AP-SDN: Action Program enabled Software-Defined Networking Architecture

Zheng Zhao[1], Xiaoya Fan[2], Xin Xie[3], Qian Mao[4], and Qi Zhao[5*]
[1] Dalian Maritime University, China
[e-mail: zhaozheng@dlmu.edu.cn]
[2] Dalian University of Technology, China
[e-mail: xiaoyafan@dlut.edu.cn]
[3] Hunan University of Information Technology, China
[e-mail: xiexin0011@gmail.com]
[4] Liaoning University, China
[e-mail: maoqian@lnu.edu.cn]
[5] Northeastern University, China
[e-mail: zhaoqi@mail.neu.edu.cn]
[*]Corresponding author: Qi Zhao

Received March 24, 2023; revised June 29, 2023; accepted July 5, 2023; published July 31, 2023

Abstract

Software-Defined Networking (SDN) offers several advantages in dynamic routing, flexible programmable control and custom application-driven network management. However, the programmability of the data plane in traditional SDN is limited. A network operator cannot change the ability of the data plane and perform complex packet processing on the data plane, which limits the flexibility and extendibility of SDN. In the paper, AP-SDN (Action Program enabled Software-Defined Networking) architecture is proposed, which extends the action set of SDN data plane. In the proposed architecture, a modified Open vSwitch is utilized in the data plane allowing the execution of action programs at runtime, thus enabling complex packet processing. An example action program is also implemented which transparently encrypts traffic for terminals. At last, a prototype system of AP-SDN is developed and experiments show its effectiveness and performance.

Keywords: Action program, controller, data plane programmability, open vSwitch, SDN.

1. Introduction

Computer networks play an increasingly important role for both industry and our daily lives. However, with the rapid expansion of network scale and the growing of applications, the limitations of traditional network architectures become more apparent. In the traditional networks, the control plane and data plane are tightly coupled, resulting in complex protocols in network devices, a lack of dynamic coordination among network nodes and difficulties in quickly deploying new packet processing methods. These defects significantly limit further application of network technology.

Software-Defined Networking (SDN) [1-4] breaks the closed system of the traditional network and decouples the control plane and data plane. It centralizes the control functions on the controller, while leaving the forwarding function on network devices. SDN enables centralized, flexible and fine-grained control and makes the network programmable through external applications. However, its data plane is not programmable and can only forward packets in Match-Action mode without supporting more advanced operations. Additionally, it lacks the ability to flexibly adapt to the rapidly evolving network protocols. To enhance the programmability of the data plane, Programming Protocol-independent Packet Processors (P4) [5, 6] is proposed. P4 switch can control the parsing and forwarding process of packets according to the custom-designed P4 program, and support dynamic extension of network protocols. However, P4 switch can only perform limited actions on packets and its reprogramming requires suspension of the running switches, which greatly constrains its programmability and brings difficulty to conducting flexible operations on packets.

To address these problems, an Action Program enabled Software-Defined Networking (AP-SDN) architecture is proposed to improve the programmability of SDN data plane. AP-SDN enables action programs to be executed on the data plane dynamically so that complex processing of packets becomes applicable, which gives a high degree of dynamic and flexibility to SDN data plane. The contributions of this paper are as follows:

1) A novel AP-SDN architecture is proposed to improve the programmability of the data plane through the use of action programs. This architecture enables dynamic extension of the network data plane with action programs and allows complex packet processing.

2) We modified Open vSwitch (OVS) so that action programs can be installed and launched dynamically, i.e., AP-OVS. Accordingly, RYU controller and OpenFlow protocol are extended to support management and execution of action programs.

3) An example action program is implemented to demonstrate the practical application of AP-SDN, which can transparently encrypt traffic for terminals.

4) A prototype system of AP-SDN is developed and its effectiveness and performance are analyzed.

2. Related Work

2.1 Programmable Data Plane

SDN simplifies traditional routers by decoupling the control plane from the data plane, thus provides a centralized and programmable control plane. In SDN architecture, devices in the data plane, i.e., SDN switches, only forward packets based on the configuration specifications of the controller [4]. SDN greatly enhances the flexibility and controllability of the network. However, the lack of programmability in the data plane limits its adaptability to new network protocols and restricts the flexibility of SDN.

To address this issue, Bosshart et al. [5] proposed P4, a domain-specific language that introduces programmability to the data plane [7]. P4 decouples protocol from hardware, which means network operators can program the packet processing behavior of switches by P4 programs. P4Runtime [8] is used as southbound protocol for communicating between P4 switches and SDN controller. There are several data plane specific languages except for P4, such as POF [9], packet [10] and PX [11]. However, programmable data planes incorporating these techniques have to suspend for a moment when they are reprogramed. In order to solve this problem, researchers proposed P4-hypervisors, such as HyPer4 [12] and HyperVDP [13]. P4-hypervisor is also a P4 program which can accommodate specific P4 programs. Those P4 programs should be converted into table entries through a translator then installed on P4-hypervisor to realize specific packet processing function. Thus, P4-hypervisor enables online updating of P4 programs. However, P4 language has limited expression power [14], resulting in its incompetence in complex processing of packets. Kataria et al. [15] proposed a programmable data plane for new IP packet processing to deal with the stochasticity in quality of services and inflexible address structure in traditional network. While these methods support address customization using programmable data planes, it has limitations when it comes to complex packet processing. However, AP-SDN extends the action set through utilizing the action programs, enabling the implementation of any packet processing logic.

2.2 Hybrid SDN switch

Hybrid SDN switches [16-18] combine the advantages of traditional network switch/router and SDN switch, achieving the benefits of both worlds. For solving the controller bottleneck of SDN, Bianchi et al. [19] proposed a hybrid switch named OpenState, which can achieve a stateful data-plane. It offloads part of the controller functions by introducing an extended finite state machine in the data plane. Therefore, the data plane can forward packets according to network state without intervention from the controller. Mekky et al. [20] proposed an extended application-aware SDN architecture, which offloads some application logic locally to the switches. This architecture reduces the switch to-controller delays involved in flow establishment and eliminates inefficient traffic detouring in the case of service chaining. Curtis et al. [21] designed DevoFlow, a hybrid SDN model, which grants partial decision-making capability to switches and allows switches to make local routing decisions without the involvement of a controller. DevoFlow reduces the control plane communication greatly and achieves high performance networks. Further, Xu et al. [22] designed a hybrid switch which integrates traditional switching and SDN switching, achieving scalability and high performance simultaneously. Alvarez-Horcajo et al. [23] offloaded certain control plane functions to the data plane. They modified BOFUSS and constructed a hybrid switch, which can take charge of basic bridging and cooperate in path recovery with the controller. Chang et al. [24] modified OVS and implemented a kind of hybrid switch with the function of topology discovery. This method reduces the CPU usage and traffic load on the controller significantly. From the security aspect, Serna et al. [25] offloaded challenge verification from the controller onto programmable switches and designed PCPP protocol, which effectively prevents control plane overload. All above-mentioned hybrid switches involve offloading certain functionalities from the control plane to the data plane to reduce controller overhead and enhance network scalability. But they do not concern about other network function in data plan.

Hybrid switch is also explored to enhance capabilities of the data plane in multiple aspects. Fukuda et al. [26] proposed PRS architecture, in which the current OpenFlow is extended and OVS is modified. They added a packet processing module in OVS for combining portions of

divided payloads into the original one. Krishnan [27] proposed a new SDN data plane architecture named OpenPATH, where SDN and network function virtualization (NFV) [28-31] are combined. OpenPATH puts network functions (NF) inside SDN data plane and concentrates network management on a controller. It provides NFs with organic, reliable and scalable support, which greatly reduces latency and improves performance. Yu et al. [32] proposed a lightweight and cooperative traffic measurement based on OVS, named CountMax. Each switch in CountMax only processes a partial set of flows, which reduces the computing overhead greatly and ensures measurement accuracy at the same time. Chen et al. [33] presented P4SC, a system for implementing service function chains on the P4-capable data plane. P4SC parses and converts the service function chain (SFC) policies into a P4 program, then deploys the P4 program on the data plane. Zhang et al. [34] proposed a compiler-oriented method named Gallium, which offloads part of network functions to programmable switches and improves throughput greatly. In response to the issue of offloading feasibility, Chen et al. [35] presented LightNF system, which provides a suite of primitives for the description of network functions and utilizes an analyzer to analyze them. Chen et al. [36] deployed and managed security-oriented network functions on programmable switches for DDoS mitigation, achieving low cost, high flexibility and high performance. The methods mentioned above are similar to our AP-SDN. However, these methods extend the data plane with fixed applications, such as forwarding decision function and network function, while our AP-SDN dynamically extends the action set of SDN. We argue that our method provides more flexibility and extendibility to the network. In addition, our action programs can achieve finer-grained processing on packets.

2.3 Network function virtualization

NFV is a network architecture and technological paradigm that seeks to migrate the functionalities of traditional dedicated network devices to generic server hardware by leveraging software-based and virtualization approaches. NFV provides high flexibility and scalability, resulting in reduced operational costs. SFC [37, 38] orchestrates virtual network functions (VNFs) in a sequential chain, allowing for flexible configuration according to specific requirements. This enables a wide range of service customization and function combinations.

To optimize the placement of VNF in data center networks, Hawilo et al. [39] proposed a mixed integer linear programming (MILP) optimization model and BACON algorithm. This approach takes into consideration the carrier-grade characteristics of VNFs while also minimizing intra- and end-to-end delays of the SFC. In the IoT scenario, Liu et al. [40] proposed an SFC dynamic orchestration frameworks based on deep reinforcement learning technique. A deep deterministic policy gradient-based algorithm is used to solve the SFC dynamic orchestration problem in dynamic and complex IoT networks. To solve the multi-domain SFC routing problem, Zhao et al. [41] proposed the cooperative multi-agent reinforcement learning based algorithms, which achieves significant improvements in total network utility. For making better use of the finite substrate resources over time, Zhang et al. [42] consider about resource requirement for VNF instances and proposed forecast-assisted SFC deployment method. They constructed a model that combines the multitask regression layer above the graph neural networks to predict the future resource requirements. Chowdhury et al. [43] found that common functionality is repeatedly implemented in monolithic VNFs on the SFCs resulting in wasted infrastructure resources. They proposed a disaggregated packet processing architecture named MicroNF. MicroNF deploys VNFs using reusable, loosely-coupled, and independently deployable components achieving same packet processing

throughput with less CPU cost on average. All of the aforementioned methods orchestrate and deploy VNFs in various scenarios, enabling flexible and fine-grained packet processing. However, it is worth noting that in these methods, VNFs are executed on servers that are separate from switches or routers. Consequently, network traffic must detour through these servers, leading to increased routing latency caused by longer paths. In contrast, AP-SDN implements packet processing logic directly within switches as actions, thus mitigating the aforementioned drawback.

2.4 Software Switch

The Click modular router [44] is the first software router, which consists of a sequence of discrete modules and supports flexible packet processing. Click is a widely adopted framework for network processing, which serves as a foundation for building innovative switches, firewalls, and packet filters. It enables the development of new types of network devices by leveraging its flexible and extensible architecture. OVS [45, 46] is one kind of software switch, which is widely employed in SDN. It contains match-action tables and can be regarded as match-action forwarding pipeline of OpenFlow [45]. The Basic OpenFlow Software Switch (BOFUSS) [47], also known as CPqD or ofsoftswitch13, is another highly popular OpenFlow software switch. It can be easily modified for extended functionality and widely used in the research to implement new features. However, the above-mentioned software switches are not protocol independence, i.e., modification is required when a new protocol is added in the network.

To build a protocol-independent software switch, Shahbaz et al. [48] proposed PISCES, a modified OVS, in which the parse, match, and action code are replaced by C code generated by a P4 compiler. When a new protocol is added, the manager only needs to modify the P4 program, and compiles it to a new software switch. However, PISCES still needs re-compilation to modify the competence of software switch. Osinski et al. [49] presented P4rt-OVS, an original extension of OVS that enables runtime programming of protocol-independent and stateful packet processing pipelines. P4rt-OVS extends OVS with an additional Berkeley Packet Filter (BPF) [50] subsystem, which enables the injection of packet forwarding programs at runtime and integration with the OVS forwarding pipeline. Osiński et al. [51] proposed a novel programmable software datapath named NIKSS. This approach utilizes P4 as a high-level programming abstraction, Portable Switch Architecture (PSA) as a fully-featured P4 forwarding model and eBPF as a packet processing engine, to achieve a high degree of programmability for data plane. While these programs are well-suited for multiple match-action model functions, such as firewall and DDoS mitigation, they are primarily designed for implementing simple packet processing logic. It remains challenging for them to execute complex actions on packets, such as packet encryption. However, the proposed AP-SDN architecture allows for the implementation of any packet processing logic through action programs.

3. AP-SDN Architecture

In traditional SDN, the control and forwarding are separated. A network operator can program the control plane to control the forwarding action of the data plane. However, the action set of the data plane is fixed, with limited types of actions, and its extension is difficult. To solve this problem, AP-SDN introduces applications into the action set, so that extensible actions can be built in the form of action programs and deployed on the data plane, facilitating flexible and dynamic control over flows. AP-SDN extends traditional SDN architecture and its overall

architecture is shown in **Fig. 1**.

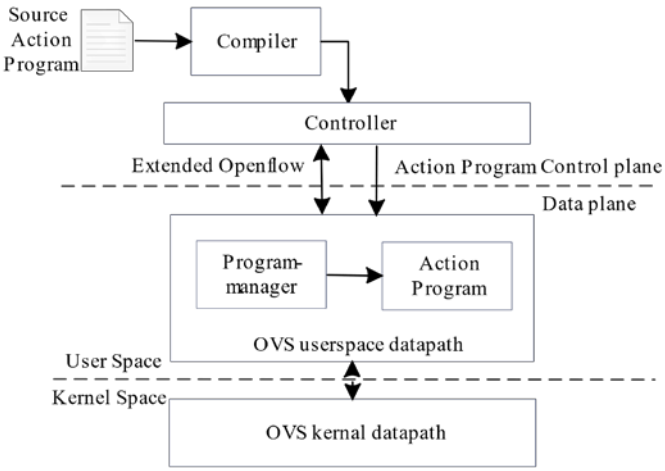

Fig. 1. The overall architecture of AP-SDN.

Similar to traditional SDN, the AP-SDN architecture consists of a control plane and a data plane. The control plane (i.e., controller) of is extended to store, compile, and distribute action programs. When an action program is needed, the corresponding source action program will be compiled into executable file by the controller using a compiler based on the platform of target switches. Then the executable action program can be distributed to the data plane upon request. At last, the action program will be received and installed by the switches of AP-SDN.

The data plane of AP-SDN is consisted of the action program enabled switches. These switches are derived from traditional SDN switches, which are able to manage and run action programs. As a manager of action programs in an action program enabled switch, the program-manager is responsible for adding, deleting and querying action programs. In terms of southbound interface, OpenFlow is extended to support the use of action programs as actions within flow table entries. Once an action program is executed on the switch, it processes flows based on the flow tables installed by the controller. Thus, complex processing to certain flows is enabled through the action programs.

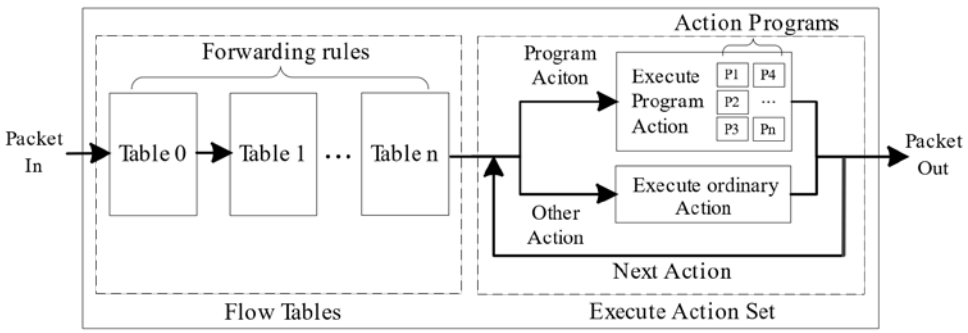

Fig. 2. The abstract forwarding model.

The abstracted forwarding model for AP-SDN is shown in **Fig. 2**. When a packet enters a switch, it is first matched with one or more flow tables to generate an action set, in which the actions will be executed sequentially. If action programs are included in the action set, they will be executed to process this packet. Otherwise, the packet will be processed by ordinary

actions. The combination of multiple traditional actions and program actions will compose more complex operations.

4. Implementation of AP-SDN

To implement AP-SDN, modifications are made to the RYU controller [52], southbound interface protocol and OVS switch, as illustrated in **Table 1**. Regarding the controller of AP-SDN, original RYU controller is modified to support the compilation and distribution of action programs. As southbound interface, OpenFlow is extended to support the distribution and installation of action programs, as well as their configuration files. The switch of AP-SDN, i.e., AP-OVS, which is constructed based on OVS, supports installation, execution and management of action programs. OVS and RYU are chosen as foundation to implement AP-SDN because of the high performance of OVS and simplicity of RYU. Moreover, both of them have active communities and a wide user base. Indeed, they are not the only options, as other software switches and controllers like BOFUSS and ONOS can also be utilized as foundations for implementing AP-SDN.

Table 1. Implementation of AP-SDN.

Level	Foundational software or protocol	New features
Controller	RYU	Compilation and distribution of action programs
Southbound Interface	OpenFlow	Distribution and installation of action programs
Switch	OVS	Installation, execution and management of action programs

4.1 The Implementation of AP-OVS

The architecture of AP-OVS is shown in **Fig. 3**, where program-manager is responsible for the management of programs and their corresponding configuration files. An action program is a binary executable file that functions as an action within a specific flow table entry, and its identification and runtime parameters are stored in a configuration file. Action programs and their configuration files are distributed by the controller and updated independently, thereby decoupling them and enhancing the switch's flexibility. To extend the action set of AP-OVS, a new action type is added in the flow table, i.e., 'program', with a parameter 'config_id' that represents the identification of the corresponding configuration file. The program type action will cause vswitchd module submitting a certain flow to the action program specified by a configuration file.

To enable dynamic distribution and online execution of action programs, an action-process table is employed in the user space of AP-OVS. Each entry in action-process table represents an action process, and is denoted as (*config_id, shm, counter*). An action process is created when an action program executes. The *config_id* specifies the configuration file of the corresponding action program. *shm* is the memory shared between vswitchd and this action process, facilitating rapid packet submission. This shared memory is organized as a circular queue to ensure that the packets are processed in order. *counter* is a reference number that tracks the number of times an action process is referenced by flow table entries.

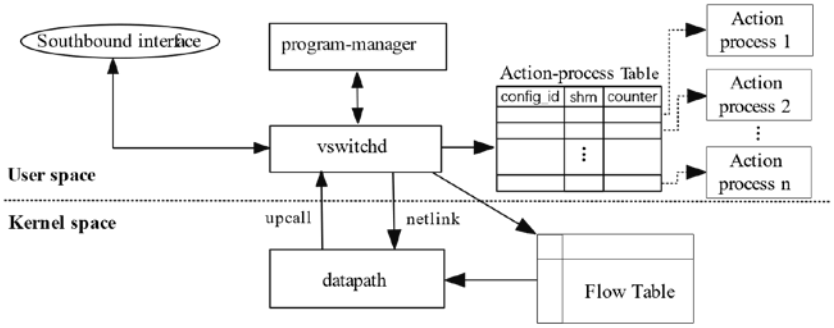

Fig. 3. The architecture of AP-OVS.

4.2 Installation of an Action Program

The installation of an action program in AP-OVS is as follows: firstly, the action program is compiled by the controller and distributed to the AP-OVS along with its configuration file using extended OpenFlow protocol. Secondly, the AP-OVS stores the action program and its configuration file in the designated folders respectively. Lastly, the action program is launched as a separate process in the AP-OVS.

The launch of an action program in AP-OVS is shown as **Pseudocode 1**. Within this process, AP-OVS receives a command from the controller (step 1) and respond accordingly by parsing the command. If the command is for adding a flow table entry and the instruction is set as a program, the *config_id* is obtained (step 2~5). If this *config_id* already exists in action-process table, the corresponding counter will be incremented by 1, indicating a new reference to this action program process (step 6~7). If the *config_id* is not present, AP-OVS reads the identification and running parameters from the configuration file specified by *config_id* and creates a new process to execute the action program using the provided running parameters, along with the creation of a new shared memory *shm* (step 8~10). Subsequently, (*config_id, shm, 1*) is added to the action-process table (step 11).

AP-OVS can support multiple running processes concurrently. These processes operate independently and each occupies a distinct entry within the action-process table.

Pseudocode 1. The launch of an action program in AP-OVS.

```
1   Receive a command from controller
2   if command is “add a flow table entry”
3     Obtain instruction type from the flow table entry
4     if instruction type is program
5       Obtain config_id
6       if config_id is in the action-process table
7             Increase the counter of the corresponding item by 1
8       else Read the identification and running parameters from the configuration
                                                        file specified by config_id
9             Create the shared memory shm
10            Create a new process to execute the action program using the running
                                                        parameters and shared memory
11            Add (config_id, shm, 1) in action-process table
12  end
```

4.3 Packet Processing in AP-OVS

AP-OVS enables complicated actions on packets through the installed action programs. The procedure of packet processing in AP-OVS is described in **Pseudocode 2**. Firstly, Upon the arrival of a packet at AP-OVS, the system looks up the current flow table to identify a matched flow table entry (step 2). If no corresponding entry is found, the packet will be forwarded to the controller (step 8). Conversely, if a match is identified, the action type of the action in flow table entry will be obtained (step 3~4). If the action type is program, this action will be added to the action set along with its parameter *config_id*. Otherwise, an ordinary action will be added (step 5~7). Iteration over the actions within the action set (step 10), if the type is not program, an ordinary action will be executed to the packet (step 15). Otherwise, AP-OVS will look up the action-process table to get the share memory *shm*, copy the packet to *shm* (step 11~13). Then, the action process will read the packet from the *shm* and process it (step 14).

Pseudocode 2. The procedure of packet processing in AP-OVS.

```
1   do
2     Look up the current flow table
3     if a matched flow table entry is identified
4       Obtain the action type of the action in flow table entry
5       if the action type is program
6           Add this program action to the action set along with its parameter config_id
7       else add this ordinary action to the action set
8     else forward the packet to the controller
9   while a new flow table is needed to be matched
10  for action in action set
11    if the type of action is program
12      Look up the action-process table to get shm
13      Copy the packet to the shm
14      Action process reads the packet from the shm and processes it
15    else execute ordinary action
16  end
```

5. An Action Program Example: Traffic Encryption

Action programs can be applied in various scenarios and operate traffic dynamically and flexibly. To illustrate the application of AP-SDN, let's consider the use case of traffic encryption. Traditionally, traffic encryption is typically implemented at the terminals or via intermediary devices known as middleboxes. However, terminal encryption cannot encrypt any traffic in the network and middlebox encryption may increase the length of the routing paths. In contrast, AP-SDN offers a more efficient and flexible approach to traffic encryption by transparently executing action programs on switches.

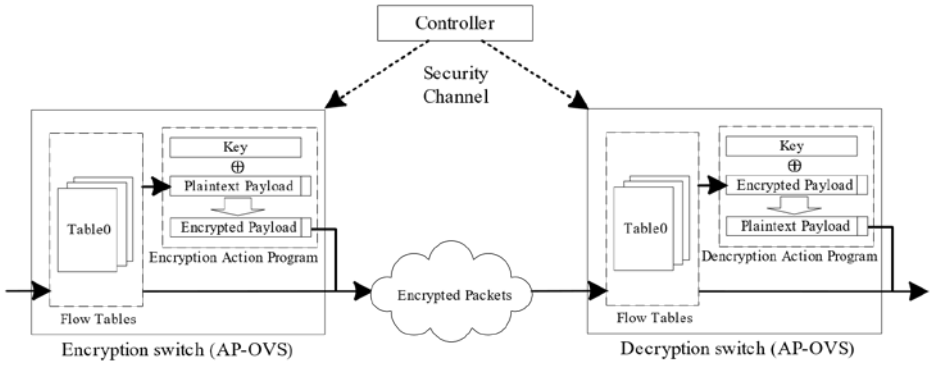

Fig. 4. Schematic diagram of the encryption and decryption action program.

To enable traffic encryption in AP-SDN, the construction and deployment of encryption and decryption action programs are necessary. The controller compiles these action program according to the target switch platforms, and installs the executable action programs and their configuration files on the target switches. After traffic is submitted to the action programs, the action programs will perform encryption or decryption for the traffic transparently. Considering the efficiency of packet encryption, stream encryption algorithm is chosen to construct the encryption and decryption action programs. Stream encryption/decryption keys are stored in the configuration files of the action programs and the length of keys is designed as the maximum payload length of IP packet. The schematic diagram of the encryption and decryption action programs is shown in **Fig. 4**. As can be seen, packets are transmitted as ciphertext between the encryption and decryption switch. In this application scenario, the encryption/decryption keys are distributed in the security channel to ensure key security. In fact, for stream encryption, the encryption action program and decryption action program are the same program.

6. Experiments and Analysis

In this section, the performance of AP-SDN and traditional SDN are compared. **Fig. 5** shows the experimental network topologies, in which **Fig. 5(a)** is a traditional SDN and **Fig. 5(b)** is a traditional SDN with two middleboxes MB1 and MB2. These middleboxes perform encryption and decryption on the traffic passing through them. **Fig. 5(c)** is a hybrid SDN, wherein S1 and S2 are hybrid switches. These switches are modified version of OVS that incorporate built-in encryption and decryption function. Lastly, **Fig. 5(d)** is an AP-SDN network, wherein S1 and S2 are AP-OVSes. Encryption and decryption action programs mentioned in Section 5 are executed on S1 and S2.

In our experiments, AP-OVS is installed on the DELL Precision 3640 tower station equipped with i7-10700 CPU running at 2.90GHz, with 8 GB RAM and one four-port Broadcom 1000Mbps NIC, running ubuntu18.04-Linux 4.17 operating system. As a controller, a modified RYU is deployed on DELL T7920 tower station equipped with one Xeon 5218R CPU running at 2.1GHz, with 64 GB RAM and one dual-port Intel 1000Mbps NIC, running ubuntu18.04-Linux 4.17 operating system. The experiments are conducted using OVS version 2.13.3, upon which the AP-OVS is developed. Hosts and middleboxes are deployed on the X86 platform equipped with i7-10510U CPU running at 1.80GHz, with 8 GB RAM and running ubuntu18.04-Linux 4.17 operating system.

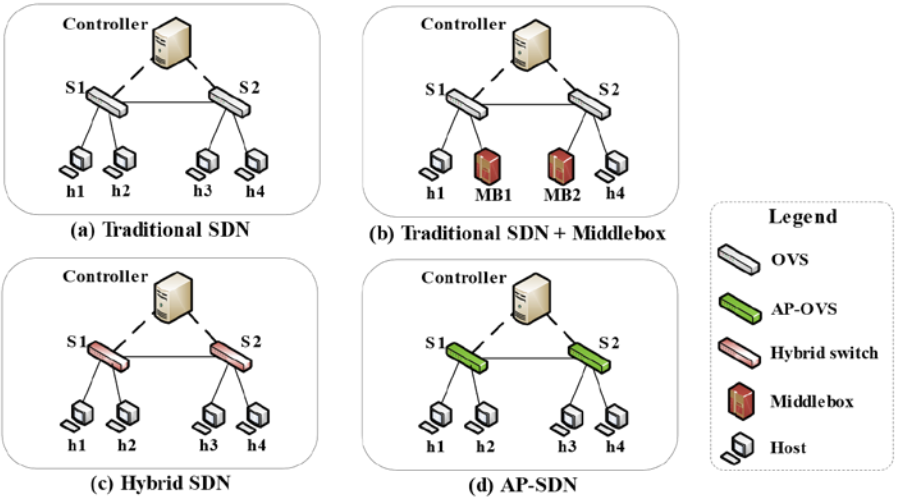

Fig. 5. Experimental network topologies.

6.1 Network Delay Analysis

To verify the data forwarding performance of AP-SDN, we compared the network delay between AP-SDN and traditional SDN from the following three aspects.

1) Comparison of network delay without action program execution in AP-SDN

In this experiment, Ping tool was used in traditional SDN (**Fig. 5(a)**) and AP-SDN (**Fig. 5(d)**) to test the round-trip time (RTT) between source host h1 and destination hosts h2, h3, h4, when no action program was executed on AP-SDN. ICMP packets were sent from host h1 to one of the destination hosts at a rate of 1 packet per second for a duration of 100 seconds. The network delay, measured in terms of RTT, is presented in **Fig. 6**. The results showed that both AP-SDN and traditional SDN had similar network delays. This outcome is expected since no action program is running in AP-SDN and the packets are processed in the same way as in traditional network. Furthermore, the results revealed a shorter delay from h1 to h2 compared to the delay from h1 to h3/h4 in both networks. This could be explained by the fact that the path between h1 and h2 is shorter.

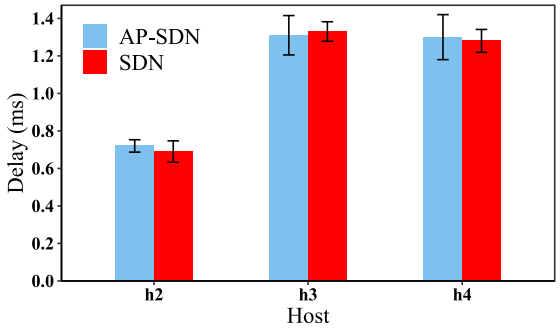

Fig. 6. Comparison of network delay without action program execution in AP-SDN.

2) Comparison of network delay with action programs execution in AP-SDN

To evaluate the forwarding performance of AP-SDN when action programs are executed, the network delays were measured in both traditional SDN and AP-SDN. In both networks,

host h2 generated traffic ranging from 100M to 900M and sent them to h3 to simulate network background traffic. The RTT between host h1 to h4 was measured. The difference was that in AP-SDN, the background traffic was encrypted and decrypted by action programs on switch S1 and S2. The RTT was measured for 100 times for both networks in this experiment. The test traffic in neither network was encrypted or decrypted. As shown in **Fig. 7**, the network delay in AP-SDN is comparable to that in traditional SDN. This implies that the increase in traffic processed by the action programs does not cause additional time consumption for traffic that is not processed by them.

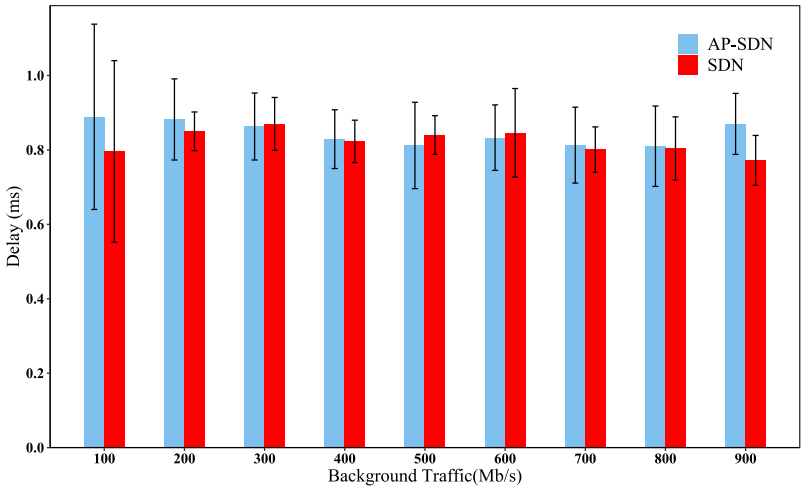

Fig. 7. Comparison of network delay with action programs execution in AP-SDN.

3) Network delay comparison when test traffic was processed by action programs

In this experiment, network delays between host h1 and h4 were compared among four different network topologies shown in **Fig. 5**. For all four networks, host h2 generated traffic ranging from 100M to 900M and sent them to h3 to simulate network background traffic. The RTT between host h1 to h4 was measured for 100 times. The differences among these networks were as follows:

- In traditional SDN, the traffic between host h1 and h4 was routed based on the shortest path.
- In SDN+Middlebox, background traffic between h1 and h4 was encrypted and decrypted by middleboxes MB1 and MB2, respectively. The traffic was sent from h1 to h4 and encrypted by MB1. It was then sent back to the network and decrypted by MB2 before being forwarded to host h4. Similarly, traffic in the opposite direction was encrypted by MB2 and decrypted by MB1.
- In hybrid SDN, traffic between h1 and h4 was encrypted and decrypted by built-in encryption and decryption modules in the switch S1 and S2.
- In AP-SDN, traffic between h1 and h4 was encrypted and decrypted by the action programs running on switch S1 and S2.

The encryption and decryption functions in middleboxes, hybrid switches and AP-OVS were implemented in C language, and the encryption algorithms were stream encryption. The results are presented in **Fig. 8**. The network delay in traditional SDN was lower than that in other three networks. This is reasonable since these latter networks involve additional encryption and decryption processes, which consumes more time. Compared with SDN+Middlebox, the network delay of AP-SDN was much lower. There are two reasons

behind this. Firstly, in SDN+Middlebox, the traffic is required to take a detour and follow a longer routing path when passing through the middleboxes, resulting in increased transmission delays. Secondly, in SDN+Middlebox, the traffic between the two hosts traverses multiple middleboxes and network protocol stacks, leading to repeated packet processing and increased processing delays. The results showed an advantage of the proposed AP-SDN in network delay over SDN+Middlebox for traffic encryption. On the other hand, the network delay of hybrid SDN is slightly lower than AP-SDN. This is because the encryption/decryption operation in hybrid SDN follows a fixed and more direct approach. However, it is important to note that this fixed approach sacrifices flexibility. Additionally, the network delay of AP-SDN barely increases as the background traffic volume increases.

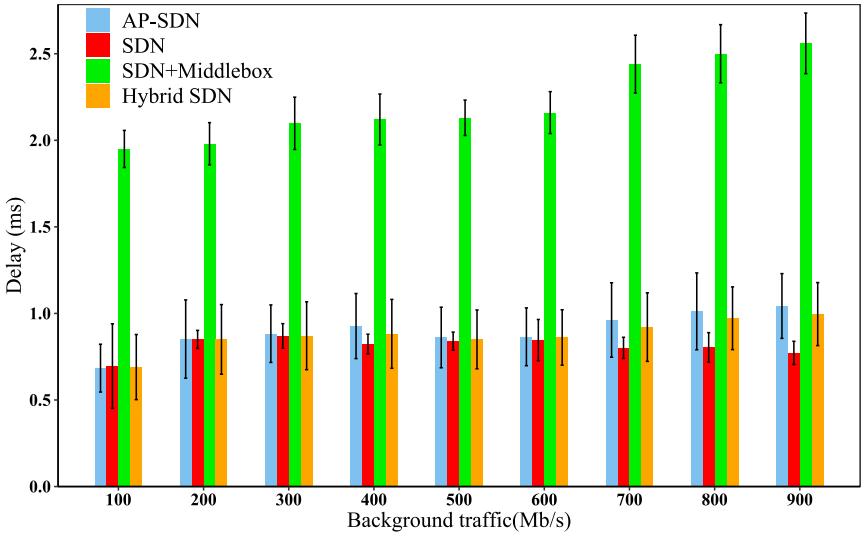

Fig. 8. Network delay comparison when test traffic was processed by action programs.

6.2 The Network Throughput of AP-SDN

In this experiment, the network throughputs of AP-SDN, SDN, SDN+Middlebox and hybrid SDN were compared. Iperf3 tool was used to measure the network throughput from host h1 to h4 using different packet sizes on the four network topologies shown in **Fig. 5**. As can be seen in **Fig. 9**, the network throughput of AP-SDN and SDN are nearly identical when the traffic is not processed by action programs. However, when the traffic is processed by the action program, the network throughput of AP-SDN decreases by 26.1% on average. Nevertheless, when the packet size reaches 1024, the network throughput of AP-SDN becomes comparable to that of traditional SDN. Similar trends are observed in SDN+Middlebox and hybrid SDN. The network throughputs in these networks were lower because the encryption and decryption of a large number of packets cannot be completed within a short time, thereby limiting the throughput. But note that encryption is a computationally intensive task and our current code is not yet fully optimized. For example, our encryption action programs only run with a signal thread. If multithreading is adopted, AP-SDN throughput would be significantly improved.

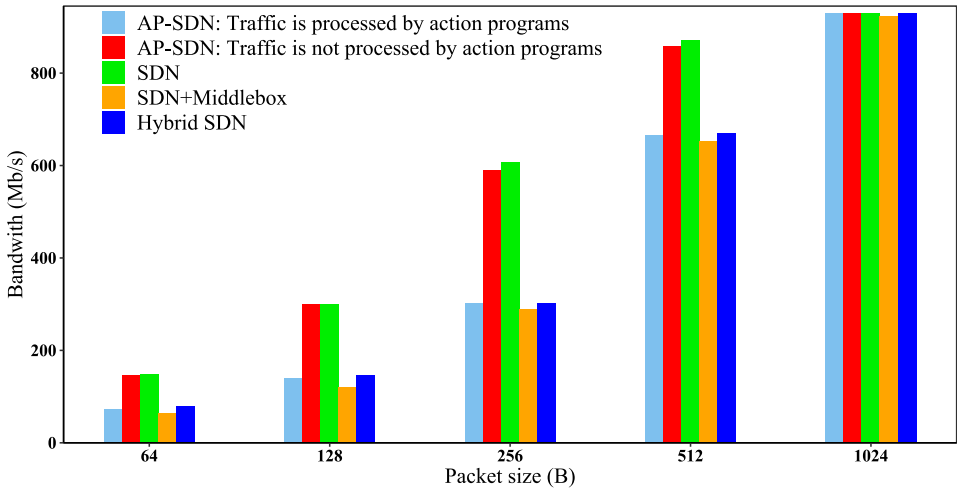

Fig. 9. The network throughput comparison.

6.3 Installation Delay of Flow Table and Action Program

In the AP-SDN architecture, the controller is responsible for distributing action programs to specific switches, which are then installed on these switches. This process is similar to the installation of flow tables in traditional SDN and incurs certain delays. In this section, we focus on evaluating the installation delay within the AP-SDN. We conducted experiments aiming at measuring the installation delays of both traditional SDN and AP-SDN in three distinct scenarios as follows.

Scenario 1: The installation delay of flow tables in traditional SDN.

Scenario 2: The installation delay of flow tables in AP-SDN.

Scenario 3: The total installation delay of flow tables and action programs in AP-SDN.

Direct measurement of installation delay is challenging, but installation delay can be inferred from the end-to-end communication delay. When there are no corresponding flow table rules or action programs installed in the switches, the communication delay increases accordingly serving as an indicator of the installation delay. Based on this approach, the installation delays were quantified by calculating the discrepancy in communication delay between networks with installed flow tables or action programs and the scenario where they were absent.

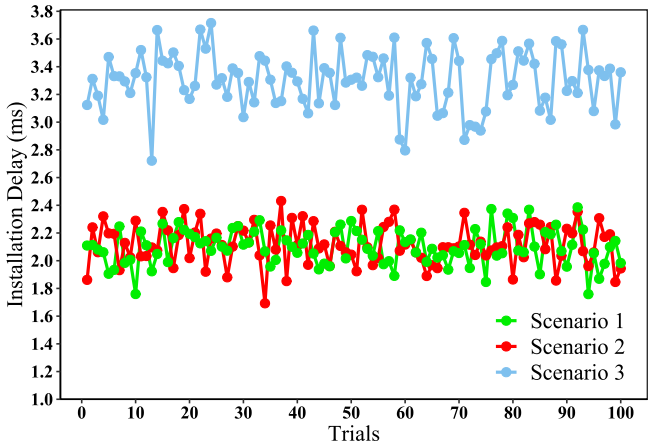

Fig. 10. Delay comparison between installation flow rules and action programs.

All measurements were conducted for 100 times. Note that the installation delay of action program excluded the compilation time of source action program. The size of executable action program file was 13K, including configuration file. Results are shown in **Fig. 10**. Results indicate that the installation delay of flow tables in AP-SDN and traditional SDN exhibits a comparable performance in Scenario 1 and Scenario 2. However, when AP-SDN installed both actions and flow table rules in Scenario 3, the installation time increased by 1.2ms on average.

6.4 Number of flow table rules and features

SDN+Middlebox (**Fig. 5(b)**), hybrid SDN (**Fig. 5(c)**) and AP-SDN (**Fig. 5(d)**) can all achieve traffic encryption. However, the encryption and decryption operations differ in their respective implementations. In SDN+Middlebox, these operations occur within the middleboxes, whereas in the other two architectures, they take place inside the switches. For SDN+Middlebox, additional flow table rules need to be configured on switch S1 and S2 to send specified traffic to the middleboxes, which increases the overhead in terms of flow table space. On the other hand, both hybrid SDN and AP-SDN perform encryption and decryption directly within the switches, requiring no additional flow table rules. The numbers of flow table rules required to enable encrypted communication between h1 and h4 in these three networks are shown in **Table 2**.

Table 2. Number of flow table rules required in SDN+Middlebox and AP-SDN.

Network type	Number of flow table rules of S1	Number of flow table rules of S2
SDN+Middlebox	4	4
Hybrid SDN	2	2
AP-SDN	2	2

In AP-SDN, the network functions are implemented in the form of action programs, which are dynamically distributed from the controller to the specified switches in the data plane. In contrast, SDN+Middlebox and hybrid SDN have fixed middleboxes and hybrid switches, making it challenging to update them dynamically. Therefore, compared to SDN+Middlebox and hybrid SDN, AP-SDN is superior in terms of flexibility, as it enables fast migration and dynamic update of action programs through dynamic distribution. Since the network functions are implemented directly within switches in hybrid SDN and AP-SDN, no middlebox is required, which reduces the deployment and management costs of middleboxes. The distinguishing features of SDN+Middlebox, hybrid SDN and AP-SDN are summarized in **Table 3**.

Table 3. Comparison about features of AP-SDN and SDN+Middlebox.

Network type	Flexibility	Fast migration	Dynamic update	Deployment cost	Management cost
SDN+Middlebox	Low	Difficult	Difficult	High	High
Hybrid SDN	Low	Difficult	Difficult	Low	Low
AP-SDN	High	Easy	Easy	Low	Low

7. Discussion

7.1 Comparison with existing schemes

AP-SDN enhances the flow processing capability of SDN data plane and supports dynamic extension of the switch functions. While P4 switch and hybrid switch can also be utilized to implement custom-designed functions, AP-SDN differs from both of them in several aspects.

Despite the high performance achieved by P4 switches for packet processing using programmable switching ASICs, AP-SDN offers advantages over P4. The reasons are threefold.

1) Incremental function extension: In AP-SDN, the functions of switches can be extended incrementally by installing specific action programs. In contrast, extending the function of a P4 switch requires rebuilding the entire packet forwarding logic. Thus, AP-SDN provides greater flexibility.

2) Online updates: Reprogramming to a P4 switch requires suspending the switch, which greatly constrains its programmability and narrows its application scenarios. But updating or adding action programs in AP-SDN can be done online without suspension of any switch.

3) Enhanced programmability: The P4 switches can only perform limited actions on packets, which greatly restricts its programmability and hinders flexible packet operations. However, the action program of AP-SDN can implement any logic for packet processing, enabling flexible packet operations.

Regarding hybrid switches, they are typically extended with special functions such as path recovery, traffic measurement and attack defense. Each custom-designed function requires modifying the switch, resulting in a fixed function that cannot be updated online. AP-SDN, on the other hand, adopts dynamic function extension, allowing all custom-designed functions to be implemented as action programs and installed on switches without modifying the switches themselves.

Overall, AP-SDN offers advantages in dynamic extension and flexible programmability, making it more suitable for the scenarios that require special or dynamic network function. P4 switch has more advantages in high-speed forwarding due to its high performance. Hybrid switch are more suitable for network with fixed functions and lower management cost. Consequently, we believe that the proposed AP-SDN is a beneficial complement to existing schemes.

7.2 Scalability

Similar to SDN, scalability is an important consideration for AP-SDN. When applying AP-SDN to large-scale networks, several scalability issues need to be addressed:

1) Excessive controller load: As the network size grows, the controller may experience a heavy workload due to the compilation, decision-making, and distribution of action programs. Compiling a large number of action programs with complex logic can be computationally intensive.

2) Switch storage space: With numerous user-specific requirements, switches need to store and install a large number of action programs, leading to significant storage pressure on the switches.

3) Switch processing delay: In AP-SDN, switches execute action programs to process packets. With a large scale of network, the switch processing delay becomes a scalability concern. Efficient execution of action programs is essential to ensure timely packet forwarding.

4) Communication delay and overhead: Communication between the controller and switches, particularly for distributing action programs and flow rules, can introduce delay and

overhead. As the network scales, communication delay and overhead between the controller and switches increases, potentially diminishing the controller's ability to quickly perceive the network status and conduct timely control.

To address these scalability challenges, several solutions can be employed. Firstly, the excessive load on the controller can be mitigated by using multiple controllers. Secondly, a hierarchical storage approach can be adopted to alleviate storage pressure on switches. Frequently used or prioritized action programs can be stored in switches, whereas less frequently used programs are stored centrally in the controller. Thirdly, switch processing delay can be minimized through techniques such as hardware acceleration and caching, ensuring efficient execution of action programs. Lastly, a proactive action program installation strategy can help improve the situation of communication delay and overhead between controller and switch.

7.3 Security Risk

AP-SDN introduces action program running inside switches and offers some advantages. But it also brings additional security risks. Three typical security risks are as following.

1) Malicious action programs: If an attacker installs malicious action programs inside switches, these switches may be controlled by the attackers and various malicious actions can be conducted.

2) Action program vulnerabilities: Software vulnerabilities cannot be avoided with current technology. A high-risk action program vulnerability may make the whole switch crashed. Attackers can exploit vulnerabilities to attack target switches, potentially resulting in a decrease in network connectivity.

3) Action program confidentiality and integrity: Action programs and their configuration files are distributed by the controller and transmitted over the network. They can be sniffed by the attackers, resulting in the disclosure of sensitive information such as encryption keys. They can also modify action programs and their configuration files.

Despite these security risks, effective security management and mechanisms can mitigate them, making AP-SDN reliable and stable. Firstly, action programs are managed by network managers and isolated from common users, making it difficult for attackers to install malicious action programs in switches. Secondly, action programs can be isolated from packet exchange process in AP-VOS. This can be achieved through containerization or virtualization technologies. That is to say, even if action programs on an AP-OVS crash, the packet exchange process will continue to work normally. Lastly, action programs and their configuration files are transmitted through secure channels between the controller and switches, employing protocols such as transport layer security (TLS) protocol used in the traditional SDN. The cost of sniffing or modifying the action programs for the attackers would be significantly high.

7.4 Application Scenarios

AP-SDN opens up the action set of SDN data plane, allowing the switches to dynamically acquire new capabilities under the control of controller. Each switch can possess different abilities based upon user requirement and network statement. Furthermore, the abilities of switches can be evolved continuously and transparently without redeploying network devices. With these advantages, AP-SDN can be applied in various fields, including network security, edge computing, network function, in-network computing and others.

For instance, AP-SDN can be used in DDoS attack traffic scrubbing. Although traditional switches can be used to scrub attack traffic with scrubbing rules, but they can only scrub a part of attack traffic in the mode of “match-action”. More sophisticated actions go beyond the

capability of the switch, such as those that require complex arithmetic operations, loops, or application layer processing [53]. AP-SDN will be a suitable choice in this scenario, because any action can implement using an action program, without limitation of the switching ASIC. Another example is network function, where various network functions can be implemented using action programs and installed on any switch flexibly instead of deploying dedicated devices.

Our AP-SDN is built based on virtual switch OVS. It is worth noting that the performance of OVS improves gradually along with the optimization on it. It has been widely deployed in real network to provide high-throughput and low-latency packet exchange. Besides, the basic architecture with separated control and forwarding in AP-SDN is the same as in traditional SDN, which means that AP-SDN can be incrementally deployed within traditional SDN to form a hybrid SDN network.

8. Conclusion

In this paper, a novel SDN architecture, action program enabled software-defined networking (AP-SDN), is proposed. AP-SDN can execute action programs distributed by controller on the data plane at runtime, thereby achieving complex packet processing. This architecture significantly enhances the programmability and packet processing ability of SDN data plane. AP-SDN is highly dynamic and flexible. We took flow encryption as a use case of AP-SDN. Experiments on the real network showed the effectiveness and potential of AP-SDN.

Although the current version of AP-SDN cannot fulfill all the possible flow processing requirements, we believe that action programmable SDN can be beneficial in many scenarios. In future studies, our focus will be on the following aspects: 1) AP-SDN will be deployed in larger network with a broader range of implemented action programs. 2) The strategies for defending AP-SDN from attacks will be further investigated. 3) AP-SDN and P4 language will be combined to construct a protocol-oblivious, action-free data plane.

References

[1] K. Kirkpatrick, “Software-defined networking,” *Communications of the ACM*, vol. 56, no. 9, pp. 16-19, 2013. Article (CrossRef Link)

[2] F. Hu, Q. Hao, and K. Bao, “A survey on software-defined network and openflow: From concept to implementation,” *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2181-2206, 2014. Article (CrossRef Link)

[3] S. Badotra, and S. N. Panda, “A survey on software defined wide area network,” *International Journal of Applied Science and Engineering*, vol. 17, no. 1, pp. 59-73, 2020. Article (CrossRef Link)

[4] D. S. Rana, S. A. Dhondiyal, and S. K. Chamoli, “Software defined networking (SDN) challenges, issues and solution,” *International Journal of Computer Sciences and Engineerin*g, vol. 7, no. 1, pp. 884-889, 2019. Article (CrossRef Link)

[5] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, and G. Varghese, “P4: Programming protocol-independent packet processors,” *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87-95, 2014. Article (CrossRef Link)

[6] S. Kaur, K. Kumar, and N. Aggarwal, “A review on P4-Programmable data planes: architecture, research efforts, and future directions,” *Computer Communications*, vol. 170, pp. 109-129, 2021. Article (CrossRef Link)

[7] A. Liatifis, P. Sarigiannidis, V. Argyriou, and T. Lagkas, “Advancing SDN from OpenFlow to P4: A Survey,” *Computing Surveys*, vol. 55, no. 9, pp. 1-37, 2023. Article (CrossRef Link)

[8] The P4.org API Working Group, "P4Runtime Specification," 2020. [Online]. Available: https://p4.org/p4-spec/p4runtime/v1.0.0/P4Runtime-Spec.html.
[9] H. Song, "Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane," in *Proc. of the 2nd ACM SIGCOMM workshop on HotSDN*, HongKong, China, pp. 127-132, 2013. Article (CrossRef Link)
[10] P. Jungck, R. Duncan, and D. Mulcahy, *packetC Programming*, NewYork, NY, USA: Springer, 2011. Article (CrossRef Link)
[11] G. Brebner, and W. Jiang, "High-speed packet processing using reconfigurable computing," *IEEE Micro*, vol. 34, no. 1, pp. 8-18, 2014. Article (CrossRef Link)
[12] D. Hancock, and J. Van der Merwe, "Hyper4: Using p4 to virtualize the programmable data plane," in *Proc. of the 12th Int. Conf. CoNEXT*, Irvine California, USA, pp. 35-49, 2016. Article (CrossRef Link)
[13] C. Zhang, J. Bi, Y. Zhou, and J. Wu, "HyperVDP: High-performance virtualization of the programmable data plane," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 556-569, 2019. Article (CrossRef Link)
[14] S. Ibanez, G. Brebner, N. McKeown, and N. Zilberman, "The p4-> netfpga workflow for line-rate packet processing," in P*roc. of the 2019 ACM/SIGDA Int. Sym. on FPGA*, Seaside, CA, USA, pp. 1-9, 2019. Article (CrossRef Link)
[15] B. Kataria, M. P. R, L. Monis, M. P. Tahiliani, and K. Makhijani, "Programmable Data Plane for New IP using eXpress Data Path (XDP) in Linux," in *Proc. of 23rd Int. Conf. on HPSR*, Taicang, Jiangsu, China, pp. 9-16, 2022. Article (CrossRef Link)
[16] S. Khorsandroo, A. G. Sánchez, A. S. Tosun, J. M. Arco, and R. Doriguzzi-Corin, "Hybrid SDN evolution: A comprehensive survey of the state-of-the-art," *Computer Networks*, vol. 192, pp. 107981, 2021. Article (CrossRef Link)
[17] R. Amin, M. Reisslein, and N. Shah, "Hybrid SDN networks: A survey of existing approaches," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3259-3306, 2018. Article (CrossRef Link)
[18] X. Huang, S. Cheng, K. Cao, P. Cong, T. Wei, and S. Hu, "A survey of deployment solutions and optimization strategies for hybrid SDN networks," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1483-1507, 2019. Article (CrossRef Link)
[19] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, "Openstate: programming platform-independent stateful openflow applications inside the switch," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 44-51, 2014. Article (CrossRef Link)
[20] H. Mekky, F. Hao, S. Mukherjee, Z.-L. Zhang, and T. Lakshman, "Application-aware data plane processing in SDN," in *Proc. of the 3rd ACM SIGCOMM workshop on HotSDN*, Chicago, Illinois, USA, pp. 13-18, 2014. Article (CrossRef Link)
[21] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," in *Proc. of Int. Conf. on SIGCOMM*, Toronto, Ontario, Canada, pp. 254-265, 2011. Article (CrossRef Link)
[22] H. Xu, H. Huang, S. Chen, and G. Zhao, "Scalable software-defined networking through hybrid switching," in *Proc. of IEEE Conf. on Computer Communication*, Atlanta, GA, USA, pp. 1-9, 2017. Article (CrossRef Link)
[23] J. Alvarez‐Horcajo, I. Martinez‐Yelmo, E. Rojas, J. A. Carral, and D. Lopez‐Pajares, "New cooperative mechanisms for software defined networks based on hybrid switches," *Transactions on Emerging Telecommunications Technologies*, vol. 28, no. 8, pp. e3150, 2017. Article (CrossRef Link)
[24] Y.-C. Chang, H.-T. Lin, H.-M. Chu, and P.-C. Wang, "Efficient topology discovery for software-defined networks," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1375-1388, 2021. Article (CrossRef Link)
[25] C. B. Serna, and C. Mas-Machuca, "Preventing Control Plane Overload in SDN Networks with Programmable Data Planes," in *Proc. of 18th Int. Conf. on Network and Service Management*, Atlanta, GA, USA, pp. 37-45, 2022. Article (CrossRef Link)

[26] H. Fukuda, and S. Kojima, "PRS: a payload inspection mechanism for software defined network," in *Proc. of 16th IEEE Ann. CCNC*, Las Vegas, NV, USA, pp. 1-6, 2019. Article (CrossRef Link)
[27] P. Krishnan, S. Duttagupta, and R. Buyya, "OpenPATH: Application aware high-performance software-defined switching framework," *Journal of Network and Computer Applications*, vol. 193, pp. 103196, 2021. Article (CrossRef Link)
[28] S. Yang, F. Li, S. Trajanovski, R. Yahyapour, and X. Fu, "Recent advances of resource allocation in network function virtualization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 2, pp. 295-314, 2021. Article (CrossRef Link)
[29] S. Mostafavi, V. Hakami, and M. Sanaei, "Quality of service provisioning in network function virtualization: a survey," *Computing*, vol. 103, pp. 917-991, 2021. Article (CrossRef Link)
[30] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90-97, 2015. Article (CrossRef Link)
[31] H. Hawilo, A. Shami, M. Mirahmadi, and R. Asal, "NFV: state of the art, challenges, and implementation in next generation mobile networks (vEPC)," *IEEE network*, vol. 28, no. 6, pp. 18-26, 2014. Article (CrossRef Link)
[32] X. Yu, H. Xu, D. Yao, H. Wang, and L. Huang, "CountMax: A lightweight and cooperative sketch measurement for software-defined networks," *IEEE/ACM Transactions on Networking*, vol. 26, no. 6, pp. 2774-2786, 2018. Article (CrossRef Link)
[33] X. Chen, D. Zhang, X. Wang, K. Zhu, and H. Zhou, "P4sc: Towards high-performance service function chain implementation on the p4-capable device," in *Proc. of IFIP/IEEE Sym. on INSM*, Arlington, VA, USA, pp. 1-9, 2019. Article (CrossRef Link)
[34] K. Zhang, D. Zhuo, and A. Krishnamurthy, "Gallium: Automated software middlebox offloading to programmable switches," in *Proc. of Ann. Conf. of the ACM on SIGCOMM*, New YorkNY, United States, pp. 283-295, 2020. Article (CrossRef Link)
[35] X. Chen, Q. Huang, P. Wang, Z. Meng, H. Liu, Y. Chen, D. Zhang, H. Zhou, B. Zhou, and C. Wu, "Lightnf: Simplifying network function offloading in programmable networks," in *Proc. of IEEE/ACM 29th Int. Sym. on IWQOS*, Tokyo, Japan, pp. 1-10, 2021. Article (CrossRef Link)
[36] X. Chen, H. Liu, D. Zhang, Q. Huang, H. Zhou, C. Wu, and Q. Yang, "Empowering DDoS Attack Mitigation with Programmable Switches," *IEEE Network*, pp. 1-7, 2022. Article (CrossRef Link)
[37] A. M. Medhat, T. Taleb, A. Elmangoush, G. A. Carella, S. Covaci, and T. Magedanz, "Service function chaining in next generation networks: State of the art and research challenges," *IEEE Communications Magazine*, vol. 55, no. 2, pp. 216-223, 2017. Article (CrossRef Link)
[38] K. Kaur, V. Mangat, and K. Kumar, "A comprehensive survey of service function chain provisioning approaches in SDN and NFV architecture," *Computer Science Review*, vol. 38, pp. 100298, 2020. Article (CrossRef Link)
[39] H. Hawilo, M. Jammal, and A. Shami, "Network function virtualization-aware orchestrator for service function chaining placement in the cloud," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 643-655, 2019. Article (CrossRef Link)
[40] X. Li, Y. Zhang, L. Xi, and D. Zhao, "Dynamic service function chain orchestration for NFV/MEC-enabled IoT networks: A deep reinforcement learning approach," *IEEE Internet of Things Journal*, vol. 8, no. 9, pp. 7450-7465, 2021. Article (CrossRef Link)
[41] D. Zhao, Y. Lu, X. Li, Z. Li, and Y. Liu, "Cross-Domain Service Function Chain Routing: Multiagent Reinforcement Learning Approaches," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 12, pp. 4754-4758, 2022. Article (CrossRef Link)
[42] J. Zhang, Y. Liu, Z. Li, and Y. Lu, "Forecast-Assisted Service Function Chain Dynamic Deployment for SDN/NFV-Enabled Cloud Management Systems," *IEEE Systems Journal*, pp. 1-12, 2023. Article (CrossRef Link)
[43] S. R. Chowdhury, H. Bian, T. Bai, and R. Boutaba, "A disaggregated packet processing architecture for network function virtualization," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 6, pp. 1075-1088, 2020. Article (CrossRef Link)
[44] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click modular router," *ACM Transactions on Computer Systems*, vol. 18, no. 3, pp. 263-297, 2000. Article (CrossRef Link)

[45] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, and P. Shelar, "The design and implementation of open vswitch," in *Proc. of 12th USENIX Sym. on NSDI*, Oakland, CA, USA, pp. 117-130, 2015. Article (CrossRef Link)
[46] W. Tu, Y.-H. Wei, G. Antichi, and B. Pfaff, "revisiting the open vSwitch dataplane ten years later," in *Proc. of the 2021 ACM SIGCOMM 2021 Conference*, Virtual Event, USA, pp. 245–257, 2021. Article (CrossRef Link)
[47] E. L. Fernandes, E. Rojas, J. Alvarez-Horcajo, Z. L. Kis, D. Sanvito, N. Bonelli, C. Cascone, and C. E. Rothenberg, "The road to BOFUSS: The basic OpenFlow userspace software switch," *Journal of Network and Computer Applications*, vol. 165, pp. 102685, 2020. Article (CrossRef Link)
[48] M. Shahbaz, S. Choi, B. Pfaff, C. Kim, N. Feamster, N. McKeown, and J. Rexford, "Pisces: A programmable, protocol-independent software switch," in *Proc. of Conf. of the ACM on SIGCOMM*, Florianopolis, Brazil, pp. 525-538, 2016. Article (CrossRef Link)
[49] T. Osiński, H. Tarasiuk, P. Chaignon, and M. Kossakowski, "P4rt-OVS: Programming Protocol-Independent, Runtime Extensions for Open vSwitch with P4," in *Proc. of Conf. of the IFIP Networking*, Paris, France, pp. 413-421, 2020. Article (CrossRef Link)
[50] S. McCanne, and V. Jacobson, "The BSD Packet Filter: A New Architecture for User-level Packet Capture," in *Proc. of Conf. of the USENIX*, San Diego, CA, pp. 1-11, 1993. Article (CrossRef Link)
[51] T. Osiński, J. Palimąka, M. Kossakowski, F. D. Tran, E.-F. Bonfoh, and H. Tarasiuk, "A novel programmable software datapath for software-defined networking," in *Proc. of the 18th Int. Conf. on CoNEXT*, Roma, Italy, pp. 245-260, 2022. Article (CrossRef Link)
[52] K. Morita, I. Yamahata, and V. Linux, "Ryu: Network operating system," in *Proc. of OpenStack Design Summit & Conference*, 2012. Article (CrossRef Link)
[53] M. Zhang, G. Li, S. Wang, C. Liu, A. Chen, H. Hu, G. Gu, Q. Li, M. Xu, and J. Wu, "Poseidon: Mitigating volumetric ddos attacks with programmable switches," in *Proc. of Sym. on NDSS*, San Diego, CA, USA, pp.1-18, 2020. Article (CrossRef Link)

Zheng Zhao received his B.S. degree from Dalian University of Technology in 2010. He received his M.S. and Ph.D degrees from Zhengzhou Science and Technology Institute in 2013 and 2017. Now he is working at College of Artificial Intelligence, Dalian Maritime University. His research interests include network security, next generation Internet and deep learning.

Xiaoya Fan received her M.S. degree in biomedical engineering from Beihang University in 2014 and Ph.D.degree from the Université libre de Bruxelles in 2018. She is currently an assistant professor with the School of Software, Dalian University of Technology. Her current research interests include network analysis and machine learning.

Xin Xie received his B.S., M.S. and Ph.D degrees from Zhengzhou Science and Technology Institute in 2008, 2011 and 2015. Now he is working at School of Computer Science and Engineering, Hunan University of Information Technology. His research interests include software security, deep learning and intelligent imaging technology.

Qian Mao is a lecture of Light Industry College of Liaoning University. Her research focuses on data analysis.

Qi Zhao is an assistant professor of College of Medicine and Biological Information Engineering of Northeastern University. His research focuses on network security, application of deep learning, and bioinformatics.