# Formal Verification and Testing of Data Plane in Software-Defined Networks: A Survey

Jiangyuan Yao[1], Min Jing[1], Shengjun Lin[1], Deshun Li[1](✉), and Xingcan Cao[2]

[1] Hainan University, Haikou 570228, Hainan, China
lideshunlily@qq.com
[2] University of British Columbia, Vancouver, BC V6T1Z1, Canada

**Abstract.** Software-defined network (SDN) separates the control plane and the data plane, which provides the programmability of the network and is widely deployed in data center networks. As the foundation of SDN, the data plane needs to be fully verified and tested to ensure its correctness and reliability. At present, formal verification and testing methods have been applied to SDN networks. The goals of verification and testing are to find the design defects and the implementation errors of the data plane, respectively. In this paper, we conduct a survey of the state-of-art methods and tools of formal verification and formal testing for SDN data plane. According to support for online verification, the related works of formal verification for the data plane fall into static verification and real-time verification. According to the requirement of source code, the existing works of formal testing for the data plane fall into white-box testing and black-box testing. Based on the state-of-art approaches of verification and testing, we also discuss the research trends of verification and testing for SDN data plane, such as artificial intelligence (AI)-based model construct and property definition, and scalable support for the stateful data plane.

**Keywords:** Software-defined network (SDN) · Data plane · Verification · Testing · Formal method

## 1 Introduction

SDN is a new network paradigm, which separates the control plane and the data plane and realizes the centralized control of the network through a central controller. Data plane also known as the forwarding plane, which consists of switches and other forwarding equipment. Compare with traditional networks, these forwarding devices are simple forwarding components without embedded intelligence to make their own decisions. The programmability of control plane is supported by data plane which is the foundation of SDN. SDN also makes it easier to apply machine learning strategies. Based on this, some scholars have proposed Intelligent Software Defined Network (ISDN) to deal with new challenges [3].

The programmability brings new possibilities to the future network, which introduces the risk of network errors. Although some methods have been proposed [1] to effectively handle network congestion and load balancing in SDN networks, it is still necessary to verify and test SDN networks. Due to the difference of network properties, the technology of verification and testing in traditional network cannot be directly applied to SDN data plane. SDN has been paid more and more attention by the academia, and this paper focuses on the verification and testing with formal methods of data plane. The formal methods employ rigorous logical reasoning to check whether the model meets the system specifications, which has high accuracy and efficiency of verification and testing in data plane.

The formal verification in data plane of is used to check whether the packet forwarding path is consistent with the expectation of controller. The main methods include reading a snapshot of the flow table of the switch at a certain time, and monitoring the communication between the controller and the switch in real time, which constructing a formal model to verify the model. With a lot of effort, a lot of work has been done to help users find design flaws in data planes, such as HSA [15], VeriFlow [16], and NetPlumber [14], and more.

The formal testing of the SDN data plane is to check the forwarding behavior of each switch by generating probe packets. According to whether the source code is needed, these works are divided into the white-box testing methods and the black-box testing methods. Testing is used to find implementation errors in the data plane devices, thereby helping users or developers to improve the reliability of the data plane. Stand for Monocle [18], RuleScope [8], RuleChecker [24], and more.

Based on the survey, we believe that the future development direction of SDN data plane verification and testing may include the following three aspects. (1) AI-based model construct, (2) AI-based property definition, (3) scalable support for the stateful data plane.

The remainder of this paper is organized as follows. In Sect. 2, we introduce the formal verification of the SDN data plane. In Sect. 3, we introduce the formal testing of the SDN data plane. In Sect. 4, we put forward a vision for future work. Section 5 concludes this paper.

## 2 Verification

When verifying the data plane of traditional networks, we can collect FIB (Forwarding Info Base) through SNMP (Simple Network Management Protocol), a terminal, or a control session, representing Anteater [17], NETSAT [25], and so on. For SDN data plane verification, the main idea is to read the information of the flow tables from the data plane devices and monitor the real-time communication between the controller and the devices, as shown in Fig 1. We roughly divide the verification works into two categories: one is static verification, and the other is real-time verification.

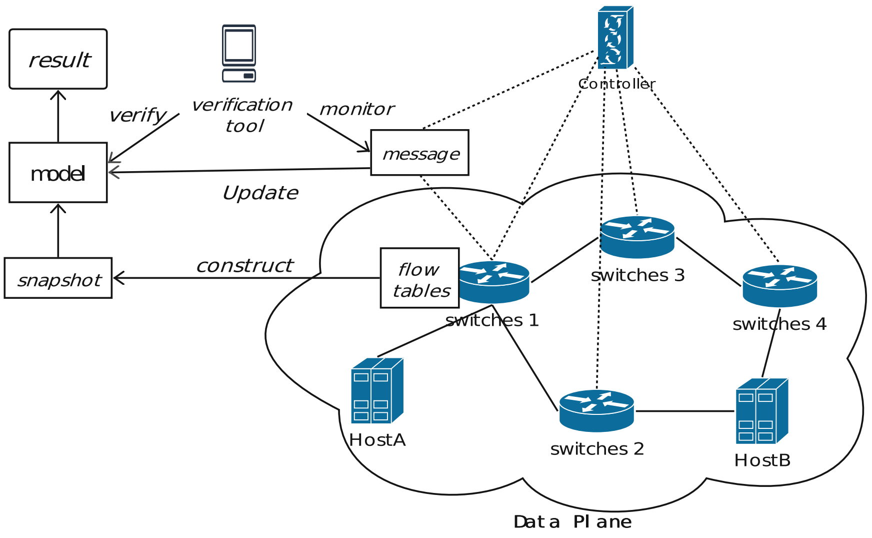

**Fig. 1.** Workflow of SDN data plane verification

## 2.1 Static Verification

The static verification of the SDN data plane is mainly to construct a snapshot at a certain moment by reading the flow table of the switch, and then build a model based on the snapshot, and then use a verification tool to check whether the model meets the defined rules, and then find possible data plane design.

Ehab et al. propose FlowChecker [2], which first applies model checking to SDN networks, and can accurately detect the correctness of the data plane. They code the flow tables as the Binary Decision Diagram (BDD), encode the forwarding rules as Boolean expressions. FlowChecker use the model checking tool NuSVM to verify the invariants of the network. Although FlowChecker can check the correctness of protocol deployment, its poor scalability indicated that it can only be used on small-scale networks.

Natali et al. propose a verification method based on the OpenFlow Specification. They propose a general OpenFlow switch model based on first-order logic. Use Alloy [19] to model and verify the OpenFlow switch, and finally use the SAT solver to solve it. It can well detect the violation of invariants such as black holes and forwarding loops, but it also has the common shortcomings of other model-based checking tools such as state space explosions, etc.

HSA [15], proposed by the Kazemian team, combines the formal methods and the network domain features to verify the data plane by checking the network boxes. First, it abstracts the data packet header as a subset of the geometric space, then uses the network transfer function and the topology transfer function to model different individual network boxes, and finally combines all the individual network boxes into a large network box, which contains all network behaviors. It can use several algorithms to check network invariants, such as reachability failure, loops, and so on. However, when the number of bits in the header space is large, the cost of exploring the header space state of an exponential packet is enormous.

Son et al. proposed FLOVER [20], a model checking system that verifies that the set of instantiated flow policies in an OpenFlow network does not violate the network security policy. They proposed a formal approach to demonstrate the consistency of dynamically generated OpenFlow flow rules with non-by-pass security attributes, including those with set and goto table actions. Using FLOVER, OpenFlow rules and network security policies are converted to an assertion set, which is then processed and validated by the SMT solver. FLOVER can detect violations that override and modify up to 200 rules in more than 100 ms.

Static verification summary: As SDN was just beginning to be known to the public, so the initial verification work was not much, and was limited to static verification. They all verify the accuracy of the network to a limited extent, but these tools find problems after the data plane goes wrong, these errors may have caused damage to the network. More importantly, static verification is not efficient and cannot be applied to dynamically change networks.

## 2.2 Real-Time Verification

The network changes with time due to the addition and deletion of the rules for the controller to issue the flow table. Previous tools were not sufficient to check the correctness of each network update. To implement real-time verification, it is necessary to get the continuous update in real-time and improve the performance of the verification methods. Fortunately, in SDN networks, forwarding rules can be obtained by monitoring the control messages, such as insertion, deletion, or modification, between the controller and switch, to achieve real-time verification.

VeriFlow [16], proposed by Khurshid, is the first real-time verification system that can check network invariants in a few hundred microseconds. It observes state changes between the control plane and the data plane, and dynamically checks the validity of network-wide invariants as each rule is inserted. The network is divided into a set of equivalent classes, and the packets belonging to the equivalent class go through the same forwarding path in the whole network. VeriFlow iterates through the paths of the equivalent classes to determine the state of one or more invariants. Then it can find where the failure occurred. However, when a rule has multiple matching fields to check, the number of equivalent classes may be too large for quick verification.

Different from VeriFlow, Yang et al. give a new method for the division of equivalence classes. They use PreCherker [9] to dynamically identify conflicting rules and classify them into equivalence classes. A multi-terminal BDD (MTBDD) structure is proposed to express the equivalent classes. This significantly improves the efficiency of network verification.

Kazemian et al. have improved on HSA and propose NetPlumber [14] a new real-time policy checking tool. They run HSA checks incrementally and use NetPlumber to check for updates in real-time. It does not have to write new code for each policy check like HSA. Once Netplumber detects the occurrence of an error, it prevents the new policy from taking effect Although it is generally fast enough, it takes a long time to update the dependency diagram when a link is up or down.

Yang et al. also inspired by HSA, proposes a new approach, NetV [10]. They redefine the rule function based on packet header space and present the BDD transformation and inverse transformation algorithm, which speeds up the rule updating, and can verify invariants among domains.

In the SDN real-time verification methods, there are some works based on atomic predicates. Atomic predicates, first proposed by the Yang [21] team, use atomic predicates to quickly calculate the intersection and union of data packets. The Zhang [26] team proposes a new verification method based on atomic predicates. First, the reachability of packets is verified, and then the reachability result of packets is used to verify the cyclic degree of freedom and the absence of black hole. In addition, they modeled the network as a directed graph, adopted the concept of atomic predicates. To improve scalability, they also proposed a parallel computing method Apache Spark to compute atomic predicates.

To solve the problem of a large number of equivalent classes that VeriFlow cannot solve, the Horn group proposes a new real-time data plane checker, Delta-net [13]. It does not construct multiple forwarding graphs to represent the packets in the network. Instead, the packets are incrementally transformed into a single edge-labelled graph that can represent all packet flows across the network. In addition, the first provable quasi-linear algorithm is proposed, which is influenced by Yang's atomic predicate verifier. The algorithm maintains the concept of atomic predicate incrementally. It analyzes all of the Boolean combinations of the IP prefix forwarding rules in the network with a set of disjoint packets.

Vermont [5], proposed by the Altukho team, is a set of tools for real-time verification. Vermont can be installed on the control plane. It intercepts the messages sent by the switch to the controller and the commands sent by the controller to the switch to observe the state changes of the network. It establishes an appropriate formal model of the entire network and checks each event, such as rule installation, deletion, or modification, against a set of formal requirements of the Packet Forwarding Policies (PFP). Before sending the network update command to the switch, Vermont predicts its execution and checks that the new network state meets the PFP. If this condition is met, the command will be passed to the corresponding switch. When a violation of the PFP is detected, VERMONT will prevent the change, warn the network administrator, and provide some additional information to locate the possible source of the error.

Due to the lack of detailed behavior of each hop, the previous work required a complicated fault location process. Zhao et al. proposed SERVE [27], a method that can automatically identify network problems in the data plane and periodically compare each rule and network behavior. SERVE provides all existing rules and generates a set of probes. After each probe is input, the actual network behavior of the output is compared with the expected network behavior of the control plane to verify the validity of each rule. They proposed a new detection generation method that can perform real-time verification in time when rules are added or deleted, which improves the verification efficiency. They considered the characteristics of pipeline processing and modeled the stateful multi-rooted tree (SMRT) in the network equipment of the data plane. The verification efficiency of SERVER is very high.

Summary of real-time verification: Formal real-time verification tools support real-time monitoring of the evolving SDN data plane of the network. They turn the packet reachability verification problem into a graph problem by modeling data packets or flow rules as graphs. This makes verification fast enough, but the existing tools each have different shortcomings, they cannot achieve all of the intended goals, such as good scalability, detection range, easy deployment, easy modeling, and detection speed. In addition, some hardware failures cannot be detected by these tools.

## 3 Testing

Data plane verification in the SDN network can check whether the devices of the data plane forward the packets according to the rules issued by the controller. However, switch failure may still exist, so it is necessary to test the data plane in the SDN network. Testing checks the forwarding behavior of each switch by generating probe data packets, thereby discovering failures in the data plane, as shown in Fig. 2. We divide this testing works into white-box testing and black-box testing according to the need for source code.

### 3.1 White-Box Testing

White-box testing needs to obtain source codes from the vendors to build formal models for checking the invariants of the data plane.

Zeng et al. propose ATPG [23], which is an automated and systematic method for testing data planes. ATPG reads the router configurations and builds a device-independent model which is used to generate a minimum set of test packets. Test packets are sent periodically to detect the failures. And then a mechanism will be triggered to locate the failures. Instead of matching rules, ATPG pays attention to whether the physical paths of the data plane are the same as the expected paths of the control plane policies. When the number of data packets exceeds a certain number, the detection speed will slow down.

Peter et al. propose Monocle [18], a system for monitoring the SDN data plane in real-time. Monocle is used as a proxy between the SDN controller and its corresponding switches, which allows Monocle to intercept all rules changes sent to any switch. Monocle maintain the desired global forwarding status on the network, and the expected content of the flow tables in each switch. After determining the expected state of the switches, Monocle can calculate the packet header space of running rules. Finally, Monocle injects these generated packets into the network as a proxy and sees how they are processed. But it requires a

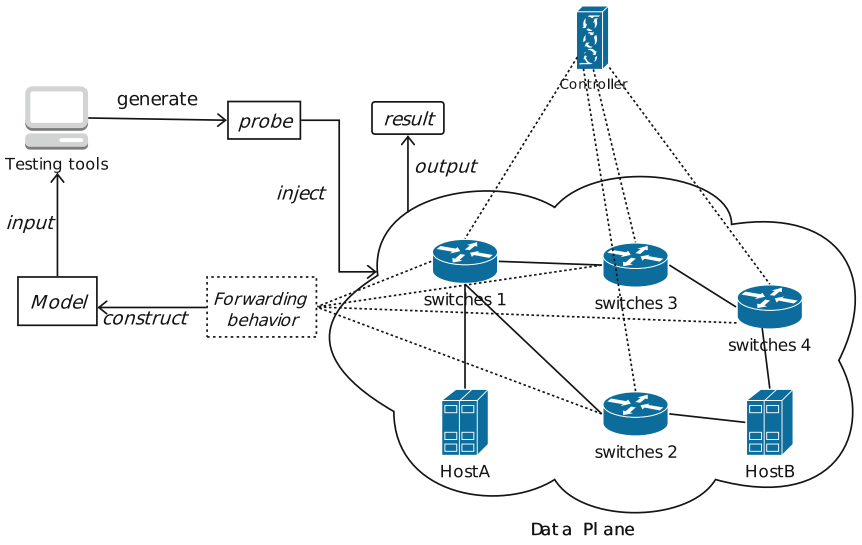

**Fig. 2.** Workflow of SDN data plane testing

Boolean satisfiability problem solution for each probe, resulting in slow probe generation. And when there are multiple missing rules associated with them, they may produce false negatives and do not support incremental probe updates.

Bu et al. propose RuleScope [8], a more comprehensive solution to check the transmission of SDN packets, mainly to detect rule missing fault and priority fault in switches. It checks the forwarding behavior with a probe, based on the established system. RuleScope accomplishes how to generate probe packets and how to handle the results of probes by introducing a monitoring application on the probe core. At the heart of the monitoring, the application is a set of algorithms they propose to detect and troubleshoot rules. However, RuleScope cannot handle rule updates, and its deployment scope is limited.

In response to previous deficiencies, Zhang et al. propose RuleChecker [24]. It is a fast SDN data plane testing tool. Unlike previous tools that solve the SAT problem by generating each probe for each data packet, RuleChecker takes the flow table as a whole and generates all probe data packets iteratively through a simple set operation. By encoding the set with a binary decision graph, RuleChecker is very fast. It is nearly 20 times faster than RuleScope and can update detections in 90.

White-box test summary: In SDN data plane formal testing, there are many methods based on white-box testing. In general, SDN data plane white-box testing can accurately detect and locate the errors, suitable for small and medium-sized networks. But it also has many shortages. On the one hand, not all source codes of data plane devices are available. On the other hand, for large networks, the model built from source codes has too many details and cost too much for testing.

### 3.2 Black-Box Testing

White box testing requires access to source code, but it is difficult for most users to obtain it, which gives white box testing certain limitations. Therefore, it is necessary to use black box testing without obtaining the source code during the testing.

Yao et al. [22] proposed a black-box testing method for the SDN data plane. They defined an extended finite state machine model for the pipeline to describe the OpenFlow switch. The data graph is extracted from the pipeline extended finite state machine model, and the data path that does not contain component details are searched on the data graph. This method can effectively reduce the cost. A leading sequence generation algorithm for I/O matching is proposed, which can process the components on each data path one by one. This method does not need to combine all the state machines in the model, thus effectively alleviating the space explosion.

Fayaz et al. proposed FlowTest [11], a test scheme for testing stateful and dynamic network strategies. They capture different DPFs by establishing abstract models, then model the network topology and forwarding strategies, and then use CBMC to generate counterexamples. Finally, these counterexamples are used as the input of the tracking generator to detect the fault and locate the location of the fault.

Fayaz et al. propose a stateful data plane testing framework BUZZ based on symbolic execution [12]. To establish an expressible and extensible data plane model, BUZZ introduces a novel network traffic abstraction called BUZZ Data Unit (BDU). It models network properties as a collection of finite state machines and triggers forwarding policies by generating test traffic. They also developed an optimized workflow based on symbolic execution to generate test traffic. To deal with the problem of state space explosion, they reduced the number and scope of symbolic variables. However, the process of BUZZ is too complicated, and it is easy to fail in the modeling process.

Summary of black-box testing: In this subsection, we introduce the SDN data plane black-box formal testing. We think that black-box testing is complementary to white-box testing. But due to the limitations of formal methods, and the strategic problems with their testing methods, they are not fast enough. In a word, the black-box testing of SDN data has made great progress, but the current testing method cannot deal with large-scale networks and hybrid networks well.

## 4 Future Research Prospects

At present, many works on formal testing and verification of SDN data planes has been recognized by the industry. But with the widespread deployment of SDN, the data plane will become more complex, so its accuracy and reliability will encounter more challenges. The current work urgently needs to be optimized, and their cost and time also need to be reduced. We divide the future development direction into the following three points. In the future, we will also work to combine AI and formal methods and apply them to the formal verification and testing of the SDN data plane. We learned that AI can effectively balance the load and improve quality of service (QoS) of SDN [7], and we will use it in combination with the related method proposed by some scholars. [4,6], thereby helping us to obtain more traffic.

(1) **AI-based model construct:** Model is the basis of formal verification and testing. Building a model requires specific knowledge. It is so difficult, that only computer experts can implement. This requirement hinders the promotion of formal methods in the industry. Some researchers are currently trying to use AI to assist model construction and have achieved preliminary results. They construct a concrete model based on the abstract model input by the user based on the AI method. We plan to apply AI-based model construct to the real-world SDN data plane verification and testing works. By comparing with existing expert modeling methods, this type of method is expected to gain advantages in ease of use and efficiency.
(2) **AI-based property definition:** In the formal method of verification and testing, researchers define attributes according to the general rules of the network. The attributes are further used as the basis for the correctness of verification and testing. So, defining attributes is very important. But defining attributes requires a lot of professional knowledge. This can only be done by highly professional people. We plan to apply AI-based property definition for formal verification and testing. By giving AI a small number of predefined attributes and network traffic, we let it learn independently and generate a large number of attributes that can be used for verification. Compared with the existing methods that completely define attributes by humans, this type of method is expected to gain advantages in accuracy and efficiency.
(3) **Scalable support for stateful data plane:** As SDN may enable richer network data processing services, the SDN network environment will be more complex and changeable. The correctness and reliability of the stateful data plane need more attention. As the scale of the data plane grows, the state space will become larger and larger, which brings challenges to the scalability of the model. For complex stateful data planes, a new formal model needs to be proposed to accurately describe the data plane while controlling the model scale.

## 5 Conclusion

Since 2010 when Flowchecker applied model checking to SDN data plane verification, more and more work has been done on SDN data plane formal verification and testing. In this paper, the existing works are divided into data plane verification and data plane testing. We have the following conclusions: data plane verification can find the defects of the data plane design in a short time and evaluate whether the design supports ideal invariants; data plane testing can check network behavior by generating probes to find implementation errors in the network. The current short-coming is that the manual construction of models and definitions of attributes re-quire professional knowledge, which hinders the promotion of formal verification and testing. And it needs to be improved in terms of verifying the stateful data plane. In the future, we will apply AI to assist modeling and attribute definition work, and make efforts to explore feasible verification and testing methods for stateful data planes.

**Acknowledgement.** This work was supported by the Hainan Provincial Natural Science Foundation of China (620RC562, 2019RC096, 620RC560), the Scientific Research Setup Fund of Hainan University (KYQD(ZR)1877), the Program of Hainan Association for Science and Technology Plans to Youth R&D Innovation (QCXM201910), and the National Natural Science Foundation of China (61802092, 62162021).

## References

1. Ahmad, S., Jamil, F., Ali, A., Khan, E., Ibrahim, M., Whangbo, T.K.: Effectively handling network congestion and load balancing in software-defined networking. CMC-Comput. Mater. Continua **70**(1), 1363–1379 (2022)
2. Al-Shaer, E., Al-Haj, S.: Flowchecker: Configuration analysis and verification of federated openflow infrastructures. In: Proceedings of the 3rd ACM workshop on Assurable and usable security configuration. pp. 37–44 (2010)
3. Alhaidari, F., et al.: Intelligent software-defined network for cognitive routing optimization using deep extreme learning machine approach (2021)
4. Ali, J., Roh, B.h.: Quality of service improvement with optimal software-defined networking controller and control plane clustering. CMC-Comput. Mater. Continua **67**(1), 849–875 (2021)
5. Altukhov, V., Podymov, V., Zakharov, V., Chemeritskiy, E.: Vermont-a toolset for checking SDN packet forwarding policies on-line. In: 2014 International Science and Technology Conference (Modern Networking Technologies) (MoNeTeC), pp. 1–6. IEEE (2014)
6. Babbar, H., Rani, S., Masud, M., Verma, S., Anand, D., Jhanjhi, N.: Load balancing algorithm for migrating switches in software-defined vehicular networks. Comput. Mater. Continue **67**(1), 1301–1316 (2021)
7. Belgaum, M.R., Ali, F., Alansari, Z., Musa, S., Alam, M.M., Mazliham, M.: Artificial intelligence based reliable load balancing framework in software-defined networks. CMC-Comput. Mater. Continua **70**(1), 251–266 (2022)
8. Bu, K., Wen, X., Yang, B., Chen, Y., Li, L.E., Chen, X.: Is every flow on the right track?: Inspect SDN forwarding with rulescope. In: IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, pp. 1–9. IEEE (2016)
9. Fang, Y., Lu, Y.: Checking intra-switch conflicts of rules during preprocessing of network verification in SDN. IEEE Commun. Lett. **23**(9), 1547–1550 (2019)
10. Fang, Y., Lu, Y.: Real-time verification of network properties based on header space. IEEE Access **8**, 36789–36806 (2020)
11. Fayaz, S.K., Sekar, V.: Testing stateful and dynamic data planes with flowtest. In: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, pp. 79–84 (2014)

12. Fayaz, S.K., Yu, T., Tobioka, Y., Chaki, S., Sekar, V.: {BUZZ}: Testing context-dependent policies in stateful networks. In: 13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 2016), pp. 275–289 (2016)
13. Horn, A., Kheradmand, A., Prasad, M.: Delta-net: real-time network verification using atoms. In: 14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17), pp. 735–749 (2017)
14. Kazemian, P., Chang, M., Zeng, H., Varghese, G., McKeown, N., Whyte, S.: Real time network policy checking using header space analysis. In: 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 2013), pp. 99–111 (2013)
15. Kazemian, P., Varghese, G., McKeown, N.: Header space analysis: static checking for networks. In: 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12), pp. 113–126 (2012)
16. Khurshid, A., Zou, X., Zhou, W., Caesar, M., Godfrey, P.B.: VeriFlow: verifying network-wide invariants in real time. In: 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 2013), pp. 15–27 (2013)
17. Mai, H., Khurshid, A., Agarwal, R., Caesar, M., Godfrey, P.B., King, S.T.: Debugging the data plane with anteater. ACM SIGCOMM Comput. Commun. Rev. **41**(4), 290–301 (2011)
18. Perešíni, P., Kuzniar, M., Kostić, D.: Rule-level data plane monitoring with monocle. ACM SIGCOMM Comput. Commun. Rev. **45**(4), 595–596 (2015)
19. Ruchansky, N., Proserpio, D.: A (not) nice way to verify the openflow switch specification: formal modelling of the openflow switch using alloy. In: Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM, pp. 527–528 (2013)
20. Son, S., Shin, S., Yegneswaran, V., Porras, P., Gu, G.: Model checking invariant security properties in openflow. In: 2013 IEEE International Conference on Communications (ICC), pp. 1974–1979. IEEE (2013)
21. Yang, H., Lam, S.S.: Scalable verification of networks with packet transformers using atomic predicates. IEEE/ACM Trans. Network. **25**(5), 2900–2915 (2017)
22. Yao, J., Wang, Z., Yin, X., Shiyz, X., Wu, J.: Formal modeling and systematic black-box testing of SDN data plane. In: 2014 IEEE 22nd International Conference on Network Protocols, pp. 179–190. IEEE (2014)
23. Zeng, H., Kazemian, P., Varghese, G., McKeown, N.: Automatic test packet generation. In: Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies, pp. 241–252 (2012)
24. Zhang, P., Zhang, C., Hu, C.: Fast data plane testing for software-defined networks with rulechecker. IEEE/ACM Trans. Network. **27**(1), 173–186 (2018)
25. Zhang, S., Malik, S.: SAT based verification of network data planes. In: Van Hung, D., Ogawa, M. (eds.) ATVA 2013. LNCS, vol. 8172, pp. 496–505. Springer, Cham (2013). https://doi.org/10.1007/978-3-319-02444-8_43
26. Zhang, Y., Li, J., Kimura, S., Zhao, W., Das, S.K.: Atomic predicates-based data plane properties verification in software defined networking using spark. IEEE J. Sel. Areas Commun. **38**(7), 1308–1321 (2020)
27. Zhao, Y., Zhang, P., Wang, Y., Jin, Y.: Troubleshooting data plane with rule verification in software-defined networks. IEEE Trans. Netw. Serv. Manag. **15**(1), 232–244 (2017)