2021 3rd International Conference on Advances in Computer Technology, Information Science and Communication (CTISC) | 978-1-6654-1868-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/CTISC52352.2021.00022

SRCV: A Source Routing based Consistency Verification Mechanism in SDN

Jiqiang Xia
Information Technology Institute
PLA Strategic Support Force Information Engineering University
Zhengzhou, China
Mainblack@126.com

Pengshuai Cui
Information Technology Institute
PLA Strategic Support Force Information Engineering University
Zhengzhou, China
cuipengshuai@gmail.com

Ziyong Li
Information Technology Institute
PLA Strategic Support Force Information Engineering University
Zhengzhou, China
17629352940@163.com

Julong Lan
Information Technology Institute
PLA Strategic Support Force Information Engineering University
Zhengzhou, China
18811378198@163.com

***Abstract*—The decoupling of the control plane and data plane in software-defined network (SDN) not only promotes the innovation of network architecture but also leads to the inconsistency faults between the two layers. Through the precious works, we realize that this issue got even worse when P4, a programming language for data plane, was introduced in SDN, since many bugs and unpredictable faults may appear whether the programs are compiled or running in devices. To verify the control-data plane inconsistency in P4-based SDN, we present the design and implementation of SRCV, a source routing based control-data plane consistency verification mechanism for P4-based SDN at runtime. Firstly, SRCV generates the active probe traffic with source routing labels, and then the probe forwards along the source routing path and collects the matching flow rules information. Finally, those collected information are compared with the control plane flow rules information through symbolic execution to conduct the consistency verification. Our evaluation results show that, compared with the existing control-data plane consistency verification mechanism in P4-based SDN, the verification time of SRCV is not affected by the network topology, and only the number of switches on the checking path is linearly correlated.**

Keywords-Software-defined network; programmable data plane; consistency verification

I. Introduction

Software-defined network (SDN) [1] separates the management and control logic of the network from the devices, decouples the control plane and the data plane, constructing a novel prototype of an efficient centralized control plane. However, this separation leads to the consistency problem: whether the flow rules populated by the controller are strictly executed by the data plane.

The emergence of Domain Specific languages (DSLs) e.g., P4 [2] and POF [3], intensifies the consistency security problem of the control plane and data plane despite making the data plane programmable like the control plane. On the one hand, as a programming language, P4 is prone to bugs inherent and that become potential security threats to the data plane. On the other hand, when deploying P4 programs to the hardware, the control-data plane inconsistency possibly appears even the programs are virginal, in the case of network faults occur, including network configuration or hardware failure, operator's misconfiguration, and malicious attacks. These issues above violate the premise consensus that the logically centralized control plane is consistent with the actual state of the data plane in SDN. Therefore, they will cause severe damage to the network infrastructure, lead to unpredictable network security compromise.

Existing network testing systems [4], [5] prevailing to inject probe packets into the network to collect the runtime status of devices, but mostly rely on layer-3 routing or layer-2 switching to forward probes. This method needs to inject large number of probes to check all the rules or paths with a long checking time, which may not only affect basic communication of network but also possibly construct fake checking results due to the lack of instantaneity. Besides, more attention should be paid to the key nodes deployed with vital network functions, e.g., firewall, and the consistency of the critical links passing through these nodes is supposed to be concerned. How to quickly discover the control-data plane inconsistency of critical links at runtime becomes an urgent security problem in P4-based SDN. Given that *source routing* [6] can specify part of or all nodes that the packet passes through in the end-to-end communication, we can set the critical links as the forwarding paths of probes, thus quickly collecting the state of the data plane at runtime. The key contributions of this paper are as follows:

- We propose SRCV, a source routing based control-data plane runtime consistency verification mechanism.

- We introduce a source routing mechanism into the data plane, set the key nodes as the intermediate nodes of the probe forwarding path to make the verification path specifically.
- We implement an SRCV prototype in Mininet [7], and evaluate it with different topologies. The results show that the verification time of SRCV is topology independent.

II. Related Work

A. Consistency Verification for OpenFlow-based SDN

Before the emergence of data plane programming languages, e.g., P4, the control-data plane consistency in SDN has aroused wide concern. Monocle [4] quickly generates probes for specific rules by expressing forwarding logic as a Boolean Satisfiability (SAT) problem. In fact, Monocle is an agent working between the controller and the data plane which can not only verify the installed flow rules in the persistent network but also track the installing of the updated flow rules during reconfiguration. Furthermore, Rulescope [8] proposes a more accurate and efficient consistency check algorithm for the rule priority faults found in commercial switches [9]. These proactive network testing methods can verify the flow rules known or synchronized to the controller, but cannot detect the faults caused by the unknown or misconfigured flow rules in the data plane.

Unlike these approaches, VeriDP [10] abstracts the configuration of the control plane into a path table then samples the real network traffic to generate a probe, and finally compares the path information collected by the probe with the path table. VeriDP can effectively verify the forwarding path of real network traffic, but this scheme needs to build the path table according to the forwarding path between any two ports in the network in advance, which increases the resource consumption of the controller. The above schemes are all aimed at the traditional SDN control-data plane consistency verification scenario, which needs to be redesigned to apply to the P4-based SDN scenario.

B. Verification for Buggy P4 Programs

Just like other programming languages, P4 suffers from inherent programming language faults [11]. For example, the handling of invalid header fields can cause unpredictable security problems in the data plane. Other similar security vulnerabilities have been specified by the P4 community [12], and many assertion-based verification efforts have been made to address vulnerabilities in P4 programs [13]-[15]. These methods can effectively find all kinds of security vulnerabilities in P4 programs, but they can't detect the faults and inconsistencies caused by the program whether it is compiled or running, and often require developers to add corresponding assertion annotations in P4 programs, which increases the burden of network developers.

C. Consistency Verification for P4-based SDN

In contrast with static program verification, inconsistency faults that occur in the data plane at runtime need more attention. One recent work [16] has inspired the study of the control-data plane in P4-enabled SmartNICs. This work highlights the serious impact on network performance such as latency throughput when only focusing on the design of the P4 program while ignoring the resource constraints of the device itself. By introducing a fuzzing mechanism based on reinforcement learning, P4RL [17] realizes the automatic verification of a single P4 switch at run time, but it is not suitable for checking the consistency of the control-data plane in SDN. Furthermore, they also proposed a control-data plane consistency verification method, P4Consist [18], for P4-based SDN. In this method, for the first time, the P4-based In-band Network Telemetry (INT) [19] technology is applied to collect the status of the data plane, and they realized the control-data plane consistency verification for P4-based SDN. But because of requiring traversal of all possible paths between a given source-destination pair, the checking time of P4consist increases dramatically as the network topology becomes more complex.

Based on the previous works, this paper proposes a control-data plane consistency verification mechanism for critical links in P4-based SDN. By utilizing a source routing mechanism, the actual state of the data plane at runtime can be obtained effectively, thus improving the efficiency and flexibility of control-data plane consistency verification.

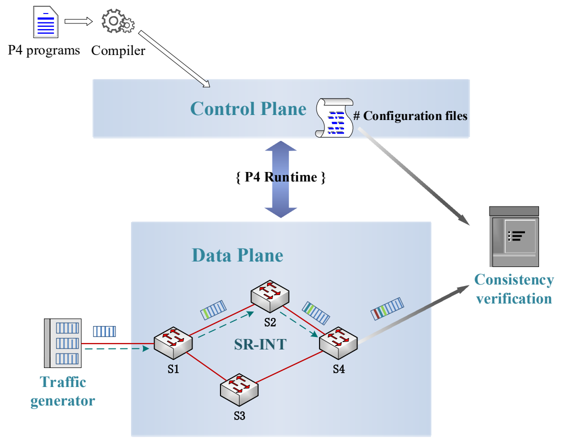

Figure 1. SRCV architecture.

III. SRCV Design

SRCV can check the control-data plane consistency of a specific path between a given source-destination address pair.

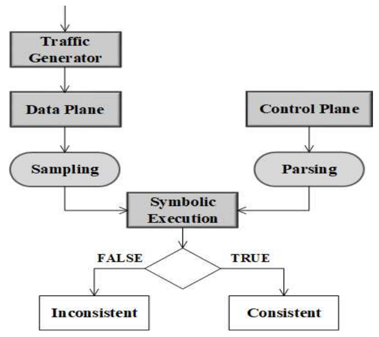

Figure 2. SRCV workflow.

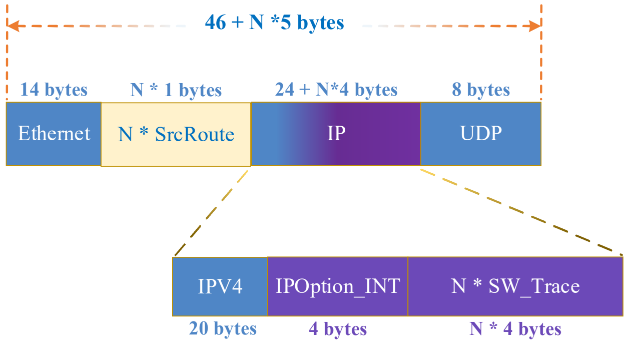

Figure 3. Headers of the probe.

Fig. 1 specifies the roles of each module, i.e., *Traffic Generator, Data Plane, Control Plane*, and *Consistency Verification* in SRCV architecture.

Besides, Fig.2 describes the workflow of SRCV as follows: Firstly, the probe traffic, consists of simple 5-tuple information and the necessary stack of source routing ports, is sent by the *Traffic Generator*. Then, the probe traffic travels to the destination based on the source routing stack, with each switch through the way pushing telemetry data into the probe header. When the probes reach the destination node, the telemetry data collected in it is extracted and delivered to the *Consistency Verification* module. Finally, according to the telemetry data, the *Consistency Verification* module generates a symbolic packet with a similar header, simulates its forwarding process along the given path, and verifies each rule with configuration files parsed from the *Control Plane* module to complete consistency check. Specifically, each module in SRCV architecture can be described as follows.

A. *Traffic Generator*

The traffic generator module is a proactive probe traffic generator. Besides, to avoid the probe traffic taking up much links bandwidth, the probe packet contains only the basic 5-tuple information and the necessary source routing port stack. The probe format is shown in Fig.3. The forwarding path of the probe packet is specified by the given source routing port stack, and the 5-tuple information is used to determine the path and rule information when the real data is transmitted.

As Fig. 3 shows, source routing port stack is assigned at the probe SrcRoute fields, and the corresponding stack bottom tag *bos* (bottom of the stack), total cost $N * 1$ bytes (N for the switch number in the path) to record the telemetry data, probe header add IPOption_INT optional fields, and the telemetry data gathered is stored in SW_Trace fields taking up $N * 4$ bytes.

B. *Data Plane*

INT [19] is adopted in SRCV to obtain the execution of the actual flow rules on the data plane. The processing of arriving probes by each switch can be roughly divided into two parts: first, the probe is sent out by the given outport according to the source routing port stack in the packet header, i.e., SrcRoute field; Second, before sending probes out, each switch makes telemetry data pushed into probe header, recognized as SW_Trace field. This field including not only the current switch ID & inports, but also recorded the rule IDs & corresponding outports in the case of the probe traffic forwarded based on layer-3 routing or layer-2 switching, namely real traffic forwarding rules. When the probes reach the destination node, the telemetry data collected will be parsed out and delivered to the Consistency Verification module to conduct a consistency check.

Moreover, the packet parsing process is abstracted as a Finite State Machine (FSM) in P4, with each state node corresponding to a field in the packet header. The parser traversal the packet header in sequence, extract the values of each field during execution and send them to the Match-Action Table for processing. We designed our parser which is stick to the packet format in Fig. 3.

C. *Control Plane*

When the consistency verification module receiving probes (5-tuple flow) from the data plane, the control plane starts to parse and store the information of the configuration file and send it to the consistency verification module. Then the symbolic execution is conducted by consistency verification module to detect inconsistency faults between the control-data plane.

In P4, the flow rules populated from the control plane to the data plane are stored in the corresponding configuration file (usually in JSON format), and each switch has an independent configuration file. These files contain the names, parameters, and other information of all the matching action lists (i.e., forwarding rules) for each switch. Once the control plane module has finished parsing the forwarding rules, the information in the file is saved into a dictionary, where the key is the switch ID and the value is a list of all

Algorithm 1 Consistency Verification

Input: The network configuration ***Rules***, the symbolic packet ***SP***, and the given verification path ***PATH_SPEC***.
Output: The checking results ***PATH_CHECK*** and the ***Error_Report*** for inconsistent switches.

```
1:  for switch ∈ PATH_SPEC do
2:    if (last switch) then
3:      for rule ∈ Rules do
4:        if check (SP, rule) == TRUE then
5:          PATH_CHECK ← TRUE // no faults in this given path
6:        else
7:          if (last rule) then
8:            SWITCH_CHECK ← False
9:            PATH_CHECK ← False   // inconsistent path
10:           Error_Report
11:   else
12:     for rule ∈ Rules do
13:       if check (SP, rule) == TRUE then
14:         SWITCH_CHECK ← TRUE // consistent switch
15:         Go to next switch // continue checking the next node
16:       else
17:         if (last rule) then
18:           SWITCH_CHECK ← False
19:           PATH_CHECK ← False // inconsistent path
20:           Error_Report
```

available rules in the JSON configuration file for that switch.

This paper adopts the modular-design method through the control plane to the data plane, inheriting the idea in SDN that the control plane and the data plane are supposed to be decoupled. Therefore, methods for managing the data plane rules of different network devices are device-independent. In fact, the configuration file based on JSON format is specific to the target model (BMV2 [20]) used in our prototype, while the consistency checking method proposed in this paper is universal. If the control plane using different file formats to store forwarding rules, nothing has to be changed except adjusting the control plane parsing process.

D. Consistency verification

This module is responsible for checking the information from the control plane module and the data plane module to complete the control-data plane consistency verification. When receiving the data plane telemetry data, it generates a Symbolic Packet (SP) according to the 5-tuple information of the probes and then simulates the forwarding process of the SP hop-by-hop along the given path, i.e., symbol execution. Algorithm 1 illustrates how we conduct the symbolic execution on SP. It is a Boolean function that conducts the bitwise checking of the information from the control plane and the data plane, including destination IP, inport/outport number, forwarding rule IDs, etc., to ensure the accuracy of the verification results.

Besides, to simulate the actual table lookup behavior of packets in the switch, symbol execution adopts the way of sequential table lookup (regardless of the rule priority). In this case, if a rule matches, the switch is marked as forwarding: True and the symbolic packet fields are updated, then continue checking the next switch in the path. If the checking of the rules fails, i.e., none of the rules forward the packet, the current switch is marked as not forwarding: False and accordingly the whole path is marked as False. But in that case, we still conduct checking of the next switch to find if there is an inconsistency error in the downstream switches of the path.

IV. EVALUATION

In this section, we present the implementation of SRCV and evaluate its effectiveness and performance. The experim-

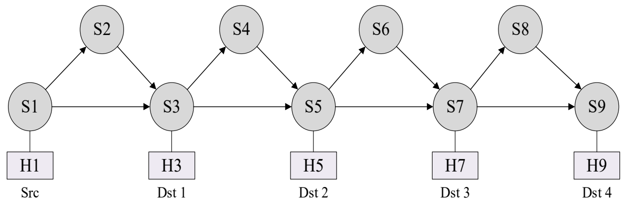

Figure 4. Illustration of our experimental topology.

-ental environment and parameter settings are described as follows:

- Our experiments are conducted on a Linux virtual machine (Ubuntu 16.04 LTS) with 4 logical CPUs at 3.00GHz and 8GB of RAM.
- We configured different network scenarios in Mininet [7] with a series of virtual switches. Each virtual switch implemented the second version of P4, known as Behavioral-Model (BMv2 [20]).
- In each experiment, the same number of forwarding rules were configured for each switching node, i.e., 15K,30K,60K, and the probe generating rate was set as 100pps.

A. Performance Test

To explore the relationship between the time of single specific path verification in our mechanism and the number of switching nodes of that path, the network topology as shown in Fig. 4 is built, in which *H1* is the source node and *H3, H5, H7* and *H9* are the destination nodes in turn.

First, *H1* pushes the single given path *PATH_SPEC* into the probe headers (as shown in Fig. 3), and then periodically sends a group of probes. After the destination node sizes the probes, consistency verification is carried out. Symbol execution time is *symexe_time*, and the time from the issuing of the probe to the completion of the consistency check on the destination node is *last_time*, that is, the total time to complete the single path check for each set of given source node pairs. The average value of multiple experiments is taken as the results (av_symexe_time, av_last_time). The experimental results are shown in Fig. 5.

Fig. 5 shows that both symbol execution time and the total checking time for a single path are linearly related to the number of switches on the path. Due to the symbolic exe

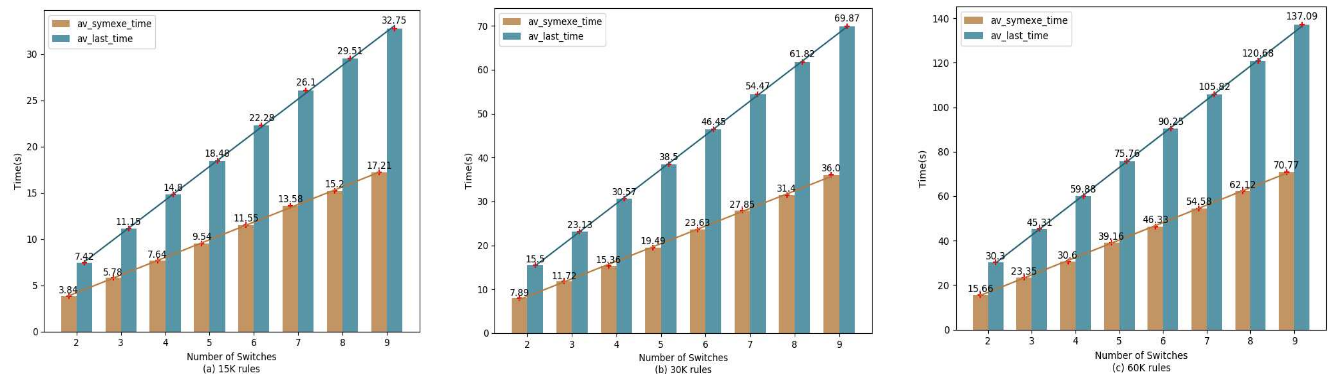

Figure 5. The symbolic execution time and the total verification time for the different number of switches on the path.

-cution with the method of the sequential lookup table, each switch is configured with the same number of flow rules, therefore the symbolic execution time is linearly related to the number of switches on the path. Also, since the time of INT for microsecond (in terms of 1 Gbps link), that is negligible to the total checking time, so the total checking time is linear-correlation with the switch quantity of that path.

B. *Data Plane Overhead*

Different from the traditional schemes which rely on the probe packet loss rate or delay and other indicators, the data plane of our scheme utilizes the INT [19] mechanism to collect the information of path and flow rules. As the source routing mechanism specifies the telemetry path, a single probe is sufficient to collect flow rule information for a single path. The format of the probe is shown in Fig. 3. The total length of a single probe is $46 + 5N$ bytes (N is the number of switches on the path). Taking the 4-layer fat-tree topology as an example ($N = 7$), the probe length is 84 bytes, and the probe traffic is injected into the network at a rate of 100pps (packets per second), resulting in only about 0.06‰ bandwidth overhead for a 1Gbps link.

V. Conclusions

To solve the problem of inconsistency between the control plane and data plane in the P4-based SDN network, we propose SRCV, a source routing based control-data plane runtime consistency verification mechanism. By introducing the source routing mechanism, the key nodes of the network are set as the intermediate nodes of the probe forwarding path, which makes the checking process more efficient and flexible. Our evaluation results demonstrate that, compared with the existing control-data plane consistency verification mechanism in P4-based SDN, the performance of SRCV is not affected by the network topology, and only the number of switches on the checking path is linearly correlated. Besides, SRCV brings as little as 0.06‰ overhead on the network data plane. For part of our future work, we will attempt to optimize the checking algorithm to further reduce the verification time.

Acknowledgment

This work is supported by the National Key Research and Development Project of China (2019YFB1802501), and the National Natural Science Foundation of China for Innovative Research Groups under Grant 61521003.

References

[1] McKeown N, Anderson T, Balakrishnan H, et al. OpenFlow: enabling innovation in campus networks. 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 H. Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane [C] //Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking. ACM, 2013, pp. 127—132.

[4] Peter Perešíni, Maciej Kuźniar, and Dejan Kostić. "Monocle: Dynamic, Fine-grained Data Plane Monitoring", In Proceedings of CoNEXT, Dec. 2015, pp. 1-13, doi:10.1145/2716281.2836117.

[5] Yu Zhao, Huazhe Wang, Xin Lin, Tingting Yu, and Chen Qian. 2017. Pronto: Efficient Test Packet Generation for Dynamic Network Data Planes. In Proceedings of ICDCS.

[6] Sunshine C A. Source routing in computer networks[J]. ACM SIGCOMM Computer Communication Review, 1977, 7(1): 29-33.

[7] PAL C, VEENA S, RUSTAGI R P, et al. Implementation of simplified custom topology framework in Mininet[C]. 2014 Asia-Pacific Conference on Computer Aided System Engineering, South Kuta, Indonesia, 2014: 48–53. doi:10.1109/APCASE.2014.6924470.

[8] Wen X, Bu K, Yang B, et al. Rulescope: Inspecting forwarding faults for software-defined networking[J]. IEEE/ACM Transactions on Networking, 2017, 25(4): 2347-2360.

[9] M. Ku´ zniar, P. Perešíni, and D. Kosti´ c, "What you need to know about SDN flow tables," in Proc. PAM, 2015, pp. 347–359.

[10] P. Zhang et al., "Mind the gap: Monitoring the control-data plane consistency in software defined networks," in Proc. 12th Int. Conf. Emerg. Netw. Exp. Technol., Dec. 2016, pp. 19–33.

[11] Dumitru M V, Dumitrescu D, Raiciu C. Can we exploit buggy P4 programs ?[C]//Proceedings of the Symposium on SDN Research. 2020: 62-68.

[12] P. L. Consortium. P4 Language and Related Specifications. Accessed: Mar. 2021. [Online]. Available: https://p4.org/specs/.

[13] Liu J, Hallahan W, Schlesinger C, et al. P4v: Practical verification for programmable data planes[C]//Proceedings of the 2018 Conference of the ACM Special Interest Group on data communication. 2018: 490-503.

[14] Eichholz M, Campbell E, Foster N, et al. How to avoid making a billion-dollar mistake: Type-safe data plane programming with SafeP4[J]. arXiv preprint arXiv:1906.07223, 2019.

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

[16] N. Gray, A. Grigorjew, T. Hosssfeld, A. Shukla, and T. Zinner, "Highlighting the gap between expected and actual behavior in p4-enabled networks," in Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage. Demos, Apr. 2019, pp. 731–732.

[17] Shukla A, Hudemann K N, Hecker A, et al. Runtime verification of p4 switches with reinforcement learning[C]//Proceedings of the 2019 Workshop on Network Meets AI & ML. 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. doi:10.1109/JSAC.2020.2999653.

[19] In-Band Network Telemetry (INT) Specification V2.1. P4 Language Consortium. Accessed: Mar. 2021. [Online]. Available: https://github.com/p4lang/p4-applications/blob/master/docs/INT_v2_1.pdf.

[20] Behavioral Model Repository. P4 Language Consortium. Accessed: Mar. 2021. [Online]. Available: https://github.com/p4lang/behavioral-model.