectively. Similar to [6], we set the resource capacity of switches: each programmable switch has 6400 action units and 46.25 MB SRAM. An NF consumes $1000 \sim 1500$ action units and $6 \sim 12$ MB SRAM [6]. Moreover, the memory consumption of an NF on a server is set to $343 \sim 525$ MB [8]. The memory resources reserved by a server for programmable switches are set to 100 GB. Besides, the system throughput of a programmable switch is set to 3.2 Tbps [11].

5.1.2. Benchmarks

The proposed network upgrade algorithm is compared with the other two benchmarks. The first one, adopted from [14], is to upgrade networks by only deploying programmable switches (ODPS) without memory expansion on external servers. Different from the KPBP algorithm, when upgrading the network, ODPS does not consider expanding programmable switches' memory space through RDMA technology supported by SmartNICs. Thus, in the case of processing all SFC requests, due to the limited memory resources of programmable switches, ODPS has to deploy a lot of programmable switches to process all the requests, resulting in a high upgrade cost. Moreover, in the case of fixed upgrade cost, the throughput of KPBP is much higher than that of ODPS.

The second benchmark upgrades the network by deploying programmable switches and configuring SmartNICs for commodity servers, which is similar to KPBP. Since there is currently no work that solves exactly the same problem proposed in this paper, we obtain this benchmark based on the modification of TEA [8]. For simplicity, this benchmark is denoted as TEA. In this method, the network upgrade process consists of the following two steps. First, it replaces the legacy switches with programmable switches. Second, for each deployed programmable switch, it selects the closest server to configure a SmartNIC as an external server for memory expansion. The difference between TEA and KPBP is that the external server provides memory expansion for only one programmable switch in the TEA method. Thus, when processing all SFC requests, this upgrade cost is slightly higher than that of KPBP.

5.1.3. Performance metrics

We adopt the following four metrics to evaluate the effectiveness of the proposed algorithm. (1) The upgrade cost. To process all the requests with SFC requirements in the network, more programmable switches and SmartNICs should be deployed when the number of requests increases. We record the total price of deployed programmable

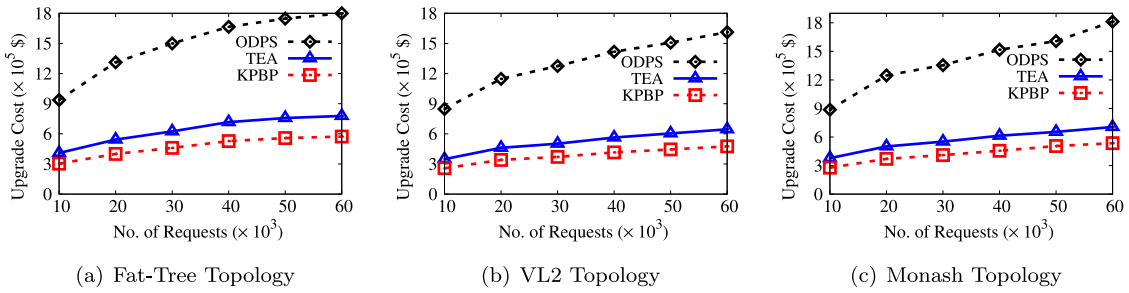


Fig. 2. Upgrade Cost vs. No. of Requests.

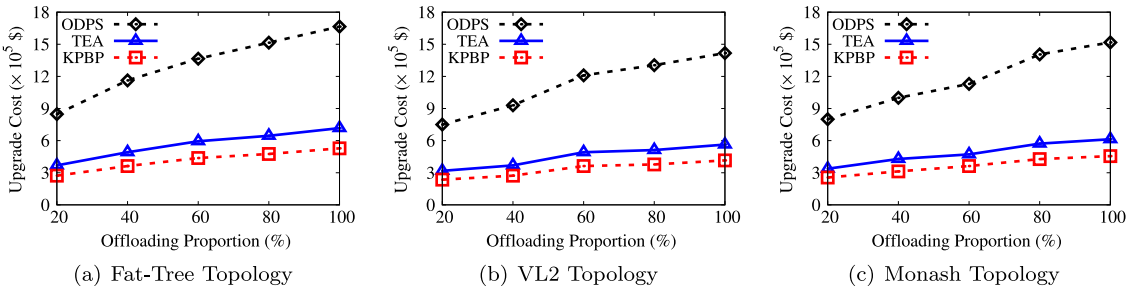


Fig. 3. Upgrade Cost vs. Offloading Proportion.

switches and SmartNICs as the upgrade cost. (2) The system throughput. After the network has been upgraded, NFs can be offloaded on the newly deployed programmable devices. We measure the maximum throughput that the offloaded NFs on programmable switches can support as the system throughput. (3) The programmable switch utilization (PSU). We divide the actual throughput of a programmable switch by its throughput capacity as the programmable switch utilization. High PSU means that the computing resources of programmable switches are fully utilized. (4) The SmartNIC utilization. Programmable switches access external servers' memory through SmartNICs. The utilization of a SmartNIC is the throughput of the SmartNIC divided by the maximum throughput. Low SmartNIC utilization means a waste of resources. The former two metrics describe the network upgrade cost and the throughput of the overall system, and the last two metrics reflect the resource utilization of the upgraded network.

5.2. Simulation results

Main observations. From our simulations, we can make the following conclusions. First, KPBP can reduce the upgrade cost by about 70.47% and 26.32%, respectively, compared with ODPS and TEA, while preserving the same system throughput. Second, given the same upgrade cost, KPBP outperforms ODPS and TEA by up to 808.19% and 82.86%, respectively, in the system throughput. Third, the KPBP algorithm can achieve similar programmable switch utilization as TEA, and improve the programmable switch utilization by about 87% compared with ODPS. Fourth, KPBP can improve the SmartNIC utilization by about 24% compared with TEA.

Upgrade cost. Figs. 2–3 exhibit the simulation results of the upgrade cost. We generate real SFC requirements for each request, and all requests should be processed by NFs offloaded on programmable switches. Then we investigate the upgrade cost through adjusting the total number of requests. The results in Fig. 2 show that the upgrade cost increases with more and more requests for all algorithms. Besides, we see that the upgrade cost of our KPBP method is much less than those of ODPS and TEA. For instance, when there are $50K$ requests for VL2 topology in the right plot of Fig. 2, the upgrade costs of ODPS, TEA, and KPBP are 15.07×10^5, 6.04×10^5 and 4.45×10^5, respectively. It means that, compared with ODPS and TEA, KPBP reduces the upgrade cost by 70.47% and 26.32%, respectively.

We define the offloading proportion as the proportion of NFs that are offloaded on programmable switches. Network operators probably cannot offload all the NFs, since some NFs are too complex to be offloaded on programmable switches or the upgrade cost is limited. For this scenario, we investigate how offloading proportion affects the upgrade cost given $40K$ requests in Fig. 3. The results show that KPBP can reduce upgrade cost by 71.11% and 26.36% compared with ODPS and TEA, respectively, given the same offloading proportion. In conclusion, KPBP outperforms ODPS and TEA in upgrade cost. That is because ODPS needs to deploy more programmable switches to handle all the requests and TEA spends more money on purchasing SmartNIC, which leads to higher network upgrade costs.

System throughput. We then investigate how the upgrade cost affects the system throughput given $40K$ requests. From the simulation results in Fig. 4, we see that the system throughputs of KPBP, ODPS and TEA increase with more upgrade cost. Besides, the system throughput of KPBP is always higher than those of ODPS and TEA. For instance, when the upgrade cost is 50×10^4 for Fat-Tree topology, as shown in the left plot of Fig. 4, the system throughputs of KPBP, TEA and ODPS are 151.93 Tbps, 82.63 Tbps and 15.50 Tbps, respectively. It illustrates that KPBP can improve system throughput by 82.86% and 808.19%, respectively, compared with TEA and ODPS. In Fig. 5, we investigate how the number of requests affects the system throughput given a fixed upgrade cost (*e.g.*, 50×10^4). We observe that the system throughput increases with more and more requests, and then flattens when the number of requests exceeds a certain value, such as $30K$ in VL2 topology, as the processing power of programmable switches is close to saturation. Similar to Fig. 4, KPBP also performs better than ODPS and TEA in terms of system throughput in Fig. 5. The reason is that KPBP can make full use of the computing power of programmable switches with memory expansion compared with ODPS. In addition, since our network upgrade method enables a server to provide memory expansion for multiple switches, KPBP can purchase more programmable switches to improve the system throughput compared with TEA, given a fixed upgrade cost.

Programmable switch utilization. Given the upgrade cost of 50×10^4, Fig. 6 shows the programmable switch utilization (PSU) performance.

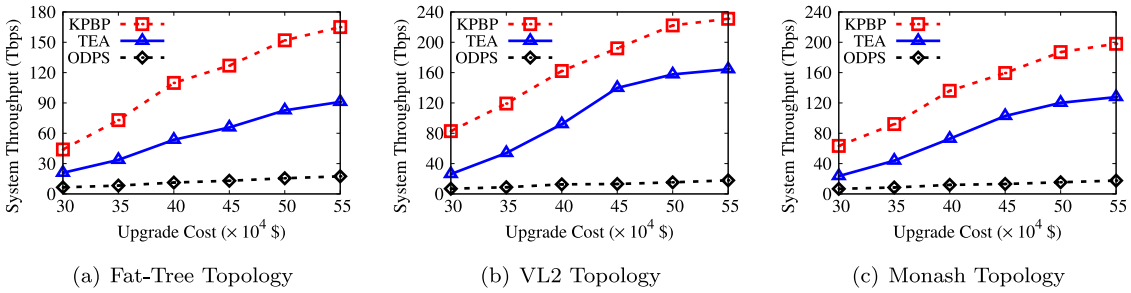


Fig. 4. System Throughput vs. Upgrade Cost.

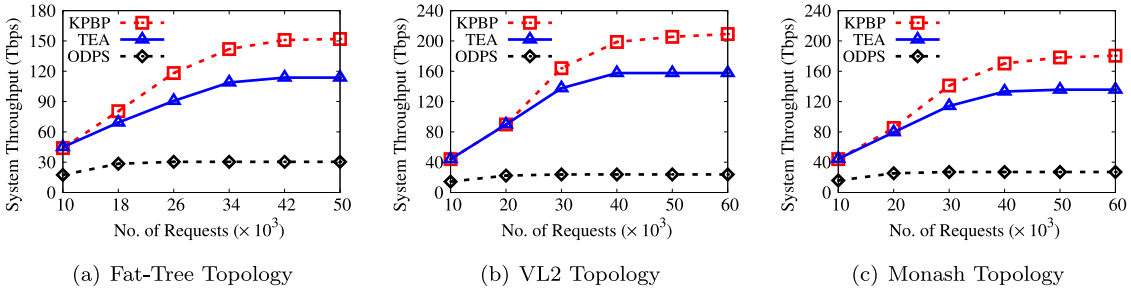


Fig. 5. System Throughput vs. No. of Requests.

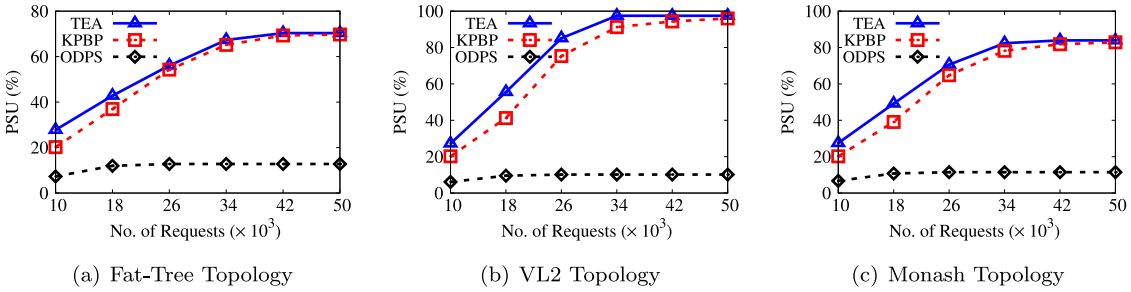


Fig. 6. Programmable Switch Utilization (PSU) vs. No. of Requests.

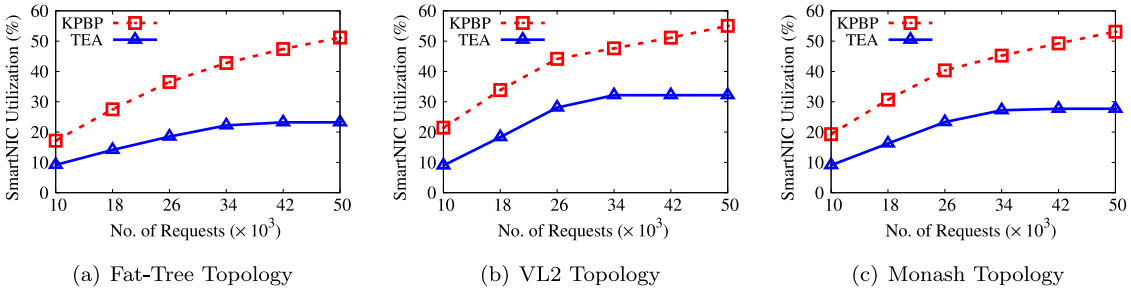
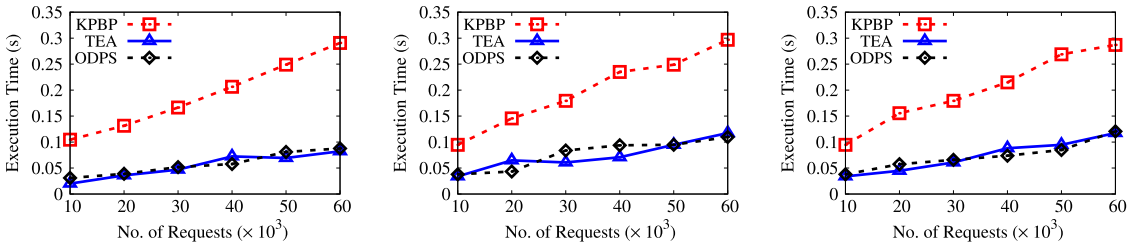


Fig. 7. SmartNIC Utilization vs. No. of Requests.

From the simulation results, we observe that the programmable switch utilization of KPBP is similar to that of TEA and greater than that of ODPS. For instance, given $34K$ requests for VL2 topology, the programmable switch utilization of KPBP, TEA, and ODPS is 91.21%, 97.47%, and 10.15%, respectively. It means that KPBP can achieve similar programmable switch utilization with TEA, and the programmable switch utilization gap between KPBP and TEA is around 6%. The main reason is that both KPBP and TEA enable programmable switches to access external memory resources so that programmable switches' computing power is fully utilized, thus improving the utilization of programmable switches. Since OPDS does not consider the limited memory resources of programmable switches, it has much lower programmable switch utilization compared with KPBP and TEA.

SmartNIC utilization. We investigate the SmartNIC utilization of KPBP and TEA under 50×10^4\$ upgrade cost in Fig. 7. Since ODPS does not adopt SmartNIC and servers to expand the memory capacity of programmable switches, this set of simulations does not include the SmartNIC utilization of ODPS. We observe that the SmartNIC utilization of KPBP is always higher than that of TEA. For example, the SmartNIC utilization gap between KPBP and TEA is 24.16% given $42K$ requests in Fat-Tree topology. KPBP supports a server with SmartNIC to provide memory expansion for multiple programmable switches, whereas TEA assumes that a server's resources are monopolized by a programmable



(a) Fat-Tree Topology (b) VL2 Topology (c) Monash Topology

Fig. 8. Execution Time vs. No. of Requests.

switch. Thus, the SmartNIC utilization of KPBP is higher than that of TEA.

Algorithm execution time. Fig. 8 shows the algorithm execution time of the proposed method and benchmarks. The results are collected from a desktop carrying an Intel i5-9256 CPU with 8 GB memory. From this simulation, we can see that the execution time increases with more and more requests in the network. Although the execution time of KPBP is longer than those of the comparison methods, KPBP can still give the network upgrade scheme in 0.3 s, given $60K$ requests. Moreover, KPBP outperforms ODPS and TEA in upgrade cost and throughput.

6. Related works

With the help of programming protocol-independent packet processors (P4) [53], apart from switch's initial functionalities like packet forwarding, programmable switches also support additional functions and packet processing details.

Since programmable switch has the advantages of low processing delay and high throughput, offloading NFs on programmable switches has attracted plenty of researchers. For example, Chen et al. [6] propose a novel system called LightNF, which facilitates NF offloading on programmable switches. It supports automatica decomposition of NF features (*e.g.*, NF performance behaviors) by performance profiling and code analysis whithout manual efforts. To generate the optimal performance offloading, it also utilize the analyzed NF features in SFC placement. Zhao et al. [9] present Flexible Network Function (FlexNF), which enables NF orchestration in fine-grained granularity on the programmable data plane to satisfy the dynamic demands of service function chain. For flexible NF orchestration, they propose an NF selection method that use the pipeline re-enter operation and labels supported by programmable switches to enable selective serving mechanism. Then they propose a service path construction algorithm based on two-stage method to realize on-path service and also achieve load balancing among links. To defend volumetric DDoS attacks, Liu et al. [10] design and implement a switch-native called Japen, which can run mitigation functions and detection totally inline on programmable switches and does not involve additional hardware in the data plane. To construct defense tactics that utilize switch capabilities with effect, they devise resource-efficient, switch-optimized detection and mitigation based on building blocks. Zhang et al. [14] design a switch-native load balancer called LBAS, which considers application server states. Without violating per-connection consistency, LBAS can store a huge number of connections with limited store space in the data plane. Moreover, to reduce the processing delay on application servers, with the help of the Ridge Regression theory, the authors design a partial dynamic weighting algorithm. Huang et al. [54] build a framework called HyperSFP to configure fault-tolerance service function chains on programmable switches. To mitigate the impact of failures in the data plane, they propose to place the active and backup network functions. When a switch failure is detected, the controller updates the forwarding rules in the routing table so that the traffic will be migrated from the failed active NFs to the backup ones.

However, the above works mainly focus on the implementation and configuration of NFs. They all assume that the network has been equipped with programmable switches. Only a few works consider the network upgrade problem. For example, Liu et al. [19] consider constructing an NFV-enabled network by deploying only commodity servers. They propose an incremental server deployment (INSD) problem and prove that the optimal solution cannot be obtained in polynomial time, and then design a approximate algorithm to solve it. Since they aims at the scenario of NFV-enabled networks, their method cannot be applied to construct a programmable network. Xue et al. [16] study the problem of transiting homogeneous NFV networks to the heterogeneous ones, in which both commodity servers and programmable switches can run NFs. They model this problem as an integer linear programming with the optimization objective of minimizing the total upgrade cost, then propose a two-step method to solve it. Nevertheless, it is hard to offload memory-intensive NFs on programmable switches, due to the limited memory resources. A promising way to expand memory is using the external servers' memory resources, which is proposed by TEA [8]. Specifically, TEA provides table abstraction for status stored across both external DRAM and local SRAM. With the help of this abstraction technique, stateful NFs offloaded on programmable switches can use 5-tuple of an data packet to look up status. The corresponding status can be found either from the remote DRAM or local SRAM using RDMA technique based on SmartNIC. TEA will prioritize lookup request and defer the processing of the packet when it accesses DRAM. This process will not block the processing of other packets in the pipeline. When the DRAM lookup completes, TEA will resume the processing of defered packet. Although Kim et al. [8] propose TEA to extend programmable switch's memory capacity, they do not present how to cost-efficiently upgrade networks for this method. To achieve efficient NF offloading, we consider deploy programmable switches and SmartNICs with minimal network upgrade cost so that NFs can be offloaded on programmable switches with external memory expansion.

7. Conclusion

In this work, we investigate the problem of upgrading networks by replacing legacy switches and equipping servers with SmartNICs so that NFs can be offloaded on programmable switches efficiently. To solve this problem, we design an efficient algorithm with an approximation ratio of $2.5 \cdot H(m \cdot n)$, where n is the maximum number of requests passing through a programmable switch, and m is the number of NF's types. The simulation results show that our solution can reduce the upgrade cost by about 70% compared with baselines.

CRediT authorship contribution statement

Huaqing Tu: Conceptualization, Methodology, Validation, Writing – original draft. **Gongming Zhao:** Formal analysis, Writing – review & editing. **Hongli Xu:** Methodology, Writing – review & editing. **Chunming Qiao:** Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

The corresponding author of this paper is Gongming Zhao. This work was supported by the National Key Research and Development Project of China (No. 2022YFB2901503), the National Science Foundation of China (No. 62372426, No. 62102392 and No. U22A2005), the National Science Foundation of Jiangsu Province (No. BK20210121), the Hefei Municipal Natural Science Foundation (No. 2022013), the Key Research Project of Zhejiang Lab (No. 2021LE0AC02), the Youth Innovation Promotion Association of Chinese Academy of Sciences (No. 2023481), and the Fundamental Research Funds for the Central Universities.

References

[1] A. Tootoonchian, A. Panda, C. Lan, M. Walls, K. Argyraki, S. Ratnasamy, S. Shenker, ResQ: Enabling SLOs in network function virtualization, in: 15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 18, 2018, pp. 283–297.

[2] J.G. Herrera, J.F. Botero, Resource allocation in NFV: A comprehensive survey, IEEE Trans. Netw. Serv. Manag. 13 (3) (2016) 518–532.

[3] A.A. Barakabitze, A. Ahmad, R. Mijumbi, A. Hines, 5G network slicing using SDN and NFV: A survey of taxonomy, architectures and future challenges, Comput. Netw. 167 (2020) 106984.

[4] I. Alam, K. Sharif, F. Li, Z. Latif, M.M. Karim, S. Biswas, B. Nour, Y. Wang, A survey of network virtualization techniques for internet of things using sdn and nfv, ACM Comput. Surv. 53 (2) (2020) 1–40.

[5] H. Tu, G. Zhao, H. Xu, Y. Zhao, Y. Qiu, L. Huang, RoNS: Robust network function services in clouds, Comput. Netw. 215 (2022) 109212.

[6] X. Chen, Q. Huang, P. Wang, Z. Meng, H. Liu, Y. Chen, D. Zhang, H. Zhou, B. Zhou, C. Wu, LightNF: Simplifying network function offloading in programmable networks, in: 2021 IEEE/ACM 29th International Symposium on Quality of Service, IWQOS, IEEE, 2021, pp. 1–10.

[7] X. Ge, Y. Liu, D.H. Du, L. Zhang, H. Guan, J. Chen, Y. Zhao, X. Hu, OpenANFV: Accelerating network function virtualization with a consolidated framework in openstack, ACM SIGCOMM Comput. Commun. Rev. 44 (4) (2014) 353–354.

[8] D. Kim, Z. Liu, Y. Zhu, C. Kim, J. Lee, V. Sekar, S. Seshan, Tea: Enabling state-intensive network functions on programmable switches, in: Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication, 2020, pp. 90–106.

[9] H. Zhao, Q. Li, J. Duan, Y. Jiang, K. Liu, FlexNF: Flexible network function orchestration on the programmable data plane, in: 2021 IEEE/ACM 29th International Symposium on Quality of Service, IWQOS, IEEE, 2021, pp. 1–6.

[10] Z. Liu, H. Namkung, G. Nikolaidis, J. Lee, C. Kim, X. Jin, V. Braverman, M. Yu, V. Sekar, Jaqen: A high-performance switch-native approach for detecting and mitigating volumetric ddos attacks with programmable switches, in: 30th USENIX Security Symposium, USENIX Security 21, 2021, pp. 3829–3846.

[11] Wedge 100B Series of Switches, URL https://people.ucsc.edu/~warner/Bufs/Barefoot%20Wedge%20100B%20OCP%20Spec-prt.pdf.

[12] Edgecore Wedge 100BF-32X, URL https://www.edge-core.com/productsInfo.php?cls=1&cls2=5&cls3=181&id=335.

[13] D.E. Eisenbud, C. Yi, C. Contavalli, C. Smith, R. Kononov, E. Mann-Hielscher, A. Cilingiroglu, B. Cheyney, W. Shang, J.D. Hosein, Maglev: A fast and reliable software network load balancer, in: 13th USENIX Symposium on Networked Systems Design and Implementation, NSDI 16, 2016, pp. 523–535.

[14] J. Zhang, S. Wen, J. Zhang, H. Chai, T. Pan, T. Huang, L. Zhang, Y. Liu, F.R. Yu, Fast switch-based load balancer considering application server states, IEEE/ACM Trans. Netw. 28 (3) (2020) 1391–1404.

[15] M. Scazzariello, T. Caiazzi, H. Ghasemirahni, T. Barbette, D. Kostić, M. Chiesa, A high-speed stateful packet processing approach for tbps programmable switches, in: 20th USENIX Symposium on Networked Systems Design and Implementation, NSDI 23, 2023, pp. 1237–1255.

[16] Y. Xue, Z. Zhu, On the upgrade of service function chains with heterogeneous NFV platforms, IEEE Trans. Netw. Serv. Manag. 18 (4) (2021) 4311–4323.

[17] M. Kablan, A. Alsudais, E. Keller, F. Le, Stateless network functions: Breaking the tight coupling of state and processing, in: 14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 17, 2017, pp. 97–112.

[18] D. Kim, J. Nelson, D.R. Ports, V. Sekar, S. Seshan, Redplane: Enabling fault-tolerant stateful in-switch applications, in: Proceedings of the 2021 ACM SIGCOMM 2021 Conference, 2021, pp. 223–244.

[19] J. Liu, H. Xu, G. Zhao, C. Qian, X. Fan, L. Huang, Incremental server deployment for scalable NFV-enabled networks, in: IEEE INFOCOM 2020-IEEE Conference on Computer Communications, IEEE, 2020, pp. 2361–2370.

[20] Z. Zeng, L. Cui, M. Qian, Z. Zhang, K. Wei, A survey on sliding window sketch for network measurement, Comput. Netw. 226 (2023) 109696.

[21] Q. Kang, L. Xue, A. Morrison, Y. Tang, A. Chen, X. Luo, Programmable in-network security for context-aware BYOD policies, in: 29th USENIX Security Symposium, USENIX Security 20, 2020, pp. 595–612.

[22] X. Fan, H. Xu, H. Huang, X. Yang, Real-time update of joint SFC and routing in software defined networks, IEEE/ACM Trans. Netw. 29 (6) (2021) 2664–2677.

[23] T. Lukovszki, M. Rost, S. Schmid, It's a match! near-optimal and incremental middlebox deployment, ACM SIGCOMM Comput. Commun. Rev. 46 (1) (2016) 30–36.

[24] V. Eramo, E. Miucci, M. Ammar, F.G. Lavacca, An approach for service function chain routing and virtual function network instance migration in network function virtualization architectures, IEEE/ACM Trans. Netw. 25 (4) (2017) 2008–2025.

[25] A. Gushchin, A. Walid, A. Tang, Scalable routing in SDN-enabled networks with consolidated middleboxes, in: Proceedings of the 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization, 2015, pp. 55–60.

[26] E.F. Kfoury, J. Crichigno, E. Bou-Harb, An exhaustive survey on p4 programmable data plane switches: Taxonomy, applications, challenges, and future trends, IEEE Access 9 (2021) 87094–87155.

[27] C. Gao, X. Yao, T. Weise, J. Li, An efficient local search heuristic with row weighting for the unicost set covering problem, European J. Oper. Res. 246 (3) (2015) 750–761.

[28] U. Feige, A threshold of ln n for approximating set cover, J. ACM 45 (4) (1998) 634–652.

[29] R. Raz, S. Safra, A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP, in: Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, 1997, pp. 475–484.

[30] S.M. Marcus, Acoustic determinants of perceptual center (P-center) location, Percept. Psychophys. 30 (3) (1981) 247–256.

[31] O.H. Ibarra, C.E. Kim, Fast approximation algorithms for the knapsack and sum of subset problems, J. ACM 22 (4) (1975) 463–468.

[32] S. Sahni, Approximate algorithms for the 0/1 knapsack problem, J. ACM 22 (1) (1975) 115–124.

[33] A. Gupta, M. Pál, R. Ravi, A. Sinha, What about Wednesday? Approximation algorithms for multistage stochastic optimization, in: Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques, Springer, 2005, pp. 86–98.

[34] D.S. Johnson, Near-Optimal Bin Packing Algorithms (Ph.D. thesis), Massachusetts Institute of Technology, 1973.

[35] J. Pei, P. Hong, M. Pan, J. Liu, J. Zhou, Optimal VNF placement via deep reinforcement learning in SDN/NFV-enabled networks, IEEE J. Sel. Areas Commun. 38 (2) (2019) 263–278.

[36] J. Xing, K.-F. Hsu, M. Kadosh, A. Lo, Y. Piasetzky, A. Krishnamurthy, A. Chen, Runtime programmable switches, in: 19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 22, 2022, pp. 651–665.

[37] R. Anand, D. Aggarwal, V. Kumar, A comparative analysis of optimization solvers, J. Stat. Manag. Syst. 20 (4) (2017) 623–635.

[38] P. Raghavan, C.D. Tompson, Randomized rounding: a technique for provably good algorithms and algorithmic proofs, Combinatorica 7 (4) (1987) 365–374.

[39] H. Tu, G. Zhao, H. Xu, Y. Zhao, Y. Zhai, A robustness-aware real-time SFC routing update scheme in multi-tenant clouds, IEEE/ACM Trans. Netw. (2022).

[40] H. Tu, G. Zhao, H. Xu, Y. Zhao, Y. Zhai, Robustness-aware real-time sfc routing update in multi-tenant clouds, in: 2021 IEEE/ACM 29th International Symposium on Quality of Service, IWQOS, IEEE, 2021, pp. 1–6.

[41] R. Cohen, L. Lewin-Eytan, J.S. Naor, D. Raz, On the effect of forwarding table size on SDN network utilization, in: IEEE INFOCOM 2014-IEEE Conference on Computer Communications, IEEE, 2014, pp. 1734–1742.

[42] G. Zhao, H. Xu, J. Liu, C. Qian, J. Ge, L. Huang, Safe-me: Scalable and flexible middlebox policy enforcement with software defined networking, in: 2019 IEEE 27th International Conference on Network Protocols, ICNP, IEEE, 2019, pp. 1–11.

[43] J. Wang, G. Zhao, H. Xu, Y. Zhai, Q. Zhang, H. Huang, Y. Yang, A robust service mapping scheme for multi-tenant clouds, IEEE/ACM Trans. Netw. (2021).

[44] D. Feldman, A. Fiat, M. Sharir, D. Segev, Bi-criteria linear-time approximations for generalized k-mean/median/center, in: Proceedings of the Twenty-Third Annual Symposium on Computational Geometry, 2007, pp. 19–26.

[45] M. Al-Fares, A. Loukissas, A. Vahdat, A scalable, commodity data center network architecture, ACM SIGCOMM Comput. Commun. Rev. 38 (4) (2008) 63–74.

[46] A. Greenberg, J.R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D.A. Maltz, P. Patel, S. Sengupta, VL2: A scalable and flexible data center network, in: Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication, 2009, pp. 51–62.

[47] H. Song, S. Guo, P. Li, G. Liu, FCNR: fast and consistent network reconfiguration with low latency for SDN, Comput. Netw. 193 (2021) 108113.
[48] Alibaba cluster data trace, URL https://github.com/alibaba/clusterdata.
[49] C. Sun, J. Bi, Z. Zheng, H. Yu, H. Hu, NFP: Enabling network function parallelism in NFV, in: Proceedings of the Conference of the ACM Special Interest Group on Data Communication, 2017, pp. 43–56.
[50] X. Chen, D. Zhang, X. Wang, K. Zhu, H. Zhou, P4SC: Towards high-performance service function chain implementation on the P4-capable device, in: 2019 IFIP/IEEE Symposium on Integrated Network and Service Management, IM, IEEE, 2019, pp. 1–9.
[51] Barefoot Tofino, URL https://bm-switch.com/index.php/netberg-aurora710-100g-bms.html.
[52] SmartNIC, URL https://itprice.com/cisco-gpl/smartnic.
[53] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, et al., P4: Programming protocol-independent packet processors, ACM SIGCOMM Comput. Commun. Rev. 44 (3) (2014) 87–95.
[54] H. Huang, W. Wu, HyperSFP: Fault-tolerant service function chain provision on programmable switches in data centers, in: NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium, IEEE, 2022, pp. 1–9.

Huaqing Tu received the Ph.D. degree in computer science at the University of Science and Technology of China in 2023. She is currently a postdoctoral fellow with Zhejiang Lab in China. Her main research interests are software-defined networks, programmable networks and cloud computing.

Gongming Zhao received the Ph.D. degree in computer software and theory from the University of Science and Technology of China in 2020. He is currently an Associate Professor with the University of Science and Technology of China. His current research interests include software-defined networks and cloud computing.

Hongli Xu received the B.S. degree in computer science and the Ph.D. degree in computer software and theory from the University of Science and Technology of China in 2002 and 2007, respectively. He is currently a Professor with the School of Computer Science and Technology, University of Science and Technology of China. He has authored or coauthored over 70 papers, and held about 30 patents. His main research interest is software-defined networks, cooperative communication, and vehicular ad hoc network.

Chunming Qiao (Fellow, IEEE) is a SUNY distinguished professor and also the current chair of the Computer Science and Engineering Department, University at Buffalo, Buffalo, NY. He was elected to IEEE fellow for his contributions to optical and wireless network architectures and protocols. His current focus is on connected and autonomous vehicles. He has published extensively with an h-index of more than 69. Two of his papers have received the best paper awards from IEEE and Joint ACM/IEEE venues. He also has seven US patents and served as a consultant for several IT and Telecommunications companies since 2000. His research has been funded by a dozen major IT and telecommunications companies including Cisco and Google, and more than a dozen NSF grants. He has chaired and co-chaired a dozen of international conferences and workshops, and IEEE Technical Committees and Subcommittee. He has served on the editorial board for several leading IEEE journal. He was elected to IEEE Fellow for his contributions to optical and wireless network architectures and protocols.