# Adaptive SmartNIC Offloading for Unleashing the Performance of Protocol-Oblivious Forwarding

Qian Zhang, Nirwan Ansari, *Fellow, IEEE* and Zuqing Zhu, *Fellow, IEEE*

***Abstract*—The growth of Internet-of-Things (IoT) has led to the convergence of heterogeneous networking systems powered by various protocols, and consequently the emergence of protocol-independent packet processing based on programmable data plane (PDP) for IoT. In this work, we study how to leverage the hardware acceleration enabled by offloading flow tables to SmartNIC to improve the performance of software PDP switches based on protocol-oblivious forwarding (POF). We design our SmartNIC offloading system (namely, OVS-POF-TC) based on the Linux kernel traffic classification (TC) system and Open vSwitch (OVS), extend OVS to enable the installation of POF-based flow tables (POF-FTs) in a SmartNIC, and design a selective offloading mechanism for purposely offloading heavy-load POF-FTs. Our experimental results indicate that OVS-POF-TC offloads and updates POF-FTs timely, supports runtime programmability, and has improved packet processing throughput by $1.52\times$ and $3.82\times$, when applying POF-FTs with *SetField* and *AddField* to packets, respectively. Moreover, to ensure that OVS-POF-TC can offload and replace POF-FTs adaptively, we formulate two mixed integer linear programming (MILP) models to respectively solve the problems of flow placement and flow replacement, and also design time-efficient heuristics for them.**

***Index Terms*—Programmable data plane (PDP), Open vSwitch (OVS), Protocol-oblivious forwarding (POF), SmartNIC.**

## I. Introduction

NOWADAYS, Internet, the glue of modern technological infrastructures, has undergone revolutionary changes [1, 2], and various new networking paradigms and technologies have been developed to adapt to the ever-increasing network services [3–8]. Among them, software-defined networking (SDN) [9], network virtualization [10–12], and network function virtualization (NFV) [13–16] are the important innovations to provide flexible, dynamic and automated network architecture for the ecosystem of Internet-of-Things (IoT) [17–19]. As IoT is a convergence of heterogeneous network systems, including embedded systems, wireless sensor networks, control and automation systems, *etc.*, various protocols can run simultaneously in different network segments of an IoT system [20, 21]. Therefore, the protocol-independent packet processing enabled by programmable data plane (PDP) is essential for interconnecting heterogeneous IoT systems effectively and realizing efficient network operations within the IoT systems.

Specifically, PDP enables network operators to define new packet fields and customize packet processing pipelines. It can be realized with the programming protocol-independent packet processors (P4) [22] or protocol-oblivious forwarding (POF) [23]. P4 specifies the rules of writing and compiling packet processing programs, and with it, we can customize the protocols and processing of packets in a PDP switch in two stages, *i.e.*, pipeline configuration before running and flow rule insertion in runtime [24]. Although P4-based hardware PDP switches can realize high-performance protocol-independent packet processing, they still bear two drawbacks, which would limit their applications in IoT systems. First, their power and cost can exceed the related budgets of lightweight IoT devices. For instance, the power consumption of a PDP switch based on the well-known Tofino chips is $\sim$600 W [25], while the power budget for an IoT device in power-constrained environments can be only a few watts or less [26, 27]. Second, the need of creating and compiling a .p4 file in P4-based PDP switches restricts their flexibility for IoT applications (*i.e.*, new protocols can hardly be added in runtime [23]).

On the other hand, POF does not rely on specific programmable chips but defines an underlying primitive instruction set [28], with which an SDN controller can program PDP switches in runtime by installing protocol-oblivious flow tables and building packet processing pipelines with them. This makes it convenient to leverage POF to design and implement lightweight software PDP switches for IoT. Therefore, we extended the famous Open vSwitch (OVS) [29, 30] to support POF, realized a software PDP switch (namely, OVS-POF) in [31], whose source codes can be found in [32]. Specifically, with the software acceleration facilitated by data plane development kit (DPDK) [33], OVS-POF achieved a packet processing capacity of $\sim$4.5 million packets per second (Mpps). Nevertheless, for small-sized packets (*e.g.*, 64 bytes), the throughput of OVS-POF still cannot reach 10 Gbps. This is because certain POF actions (*i.e.*, *SetField* and *AddField*) invoke time-consuming memory operations and take many CPU cycles. Hence, even though OVS-POF can be considered as a potential candidate of lightweight software PDP switches for IoT, efforts are still needed to improve its throughput.

This has motivated us to study how to fully unleash the performance of POF-based software PDP switches for IoT. Here, one promising approach is to consider the hardware acceleration with SmartNIC. This is because SmartNIC [34, 35] is a lightweight and cost-effective platform that consists of programmable hardware cores, dedicated packet engines, and a variety of domain-specific accelerators for accelerating packet processing [36]. Hence, we can offload operations of a software PDP switch to SmartNICs for hardware acceleration. For instance, major SmartNIC manufacturers have already

Q. Zhang and Z. Zhu are with the School of Information Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, P. R. China (email: zqzhu@ieee.org).

N. Ansari is with the Advanced Networking Laboratory, Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102, USA (email: nirwan.ansari@njit.edu).

Manuscript received July 1, 2022.

developed the drivers and firmware for offloading OVS onto their SmartNICs, to greatly improve the packet processing performance of OVS [37]. However, since the underlying primitive instruction set of POF is completely different from that of OpenFlow [28], we cannot directly leverage these existing techniques to offload OVS-POF onto SmartNICs.

This work studies how to realize adaptive SmartNIC offloading to unleash the packet processing performance of OVS-POF. Specifically, we select POF-based flow tables (POF-FTs) from the software system of OVS-POF in runtime and offload the selected POF-FTs into a SmartNIC for hardware acceleration, such that packet processing throughput of the resulting software/hardware system can be effectively improved. Note that, our SmartNIC offloading scheme is an "adaptive" one, that is, it can choose the best POF-FTs to offload based on the current status of the software/hardware system such that the throughput improvement achieved by the offloading is maximized. Hence, our proposed SmartNIC offloading scheme is different from the conventional ones that simply offload flow tables in the first-come-first-serve way and stop when the SmartNIC's memory is used up [38]. Meanwhile, to ensure that our proposal is adaptive, we need to cover both system design and implementation and algorithm design. To the best of our knowledge, this work is the first one that realizes adaptive POF-based SmartNIC offloading.

As for the system part, we design our SmartNIC offloading system based on the Linux kernel traffic classification (TC) subsystem, extend the offloading scheme for OVS [38] to enable the installation of POF-FTs in a SmartNIC, and upgrade OVS-POF to implement an adaptive offloading mechanism for choosing proper POF-FTs to offload. The proposed PDP switch, namely, OVS-POF-TC, is evaluated in a real-world network testbed. Experimental results indicate that OVS-POF-TC can support the features of OVS-POF for runtime programmability, offload and update POF-FTs adaptively, and provide significantly improved packet processing throughput.

As for the algorithm part, we formulate two mixed integer linear programming (MILP) models to respectively tackle the problems of flow placement and flow replacement, to ensure that OVS-POF-TC can always offload and replace POF-FTs adaptively. The MILPs consider realistic parameters from our experimental measurements to maximize packet processing throughput. We also propose time-efficient heuristics for the problems to shorten the decision time of adaptive SmartNIC offloading. Simulation results confirm that even for very large-scale problems, the proposed heuristics run fast enough and can approximate the optimal solutions of the MILPs well.

The rest of the paper is organized as follows. Section II briefly surveys the related work. We present the system design and implementation of OVS-POF-TC in Section III, and the experiments for evaluating its performance are discussed in Section IV. Section V describes the algorithm design for adaptive SmartNIC offloading, and simulation results are shown in Section VI. Finally, Section VII summarizes the paper.

## II. Related Work

The centralized network control and management provided by SDN has been considered as a key technology to address the heterogeneous network environments for IoT [39–41]. The study in [39] discussed the SDN architecture for wireless sensor and actuator networks, and an interface protocol was proposed to bridge the communications between SDN controller and IoT devices. Muppet [40] is a P4-based multi-protocol switch for large-scale IoT deployment and service automation. An SDN controller [41] was designed to support various wireless communication protocols for heterogeneous and complex IoT networks. To further improve the programmability of data plane for IoT applications, the evolution of PDP networks and cloud-oriented network virtualization have promoted the research and development (R&D) on SmartNICs. SmartNICs can be architected based on various types of chips, including application-specific integrated circuits (ASICs), field-programmable gate arrays (FPGAs), and network processors units (NPUs) [42]. The ASIC-based SmartNICs usually can achieve excellent packet processing performance due to their high core-count, but their programmability is not as good as that of FPGA- and NPU-based SmartNICs and thus they only have limited deployment.

SmartNIC has enabled a number of applications. For instance, Li *et al.* [43] used SmartNICs to extend the primitives of remote direct memory access (RDMA) and enable remote key-value access to host memory, while the study in [44] tried to accelerate the data processing in a server by extending critical software components of the server onto SmartNICs. However, these applications did not consider the hardware acceleration of packet processing with SmartNIC offloading. Gao *et al.* [45] developed OVS-CAB, which is a SmartNIC offloading system for OVS, and designed a rule-caching algorithm to improve the hit rate of flow rules in a SmartNIC for enhancing the throughput of packet processing. Nevertheless, as we have already explained, OVS is still protocol-dependent and thus cannot work as a software PDP switch for IoT.

In addition to POF-FT offloading, SmartNICs can also be directly programmed with the P4 language to realize hardware PDP switches [46]. Although this leverages the advantages of P4 to fully offload the data plane, it can hardly change packet processing pipelines in runtime due to the working principle of P4-enable chips (*i.e.*, the reconfiguration of a P4-based PDP switch has to go through the configuration and runtime stages [24]). For example, the results in [46] suggested that it took a few seconds for a SmartNIC to be reprogrammed for new packet processing pipelines, while the latency could still be too long in a highly dynamic IoT environment.

The aforementioned drawbacks of existing approaches motivated us to study how to combine POF, which has runtime programmability, and SmartNIC offloading, to realize a runtime configurable and accelerated hybrid software/hardware PDP switch system. Such a hybrid software/hardware PDP switch (*i.e.*, OVS-POF-TC) leverages our previous work on POF-based software PDP switches, whose development path can be summarized as follows. We started the project based on the POF protocol and software architecture proposed in [47]. By leveraging DPDK, the first version of our POF-based software switch (PVS) achieved a packet processing rate of 1 Gbps [23]. Then, we optimized the processing logic of PVS and implemented a flow table management scheme in it, to

improve its throughput to 10 Gbps for packets with sizes of 512 bytes or longer [48]. Next, we decided to switch to implementing POF-based software PDP switches based on OVS, because it has superior performance and a supportive ecosystem for development. This led to OVS-POF [31, 32], which could achieve a throughput of 10 Gbps when the packet size was set as 256 bytes. In the following, we will leverage adaptive SmartNIC offloading to further improve the throughput of OVS-POF, and use it for performance benchmarking.

TABLE I
MAJOR ABBREVIATIONS

| Abbrev. | Full Name |
|---|---|
| SDN | Software-defined networking |
| POF-FTs | POF-based flow tables |
| IoT | Internet-of-Things |
| P4 | Programming protocol-independent packet processors |
| PDP | Programmable data plane |
| MILP | Mixed integer linear programming |
| POF | Protocol-oblivious forwarding |
| DPDK | Data plane development kit |
| TC | Linux kernel traffic classification |
| OVS | Open vSwitch |

## III. System Design and Implementation

In this section, we describe the system design of OVS-POF-TC and the implementation of SmartNIC offloading in it. Since several abbreviations are frequently used in this paper, we list them here in Table I for the convenience of readers.

### A. System Design

Different from the SmartNIC offloading of OVS, OVS-POF-TC needs to offload POF-FTs and their processing from Linux kernel to SmartNIC. In order to make the data plane completely protocol-independent, POF uses a tuple of <*offset, length, value*> to generalize the description of a packet field [23, 47], where *offset* denotes the start bit-location of the field in a packet, *length* indicates the field's length in bits, and *value* describes the content of the field. Meanwhile, POF defines an underlying primitive instruction set (*i.e.*, POF-FIS [28]) to enable PDP switches to operate on the POF-FTs. Therefore, the SDN controller can compose packet processing pipelines based on any type of packet fields without referring to a specific protocol, encode the pipelines as POF-FTs, and install them in PDP switches in runtime. This ensures the runtime programmability that can hitlessly update the pipelines in POF-based PDP switches in milliseconds. On the other hand, considering the generality of POF-FTs, it would be challenging to realize SmartNIC offloading for them.

We design our SmartNIC offloading system based on the Linux kernel TC subsystem, extend the offloading scheme for OVS [38] to facilitate the installation of POF-FTs in a SmartNIC. The TC subsystem has a few classifiers, among which the TC flower classifier can support the match-action mechanism. Specifically, its TC match can check packet fields and tunnel metadata in layers 2-4, while its TC action supports normal actions on packets (*e.g.*, modify, drop and output). Note that in version 4.6 of Linux kernel, the support for the hardware offloading of TC flow classifier has been added to the driver of Netronome SmartNICs [37]. Hence, we architect OVS-POF-TC based on the SmartNIC of Netronome Agilio CX 2×40GbE. Fig. 1 explains the principle of the SmartNIC offloading considered in this work, which orchestrates the processing of packets over three data paths:

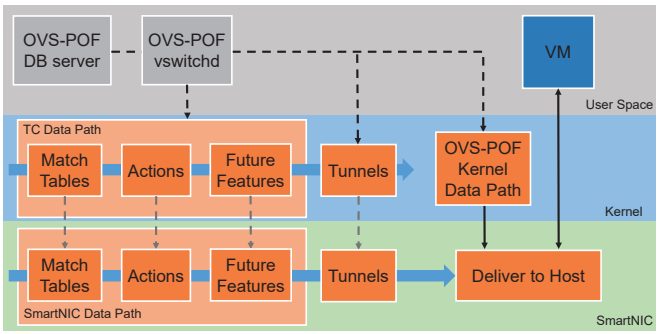



Fig. 1. Principle of SmartNIC offloading in OVS-POF-TC.

- *Kernel data path*: This is the fast data path for packet processing in OVS-POF [31], which resides in Linux kernel for improved throughput. However, its performance is limited by the frequent interrupt calls of Linux kernel, which can compete for CPU resources with it.
- *TC data path*: It resides in Linux kernel and consists of TC match and TC action. It normally does not process packets, but only serves as a tunnel to transfer POF-FTs from Linux user space to SmartNIC.
- *SmartNIC data path*: It executes the hardware acceleration for packet processing in SmartNIC, with the POF-FTs received from the TC data path.

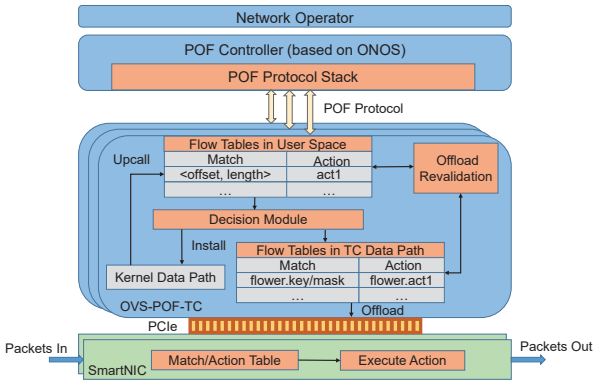



Fig. 2. System architecture of OVS-POF-TC.

Fig. 2 shows the overall system architecture of OVS-POF-TC. The control plane is realized with a POF controller, which is developed based on ONOS [49] (*i.e.*, we extended the southbound protocol stack to support POF-based control messages such as *FlowMod* and *TableMod*). The data plane consists of a software OVS-POF-TC switch running on a general-purpose server and a SmartNIC. Our work to extend OVS-POF for OVS-POF-TC is detailed below.

- We extend the TC data path to respectively map the match/action of POF-FTs to TC match/action. Then, by leveraging the hardware offloading of TC flower classifier, we can offload POF-FTs to the SmartNIC.

- We design and implement a decision module to determine whether a POF-FT should be installed in the kernel data path or the TC data path. Specifically, the decision module uses a flow placement algorithm to offload POF-FTs to the SmartNIC adaptively, *i.e.*, it checks the action(s) in each POF-FT and optimizes the SmartNIC offloading scheme to maximize the performance of OVS-POF-TC.
- In order to maintain the optimality of SmartNIC offloading during dynamic operation, we design and implement an offload revalidation module. It checks the POF-FTs in the SmartNIC, and leverages a flow replacement algorithm to determine how to offload, replace and delete POF-FTs in the SmartNIC adaptively.

We will design the flow placement and flow replacement algorithms in Section V. During operation, if a POF-FT is installed in the kernel data path, the packets that correspond to it will be processed in the software system of OVS-POF-TC. Otherwise, if a POF-FT is offloaded to the SmartNIC data path, its packets will be processed directly in the SmartNIC, without taking any CPU resources of the server.

### B. System Implementation

We implement OVS-POF-TC by extending the OVS-POF that was developed based on OVS v2.13, and the key implementations are summarized as follows.

*1) Offload Support for POF-FTs:* To realize the hardware acceleration of POF-based software PDP switches, our design uses the TC data path as the tunnel to transfer POF-FTs from Linux user space to SmartNIC. In the SmartNIC offloading of OVS [38], TC data path uses *flower.key/mask* as the match item, where the *flow.key* refers to a packet field that has been defined in OpenFlow according to existing protocols, and the *mask* is a bitmap that can be applied on the *flow.key*. Hence, after extracting a match field with the *flower.key*, OVS applies the *mask* on it to get the real match item. However, the match items in POF-FTs are in the form of <*offset, length*>, not based on specific protocols. Therefore, we program the TC data path to first extract a field based on the tuple of <*offset, length*> and then fill it in *flower.key*. As for the *mask*, we let the TC data path derive it from the wildcards in each POF-FT.

We program OVS-POF-TC to support the offloading of two representative heavy-load POF-FIS actions, *i.e.*, *AddField* and *SetField* [50]. We realize *SetField* by leveraging the *pedit* action in TC action. As for *AddField*, we hope to point out that due to the restriction of the SmartNIC's driver, we have to sacrifice certain flexibility of the *AddField* in POF-FIS. Specifically, the SmartNIC's driver does not support the insertion of packet fields in arbitrary lengths, and the length of a new field has to follow a fixed granularity (*e.g.*, 4 bytes). We implement *AddField* based on the *push_mpls* action in TC action, by merging the subfields in the multi-protocol label switching (MPLS) field to obtain an empty field in 4 bytes. Then, the *AddField* can insert any fields whose length is a multiple of 4 bytes into packets. Note that this compromise actually will not significantly restrict the application of OVS-POF-TC because most of the known network techniques involving packet field insertion have the lengths of their new fields in the granularity of 4 bytes, such as MPLS, in-band network telemetry (INT) [51], and segment routing (SR) [52].

*2) Decision Module:* The decision module checks the action(s) in each POF-FT and quickly decides whether to offload it to the SmartNIC according to the flow placement algorithm. If not, decision module will instruct OVS-POF-TC to install the POF-FT in the kernel data path.

*3) Offload Revalidation:* During dynamic operation, OVS-POF-TC should monitor the POF-FTs that have been offloaded onto the SmartNIC in realtime and update them adaptively if necessary. Specifically, when the POF-FTs have been changed in the software OVS-POF-TC switch, the offload revalidation module needs to capture these changes immediately, revalidate and update the POF-FTs in the SmartNIC to ensure that they are up-to-date and should still be placed there.

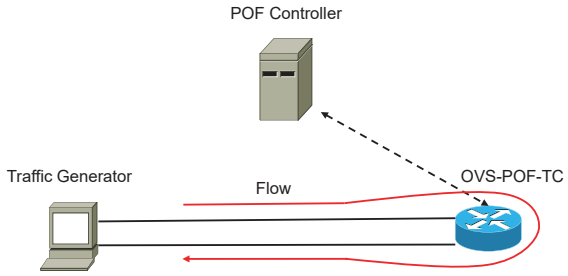



Fig. 3. Experimental setup.

As OVS has a revalidator thread that periodically monitors the flow tables in the kernel data path, we implement the offload revalidation module based on it. Specifically, we program the offload revalidation module to monitor the status of the POF-FTs on the SmartNIC periodically (*e.g.*, every one second) and update them according to the decision made by the flow replacement algorithm. Moreover, the offload revalidation module is also in charge of the POF-FT synchronization between software OVS-POF-TC switch and SmartNIC.

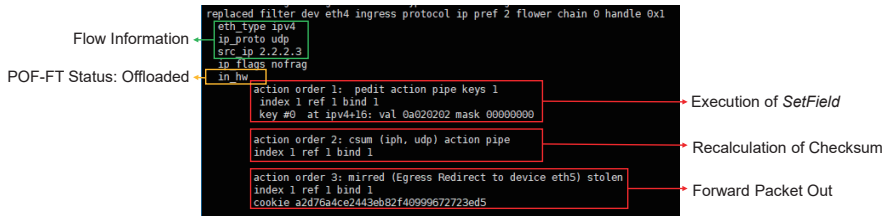
## IV. Experimental Evaluations

In this section, we perform experiments to validate the design of OVS-POF-TC and evaluate its performance. Fig. 3 shows the experimental setup that consists of a OVS-POF-TC system, an ONOS-based POF controller, and a traffic generator. The OVS-POF-TC system includes a Linux server equipped with a Netronome Agilio CX 2×40GbE SmartNIC. The traffic generator is a commercial product that can perform traffic analysis at the data-rate up to 40 Gbps.

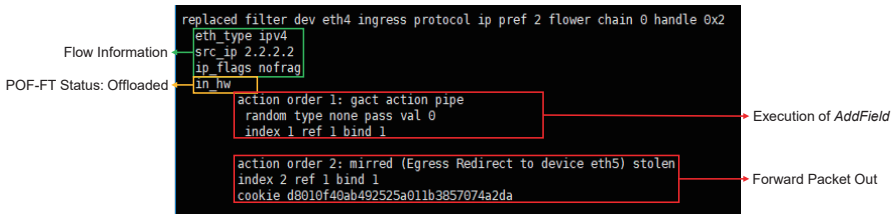
### A. Feature Validation

We first conduct experiments to validate that the functionalities designed for OVS-POF-TC have been realized correctly.

*1) Offloading POF-FTs to SmartNIC:* To verify that OVS-POF-TC can offload POF-FTs to the SmartNIC correctly, we perform an experiment to offload two POF-FTs. The first POF-FT has its match item in the form of <*offset, length, value*> as $<208, 32, 0x02020203>$, and its action is *SetField* $<240, 32, 0x0a020202>$, *i.e.*, it matches to the flow whose source IP address is 2.2.2.3 and modifies the flow's destination




(a) Status of the POF-FT that executes *SetField*




(b) Status of the POF-FT that executes *AddField*

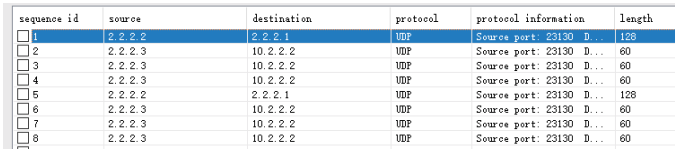
Fig. 4. Results of SmartNIC offloading obtained by TC monitor of TC flower classifier.

IP address to 10.2.2.2. The match item of the second POF-FT is $<208, 32, 0x02020202>$, and its action is *AddField* $<112, 32, 0x0a0a0908>$, *i.e.*, the POF-FT matches to the flow whose source IP address is 2.2.2.2 and inserts a 4-byte field of $0x0a0a0908$ in each of its packets after the Ethernet header.

Figure 4 shows the experimental results obtained by the TC monitor of the TC flower classifier, which can reveal the status of SmartNIC offloading. Fig. 4(a) shows that the first POF-FT has been offloaded to the SmartNIC successfully (*i.e.*, suggested by the flag "*in_hw*") and the SmartNIC data path processes a packet correctly: 1) executing the *SetField* action to modify the destination IP address, 2) recalculating the IP header checksum, and 3) forwarding the packet out through the port of *eth5*. Similarly, Fig. 4(b) shows the offloading result of the second POF-FT, where the *AddField* action is first executed to insert a new field in the packet and then the packet is forwarded out through *eth5*.

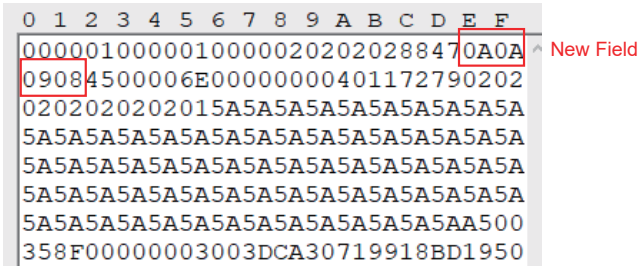
In order to further confirm that the SmartNIC has correctly executed the offloaded POF-FTs on related packets, we use the traffic generator to analyze the received packets and plot the results in Fig. 5. Fig. 5(a) suggests that the first POF-FT has been executed correctly, because all the packets whose source IP addresses are 2.2.2.3 have their destination IP addresses been modified to 10.2.2.2. The packet content in Fig. 5(b) indicates that a new 4-byte field of $0x0a0a0908$ has been inserted in a packet by the second POF-FT in the SmartNIC. In summary, the results in Figs. 4 and 5 confirm that the SmartNIC offloading in OVS-POF-TC has been implemented correctly for handling POF-FTs well.

*2) Selective Offloading by Decision Module:* Then, we conduct an experiment to verify the feature of selective offloading enabled by the decision module. Specifically, we implement a simple flow placement algorithm in the decision module to offload all the POF-FTs that contain *SetField* or *AddField*. Then, we let the traffic generator generate three flows with different source-destination IP address pairs and send them through the OVS-POF-TC to loop back. Meanwhile, the POF controller installs three POF-FTs for the flows, respectively, and among the POF-FTs, the first one contains *SetField* and



| sequence id | source | destination | protocol | protocol information | length |
|---|---|---|---|---|---|
| 1 | 2.2.2.2 | 2.2.2.1 | UDP | Source port: 23130 D... | 128 |
| 2 | 2.2.2.3 | 10.2.2.2 | UDP | Source port: 23130 D... | 60 |
| 3 | 2.2.2.3 | 10.2.2.2 | UDP | Source port: 23130 D... | 60 |
| 4 | 2.2.2.3 | 10.2.2.2 | UDP | Source port: 23130 D... | 60 |
| 5 | 2.2.2.2 | 2.2.2.1 | UDP | Source port: 23130 D... | 128 |
| 6 | 2.2.2.3 | 10.2.2.2 | UDP | Source port: 23130 D... | 60 |
| 7 | 2.2.2.3 | 10.2.2.2 | UDP | Source port: 23130 D... | 60 |
| 8 | 2.2.2.3 | 10.2.2.2 | UDP | Source port: 23130 D... | 60 |

(a) List of received packets




(b) Content of a packet experienced the second POF-FT

Fig. 5. Packet captures obtained by traffic generator.

*Output* actions, the second one includes *AddField* and *Output*, while the last one only contains *Output*.

Figure 6 shows the packet processing latencies on OVS-POF-TC for the three flows. We can clearly see that the packet processing latency of the last flow (55.04 $\mu$s) is much longer than those of the first two (3.99 $\mu$s and 4.00 $\mu$s) because the decision module selects the POF-FTs of the first two flows to be offloaded and the flows are processed with the hardware acceleration in the SmartNIC. On the other hand, the last flow (*i.e.*, the one whose POF-FT only does not contain *SetField* or *AddField*) is not offloaded and gets processed in the kernel data path of OVS-POF-TC. Therefore, the results in Fig. 6 verify the functionality of the decision module.

*3) Offload Revalidation and Runtime-programmability:* Finally, we design an experiment to validate the functionality of the offload revalidation module and confirm the runtime programmability of OVS-POF-TC. The experiment lets the traffic generator generate a flow at 20 Gbps with the packet size of 64 bytes. For the flow, the POF controller first installs a POF-FT that only contains *Output* in the OVS-POF-TC, which

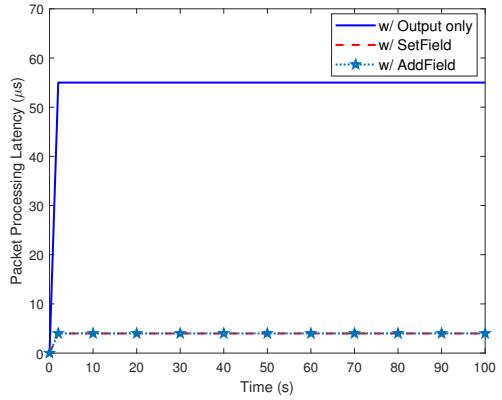



Fig. 6. Results on packet processing latency for verifying selective offloading.

offloads the POF-FT to the SmartNIC immediately, and after ∼50 seconds, the controller updates the POF-FT to include *AddField* and *Output*, which is synchronized to the SmartNIC.

Figure 7 illustrates how the throughput of the flow changes over time. We can see that with SmartNIC offloading, the flow's throughput is 28.64 Mpps at the beginning, when its POF-FT only contains *Output*, which corresponds to a maximum data-rate of 19.25 Gbps. At ∼50 seconds, there is a throughput change, which is caused by the POF controller updating the flow's POF-FT. We can clearly see that the POF-FT update only decreases the throughput a little bit but does not interrupt it, and the transition is in milliseconds only. This indicates that the offload revalidation module captures the POF-FT update accurately and performs a revalidation to synchronize the new POF-FT to the SmartNIC on time, which verifies the functionality of the offload revalidation module and confirms the runtime programmability of OVS-POF-TC.

Meanwhile, we observe that after the POF-FT update, the flow's throughput changes to 26.65 Mpps (a data-rate of 18.76 Gbps). This is because compared to *Output* only, the actions of *AddField* and *Output* are more heavy-load in terms of the complexity of packet processing [50], especially for the POF-based *AddField*. Hence, even with the hardware acceleration in the SmartNIC, the flow's throughput still decreases slightly.

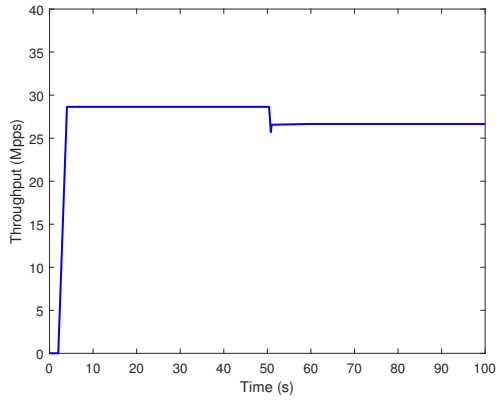



Fig. 7. Throughput of flow when its POF-FT changes in runtime.

Moreover, we measure the average total latency in OVS-POF-TC for offloading/removing a set of POF-FTs to/from the SmartNIC, and plot the results in Fig. 8. Specifically, we offload/remove different numbers of POF-FTs to/from the SmartNIC in OVS-POF-TC and obtain the total latency, and for each data point in Fig. 8, we repeat the experiment for 20 times and show the average result. It can be seen that the average total latency is around 23 ms for all the test cases, and it does not change significantly with the number of POF-FTs. This is because the latency mainly comes from the communication latency between the SmartNIC and server and the running time of the decision module or offload revalidation module, which are generally irrelevant to the number of POF-FTs. This confirms not only the runtime programmability but also the scalability of OVS-POF-TC. In all, the average total latencies of offloading and removing a set of POF-FTs in OVS-POF-TC are 22.5 ms and 25.3 ms, respectively.

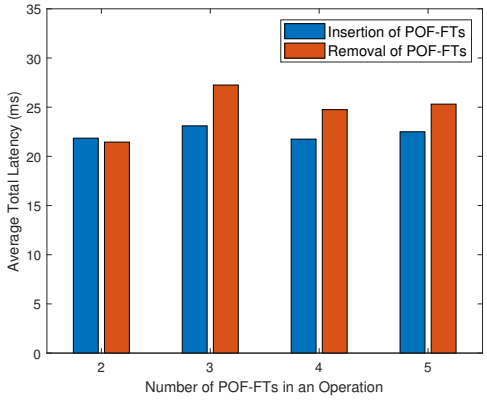



Fig. 8. Latency of offloading/removing POF-FTs to/from in OVS-POF-TC.

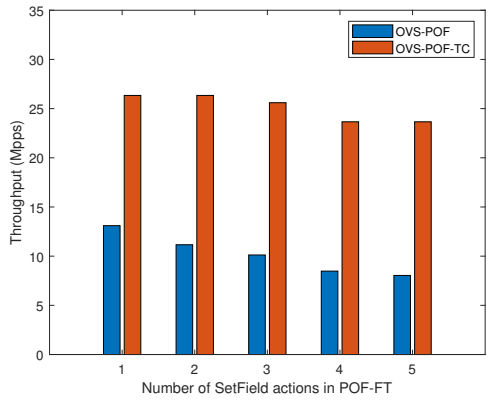
### B. Performance Benchmarking

In order to clearly present the advantages of the hardware acceleration in OVS-POF-TC, we use the existing POF-based software PDP switch in the literature (*i.e.*, OVS-POF [31]) as the benchmark to compare their throughputs. Specifically, we use the POF controller to install POF-FTs with different numbers of *SetField* actions or *AddField* actions in OVS-POF-TC and OVS-POF, where OVS-POF-TC offloads the POF-FTs to the SmartNIC for hardware acceleration, while OVS-POF just handles the POF-FTs in the software system. Then, we use the traffic generator to measure the throughputs of OVS-POF-TC and OVS-POF with the packet size of 64 bytes (*i.e.*, the shortest packets for stressing the PDP switches mostly).

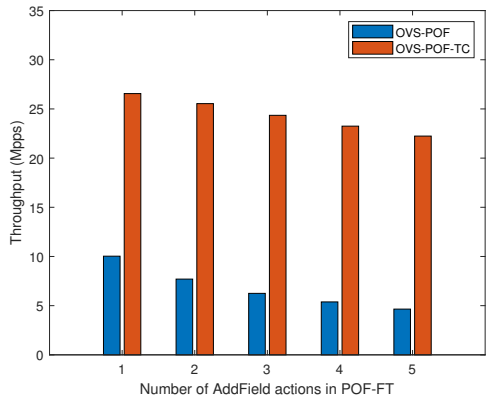
Fig. 9 shows the results on packet processing throughput. After analyzing the results, we can get the following insights. First, the hardware acceleration enabled by our SmartNIC offloading does significantly improve the packet processing performance of POF-based PDP switch. Specifically, for the cases with *SetField* actions, OVS-POF-TC increases the throughput by 1.94 times at maximum and 1.52 times on average, related to OVS-POF, and for the cases with *AddField* actions, the acceleration ratios achieved by OVS-POF-TC over OVS-POF have a maximum of 4.78 times and an average of 3.28 times. There are two main reasons for the improvements achieved

by OVS-POF-TC: 1) the offloaded flow is processed directly by the hardware of SmartNIC, saving the time for sending its packets to the software system of OVS-POF-TC and thus increasing the throughput, and 2) the CPU cycles used by the memory operations of *SetField* and *AddField* are also saved.

Second, because the packet processing related to *AddField* is more complex than that of *SetField* in OVS-POF, its throughputs decrease significantly from Fig. 9(a) to Fig. 9(b). However, as OVS-POF-TC processes packets in hardware without taking any CPU resources, its throughputs in Figs. 9(a) and 9(b) are similar. Finally, for the reason that when there are more *SetField*/*AddField* actions in the POF-FT, the packet processing becomes more complex, the throughput of OVS-POF decreases greatly with the number of actions in Figs. 9(a) and 9(b). On the other hand, the throughput decrease of OVS-POF-TC due to the increase of actions is much less obvious in Fig. 9. Therefore, the results in Fig. 9 suggest that with our SmartNIC offloading, OVS-POF-TC achieves much larger throughput than OVS-POF and its packet processing performance is much less sensitive to the content of POF-FTs.




(a) SmartNIC offloading of *SetField* actions




(b) SmartNIC offloading of *AddField* actions

Fig. 9. Performance benchmarking of OVS-POF-TC's throughput.

Moreover, to further evaluate the performance of OVS-POF-TC, we design a more sophisticated experimental scenario that involves multiple flows. Specifically, we send 10 flows with different total data-rates to OVS-POF-TC and turn the SmartNIC offloading on and off to check the throughput and

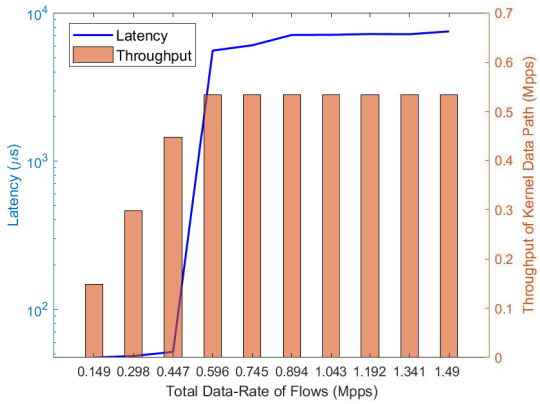



Fig. 10. Performance of kernel data path in OVS-POF-TC (SmartNIC offloading is turned off).

latency of packet processing. We first turn on the SmartNIC offloading and confirm that the throughput of OVS-POF-TC always equals the total data-rates of the flows, and the average latency of packet processing is $\sim$4 $\mu$s. Then, we turn off the SmartNIC offloading, and obtain the results in Fig. 10. This time, the maximum throughput of the OVS-POF-TC is only 0.534 Mpps. This is because in this case, all the packets are processed by the kernel data path in the software system; to support SmartNIC offloading in OVS-POF-TC, the software acceleration enabled by DPDK [33] cannot be used. Meanwhile, Fig. 10 also shows that the latency increases abruptly from $\sim$50 $\mu$s to $\sim$7,000 $\mu$s when the total data-rate of the flows increases from 0.447 Mpps to 0.596 Mpps. This is because when the input data-rate exceeds the maximum throughput of the kernel data path, the CPU for the software system is overloaded, *i.e.*, the software part of OVS-POF-TC can hardly process packets timely. The results in Fig. 10 further justify the necessity of our proposed adaptive SmartNIC offloading (*i.e.*, the offloading schemes of flows should be determined according to their packet processing performance in the different data paths in OVS-POF-TC).

## V. Algorithm Design

Although the OVS-POF-TC designed in Section III can effectively improve the packet processing performance of OVS-POF, the performance improvement can hardly be maximized without an adaptive algorithm that can select the best POF-FTs to offload in runtime, based on the current status of the software/hardware system of OVS-POF-TC. In other words, as the memory in a SmartNIC for POF-FTs is normally limited, the conventional offloading scheme that simply offloads POF-FTs in the first-come-first-serve manner cannot achieve the maximum performance improvement, and thus should be further optimized. Hence, in this section, we design algorithms to facilitate adaptive SmartNIC offloading, which enable OVS-POF-TC to select proper POF-FTs to offload by collecting and analyzing flow status information and update offloaded POF-FTs timely to address dynamic network changes. Specifically, we propose flow placement and flow replacement algorithms

to be implemented in the decision module and offload revalidation module in OVS-POF-TC, respectively.

### A. MILP Models

The performance of OVS-POF-TC can mainly be affected by three factors: 1) the memory space in the SmartNIC for POF-FTs, 2) the latency of offloading/removing a POF-FT to/from the SmartNIC, and 3) the packet processing latency of a flow when it is processed in the kernel and SmartNIC data paths in OVS-POF-TC. Hence, we formulate the problems of flow placement and replacement based on the parameters measured in a real-world system, and design MILP models.

Note that with numerous experimental measurements, we find that the packet processing latency of a flow in different data paths of OVS-POF-TC can be empirically modeled as

$$T_i = \begin{cases} \tau_2, & \text{in kernel and } C_i \le C_i^{kr}, \\ \tau_3, & \text{in kernel and } C_i > C_i^{kr}, \\ \tau_1, & \text{in SmartNIC}, \end{cases} \tag{1}$$

where $T_i$, $C_i$ and $C_i^{kr}$ are the packet processing latency, sending rate, and the kernel data path's throughput of *Flow* $i$, respectively. As for the MILPs of flow placement and replacement problems, below are the common parameters.

**Common Notations:**

- $N$: the set of POF-FTs that the POF controller installs in OVS-POF-TC, each of which corresponds to a flow, and thus the number of flows to be considered is $|N|$.
- $M^{sn}$: the maximum number of POF-FTs that can be offloaded to the SmartNIC according to its memory space.
- $\tau_{add}$: the average latency of offloading a set of POF-FTs to the SmartNIC in OVS-POF-TC.
- $C_i^{kr}$: the throughput of *Flow* $i$ when it is processed in the kernel data path of OVS-POF-TC.
- $C_i$: the sending rate of *Flow* $i$.
- $\tau_1$, $\tau_2$ and $\tau_3$: the average packet processing latencies in OVS-POF-TC, for the conditions defined in Eq. (1).
- $M$: a very large positive constant.
- $\epsilon$: a small positive constant.

*1) MILP for Flow Placement (MILP-FP):*

**Variables:**

- $x_i$: the boolean variable that equals 1 if the POF-FT of *Flow* $i$ is offloaded to the SmartNIC, and 0 otherwise.
- $y_i$: the boolean variable that equals 1 if the POF-FT of *Flow* $i$ is in the kernel data path, and 0 otherwise.
- $T_i$: the packet processing latency of *Flow* $i$.
- $\gamma$: the boolean variable that equals 1 if there is at least one POF-FT being offloaded to the SmartNIC.
- $a_i$, $f_{i,1}$, $f_{i,2}$, $f_{i,3}$, $g_1$, $g_2$ and $g_3$: the auxiliary variables introduced for linearization.

**Objective:**

As both the packet processing latencies of flows and the latency of offloading POF-FTs to the SmartNIC can affect the performance of OVS-POF-TC, we design the optimization objective as to minimize the sum of them:

$$\text{Minimize} \quad \gamma \cdot \tau_{add} + \sum_{\forall i \in N} T_i. \tag{2}$$

**Constraints:**

$$\sum_{\forall i} x_i \le M^{sn}. \tag{3}$$

Eq. (3) ensures that the POF-FTs offloaded to the SmartNIC can be accommodated by its memory space.

$$x_i + y_i = 1, \quad \forall i \in N. \tag{4}$$

Eq. (4) ensures that each POF-FT in $N$ is deployed either in the SmartNIC or the kernel data path.

$$T_i = \begin{cases} \tau_2, & C_i \le C_i^{kr} \text{ and } x_i = 0, \\ \tau_3, & C_i > C_i^{kr} \text{ and } x_i = 0, \\ \tau_1, & x_i = 1, \end{cases} \quad \forall i. \tag{5}$$

Eq. (5) gets the packet processing latency of *Flow* $i$ according to Eq. (1). It is nonlinear and will be linearized later.

$$\gamma = \begin{cases} 0, & \sum_{\forall i} x_i = 0, \\ 1, & \text{otherwise}. \end{cases} \tag{6}$$

Eq. (6) ensures that the value of $\gamma$ is set correctly, which is also a nonlinear constraint that needs to be linearized later.

$$a_i = \frac{C_i}{C_i^{kr}}, \tag{7}$$

$$f_{i,1} + f_{i,2} + f_{i,3} = 1, \tag{8}$$

$$\tau_1 \cdot f_{i,1} + \tau_2 \cdot f_{i,2} + \tau_3 \cdot f_{i,3} = T_i, \tag{9}$$

$$a_i \cdot (1 - x_i) \ge \epsilon + f_{i,3}, \tag{10}$$

$$a_i \cdot (1 - x_i) \le f_{i,2} + M \cdot f_{i,3}, \tag{11}$$

$$g_1 + g_2 + g_3 = 1, \tag{12}$$

$$\gamma = g_1 + g_3, \tag{13}$$

$$\sum_{\forall i} x_i \le -\epsilon \cdot g_1 + M \cdot g_3, \tag{14}$$

$$\sum_{\forall i} x_i \ge -M \cdot g_1 + \epsilon \cdot g_3. \tag{15}$$

Eqs. (7)-(11) leverage variables $a_i$, $f_{i,1}$, $f_{i,2}$ and $f_{i,3}$ to linearize Eq. (5), and Eqs. (12)-(15) use variables $g_1$, $g_2$ and $g_3$ to linearize Eq. (6). Finally, we can linearize the objective in Eq. (2) with the auxiliary variables as

$$\text{Minimize } (g_1 + g_3) \cdot \tau_{add} + \sum_{\forall i} (\tau_1 f_{i,1} + \tau_2 f_{i,2} + \tau_3 f_{i,3}). \tag{16}$$

*2) MILP for Flow Replacement (MILP-FR):*

In addition to the common notations defined above, the MILP for flow replacement uses the following parameters.

**Notations:**

- $N^{nw}$: the set of new POF-FTs that the controller installs in OVS-POF-TC, each of which is for a new flow.
- $\tau_{del}$: the average latency of removing a set of POF-FTs from the SmartNIC in OVS-POF-TC.
- $z_i$: the boolean that equals 1 of the POF-FT of an existing flow $i \in N$ was in SmartNIC before the flow replacement, and 0 if the POF-FT is in kernel data path.

**Variables:**

- $x_i$, $y_i$, $T_i$ and $\gamma$: same variables as those in MILP-FP.
- $\xi$: the boolean variable that equals 1 if there is at least one POF-FT being removed from the SmartNIC.

- $a_i$, $f_{i,1}$, $f_{i,2}$, $f_{i,3}$, $h_1$, $h_2$, $k_1$ and $k_2$: the auxiliary variables introduced for linearization.

**Objective:**

The optimization objective is similar to that of the flow placement problem, except that it should also consider the latency of removing POF-FTs from the SmartNIC.

$$\text{Minimize} \quad \gamma \cdot \tau_{add} + \xi \cdot \tau_{del} + \sum_{\forall i \in (N \cup N^{nw})} T_i. \tag{17}$$

**Constraints:**

$$\sum_{\forall i \in (N \cup N^{nw})} x_i \leq M^{sn}, \tag{18}$$

$$x_i + y_i = 1, \quad \forall i, \tag{19}$$

$$T_i = \begin{cases} \tau_2, & C_i \leq C_i^{kr} \text{ and } x_i = 0, \\ \tau_3, & C_i > C_i^{kr} \text{ and } x_i = 0, \\ \tau_1, & x_i = 1, \end{cases} \quad \forall i, \tag{20}$$

$$\gamma = \begin{cases} 0, & \sum_{\forall i \in N^{nw}} x_i = 0, \\ 1, & \text{otherwise.} \end{cases} \tag{21}$$

Eqs. (18)-(21) are similar to Eqs. (3)-(6).

$$\xi = \begin{cases} 0, & \sum_{i \in N} x_i \geq \sum_{i \in N} z_i, \\ 1, & \text{otherwise.} \end{cases} \tag{22}$$

Eq. (22) ensures that the value of $\xi$ is set correctly, which is also a nonlinear constraint that needs to be linearized later.

$$h_1 + h_2 = 1, \tag{23}$$

$$\gamma = h_2, \tag{24}$$

$$\sum_{\forall i \in N^{nw}} x_i \leq |N^{nw}| \cdot h_2, \tag{25}$$

$$\sum_{\forall i \in N^{nw}} x_i \geq \epsilon \cdot h_2, \tag{26}$$

$$k_1 + k_2 = 1, \tag{27}$$

$$\xi = k_2, \tag{28}$$

$$\sum_{\forall i \in N} x_i \leq k_1 \cdot M + k_2 \cdot \sum_{\forall i \in N} z_i, \tag{29}$$

$$\sum_{\forall i \in N} x_i \geq k_1 \cdot \sum_{\forall i \in N} z_i. \tag{30}$$

Eqs. (23)-(26) linearize Eq. (21), and Eqs. (27)-(30) linearize Eq. (22). Finally, we can linearize the objective in Eq. (17) as

$$\text{Minimize } h_2\tau_{add} + k_2\tau_{del} + \sum_{\forall i}(\tau_1 f_{i,1} + \tau_2 f_{i,2} + \tau_3 f_{i,3}). \tag{31}$$

### *B. Heuristic Algorithms*

Note that MILP models might not be scalable and solving them can be time-consuming, especially for large-scale problems. Meanwhile, to maintain the runtime-programmability of OVS-POF-TC, we have to solve the flow placement and replacement problems quickly. Therefore, we propose two heuristics to solve the problems time-efficiently. When solving the MILPs, we find that the arranging POF-FTs according to the sending rates of their flows helps to minimize the objectives. Therefore, the heuristics are designed accordingly.

*1) Heuristic for Flow Placement: Algorithm* 1 shows the procedure of our heuristic for flow placement, which decides which POF-FTs should be offloaded to the SmartNIC when it is empty after system initialization. Here, the motivation of introducing the set $B$ to store the kernel data path's throughputs for different packet sizes is that we would like to determine whether to offload each flow's POF-FT quickly. Specifically, the kernel data path's throughput increases with a flow's average packet size, and thus we select the throughputs of a few representative packet sizes to divide the main search in *Algorithm* 1 into branches, for saving running time.

**Algorithm 1:** Flow Placement Algorithm

**Input**: parameters of MILP-FP, the set $B$ that stores kernel data path's throughputs for the packet sizes of $\{1024, 512, 256, 128, 64\}$ bytes, and the coefficients $n_1$ and $n_2$.
**Output**: the set of offloaded POF-FTs $N'$.

```
1  N' = ∅, flag = 0;
2  sort flows in N in descending order of sending rates;
3  for each throughput b ∈ B in descending order do
4      P_0 = P_1 = P_2 = ∅;
5      insert each flow i ∈ N with C_i > b into set P_0;
6      for each flow i ∈ P_0 do
7          if C_i > C_i^kr then
8              insert Flow i into set P_2;
9          else
10             insert Flow i into set P_1;
11         end
12     end
13     p_1 = |P_1|, p_2 = |P_2|;
14     if (n_2 < p_2 ≤ M^sn) OR (n_1 < p_1 ≤ M^sn) then
15         N' = N, flag = 1;
16         break;
17     end
18     if p_2 > M^sn then
19         insert the first M^sn flows in P_2 into N';
20         flag = 1;
21         break;
22     end
23     if p_1 + p_2 > M^sn then
24         insert the flows in P_2 into N';
25         insert the first M^sn − p_2 flows in P_1 into N';
26         flag = 1;
27         break;
28     end
29 end
30 if flag = 0 then
31     if (τ_3 − τ_1) · p_2 + (τ_2 − τ_1) · p_1 > τ_add then
32         N' = N;
33     end
34 end
35 return N';
```

*Lines* 1-2 are for the initialization, where $flag$ is introduced to indicate whether the algorithm has finalized the set of offloaded POF-FTs in $N'$. Then, the for-loop covering *Lines*

3-29 divide the flows in $N$ into a few subsets by comparing their sending rates with the throughputs of kernel data path in $B$, for reducing the time complexity of the search. Specifically, the throughput of kernel data path increases with the packet size. Hence, in *Line* 4, we store the flows whose sending rates is greater than the current throughput of kernel path $b$ in set $P_0$, for processing the flows with larger sending rates earlier. Next, *Lines* 6-13 put the flows in $P_0$ in $P_1$ or $P_2$ depending whether their sending rates are larger than the throughputs with which kernel data path can process them, and the numbers of flows in $P_1$ and $P_2$ are stored in $p_1$ and $p_2$, respectively (*Line* 13).

Then, *Lines* 14-28 select flows to put into $N'$ according to system parameters. Note that $n_1$ and $n_2$ are the coefficients empirically set according to the values of $\tau_1$, $\tau_2$, $\tau_3$ and $\tau_{add}$, and we have $n_1 = 500$ and $n_2 = 3$ in the simulations in the next section. Finally, if $N'$ still has not been determined, *Lines* $30-34$ finalize it according to the latencies in the system. The time complexity of *Algorithm* 1 is $O(|B| \cdot |N|)$.

*2) Heuristic for Flow Replacement: Algorithm* 2 shows the procedure of our heuristic for flow replacement, which decides how to replace the existing POF-FTs in the SmartNIC when new POF-FTs are installed in the OVS-POF-TC. This time, the SmartNIC might not be empty, and the POF-FTs that have already been offloaded to it are stored in set $N'$.

Similar to *Algorithm* 1, we still leverage the set of kernel data path's throughputs $B$ to save running time. Therefore, the procedure in *Lines* 4-14 is similar to that in the *Lines* 4-13 of *Algorithm* 1. Then, *Lines* 15-25 check whether there exists the condition of flow replacement, *i.e.*, replacing certain POF-FTs currently in the SmartNIC with new ones helps to reduce the objective in Eq. (17). Here, the coefficient $n_3$, which is used in *Line* 15, is empirically set according to the values of $\tau_1$, $\tau_2$, $\tau_3$, $\tau_{add}$ and $\tau_{del}$, and we have $n_3 = 6$ in the simulations in the next section. Finally, *Lines* 24-30 take care of the situation in which the SmartNIC was empty before the flow replacement. The time complexity of *Algorithm* 2 is $O(2 \cdot |B| \cdot (|N| + |N^{nw}|))$.

**Algorithm 2:** Flow Replacement Algorithm

**Input**: parameters of MILP-FR, the set $B$ that stores kernel data path's throughputs for the packet sizes of $\{1024, 512, 256, 128, 64\}$ bytes, and the coefficient $n_3$.
**Output**: the new set of offloaded POF-FTs $N''$.

1 $N'' = N'$, $flag = 0$;
2 sort flows in $N$ and $N^{nw}$ in descending order of sending rates;
3 **for** *each throughput* $b \in B$ *in descending order* **do**
4  $P_0 = P_1 = P_2 = Q_0 = Q_1 = Q_2 = \emptyset$;
5  insert each flow $i \in N$ with $C_i > b$ into set $P_0$;
6  **for** *each flow* $i \in P_0$ **do**
7  **if** $C_i > C_i^{kr}$ **then**
8  insert *Flow* $i$ into set $P_2$;
9  **else**
10  insert *Flow* $i$ into set $P_1$;
11  **end**
12  **end**
13  $p_1 = |P_1|$, $p_2 = |P_2|$;
14  repeat the procedure in *Lines* 5-13 for flows in $N^{nw}$ to get $Q_0$, $Q_1$, $Q_2$, $q_1$ and $q_2$ accordingly;
15  **if** $(p_1 > n_3)$ *AND* $(q_2 > n_3)$ *AND* $(|N| > M^{sn})$ **then**
16  **for** $i \in [0, \min(p_1, q_2)]$ **do**
17  remove each flow $i \in P_1$ from $N''$;
18  insert each flow $i \in Q_2$ into $N''$;
19  **end**
20  $flag = 1$;
21  **break**;
22  **end**
23 **end**
24 **if** $flag = 0$ **then**
25  **if** $N' = \emptyset$ **then**
26  run *Algorithm* 1 with flow set $N \cup N^{nw}$;
27  **else if** $|N| < M^{sn}$ **then**
28  run *Algorithm* 1 with flow set $N^{nw}$ and SmartNIC memory space $M^{sn} = M^{sn} - |N|$;
29  **end**
30 **end**
31 **return** $N''$;

## VI. Numerical Simulations

In this section, we evaluate the algorithms designed for flow placement and flow replacement with numerical simulations. In the simulations, we assume that the maximum number of POF-FTs that can be offloaded on the SmartNIC is $M^{sn} = 4,800$. The sizes of the packets in each flow are randomly selected from $\{64, 128, 256, 512, 1024\}$ bytes, and the sending rate of each flow is randomly distributed within $[100, 40000]$ Mbps. The throughput of kernel data path for each flow ($C_i^{kr}$) is determined according to experimental measurement.

Tables II and III show the simulation results regarding flow placement and flow replacement, respectively. We can see that our proposed heuristics run $\sim$10 times faster than their corresponding MILPs, while the MILPs can become intractable when the scales of the problems are the largest. Meanwhile, the heuristics can approximate the optimal solutions from the MILPs well. Specifically, the maximum relative gaps between the heuristics and MILPs are $0.203\%$ and $3.23\%$ for flow placement and flow replacement, respectively. Therefore, the results prove the performance of our heuristics. Note that the simulations can use very large numbers of POF-FTs in $N$ and $N^{nw}$ to stress the algorithms, but in practical network systems, the POF controller might not install hundreds of thousands or millions of POF-FTs into an OVS-POF-TC in one batch.

## VII. Conclusion

In this paper, we designed and implemented a SmartNIC offloading system, namely, OVS-POF-TC, to improve the performance of POF-based software PDP switches. Our study covered both system design and implementation and algorithm design. Experimental results indicate that our proposal facilitated runtime programmability, offloaded and updated POF-FTs adaptively, and significantly improved the packet processing performance of POF-based software PDP switches.

TABLE II
PERFORMANCE EVALUATIONS FOR FLOW PLACEMENT

| | MILP-FP | | Flow Placement Heuristic | |
|---|---|---|---|---|
| # of POF-FTs ($\|N\|$) | Overall Objective ($\mu$s) | Running Time (s) | Overall Objective ($\mu$s) | Running Time (s) |
| 100 | 23,000 | 0.01 | 23,000 | 0.001 |
| 1,000 | 27,500 | 0.03 | 27,500 | 0.001 |
| 10,000 | 36,372,000 | 0.16 | 36,379,400 | 0.0070 |
| 100,000 | 684,612,000 | 1.51 | 684,611,550 | 0.1249 |
| 1,000,000 | 7,171,470,000 | 26.78 | 7,171,470,100 | 1.5211 |
| 3,000,000 | – | – | 21,595,875,650 | 6.0512 |

TABLE III
PERFORMANCE EVALUATIONS FOR FLOW REPLACEMENT

| | | MILP-FR | | Flow Replacement Heuristic | |
|---|---|---|---|---|---|
| # of POF-FTs | | Overall Objective ($\mu$s) | Running Time (s) | Overall Objective ($\mu$s) | Running Time (s) |
| $\|N\|$ | $\|N^{nw}\|$ | | | | |
| 60 | 40 | 23,000 | 0.01 | 23,000 | 0.001 |
| 600 | 400 | 27,500 | 0.01 | 27,500 | 0.0040 |
| 6,000 | 4,000 | 36,267,500 | 0.16 | 37,440,250 | 0.0170 |
| 60,000 | 40,000 | 685,200,000 | 1.55 | 686,260,950 | 0.2289 |
| 600,000 | 400,000 | 7,172,680,000 | 26.12 | 7,173,737,850 | 3.2770 |
| 600,000 | 2,400,000 | – | – | 21,595,200,650 | 12.4293 |

Moreover, to make sure that OVS-POF-TC can offload and replace POF-FTs adaptively, we formulated two MILPs to respectively solve the problems of flow placement and flow replacement, and also proposed time-efficient heuristics for the problems. Simulation results confirmed that the heuristics run much faster than the MILPs and could approximate the optimal solutions of the MILPs well. In our future work, we will extend OVS-POF-TC to support the offloading of more types of POF-FTs and consider the actual applications of OVS-POF-TC for low-latency, flexible and scalable IoT networks.

## ACKNOWLEDGMENTS

This work was supported by NSFC project 61871357 and Fundamental Fund for Central Universities (WK3500000006).

## REFERENCES

[1] Cisco Annual Internet Report (2018-2023). [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

[2] P. Lu *et al.*, "Highly efficient data migration and backup for Big Data applications in elastic optical inter-data-center networks," *IEEE Netw.*, vol. 29, pp. 36–42, Sept./Oct. 2015.

[3] P. Marsch *et al.*, "5G radio access network architecture: Design guidelines and key considerations," *IEEE Commun. Mag.*, vol. 54, pp. 24–32, Nov. 2016.

[4] W. Lu *et al.*, "AI-assisted knowledge-defined network orchestration for energy-efficient data center networks," *IEEE Commun. Mag.*, vol. 58, pp. 86–92, Jan. 2020.

[5] Z. Zhu, W. Lu, L. Zhang, and N. Ansari, "Dynamic service provisioning in elastic optical networks with hybrid single-/multi-path routing," *J. Lightw. Technol.*, vol. 31, pp. 15–22, Jan. 2013.

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

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

[8] Q. Duan, S. Wang, and N. Ansari, "Convergence of networking and cloud/edge computing – status, challenges, and opportunities," *IEEE Netw.*, vol. 34, pp. 148–155, Nov./Dec. 2020.

[9] D. Kreutz *et al.*, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, pp. 14–76, Jan. 2014.

[10] L. Gong, Y. Wen, Z. Zhu, and T. Lee, "Toward profit-seeking virtual network embedding algorithm via global resource capacity," in *Proc. of INFOCOM 2014*, pp. 1–9, Apr. 2014.

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

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

[13] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Commun. Mag.*, vol. 53, pp. 90–97, Feb. 2015.

[14] M. Zeng, W. Fang, and Z. Zhu, "Orchestrating tree-type VNF forwarding graphs in inter-DC elastic optical networks," *J. Lightw. Technol.*, vol. 34, pp. 3330–3341, Jul. 2016.

[15] W. Fang *et al.*, "Joint spectrum and IT resource allocation for efficient vNF service chaining in inter-datacenter elastic optical networks," *IEEE Commun. Lett.*, vol. 20, pp. 1539–1542, Aug. 2016.

[16] Q. Duan, N. Ansari, and M. Toy, "Software-defined network virtualization – an architectural framework for integrating SDN and NFV for service provisioning in future networks," *IEEE Netw.*, vol. 30, pp. 10–16, Sept./Oct. 2016.

[17] Z. Lv and W. Xiu, "Interaction of edge-cloud computing based on SDN and NFV for next generation IoT," *IEEE Internet Things J.*, vol. 7, pp. 5706–5712, Jul. 2020.

[18] M. Ojo, D. Adami, and S. Giordano, "A SDN-IoT architecture with NFV implementation," in *Proc. of GC Wkshps 2016*, pp. 1–6, Dec. 2016.

[19] P. Ray and N. Kumar, "SDN/NFV architectures for edge-cloud oriented IoT: A systematic review," *Comput. Commun.*, vol. 169, pp. 129–153, Mar. 2021.

[20] IoT technologies and protocols. [Online]. Available: https://azure.microsoft.com/en-us/overview/internet-of-things-iot/iot-technology-protocols/.

[21] E. Municio, S. Latre, and J. Marquez-Barja, "Extending network programmability to the things overlay using distributed industrial IoT protocols," *IEEE Trans. Ind. Informat.*, vol. 17, pp. 251–259, Jan. 2021.

[22] P. Bosshart *et al.*, "P4: Programming protocol-independent packet pro-

cessors," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 87–95, Jul. 2014.
[23] S. Li *et al.*, "Protocol oblivious forwarding (POF): Software-defined networking with enhanced programmability," *IEEE Netw.*, vol. 31, pp. 58–66, Mar./Apr. 2017.
[24] Protocol Indepedent Forwarding. [Online]. Available: https://opennetworking.org/news-and-events/protocol-independent-forwarding/
[25] Intel Tofino. [Online]. Available: https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html.
[26] K. Zhang *et al.*, "Mobile edge computing and networking for green and low-latency internet of things," *IEEE Commun. Mag.*, vol. 56, pp. 39–45, May 2018.
[27] Z. Qin *et al.*, "Low-power wide-area networks for sustainable IoT," *IEEE Wirel. Commun.*, vol. 26, pp. 140–145, Jun. 2019.
[28] J. Yu *et al.*, "Forwarding programming in protocol-oblivious instruction set," in *Proc. of ICNP 2014*, pp. 577–582, Oct. 2014.
[29] B. Pfaff *et al.*, "The design and implementation of Open vSwitch," in *Proc. of NSDI 2015*, pp. 117–130, May 2015.
[30] Open vSwitch. [Online]. Available: http://www.openvswitch.org/.
[31] S. Tang *et al.*, "Sel-INT: A runtime-programmable selective in-band network telemetry system," *IEEE Trans. Netw. Serv. Manag.*, vol. 17, pp. 708–721, Jun. 2019.
[32] OVS-POF with Sel-INT module by USTC-INFINITELAB. [Online]. Available: https://github.com/USTC-INFINITELAB/OpenvSwitch-pof/tree/fast-path.
[33] DPDK: Data Plane Development Kit. [Online]. Available: https://www.dpdk.org/.
[34] NVIDIA Mellanox Innova-2 Flex open programmable SmartNIC. [Online]. Available: https://www.nvidia.com/en-us/networking/ethernet/innova-2-flex/.
[35] SmartNIC overview - Netronome. [Online]. Available: https://www.netronome.com/products/smartnic/overview/.
[36] O. Michel, R. Bifulco, G. Retvari, and S. Schmid, "The programmable data plane: abstractions, architectures, algorithms, and applications," *ACM Comput. Surv.*, vol. 54, pp. 1–36, May 2021.
[37] Netronome 25GbE SmartNICs with Open vSwitch hardware offload drive unmatched cloud and data center infrastructure performance. [Online]. Available: https://www.netronome.com/agilio-ovs-tc/.
[38] S. Horman, "OVS hardware offload with TC flower," 2017. [Online]. Available: http://www.openvswitch.org/support/ovscon2017/horman.pdf.
[39] A. Al-Shaikhli, C. Ceken, and M. Al-Hubaishi, "An interface protocol between SDN controller and end devices for SDN-oriented WSAN," *Wirel. Pers. Commun.*, vol. 101, pp. 755–773, Apr. 2018.
[40] M. Uddin, S. Mukherjee, H. Chang, and T. Lakshman, "SDN-based Multi-Protocol edge switching for IoT service automation," *IEEE J. Sel. Areas Commun.*, vol. 36, pp. 2775–2786, Sept. 2018.
[41] M. Cicioglu and A. Calhan, "A multiprotocol controller deployment in SDN-based IoMT architecture," *IEEE Internet Things J.*, vol. 9, pp. 20 833–20 840, May 2022.
[42] D. Firestone *et al.*, "Azure accelerated networking: SmartNICs in the public cloud," in *Proc. of NSDI 2018*, pp. 51–66, Apr. 2018.
[43] B. Li *et al.*, "KV-Direct: High-performance in-memory key-value store with programmable NIC," in *Proc. of SOSP 2017*, pp. 137–152, Oct. 2017.
[44] M. Liu, S. Peter, A. Krishnamurthy, and P. Phothilimthana, "E3: Energy-efficient microservices on SmartNIC-Accelerated servers," in *Proc. of USENIX ATC 2019*, pp. 363–378, Jul. 2019.
[45] P. Gao, Y. Xu, and H. Chao, "OVS-CAB: Efficient rule-caching for Open vSwitch hardware offloading," *Comput. Netw.*, vol. 188, p. 107844, Apr. 2021.
[46] Y. Yan, A. Beldachi, R. Nejabati, and D. Simeonidou, "P4-enabled Smart NIC: Enabling sliceable and service-driven optical data centres," *J. Lightw. Technol.*, vol. 38, pp. 2688–2694, Jan. 2020.
[47] H. Song, "Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane," in *Proc. of HotSDN 2013*, pp. 127–132, Aug. 2013.
[48] Q. Sun, Y. Xue, S. Li, and Z. Zhu, "Design and demonstration of high-throughput protocol oblivious packet forwarding to support software-defined vehicular networks," *IEEE Access*, vol. 5, pp. 24 004–24 011, Oct. 2017.
[49] ONOS. [Online]. Available: https://onosproject.org/.
[50] Q. Zheng, S. Tang, B. Chen, and Z. Zhu, "Highly-efficient and adaptive network monitoring: When INT meets segment routing," *IEEE Trans. Netw. Serv. Manage.*, vol. 18, pp. 2587–2597, Sept. 2021.
[51] C. Kim *et al.*, "In-band network telemetry (INT)," *Tech. Spec.*, Jun. 2016. [Online]. Available: https://p4.org/assets/INT-current-spec.pdf.
[52] Segment Routing. [Online]. Available: https://www.segment-routing.net/.