Postprint: P4-Based Consistency Verification of SDN Control-Data Plane Flow Rules

Authors: Xia Jiqiang, Cui Pengshuai, Li Ziyong, Lan Julong

Date: 2022-04-07T15:01:56Z

Abstract

To address the problem of inconsistency between control-plane and data-plane flow rules caused by hardware/software failures and misconfigurations in the SDN data plane, we propose a P4-based consistency verification mechanism for SDN control-data plane flow rules (P4CV, P4-based Consistency Verification mechanism for SDN control-data plane). P4CV first sends specially structured probes to the data plane, then each P4 switch embeds the actual flow rule execution information from the data plane into the probes, and finally P4CV employs a symbolic-execution-based consistency verification algorithm to complete consistency verification between control-plane flow rule configurations and data-plane telemetry information. Simulation results demonstrate that the single-path verification time of P4CV is independent of network topology and exhibits only linear correlation with the number of switching nodes along the path. In multi-path forwarding scenarios with equivalent network scale and flow rule configuration, P4CV, while incurring only approximately 0.06‰ bandwidth overhead, reduces verification time by approximately 42% on average compared to existing solutions.

Full Text

Preamble

Vol. 39 No. 8
Application Research of Computers
ChinaXiv Cooperative Journal

P4-based Rules Consistency Verification for SDN Control-Data Plane

Xia Jiqiang, Cui Pengshuai†, Li Ziyong, Lan Julong
(Information Technology Research Institute, People's Liberation Army Strategic Support Force Information Engineering University, Zhengzhou 450000, China)

Abstract: To address the inconsistency of flow rules between the control plane and data plane caused by software and hardware faults, misconfigurations, and other issues in SDN data planes, this paper proposes a P4-based Consistency Verification mechanism for SDN control-data plane (P4CV). P4CV first sends probes with a specific structure to the data plane, after which each P4 switch embeds actual flow rule execution information from the data plane into the probes. Finally, P4CV employs a symbolic execution-based consistency verification algorithm to validate the consistency between control plane flow rule configurations and data plane telemetry information. Simulation results demonstrate that P4CV' s single-path verification time is unaffected by network topology, being only linearly related to the number of switching nodes along the path. In multipath forwarding scenarios with equivalent network scale and flow rule configuration, P4CV generates only approximately 0.06‰ bandwidth overhead while reducing verification time by an average of 42% compared to existing solutions.

Keywords: SDN; P4; consistency; In-band network telemetry

0 Introduction

In traditional network architectures, the control plane and data plane are tightly coupled, with software and hardware upgrades constraining each other. Software Defined Networking (SDN) decouples the control and data planes by extracting network management and control logic from devices, creating a new network architecture that enables efficient control and management. Initially, SDN programmability was primarily realized at the control layer. However, with the emergence of data plane domain-specific languages (DSLs) such as P4 and POF, SDN has achieved programmable data planes with line-rate processing performance. Network developers can flexibly design data plane processing logic using P4' s abstract forwarding model, enabling more efficient deployment of new protocols in the data plane and facilitating widespread applications in traffic engineering and network measurement. Nevertheless, while P4 brings programmability to the data plane, it also exacerbates consistency and security issues between the SDN control plane and data plane, where the actual forwarding behavior of the data plane diverges from the expected behavior of the control plane. On one hand, as a data plane programming language, bugs in P4 programs themselves can lead to control-data plane inconsistencies. On the other hand, inconsistencies can also arise during network operation due to hardware/software failures, administrator misconfigurations, or malicious attacks.

Even before the advent of data plane programming languages like P4, the consistency problem between SDN control and data plane flow rules had already received extensive attention. Monocle addresses this by logically expressing switch forwarding tables as Boolean satisfiability problems to rapidly generate probes for specific rules, enabling verification of both loaded flow rules in stable network states and tracking of rule updates during reconfiguration. To tackle the flow rule priority faults identified in commercial switches, Rulescope proposes a more precise and efficient consistency verification algorithm. VeriDP validates the actual transmission paths of real network traffic by collecting probe path information and comparing it with abstract path tables from the control plane, though this approach requires pre-constructed path tables, increasing controller resource consumption. These active probing-based consistency verification mechanisms can validate controller-known or synchronized flow rules but cannot detect faults caused by unknown or misconfigured flow rules in the data plane. Moreover, such mechanisms are designed for OpenFlow-based SDN flow rule verification scenarios and are not applicable to current programmable data planes.

For SDN programmable data plane flow rule consistency verification scenarios, researchers have recently proposed a series of P4 program bug detection schemes. Similar to traditional program analysis methods, these approaches primarily perform static vulnerability analysis of P4 programs through assertion checking. Obviously, such methods cannot detect runtime errors and inconsistencies that occur after program compilation. Consequently, recent research in the programmable data plane domain has attempted to address consistency issues during actual network operation. One study pioneered research on control-data plane inconsistencies in P4-enabled smart NICs, analyzing how focusing solely on P4 program bugs can severely impact network performance metrics such as latency and throughput. P4RL introduces reinforcement learning-based fuzzing to achieve automated runtime verification of individual P4 switches, but this mechanism is not suitable for control-data plane consistency verification at SDN network scale. The same team further proposed P4Consist, a control-data plane consistency verification method for P4-enabled SDNs. Similar to traditional SDN data plane measurement methods, P4Consist requires injecting a large number of probes to inspect all rules or paths, which not only impacts normal network communication but also increases data plane measurement latency, making test results lack real-time capability. Additionally, since P4Consist needs to traverse all paths between source and destination nodes, its verification time grows dramatically with increasing network complexity.

Therefore, achieving rapid flow rule consistency verification for SDN programmable data planes during runtime has become an urgent security challenge in the programmable data plane domain. To address this, we propose P4CV (P4-based rules Consistency Verification mechanism for SDN control-data plane) and have developed and evaluated a prototype system based on the BMv2 software switch. The main contributions are: (1) We propose a P4-based SDN control-data plane flow rule consistency verification mechanism that enables rapid consistency verification of SDN programmable data plane forwarding behavior at runtime; (2) We introduce a P4-based network telemetry mechanism and a symbolic execution-based consistency verification algorithm that enables rapid collection of flow rule execution information from programmable data planes and performs efficient, accurate consistency verification; (3) We build a simulation system, and experimental results show that in single-path verification scenarios, P4CV verification time is unaffected by network topology complexity, being only linearly related to the number of switching nodes on the path. In multipath verification scenarios, P4CV reduces verification time by approximately 42% compared to P4Consist while generating only about 0.06‰ bandwidth overhead.

1.1 P4 Abstract Forwarding Model

Unlike the traditional OpenFlow model in SDN, the P4 abstract forwarding model designs programmable parsers and packet processing actions, supports custom protocol types, and enables flexible handling of data packets. In the P4 abstract forwarding model presented in the literature, the packet forwarding process in switches primarily relies on three components: a programmable packet parser, multi-stage match-action pipelines, and buffers.

a) Parser. In the P4 forwarding model, the parser first processes data packets by extracting packet headers from the payload and caching them separately. Administrators can customize packet header structures and parsing processes, which are compiled by the compiler into packet header parse graphs configured on the parser. The extracted header portion is parsed according to this parse graph.

b) Multi-stage Pipeline. After parsing, packet headers pass through multiple match-action tables that can be executed sequentially, in parallel, or in combination. These tables are organized in a pipeline format, divided into ingress and egress pipelines. The ingress pipeline determines the output port and queue for packets, while the egress pipeline modifies packet header information. Logically, P4' s multi-stage pipeline forms a directed acyclic graph composed of a series of match-action tables. The underlying platform executing the P4 program processes packets strictly according to this directed acyclic graph logic.

c) Buffer. Buffers temporarily store parsed packet headers that have not yet entered pipeline processing, as well as the remaining payload after parsing.

From the perspective of P4 program workflow, the P4 abstract forwarding model includes a configuration phase and a runtime phase. The configuration phase primarily involves designing parser processes, setting the execution order of match-action tables, and specifying which header fields each match-action table processes. These configurations determine which protocols the switch supports and how it processes packets. During the runtime phase, entries can be added to or deleted from the match-action tables specified in the configuration phase, and the runtime control interface (P4Runtime) generated during configuration can be invoked at specific times to deliver matching rules to the data plane, thereby applying configuration policies to data packets.

1.2 Research Motivation

During SDN network operation, various internal and external disturbances can cause inconsistent forwarding behavior in programmable data planes, such as hardware failures or malicious attacks. Figure 1 illustrates a typical example of inconsistent forwarding behavior in an SDN programmable data plane. The control plane configures a forwarding path for the data plane through the P4Runtime interface (shown as solid lines), along with corresponding P4 processing logic and flow rules. Network administrators configure firewall rules in switch S_3 to protect downstream server resources. If switch S_2 is compromised by a malicious attack (or experiences unknown hardware failures) causing flow rules to be tampered with (shown as dashed lines), the actual traffic forwarding path changes to the alternative path shown. This causes traffic to bypass the firewall in S_3, potentially damaging downstream server resources.

Additionally, flow rule priority errors can cause similar inconsistent forwarding behavior. For example, PicOS 2.1.3 implements a mechanism where switches use software caching for flow tables when hardware tables are full. For a set of overlapping flow rules r_1 (high priority) and r_2 (low priority), if r_1 is cached in switch software while r_2 is stored in hardware tables, traffic will violate flow rule priorities and execute actions specified in r_2' s match-action table. Similarly, HP 5406zl commercial switch chips do not support flow rule priorities and always process packets according to the most recent rule, ignoring priorities. These situations prevent the programmable data plane processing logic developed by network administrators from being executed effectively and consistently.

2.1 Overall Framework

Since the aforementioned inconsistent forwarding behaviors in programmable data planes occur during network runtime, the control plane cannot obtain these security-threatening misconfigurations through static analysis of P4 programs. To address this, we propose P4CV, a consistency verification method for programmable data planes. The P4CV workflow is shown in Figure 2. First, a traffic generator injects probe flows into the network data plane. To minimize bandwidth impact, probes consist of simple five-tuple information (source/destination IP addresses, source/destination MAC addresses, and transport layer protocol) and necessary source-routed port sequences. The probe flows then reach the destination node according to data plane flow rule configurations and specified source-routed port sequences, with each switch along the path adding five-tuple matching flow rule information to the probe. When probe flows reach the destination node, P4CV samples them using the existing sFlow method. The telemetry data collected by probes (including actual forwarding paths and flow rule information) is parsed and delivered to the consistency verification module. Finally, the consistency verification module completes consistency validation of data plane forwarding behavior by analyzing forwarding paths and employing symbolic execution based on telemetry data from the data plane and reachable path graphs returned from the control plane.

P4CV consists of four functional modules:

a) Data Plane Module. Designs and implements P4-based packet processing logic to forward network traffic and embeds telemetry data into probes.

b) Control Plane Module. For given probes and source-destination nodes, returns reachable path graphs based on network topology and configuration information.

c) Consistency Verification Module. Validates data plane forwarding behavior consistency based on reachable path graphs and telemetry data.

d) Input/Output Module. P4CV' s input is a probe traffic generator, and its output is all inconsistency information on forwarding paths (or confirmation of no inconsistent forwarding behavior in the data plane).

The following sections provide detailed descriptions of each functional module.

2.2 Data Plane Module

To reduce the impact of telemetry processes on normal network communication while improving flow rule consistency verification accuracy, we propose an efficient P4-based network telemetry mechanism. Traditional In-band Network Telemetry (INT) is a passive telemetry technique that encapsulates telemetry instructions and data into normal data packets to probe specific telemetry data along paths. This approach reduces the effective payload ratio of data packets, increases data plane overhead, and impacts normal data communication in the network. In response, we employ probes with specific quantities and structures to collect flow rule execution information such as flow rule IDs matched during normal packet forwarding and input/output port numbers. This simplifies probe structure, reduces data plane overhead, and enables collection of necessary information for consistency verification.

In P4-based programmable data planes, the packet parsing process is abstracted as a Finite State Machine (FSM), where each state node represents a field in the packet header and edges represent state transitions. The parser traverses packet headers sequentially, extracting field values for processing by predefined match-action tables. P4CV' s probe parsing process is represented by the parser graph shown in Figure 3. The figure illustrates that each switch node in the P4CV data plane processes network packets using two logic paths:

a) Normal Packet Processing. Consistent with traditional IP packet processing (shown as dashed lines in Figure 3), P4CV forwards normal data packets based on Layer 3 routing or Layer 2 switching to ensure normal network communication services.

b) Probe Processing. As shown by solid lines in Figure 3, P4CV forwards probes from specified output ports according to the source-routed port sequence (SrcRoute field) in the probe header. Before forwarding probes, telemetry data is embedded into the probe packet. This data is recorded in the SW_Trace field, including the current switch ID, probe packet input port number, and the flow rule ID matched when the probe is forwarded according to Layer 3 routing or Layer 2 switching, along with the corresponding output port number—essentially the actual flow rule execution information during real data flow forwarding. The specific format will be detailed in Section 2.5.

2.3 Control Plane Module

In P4CV, the control plane primarily provides reference data (including forwarding paths and flow rules) for consistency verification. It abstracts network configuration as a triple $G = \langle V, E, R \rangle$, where V represents all nodes in the network, E represents directed edges between nodes in V, and R represents forwarding rules. For a directed edge (s, t) in G, $R(s, t)$ represents the forwarding rules existing at node s. For given probes and source-destination nodes, the control plane module delivers the reachable path graph between source and destination switching nodes to the consistency verification module based on network topology and flow rule configuration information to complete final verification.

Additionally, flow rules issued by the control plane to the data plane in P4 are stored in corresponding configuration files (typically in JSON format), with each switch node having its own independent configuration file. These files contain information about all match-action tables (i.e., forwarding rules) for each switch node, including names and parameters. The control plane module is responsible for parsing this stored switch configuration file information. When the consistency verification module receives a probe (five-tuple flow) from the data plane, the control plane parses the configuration file information, with results stored as a dictionary where keys are switch IDs and values are lists of all available rules from that switch' s JSON configuration file. Finally, the consistency verification module compares the parsed control plane configuration information with data plane telemetry data using symbolic execution to detect control-data plane inconsistencies.

Our modular design for control and data planes maintains the fundamental SDN principle of decoupling control and data planes. Therefore, the control plane' s method for managing rules on data plane network devices is device-agnostic. While JSON-based configuration files are specific to the software switch model (BMv2) used in our experiments, the proposed consistency verification method is generic. If the control plane uses other file formats to store forwarding rules, only the parsing method in the control plane module needs adjustment.

2.4 Consistency Verification Module

The consistency verification module compares telemetry data from the data plane with configuration information from the control plane to validate data plane forwarding behavior consistency. The module comprises two functional components: forwarding path analysis and symbolic execution.

As shown in Figure 2, upon receiving data plane telemetry data, the consistency verification module sends probe information to the control plane module and obtains the corresponding reachable path graph. The sent probe information includes probe packet structure and given source-destination nodes, formatted as $[packet, src, dst]$. The forwarding path analysis component employs a DFS search algorithm to traverse the reachable path graph starting from src, obtaining all paths between (src, dst) that conform to network configuration, thereby determining whether the probe' s telemetry path is a valid forwarding path for the given source-destination nodes. To accelerate traversal, the search algorithm uses a pruned DFS where depth equals the length of the probe' s forwarding path.

To improve P4CV verification accuracy, the consistency verification module must also validate matched flow rules after completing forwarding path verification. The symbolic execution component generates a symbolic packet (SP) with the same header as the probe based on the probe' s five-tuple information, then simulates the SP' s forwarding process hop-by-hop along the probe' s forwarding path. The SP simulation forwarding process uses Boolean functions as described in Section 2.2. For a directed edge (u, v) in the forwarding path graph, if node u contains a forwarding rule $r(u, v)$ consistent with control plane flow rules, node u is marked as True and SP is updated. The verification algorithm is as follows:

Algorithm 1: Consistency Verification Algorithm (Symbolic Execution)

Input: Symbolic packet SP, verification path $path\_spec$, flow rule configuration information $Rules$.

Output: Verification result $PATHS\_CHECK$, inconsistent switch node information $Error\_Report$.

```
1. for switch $\in path\_spec$ do

2. if (last switch) then

3. for rule $\in Rules$ do
4.   if check(SP, rule) == TRUE then
5.     $PATHS\_CHECK \leftarrow$ TRUE // No consistency issues on this path
6.   elseif (last rule) then
7.     $Error\_Report \leftarrow$ switch
8.     $PATHS\_CHECK \leftarrow$ False // Inconsistent path
9. else
```

```
10. for rule ∈ $Rules$ do
11.  if check(SP, rule) == TRUE then
12.    $SP \leftarrow$ Update(SP)
13.    Go to next switch // Continue verifying next node
14.  elseif (last rule) then
15.    $Error\_Report \leftarrow$ switch
16.    $PATHS\_CHECK \leftarrow$ False
17.    $SP \leftarrow$ Update(SP)
18.    Go to next switch // Continue verifying next node
```

Furthermore, to simulate actual table lookup and forwarding behavior of packets in switches, symbolic execution employs sequential table lookup (without considering flow table priority). To ensure verification accuracy, P4CV validates flow rule information including destination IP, input/output port numbers, and flow rule IDs. If flow rules match, the switch is marked as True, SP field information is updated, and verification continues to the next node. If flow rule matching fails, the node and corresponding forwarding path are marked as False, fault information is output, and verification similarly continues to the next node to detect other inconsistent switch nodes on the path.

2.5 Input/Output Module

This section describes P4CV' s input/output module.

1) Input Module
P4CV employs INT to obtain actual forwarding behavior information from the data plane, requiring probe flows to be injected into the network. Therefore, this solution' s input module is the probe traffic generator. To avoid probe flows consuming excessive link bandwidth, each probe contains only basic five-tuple information and necessary source-routed port sequences. The probe packet format is shown in Figure 4. The probe' s forwarding path is determined by the given source-routed port sequence (i.e., specifying the output port at each switch node), while the five-tuple information determines the flow rule information matched when real data flows are forwarded based on Layer 3 routing or Layer 2 switching. The information fields are set as follows:

a) SrcRoute (n bytes). Source-routed port sequence, storing output ports (7 bits) at each switch node during probe forwarding and a bottom-of-stack flag bos (1 bit) in a stack structure, occupying n bytes (where $n \leq N$, and N is the number of switching nodes on the path).

b) IPOption_INT (4 bytes). A telemetry data flag field in IP options, simultaneously recording brief telemetry information.

c) SW_Trace (4 bytes). Records flow rule information collected during forwarding, including node ID, flow rule ID, and input/output port numbers.

2) Output Module
During consistency verification, this solution compares each switch traversed by the probe and marks switches with flow rule inconsistencies. After the consistency verification module completes its work, it outputs verification results for each forwarding path. If certain paths contain inconsistencies, corresponding fault node information (including node ID and flow rule information) is also output. Additionally, since P4CV' s consistency verification algorithm continues verifying downstream switches after discovering inconsistent nodes on a path, all fault node information on the given path is output when inconsistencies exist.

3.1 Experimental Setup

To verify the feasibility and effectiveness of the proposed programmable data plane consistency verification mechanism, we designed two experiments: a single-path verification scenario shown in Figure 5 and a multipath verification scenario shown in Figure 6. The first experiment uses results from the topology in Figure 5a as baseline values for single-path verification, while the second experiment employs the fat-tree topology widely deployed in current data centers. Each experiment uses the average of 10 runs as the final result.

Experimental environment and parameter settings: The experiments use BMv2 software switches to construct network topologies, tested on a Mininet emulator in a virtual machine. The virtual machine runs Ubuntu 18.04-LTS with an Intel Core i7-9700 3.00GHz processor and 8GB RAM. In each experiment, all network switching nodes are configured with the same number of forwarding rules (15K, 30K, 60K). To ensure real-time verification results while minimizing probe flow transmission delay, the source node probe sending rate is set to 100 pps. Additionally, error configuration injection uses the same method as the example in Figure 1, modifying flow rules through the programmable switch' s command-line configuration interface. These misconfigurations are randomly distributed across nodes on forwarding paths.

3.2 Single-Path Verification

To evaluate our solution' s consistency verification performance in single-path scenarios, we constructed the network topology shown in Figure 5. In Figure 5a, S_1 serves as the source node, with S_3, S_5, S_7, and S_9 as destination nodes. The source node first pushes the source-routed port sequence corresponding to a given single path into the probe header (format shown in Figure 4), then periodically sends a group of probes. The destination node samples, parses, and completes consistency verification upon receiving probes. The time from probe transmission to P4CV completing consistency verification is recorded as the total time T for single-path verification, with symbolic execution duration recorded as T_{se}.

Experimental results in Figure 7 show that both single-path symbolic execution time and total verification time are linearly related to the number of switching nodes on the path. Since symbolic execution employs sequential table lookup and each switch contains the same number of flow rules, both symbolic execution time and configuration file parsing delay are linearly related to the number of switches on the path. Meanwhile, as probe telemetry time is at the microsecond level (calculated for 1 Gbps links) and negligible, total verification time is also linearly related to the number of switches on the path. Figure 7a demonstrates that when nodes on the verification path do not exceed 8 and the flow rule count per node is 15K, P4CV can precisely locate all error configurations within 30 seconds, providing strong real-time capability and reliability that ensures verification accuracy.

Furthermore, to verify that P4CV' s control-data plane consistency verification time is only linearly related to the number of switching nodes on the path and independent of network topology, we also conducted single-path consistency verification in the network topology shown in Figure 5b. Table 1 results show that with the same number of switching nodes on forwarding paths, P4CV' s single-path verification time across different network topologies is essentially equal, with error margins not exceeding 3%.

Table 1: Total Verification Time of Single-Path Scenario for Each Topology

Path Length (switches)	3	4	5	6	7	8
Total Time (s)						
baseline						
grid_4sw						
grid_9sw						
grid_16sw						

3.3 Multipath Verification

Data center networks represent the most widely deployed scenario for programmable data planes today. Therefore, we constructed the fat-tree topology shown in Figure 6 (pod=4) to verify the feasibility and effectiveness of our proposed programmable data plane consistency verification mechanism in practical deployments. This topology consists of core, aggregation, and edge layer switching nodes, including 4 core nodes, 8 aggregation nodes, and 8 edge nodes, with each edge node connecting to 2 servers. After pruning DFS calculation (depth=7), there exist 20 different forwarding paths between any pair of edge switches in different pods.

As shown in Figure 6, experiments select two servers in pod1 and pod3 as probe sending and receiving nodes. The figure illustrates one forwarding path between the two nodes (shown as dashed lines). Each experiment injects 20 error configurations into n forwarding paths between source and destination nodes using bash scripts, recording the time required to detect each error configuration.

Figure 8 presents the cumulative distribution function (CDF) of error configuration detection time for both P4CV and P4Consist verification methods. Results show that both can detect all error configurations in the network, but P4CV consistently detects errors on each path earlier than P4Consist. By introducing source-routed forwarding, P4CV accelerates probe flow traversal of forwarding paths. Moreover, P4CV' s consistency verification algorithm continues verifying downstream nodes after discovering error configurations on a path, enabling a single probe to complete consistency verification for an entire path and significantly reducing probe flow detection time for error configurations. Additionally, Figure 8 shows that the difference in total verification delay between the two methods increases with more flow rules per node. With 15K, 30K, and 60K flow rules per switch, P4Consist requires 7 minutes, 18 minutes, and 36 minutes respectively to detect all 20 error configurations, while P4CV requires only 4 minutes, 10 minutes, and 20 minutes–improving detection efficiency by an average of 42%.

3.4 Data Plane Overhead

Unlike traditional schemes that rely on probe packet loss rate and delay metrics for verification, our solution employs a P4-based network telemetry mechanism in the data plane to collect actual forwarding behavior information. Since source routing explicitly defines the telemetry path, a single probe suffices to collect flow rule information for one path. As shown in Figure 4, the probe format yields a maximum total length of $46 + 5N$ bytes (where N is the number of switches on the path). Using the 4-layer fat-tree topology in Figure 6 as an example ($N = 7$), the maximum probe length is 84 bytes. With probe flows injected at 100 pps, this consumes at most approximately 0.06‰ of bandwidth on a 1 Gbps link.

4 Conclusion

This paper proposes P4CV, a P4-based SDN control-data plane flow rule consistency verification mechanism, to address flow rule inconsistencies between SDN control and data planes. P4CV leverages a P4-based network telemetry mechanism that fully exploits the flexible, definable packet processing capabilities of programmable data planes to rapidly collect flow rule execution information from the data plane. Additionally, we propose a symbolic execution-based consistency verification algorithm that completes flow rule consistency verification for both single-path and multipath forwarding scenarios between given source-destination switching nodes. Simulation results demonstrate that compared to existing flow rule consistency verification mechanisms, P4CV' s single-path verification time is only linearly related to the number of switching nodes on the path and unaffected by network topology changes. For multipath forwarding scenarios, P4CV significantly improves verification efficiency while further reducing data plane overhead. Moreover, as our solution is designed for P4-based SDN networks, its core verification model can be rapidly deployed on all P4-programmable switching devices, offering excellent scalability. Future work will focus on optimizing the verification algorithm to further reduce verification time.

References

[1] McKeown N, Anderson T, Balakrishnan H, et al. OpenFlow: enabling innovation in campus networks [J]. ACM SIGCOMM Computer Communication Review, 2008, 38(2): 69-74.

[2] Bosshart P, Daly D, Izzard M, et al. P4: programming protocol-independent packet processors [J]. ACM SIGCOMM Computer Communication Review, 2013, 44(3): 87-95.

[3] Song Haoyu. Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane [C]// Proc of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking. New York: ACM Press, 2013: 127-132.

[4] Hauser F, Häberle M, Merling D, et al. A survey on data plane programming with P4: fundamentals, advances, and applied research [J]. arXiv Preprint arXiv:2101.10632, 2021.

[5] 林耘森箫, 毕军, 周禹, 等. 基于 P4 的可编程数据平面研究及其应用 [J]. 计算机学报, 2019, 42(11): 22. (Lin Yunsenxiao, Bi Jun, Zhou Yu, et al. Researches and applications of programmable data plane based on P4 [J]. Chinese Journal of Computers, 2019, 42(11): 22.)

[6] Dumitru M V, Dumitrescu D, Raiciu C. Can we exploit buggy P4 programs? [C]// Proc of the Symposium on SDN Research. New York: ACM Press, 2020: 62-68.

[7] Perešíni P, Kuźniar M, Kostić D. Monocle: dynamic, fine-grained data plane monitoring [C]// Proc of the 11th ACM Conference on Emerging Networking Experiments and Technologies. New York: ACM Press, 2015: 1-13.

[8] Kuźniar M, Perešíni P, Kostić D. What you need to know about SDN flow tables [C]// International Conference on Passive and Active Network Measurement. Berlin: Springer, 2015: 347-359.

[9] Wen Xitao, Bu Kai, Yang Bo, et al. Rulescope: inspecting forwarding faults for software-defined networking [J]. IEEE/ACM Trans on Networking, 2017, 25(4): 2347-2360.

[10] Zhao Yu, Wang Huazhe, Lin Xin, et al. Pronto: efficient test packet generation for dynamic network data planes [C]// IEEE the 37th International

Conference on Distributed Computing Systems. Brussels: IEEE Computer Society, 2017: 13-22.

[11] Zhang Peng, Li Hao, Hu Chengchen, et al. Mind the gap: monitoring the control-data plane consistency in software defined networks [C]// Proc of the 12th International on Conference on emerging Networking Experiments and Technologies. New York: Association for Computing Machinery, 2016: 19-33.

[12] 赵会, 吕光宏, 杨洋, 等. SDN 故障分析研究综述 [J/OL]. 计算机应用研究, 2020, 37(10). (2019-10-24) [2021-10-19]. https://www.arocmag.com/article/01-2020-10-003.html (Zhao Hui, Lyu Guanghong, Yang Yang, et al. SDN fault analysis research [J/OL]. Application Research of Computers, 2020, 37(10). (2019-10-24) [2021-10-19]. https://www.arocmag.com/article/01-2020-10-003.html)

[13] Liu J, Hallahan W, Schlesinger C, et al. P4v: practical verification for programmable data planes [C]// Proc of the 2018 Conference of the ACM Special Interest Group on Data Communication. New York: ACM Press, 2018: 490-503.

[14] Freire L, Neves M, Leal L, et al. Uncovering bugs in p4 programs with assertion-based verification [C]// Proc of the Symposium on SDN Research. New York: ACM Press, 2018: 1-7.

[15] Stoenescu R, Dumitrescu D, Popovici M, et al. Debugging P4 programs with Vera [C]// Proc of the 2018 Conference of the ACM Special Interest Group on Data Communication. New York: ACM Press, 2018: 518-532.

[16] Gray N, Grigorjew A, Hosssfeld T, et al. Highlighting the gap between expected and actual behavior in p4-enabled networks [C]// IFIP/IEEE Symposium on Integrated Network and Service Management. Piscataway, NJ: IEEE Press, 2019: 731-732.

[17] Shukla A, Hudemann K N, Hecker A, et al. Runtime verification of p4 switches with reinforcement learning [C]// Proc of the 2019 Workshop on Network Meets AI & ML. New York: ACM Press, 2019: 1-7.

[18] Shukla A, Fathalli S, Zinner T, et al. P4Consist: toward consistent P4 SDNs [J]. IEEE Journal on Selected Areas in Communications, 2020, 38(7): 1293-1307.

[19] Behavioral Model Repository. P4 Language Consortium [EB/OL]. [2021-10-11]. https://github.com/p4lang/behavioral-model.

[20] Zhang Peng, Zhang Cheng, Hu Chengchen. Fast data plane testing for software-defined networks with RuleChecker [J]. IEEE/ACM Trans on Networking, 2018, 27(1): 173-186.

[21] Phaal P, Panchen S, McKee N. InMon corporation’ s sFlow: a method for monitoring traffic in switched and routed networks [S]. RFC3176, 2001.

[22] Zhu Yibo, Kang Nanxi, Cao Jiaxin, et al. Packet-level telemetry in large datacenter networks [C]// Proc of the 2015 ACM Conference on Special Interest

Group on Data Communication. New York: ACM Press, 2015: 479-491.

[23] Pal C, Veena S, Rustagi R P, et al. Implementation of simplified custom topology framework in Mininet [C]// Asia-Pacific Conference on Computer Aided System Engineering. Piscataway, NJ: IEEE Press, 2014: 45-50.