International Journal of Information Security (2025) 24:181
https://doi.org/10.1007/s10207-025-01035-x

REGULAR CONTRIBUTION



A formal technique for automatic resolution of OpenFlow anomalies

Ramtin Aryan[1,4] · Anis Yazidi[2] · Adel Bouhoula[3] · Paal E. Engelstad[1]

Published online: 21 July 2025
© The Author(s) 2025

Abstract

Conflicts between network policies are frequent in today's computer networks due to the increasing complexity of network configuration. Resolving policy anomalies usually requires network administrator intervention, which is a time-intensive and error-prone process. This paper presents inference systems for the automatic resolution of OpenFlow anomalies. The approach uses high-level policies to detect conflict correction without policy violations. Our approach is fully automated and does not require interaction with the network administrator. Although there is a multitude of research papers on detecting anomalies in SDN, research to correct those anomalies in an automatic manner is very scarce, if not non-existent. We have formally proven the soundness and completeness of our inference systems. Furthermore, we provide experimental results based on real-life network configurations involving more than 1200 rules. The detection and correction processes exhibit very low computation overhead, in the order of milliseconds, when parallelization is used.

Keywords OpenFLow anomaly · Auto-correct · Resolving anomaly · Conflicts · Violation · SDN

1 Introduction

The demands for the Internet of Things (IoT) in the industry and human daily life have dramatically increased. It is expected that the IoT share of hypercritical and critical applications will increase steadily each year. As a result, network administrators face an increasing need to ensure network reliability. Software Defined Networking (SDN) offers a paradigm shift in managing complex networks in a logical centralized manner, moving away from traditional distributed management approaches. However, since SDN networks have to connect multiple devices with a variety of configurations, network management still remains a complex task. Furthermore, SDN networks are highly dynamic networks where rules can get updated dynamically, making the task of maintaining an anomaly free network state quite a challenge [2]. SDN anomalies were shown in the literature to be responsible for various network policy violations such data breaches, segmentation violations, and complex loops. It is neither practical nor efficient to delegate the correction task to network administrators in SDN networks since human-based correction is a time-intensive process that is prone to error. Furthermore, human correction might be infeasible as the dynamicity of the network might vary in the order of seconds. Therefore, there is a need for new automatic methods to correct SDN anomalies, which is the objective of this paper.

There are numerous research papers on rule anomaly detection in SDN networks exist. Some, such as [2], try to expand legacy firewall anomaly classification taxonomies found in the literature and adopt them in the context of SDN networks [3, 4]. Broadly speaking, there are two major methods for detecting anomalies in SDN: snapshot based methods

A preliminary version of this paper was presented at LCN 20, the 45th International Conference on Local Computer Networks, in October 2020 [1].

✉ Ramtin Aryan
ramtina@ifi.uio.no

Anis Yazidi
anis.yazidi@ntnu.no

Adel Bouhoula
a.bouhoula@agu.edu.bh

Paal E. Engelstad
paal.engelstad@its.uio.no

1 Department of Technology Systems, University of Oslo, Oslo, Norway

2 Department of Computer Science, Norwegian University of Science and Technology (NTNU), Trondheim, Norway

3 Information Technology Department, Arabian Gulf University, Manama, Kingdom of Bahrain

4 Department of Computer Science, OsloMet - Oslo Metropolitan University, Oslo, Norway

[2] and probing based methods [5]. In probing based methods [5–8], probing packets are used to examine the packet behavior, while offline approaches take snapshots from the flow tables in order to analyze dependencies between rules. However, these various detection methods do not offer any solutions to correct the detected anomalies, as this task is implicitly left to the human administrator.

There are a handful number of research papers that also try to automatically correct the anomalies, rether than just detecting them. Yazidi and Bouhoula [9] suggested a semi-automated method to resolve part of the anomalies occurring in firewall rules, where for some anomalies automatic correction actions are performed without human interaction, while for others a feedback from the network administrator is needed.

In addition, Zhou et al. [10] proposed an auto-correct tool, which is called NEAt, to resolve policy conflicts before they occur as a side effect of a newly installed rule. NEAt uses a formalism to express high-level policies. Then NEAt tries to map and compare the network rule and configuration with the high-level policy model. As a correction action, NEAt tries to remove the policies that lead to the anomalies or loops. However, the proposed method for modeling the policies and conflict resolution is complex and difficult for the network administrators to understand.

Although there is a multitude of research papers on detecting anomalies in SDN, research to correct those anomalies in an automatic manner is very scarce, if not absent. The task of correction is usually left to the human administrator, who should decide on the high-level policy to which different SDN applications need to adhere. However, controversially, due to the complexity of modern network architecture, humans can introduce new errors while trying to correct the detected ones. Furthermore, due to the stringent real-time requirements in the network, it is not practical to depend on human intervention to correct anomalies as it might increase the downtime or even lead to policy breaches due to delayed human troubleshooting.

This paper aims to propose an automatic correction method to detect and resolve the conflicts between policies. To this end, we try to categorize the anomalies that cause segmentation violations or loops in the data layer. Then via formal methods, we provide an inference system to resolve each category. Resolving the anomalies may cause some side effects and policy violations. To avoid this problem and the interaction with the network administrator, we propose automatic resolution actions, which are validated via the high-level policies. High-level policies describe the main constraints, which are defined in the application layer, and the rules in the data layer should follow them. In this paper we assume that the high-level policies are conflict-free. The conflict-freedom property can be guaranteed by methods such as PGA [11]. Receiving one switch as a faulty switch at a time is considered a second assumption. A defective switch refers to a switch that the troubleshooting regarding the high-level policies shows does not send a specific type of packet to the expected switch. Using inference systems makes our method simple and easy for network administrators to comprehend. Furthermore, we can easily prove its soundness and completeness. According to the provided inference systems, an detection and correction engine is implemented that is called *Net Auto-Solver*.

The Inference systems and automatic resolving is described in a four-page paper [1] describing the general idea without formal proof and comprehensive experiments. However, the presented draft is an extended version of the preliminary paper, which contains a whole idea with several definitions, formal proof, examples, and comprehensive evaluations.

Although the second assumption posits only one faulty switch as an input to the inference system, the combination of the SDNSpotlight [5] framework and the Net Auto-Solver can detect and correct multiple faulty switches simultaneously using a parallel approach. SDNSpotlight monitors the flows-disjoint sets of packet headers representing possible traffic that passes through the network from the ingress switches-to detect faulty switches. This method enables the detection of potential failures in multiple disjoint flows in the network concurrently. After correcting the faulty rule, the SDNSpotlight framework can continue monitoring and detecting potential faulty rules in the updated flow.

The primary contributions of the paper are as follows.

- In this paper, we present a new approach to the automatic correction of anomalies based on an inference system, operating with both the forwarding SDN rules and high-level policy constraints.
- By using inference systems, our method is simple and easy for network administrators to comprehend. Furthermore, we can also prove its correctness.
- The correctness of the inference systems is provable.
- We define some new automatic corrective actions: adding the intersection rule, swapping rules, and altering the route.
- We have categorized the root causes of policy violations into three types which include: overlap anomalies, misordered anomalies, and contradictions anomalies.
- Compared to the previous research work on detection, Net Auto-Solver corrects the root causes of loops and forwarding failures with respect to high-level constraints.
- The inference systems are capable of parallelization, and have a fast response time to the frequent SDN policy updates.

The remainder of the paper is organized as follows. In Sect. 2, we provide an overview of the state-of-the-art on correcting the anomalies in SDN networks. Sections 3 and 4

explain the basic notations and policy models that are used in Sect. 5 to describe the anomalies and the correction methods in Sect. 6. Finally, in Sect. 7 we provide an implementation of our approach that demonstrates its applicability and tests its performance.

2 Related work

Anomaly detection and correction are of utmost importance for the network administrators. There is not much research that has focused on this problem in the context of SDN. In this section, we have reviewed the main research in this field.

Zhou et al. [10] present an auto-correction method that is called NEAt. NEAt tries to check the new or updated rule and detects possible conflicts with the high-level policy. Then, in relation to the configured mode, it suggests a solution or resolves the conflict one-the-fly and installs the corrected rule on the switch. The proposed method uses the optimization algorithms on the policy and network models in order to get a real-time response to the rules updates. The optimization is applied to two levels. In the First level, NEAt uses ECs based on Veriflow [8] to model the packet forwarding behavior. Then the correction is applied only to the affected ECs. Zhou et al. use the undirected graph to model the policies and network. Threfore, in the second level, they apply an optimization method not only to decrease the number of modified edges but also minimize the size of the graph.

Yazidi and Bouhoula [9] present a method to correct the anomalies within firewall rules. They use the anomaly classification defined by [12]. Yazidi and Bouhoula show that for the correlation anomaly, simple reordering or adding a new rule does not work properly for all types of anomalies. Therefore, they described a semi-automatic correction method to correct the anomalies found in the rules of a firewall. To solve the correlation anomalies, Yazidi and Bouhoula calculate the overlap part of the rules and then try to add it as a new rule with a higher priority.

Asif et al. [13] preset a similar approach, called Auto-resolving, to detect the overlap between rules in flow tables and solve them via merging. Auto-resolving uses set theory and 3D structure to represent the rules and to detect the overlap. However, it does not detect all types of conflicts. Moreover, the proposed method does not validate the merging process to prevent the side effect on the network's general policies.

Jinjing [14] is an automatic configuration validator that is presented to aid Alibaba's network maintenance. Tian et al. receive the network operator's intent via LAI language and revise the BGP rules based on it. Although this novel method works very fast and has an effective response time, it is not compatible with the complexity of the high-level policies and the root causes of conflicts in the SDN context.

Kheradmand [15] presents a method that is called Anime. Anime suggests an alternative route for bridging the forwarding gap. It introduces objective measures to apply the data mining method to forwarding policies. The method tries to deduct a new high-level policy to solve the forwarding gap. Moreover, the tool is useful for continuous verification and anomaly detection. It uses several features such as packet header, device and the link-state, and the nodes along each path as input. As an output, Anime calculates and predicts all possible network behaviors, whether expected or not. The predicted path is used for bridging the potential forwarding gaps.

Fan et al. [16] present an algorithm that is called NSOA. The algorithm aspires to detect an optimum alternative route after a failure occurs on a switch. NSOA tries to keep most of the network configuration unchanged and updates the affected switches. The newly calculated route and switch modifications are calculated under the constraint of QoS. This means that the algorithm tries to detect the alternative route and selects the new routes based on the minimum costs. The cost could be defined based on various parameters such as link latency and packet loss.

GeckoNet [17] leverages Software-Defined Networking (SDN) to simplify network management, addressing issues like complex configurations and slow change propagation. By separating the control and data planes, it allows a central controller to manage network logic while switches forward traffic. Băluţă et al. integrate virtualization, monitoring, and a command-line interface to enhance configuration and issue resolution. Using Mininet, it creates a virtual network managed by the Faucet Controller and collects operational stats. Prometheus [18] and Grafana [19] provide real-time monitoring and alerting. The framework's modular design allows customization, and its self-healing capabilities, proven through testing, improve network resilience and efficiency.

Traditional network verification tools identify configuration errors but struggle to pinpoint exact issues. They often separate diagnosis from repair, leading to isolated solutions. Yang et al. [20] propose a novel method that treats distributed routing configurations as sequential programs using causal relationships. This approach links error localization with repair, transforming distributed systems into sequential programs to guide configuration repairs effectively. This method overcomes limitations of existing tools by providing a unified solution for diagnosing and fixing configuration errors.

Yao et al. [21] introduces FastCUV, a tool designed to efficiently verify configuration updates in distributed control planes of medium to large networks. Current network verifiers struggle with the efficiency of large-scale network configuration changes. FastCUV enhances the efficiency of distributed control plane verification while ensuring cor-

rectness by determining the network range affected by a configuration change. The paper presents a flow model and graph structure to facilitate the design of verification algorithms and accelerate the verification process. By comparing the Forwarding Information Base (FIB) before and after a change, FastCUV verifies the affected network area and supports rich network attributes, demonstrating high efficiency and correctness. Experimental results indicate that FastCUV outperforms existing methods. Additionally, the paper reviews current control plane verification methods, including simulation-based, model-checking-based, graph-based, and SMT-based approaches, and highlights the challenges of accuracy and scalability in large-scale networks.

The automated firewall configuration presented by Bringhenti et al. [22] focuses on the allocation and configuration of distributed firewalls in a service graph (SG). Existing solutions often lack formal correctness guarantees, full automation, or optimization, and many do not support modern software networking. Some introduce formal verification but miss optimization or the ability to configure generic firewalls from scratch. This methodology is the first to perform both the allocation and configuration of packet filter firewalls in a user-provided virtualized SG, achieving both optimization and formal correctness assurance. By jointly solving the firewall allocation and configuration problems in an optimized and formal manner, this method builds on a preliminary conference article. It automatically allocates packet filters in an SG and generates firewall rules to meet specified security requirements, ensuring the final solution satisfies these requirements through a formal model.

Shahriyar et al. [23] present DEFT, a solution for managing state in network function virtualization that uniquely integrates both elastic scaling and network function (NF) failure considerations, unlike most studies that focus on one aspect. It introduces a distributed state management system that ensures strong consistency in global state updates without relying on a central controller or remote storage. DEFT's design includes mechanisms for loss-free and order-preserving state migration and failure recovery, ensuring that per-flow states are updated in the order packets are received, regardless of flow migration or NF failure. The system employs a distributed consensus mechanism to maintain strongly consistent global states across NFs. DEFT addresses state management comprehensively, eliminating central dependency while supporting both elastic scaling and fault tolerance.

Sekar and Vasudevan's work [24] explores the management of link fault tolerance within software-defined networking (SDN), which involves various technologies for designing, building, and managing networks. SDN's abstraction of lower-level functionalities allows network administrators to effectively manage network services. The authors address the challenge of link fault tolerance, traditionally managed using either protection or restoration schemes, by proposing a hybrid scheme that combines both approaches. They implement per-link bidirectional forwarding detection sessions to manage each link and prevent unnecessary control transfers from the switch to the controller, thereby reducing response times for link resiliency. Their approach ensures effective handling of link failures within SDN. Comparative analysis with traditional scheduling and link fast failover methods shows that their hybrid scheme offers superior response times for both scheduling and recovery, highlighting its efficacy in managing link fault tolerance in SDN environments.

Menaceur et al. [25] conduct a comprehensive investigation into fault tolerance strategies within software-defined networking (SDN) environments. Their analysis reviews existing research to foster the development of improved methods for achieving greater reliability and resiliency in future SDN networks. The review highlights current state-of-the-art services and solutions for managing faults in SDN communications and aims to provide advanced fault tolerance frameworks incorporating modern networking techniques and technologies. The authors' key contributions include a historical overview of fault tolerance practices transitioning from traditional networks to SDN and a taxonomy categorizing control and data plane fault tolerance strategies within SDN. They review research efforts addressing fault tolerance in both the data and control planes of SDN, analyzing the implications of these approaches in existing deployments. The paper summarizes contemporary solutions for enhancing fault tolerance in SDN environments, including machine learning techniques, and identifies open research challenges and future directions in SDN-enabled fault tolerance.

Shi et al. [26] introduce NetRep, an automated tool for debugging and fixing network programs. NetRep takes a faulty program and unit tests, producing a corrected version that passes all tests using symbolic reasoning for fault localization and patch synthesis. Key features include domain-specific abstractions for efficient analysis, modular analysis that targets one function at a time and reuses function summaries, fault localization with symbolic execution and constraint solving for imperative object-oriented programs, and patch synthesis using a context-free grammar to find patches. NetRep's effectiveness was validated with benchmarks from real-world network programs, proving its capability in fault localization and patch generation.

Saâdaoui et al. [27] presented a firewall anomaly resolution tool that is called FARE. The proposed method tries to detect and correct the anomalies in simple firewalls. Moreover, a new classification of anomalies in multi-firewalls is also presented. According to this classification, a novel correction method is defined to resolve anomalies in distributed firewalls. The authors describe their method by means of a formal expression that is called an inference system. They

also approve of the correctness and completeness of the correction method.

Hussain and Shah [28] present an approach called Efficient Policy Enforcement (EPE), monitoring the policy changes at the controller level. EPE keeps the newly installed rules and, by examining the new rules and already installed rules, removes the affected rules from the switches. According to the experiments, the presented method has an efficient performance and causes low latency on the network. However, the accuracy of the method and possible side effects of purging the rules are not discussed clearly in the paper. However, a significant amount of research has focused on anomaly detection in the context of SDN. Some of the most relevant research is reviewed as follows.

Aryan et al. [2] presented an offline detection method. The method tries to detect anomalies based on the proposed comprehensive classification. The authors, just as in [6], apply the detection algorithm to the snapshot of the network layer policies. However, this approach has a slow performance that is not suitable for frequent rule updates. Therefore, an incremental method [29] is proposed to increase detection capability in the dynamic context.

Since the offline approaches are slow and are not capable of detecting physical failures, [5, 7, 8, 30] use a real-time approach to identify the forwarding failure, physical failure, and loops via the injection of the probing packet. Moreover, Ibrar et al. [31] believe that browsing rule tables and evaluating relations between the rules have a delay. Therefore, they present a proactive method based on machine learning, called PrePass-Flow, to predict faulty rules and find a new position for the faulty rules to solve the problem.

Heng et al. [32] explore active watermarking techniques to trace flows in anonymous networks used for illegal activities. Unlike passive methods that require extensive data recording, active methods embed watermarks but are usually protocol-specific. They introduce the Software Defined Networking (SDN)-based Meter Table Watermarking Traceback scheme (SMWT), which adjusts flow rates and encodes watermarks using Bose-Chaudhuri-Hocquenghem (BCH) codes. Experiments in a real Tor network showed a high detection rate of 96.7% and a low false-positive rate of 0.3%. SMWT is noted for its flexibility, stealthiness, and ease of deployment, leveraging OpenFlow meter entries to adjust flow rates without limiting communication protocols. Its deployment does not require modifications to existing network infrastructure, indicating low complexity. The use of BCH codes enhances error correction, making SMWT adaptable to various network environments.

Binghao et al. [33] introduce an innovative flowlet-level multipath routing scheme utilizing a graph neural network (GNN) within OpenFlow-based SDN. This scheme addresses the shortcomings of conventional flow-based and packet-based routing methods, achieving a balance between load balancing and packet reordering. Traditional flow-based routing often results in uneven link loads due to differing flow sizes, while packet-based routing can cause packet reordering issues. To overcome these challenges, the proposed scheme employs flowlets as the core processing unit, fragmenting flows to enhance multipath transmission and prevent congestion. The approach also leverages OpenFlow's idle timeout mechanism to adaptively implement flow splitting. Moreover, GNN integration allows for path state prediction during multipath forwarding, reducing the need for frequent statistical operations. Simulations show that this scheme effectively improves time overhead, end-to-end delay, flow completion time, and throughput.

The attack detection methods, similar to real-time conflict detection methods, attempt to identify policy breaches through traffic analysis. This paper reviews these methods to highlight their effectiveness in detecting such breaches.

Attarian et al. [34] introduce the MixFlow model, a novel traffic correlation attack targeting short-flow applications tunneled through Mixnets like Loopix [35], which were thought to be resistant to such attacks. MixFlow employs a contrastive model to identify semantic relationships between entry and exit flows, despite Poisson mixing delays and cover traffic. Unlike existing flow correlation studies, MixFlow uses meta-information from unrelated flows to create more effective differential representations for attacks. The model incorporates two contrasting approaches, ConFEm and ConCEn, to extract semantic network traffic features from different perspectives, enhancing adaptability to varying anonymity levels and traffic analysis scenarios. Using a Poisson Hidden Markov Model (PHMM) as its feature embedding network, MixFlow detects complex semantic patterns that correlate inflow and outflow.

FlowTracker, an advanced flow correlation attack model presented by Zhong et al. utilizes denoising and contrastive learning techniques. FlowTracker overcomes limitations of existing methods, such as inadequate traffic shape features and network noise interference. The model includes four key modules: flow cumulative representation, noise-resistant mapping, distance optimization, and hierarchical judgment. Using a hierarchical correlation decision algorithm, FlowTracker accurately links users with their visited Internet services. Experimental results show a high attack success rate, even with significant traffic obfuscation. The model uses raw and normalized cumulative sequences to represent traffic shape and applies contrastive learning to distinguish between unrelated flows.

In contrast to the proposed method, most existing research primarily addresses firewall anomalies in single or multiple firewall setups. Detection methods within the context of Software-Defined Networking (SDN) generally lack a correction mechanism. Furthermore, the studies that do propose correction methods predominantly rely on machine learning

techniques, which inherently include an error margin and cannot formally prove correctness. NEAt also concentrates on new rules. This paper, however, introduces a method to detect and correct most anomalies that lead to loops, forwarding failures, and network slicing violations.

3 Primary notation

In this section, we define some necessary notations that are used by all the formal models described in this paper. As presented in Equation 1, $rule_{i,s}$ defines a rule that is installed on the switch s via priority i. It consists of two parts, $Condition$ and $Action$. $Condition$ can have different fields, but in this paper, we assume that it includes Src_IP, Dst_IP, Dst_Port, and protocol. The $Action$ field defines what will happen to the packet if its header is matched with the $Condition$ field. The action field in OpenFlow rules can have different values such as forward, drop, go, set, queue, etc. The proposed method supports OpenFlow actions. However, for simplicity, in this paper we assume the action field has the value "forward" or "drop".

$$rule_{i,s} : Condition \Rightarrow Action \tag{1}$$

Moreover, the set of rules in switch sw are stored as a $Flowtable_{sw} = \{rule_{1,sw}, rule_{2,sw}, \ldots, rule_{n,sw}\}$.

The packet header plays a crucial role for the switches by assigning them with their corresponding rules. The packet header length can vary, but in this paper, we assumed the following format of the packet header:

$$Ph = Src_IP \wedge Dst_IP \wedge Dst_Port \wedge protocol$$

The packet header is defined as conjunct boolean expressions of source IP address, destination IP address, destination port, and protocol. The packet header items are checked with the rule's condition to detect the packet property and assign a relevant action. Packet headers (Ph) are stored in a list defined as: $Ph_List : \{Ph_1, Ph_2, \ldots, Ph_n\}$.

Moreover, Grid representation is used to model the high-level policy. Grid representation is suggested by [36] in order to visualize the relationship between rules and possible anomalies. It is a matrix-based model with a horizontal axis representing rules and a vertical axis showing disjoint header space. Since OpenFlow rules are assumed as an access control rule list, which contains condition fields and action, can be modeled by the Grid representation.

Disjoint header spaces (DHS) in a rule table consist of sets of packet headers (Ph), and there is no overlap between DHS ($\forall ph_i, ph_j \in DHS, i \neq j, ph_i \cap ph_j = \varnothing$). The algorithm for generating the DHS is presented in Algorithm 1. The subtraction operation among rules are defined based on the disjoint query which is defined in [29].

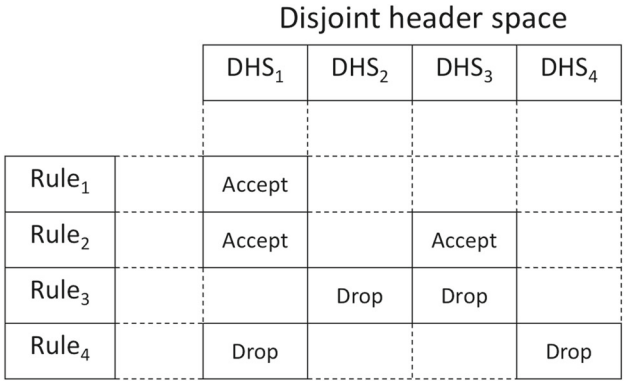


Fig. 1 Example of the grid representation

Algorithm 1: Generating disjoint header spaces (DHS) from a rule table.

```
Input: RuleList
Output: DHSList
foreach tartget_rule in RuleList do
    foreach rule in RuleList do
        if rule.Index < target_rule.Index then
            target_rule = target_rule − rule;
        end
    end
    DHSList ← target_rule
end
return DHSList;
```

In Algorithm 1, the DHS is generated based on the rules in the rule table ($RuleList$). For each rule ($target_rule$), the overlaps between the $target_rule$ and the previous rules are subtracted to generate a new disjoint header space (DHS).

The relationship between the rules in Table 1 is depicted in Fig. 1. For example, $Rule_2$ and $Rule_3$ have overlap in DHS_1, and since they have different actions they have an anomaly.

The definition of anomalies and anomaly solutions are discussed in Sects. 5 and 6, respectively.

Table 1 Example of a rule table

Rules	Src IP	Dst IP	Port	Action
$Rule_1$	*	192.168.*.*	80	Accept
$Rule_2$	*	192.168.*.*	80,53	Accept
$Rule_3$	*	192.168.*.*	25,53	Drop
$Rule_4$	*	192.168.*.*	80,443	Drop

Reach ()

	EC_1	EC_2	EC_3	EC_4
Sw_1	✓	✗	✓	✗
Sw_2	✗	✓	✓	✓
Sw_3	✓	✗	✗	✓
Sw_4	✗	✓	✗	✗

Fig. 2 Example of the grid representation for the $Reach()$ function

Forward ()

	EC_1	EC_2	EC_3	EC_4
Sw_1	✓	✗	✓	✗
Sw_2	✗	✓	✓	✓
Sw_3	✗	✗	✗	✓
Sw_4	✗	✓	✗	✗

Fig. 3 Example of the grid representation for the $Forward()$ function

4 Policy model

To validate any changes in network policies reflected by the data plane rules, we should check with the high-level policy. It is up to the network architect to specify the high-level policy. The high-level policy model describes the general overview of the reachability and forwarding criteria.

In this section, we present a formal model that is referred to as the *Policy Model*. This model describes the necessary restrictions defined by the network architect and that must be followed by all the rules in the data layer. The policy model contains a list of packet headers and their access status, i.e. reachability and forwarding status. To this end, the model uses two functions, $Reach()$ and $Forward()$, that are explained in detail in the following.

4.1 Reach() function

This function helps us to understand whether a specific set of packet headers can reach a particular node or not. $Reach(Ph_List, sw)$ function returns true for the Ph_List and Switch sw iff all of the packet headers (Ph) from the Ph_List are allowed to enter the switch sw.

The $Reach$ function for a specific DHS and switch can be represented using the grid representation model, as shown in Fig. 2.

4.2 Forward() function

This function helps us to understand whether a particular node has permission to forward a specific set of packet headers or not. $Forward(Ph_List, sw)$ function returns true for the Ph_List and switch sw iff all of the packet headers (Ph) from the Ph_List are allowed to leave switch sw.

The $Forward$ function for a specific DHS and switch can be presented using the grid representation model, as shown in Fig. 3.

5 Anomaly definition

Anomalies in this paper refer to conflicts that lead to the fact that a set of packets do not follow their expected route and end up at an unintended destination. These anomalies may cause loops or other types of anomalies, such as network slicing violations. In this section, we use the specific notations for the switches and rules that play an important role in the anomaly categorization and repairing process. Before continuing, some key notations must be defined.

5.1 Key notation

The switch in which the anomaly occurs is known as a *faulty switch* (Sw^f). The rule that is expected to catch the list of packets is called the *target rule* $(rule_t)$. The unwanted rule that changes the route of the packets and does not let them be matched with the target rule is refereed to as an *unwanted rule* $(rule_{unwt})$. Additionally, the node that the unwanted rule forwards the packets to is referred to as an *unwanted node* (Sw^{unwt}) and the expected node that the target rule forwards the packets to is known as an *expected node* (Sw^{expt}). There are several methods to detect faulty switches and faulty rules. This paper assumes that the faulty rules and switches are detected based on the SDNSpotlight [5] method. In this section, three types of anomalies are defined and a formal description is presented for each one.

5.2 Overlap anomaly

According to the high-level policy, the *overlap anomaly* occurs if a specific set of packet headers has permission to access both expected and unwanted nodes. Moreover, the faulty switch has permission to forward the specific packet header. Equation 2 presents the formal expression of the overlap anomaly condition. It shows that if a set of packets (Ph_List) does not follow the expected path, in which the Sw^f, Sw^{unwt} and Sw^{expt} are true, and the $Reach()$ function for Sw^{expt} and Sw^{unwt}, and the $Forward()$ for Sw^f are true, then the *overlap anomaly* will occur.

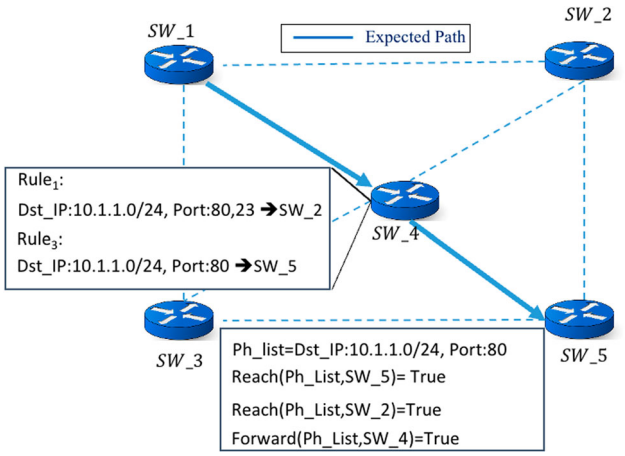


Fig. 4 Example of the overlap anomaly

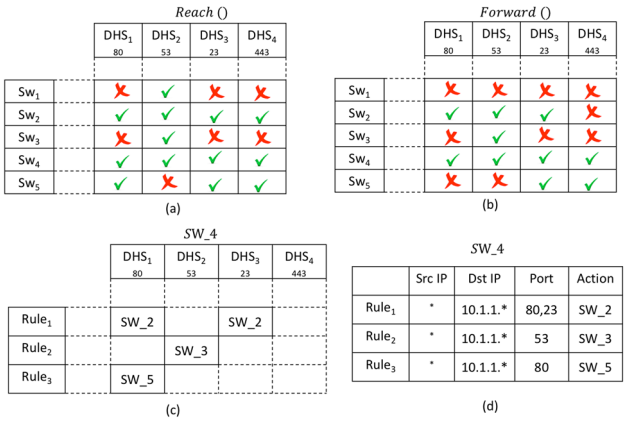


Fig. 5 The grid presentation of the overlap anomaly

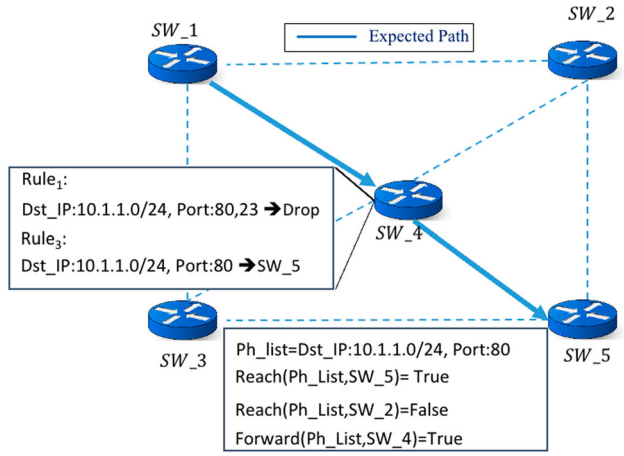


Fig. 6 Example of the misordered anomaly

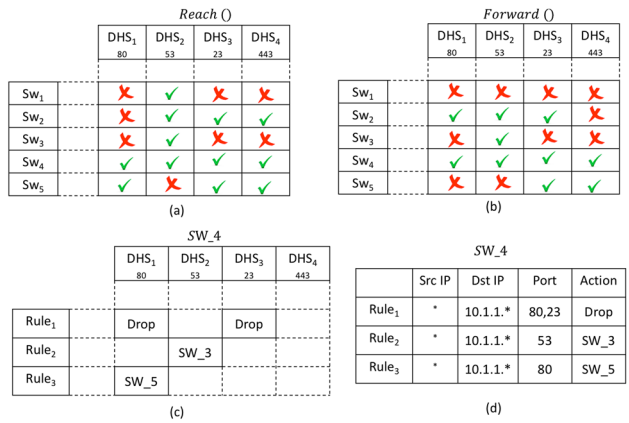


Fig. 7 The grid presentation of the misordered anomaly

$$overlapanomaly = Reach(Ph_List, Sw^{unwt}) \wedge Reach(Ph_List, Sw^{expt}) \wedge Forward(Ph_List, Sw^{f}) \quad (2)$$

According to the high-level policy in Fig. 5a, b, for packets with $Dst_IP : 10.1.1.0/24$, and $Port : 80(DHS_1)$ there is an overlap anomaly between $Rule_1$ and $Rule_3$ on switch SW_4, and the packets do not follow the expected path in Fig. 4.

5.3 Misordered anomaly

According to the high-level policy, the *misordered anomaly* occurs if a specific set of packet headers has the permission to access the expected node, while the faulty switch has permission to forward the specific packet header. Equation 3 presents the formal expression of the misordered anomaly condition. It shows that if a set of packets (Ph_List) does not follow the expected path, in which the Sw^{f}, Sw^{unwt} and Sw^{expt} are true, and the $Reach()$ function for S^{expt} is true and Sw^{unwt} is false, and the $Forward()$ for Sw^{f} is true, then the *misordered anomaly* occurs. We call this type of anomaly a misordered anomaly since there is no conflict with the high-level policy. However, the corresponding rules have the wrong priorities.

$$misorderedanomaly = \left(!Reach(Ph_List, Sw^{unwt})\right) \wedge Reach(Ph_List, Sw^{expt}) \wedge Forward(Ph_List, Sw^{f}) \quad (3)$$

As can be seen in the high-level policy in Fig. 7a, b, packets with the header space of $Dst_IP : 10.1.1.0/24$, and $Port : 80\ (DHS_1)$ cannot follow the expected path in Fig. 6, since there is a misordered anomaly between $Rule_1$ and $Rule_3$ on switch SW_4.

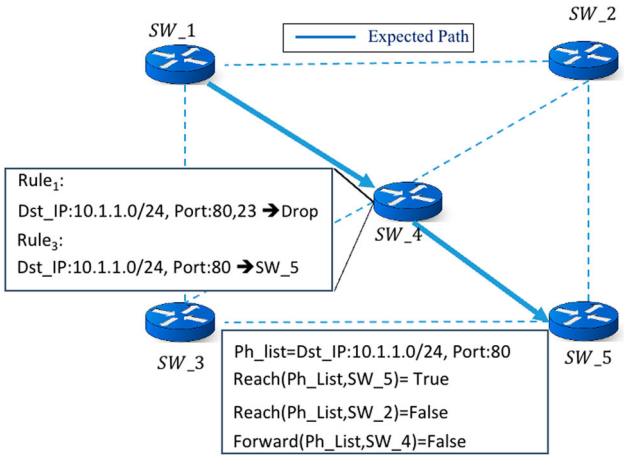


Fig. 8 Example of the contradiction anomaly

5.4 Contradiction anomaly

According to the high-level policy, the *contradiction anomaly* takes place if a specific set of packet headers has permission to access the expected node, while the faulty switch does not have permission to forward the specific packet header. Equation 4 presents the formal expression of the contradiction anomaly condition. It shows that if a set of packets (Ph_List) does not follow the expected path, which the Sw^f and Sw^{expt} are true, and the $Reach()$ function for Sw^{expt} is true and for Sw^{unwt} is false, and the $Forward()$ for Sw^f is false then the *contradiction anomaly* takes place.

$$contradictionanomaly = \left(!Reach(Ph_List, Sw^{unwt})\right) \wedge Reach(Ph_List, Sw^{expt}) \wedge \left(!Forward(Ph_List, Sw^f)\right) \quad (4)$$

For instance, an expected path is assumed, as shown in Fig. 8. According to the high-level policy in Fig. 9a, b, packets with a header space of Dst_IP : 10.1.1.0/24 and $Port$: 80 (DHS_1) cause a contradiction anomaly for $rule_1$ and $rule_3$. Since forwarding the header space from SW_4 is not allowed in the high-level policy, there is no solution to guarantee that the expected path contains the SW_4.

6 Repairing the anomalies

This section discusses four inference systems to repair the three types of anomalies. The correctness of each system is also proved.

6.1 Basic notation

This section defines the global values and functions that are used in inference systems.

Let RL be a list of pairs of switches and the rules that is belongs to them. Then we can define the list as : $RL = \left\{\bigcup^{i,sw}(r_i^{sw}, sw)\right\}$.

Let P be a list of high-level policies that is defined by the network architecture and is considered to be a global variable.

Moreover, the $modify()$ is a global function that assigns new values to the local variables. For example, $modify\ ((a, 1), (b, 2))$ assigns value 1 to a, and value 2 to b.

6.2 Adding a new rule

This solution is designed to repair a group of rules with overlap anomaly. To resolve this type of anomaly, a new rule ($r_k^{sw^f}$) is added to the flow table of the faulty switch (sw^f). This rule should have a priority higher than the unwanted rule ($r_j^{sw^f}$) and the target rule ($r_i^{sw^f}$). The condition field of the new rule contains the overlap of the condition of the unwanted rule and the condition of the target rule ($c_k^{sw^f} = (c_i^{sw^f} \cap c_j^{sw^f})$). However, the union between the action of the unwanted rule and the target rule creates the action field of the new rule ($a_k^{sw^f} = (a_i^{sw^f} \cup a_j^{sw^f})$). The Inference System 5 presents a formal expression of adding a new rule.

Inference System for Adding a New Rule:

$$\text{Init: } \frac{}{(Pr, R)} \quad \text{where } \begin{cases} \text{Pr} = \text{The list of all distinct pairs of rules in } R \\ \text{R} = \text{The set of rules of the switch } sw \end{cases}$$

$$\text{Add: } \frac{((r_i, r_j) :: Pr, \{r_i, r_j\} \cup R)}{(Pr', R \cup \{r_k, r_i', r_j'\})} \quad \text{if } \begin{cases} c_i \cap c_j \neq \emptyset \\ P \models \text{Reach}(c_i, sw^{exp}) \wedge \text{Reach}(c_j, sw^{unwt}) \wedge \text{Forward}(c_i, sw^f) \end{cases}$$

$$\text{where } \begin{cases} r_i = (i, c_i, a_i) \\ r_j = (j, c_j, a_j) \\ r_i' = (i, c_i / (c_i \cap c_j), a_i) \\ r_j' = (j, c_j / (c_i \cap c_j), a_j) \\ Pr' = \text{modify}\left(Pr, \left[(r_i, r_i'), (r_j, r_j')\right]\right) \\ r_k = (\min(i, j) - 1, c_i \cap c_j, a_i \cup a_j) \end{cases}$$

$$\text{Skip: } \frac{((r_i, r_j) :: Pr, R)}{(Pr, R)} \quad \text{if no other rules apply to the } (r_i, r_j)$$

$$\text{Success: } \frac{(\emptyset, R)}{R} \quad (5)$$

The inference system is initiated by the pairs of rules as candidates Pr and the set of rules R, which contains all of the rules in the network.

Then the condition of the *Add* inference rule is checked for each candidate c. The condition will be true if there is an overlap anomaly in the pair of candidate rules. Without loss of generality, we refer to a pair of candidate rules as a candidate. Then, the new rule will be created and added to the rule list, and the target rule r_i and the unwanted rule r_j will be modified.

If the candidate does not match with the inference rule Add, the condition of the *Skip* inference rule will be checked. The skip inference rule is applied to the candidate that has no anomalies. This inference rule will then remove the candidate

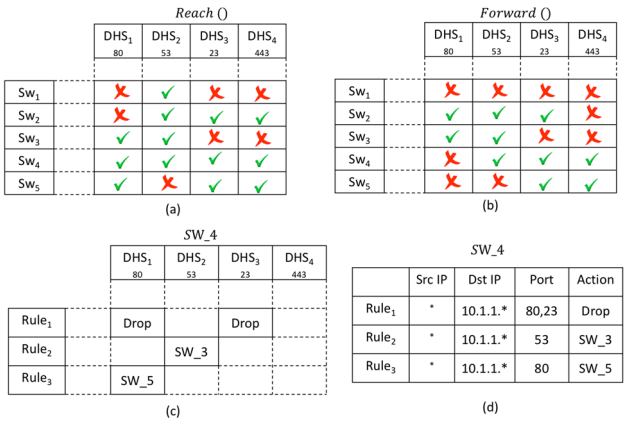


Fig. 9 The grid presentation of the contradiction anomaly

from the list. Thus, this step demonstrates that the length of the candidate list will decrease and the system will be finite. When the list of candidates is empty, the system will terminate successfully. At this juncture, we should provide a theorem that shows the correctness of our inference system when adding a new rule.

Theorem 1 *(Correctness) Let R be the set of rules of the switch sw, and let P the High-Level Policy Formula. If the following is true:*

$$(\emptyset, R) \vdash_I \cdots \vdash_I R^{'}$$

then $R \Leftrightarrow R^{'}$ with respect to P that means R and $R^{'}$ apply the same action to each unique packet. Moreover, there are no overlap *anomalies in $(sw, R^{'})$.*

Proof To prove the correctness of the Inference System 5, we must first prove that R and $R^{'}$ are equivalent. This is obvious since after the application of the inference rule *Add*, the two rule sub-lists (r_i, r_j) and $(r_k, r_i^{'}, r_j^{'})$ are equivalent.

Now it is easy to prove that there are no overlap anomalies in $(sw, R^{'})$. This is due to the fact that if the rule *Success* is applied, then, for all distinct pairs of rules (i, c_i, a_i) and (j, c_j, a_j) in $R^{'}$, we have either $c_i \cap c_j = \emptyset$ or $P \not\models Reach(c_i, sw^{exp}) \wedge Reach(c_j, sw^{unwt}) \wedge Forward(c_i, sw)$. Please note that sw^{exp} refers to the expected switch that a specific packet should reach it via the switch sw and rule $r_i^{sw} = (i, c_i^{sw}, a_i^{sw})$. sw^{unwt} refers to the unwanted switch that a specific packet should not reach it from switch sw and rule r_i^{sw}.

Theorem 2 *(Completeness) Let R be the set of rules of the switch sw, and let P the High-Level Policy Formula. Then there exists a finite derivation:*

$$(\emptyset, R) \vdash_I \cdots \vdash_I R^{'}$$

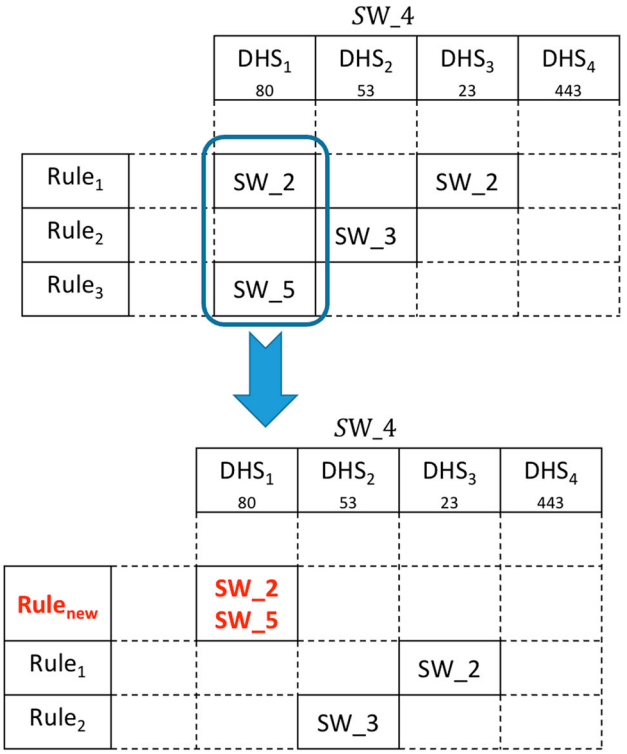


Fig. 10 Example of repairing the misordered anomaly

such that $R \Leftrightarrow R^{'}$ with respect to P, which means R and $R^{'}$ apply the same action to each unique packet. Moreover, there are no overlap anomalies in $(sw, R^{'})$.

Proof To prove the completeness of the Inference System 5, we must show that the inference system with input R, which has no overlap between the members, terminates successfully.

In this sense, we must show that the system always terminates successfully. Since the R is totally disjoint, all members of list Pr meet the rule skip condition. Therefore, the length of list Pr decreases in each interaction, and the system will terminate successfully.

As an example, in Fig. 10 and regarding the high-level policies(Fig. 5a, b), $Rule_{new}$ is defined to resolve the conflict between $Rule_1$ and $Rule_3$. This rule is installed above $Rule_1$ and guarantees the high-level policy not only for both packet header spaces but for the part that overlaps as well.

6.3 Swapping rules

This solution is useful for repairing groups of rules with a misordered anomaly. In this method, the priority of the unwanted rule (r_j) and the target rule (r_i) are swapped. This solution can guarantee the criteria of the high-level policy. The formal expression of the swapping solution is presented in the Inference System 6.

Inference System Swapping Rules:

$$\text{Init: } \frac{}{(Pr, R)} \quad \text{where } \begin{cases} \text{Pr} = \text{The list of all distinct pairs of rules in } R \\ \text{R} = \text{The set of rules of the switch } sw \end{cases}$$

$$\text{Add: } \frac{((r_i, r_j) :: Pr, \{r_i, r_j\} \cup R)}{\left(Pr', R \cup \{r_i', r_j'\}\right)} \quad \text{if } \begin{cases} c_i \cap c_j \neq \emptyset \\ P \models \text{Reach}(c_i, sw^{exp}) \wedge \neg\text{Reach}(c_j, sw^{exp}) \wedge \text{Forward}(c_i, sw^f) \end{cases}$$

$$\text{where } \begin{cases} r_i = (i, c_i, a_i) \\ r_j = (j, c_j, a_j) \\ r_i' = (j, c_i, a_i) \\ r_j' = (i, c_j, a_j) \\ Pr' = \text{modify}\left(Pr, \left[(r_i, r_i'), (r_j, r_j')\right]\right) \end{cases}$$

$$\text{Skip: } \frac{((r_i, r_j) :: Pr, R)}{(Pr, R)} \quad \text{if no other rules apply to the } (r_i, r_j)$$

$$\text{Success: } \frac{(\emptyset, R)}{R} \tag{6}$$

The inference system is initiated by the pairs of rules Pr as candidates, which are pairs of rules, and the sets of rules of the switches in the network. If the condition of the *Add* rule is correct for a candidate c, then it means there is a misordered anomaly in the pair of rules. Thus, in this step, the two rules are swapped in the rule table, and the candidate is removed from Pr. If there is no anomaly in the pair of rules, the *Skip* rule is applied. The skip rule removes the pair from the Pr, leaving the rule table unchanged. At the end, if the list is empty, it means there are no misordered anomalies among the rules, and the system terminates successfully via the *Success* rule. As explained, the length of the candidate list will decrease, and this means that the system is finite. At this juncture, we should provide a theorem that shows the correctness of our inference system when swapping rules.

Theorem 3 *(Correctness) Let R be the set of rules of the switch sw along the alternate route. If the following is true:*

$$(\emptyset, R) \vdash_I \cdots \vdash_I R'$$

then $R \Leftrightarrow R'$ with respect to P, which means R and R' apply the same action to each unique packet. Moreover, there are no misordered *anomalies in* (sw, R').

Proof To prove the correctness of the Inference System 6, we must first prove that R and R' are equivalent. This is obvious since after the application of the inference rule *Add*, the two rule sub-lists (r_i, r_j) and (r_i', r_j') are equivalent.

It is now obvious that there are no misordered anomalies in (sw, R'), since if the rule *Success* is applied, then for all distinct pairs of rules (i, c_i, a_i) and (j, c_j, a_j) in R', we have either $c_i \cap c_j = \emptyset$ or $P \not\models Reach(c_i, sw^{exp}) \wedge !Reach(c_j, sw^{exp}) \wedge Forward(c_i, sw)$. Where sw^{exp} refers to the expected switch that a specific packet should reach it via the switch sw and rule $r_i^{sw} = (i, c_i^{sw}, a_i^{sw})$.

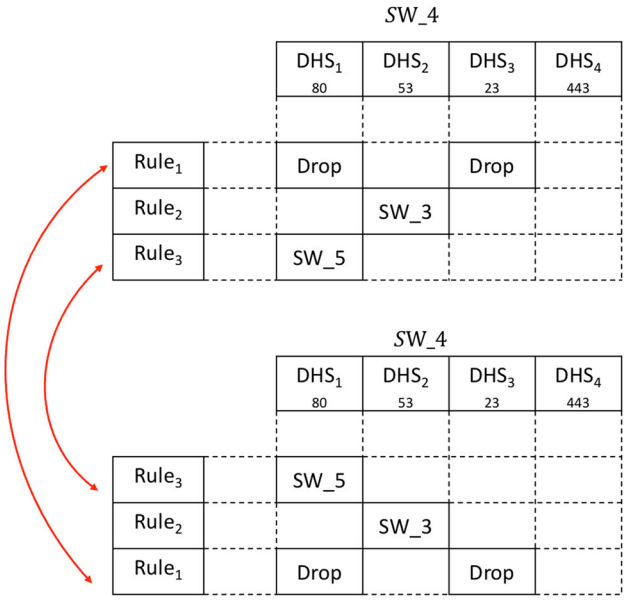


Fig. 11 Example of repairing the overlap anomaly.

Theorem 4 *(Completeness) Let R be the set of rules of the switch sw, and let P be the High-Level Policy Formula. Then there exists a finite derivation:*

$$(\emptyset, R) \vdash_I \cdots \vdash_I R'$$

such that $R \Leftrightarrow R'$ with respect to P, which means R and R' apply the same action to each unique packet. Moreover, there are no overlap anomalies in (sw, R').

Proof To prove the completeness of the Inference System 6, we must show that the inference system with input R, which has no overlap between the members, terminates successfully.

Since R is totally disjoint, all members of list Pr meet the rule skip condition. Therefore, the length of the list Pr decreases with each interaction, and the system will terminate successfully.

As an example, in Fig. 11 and regarding the high-level policies (Fig. 7a, b), $Rule_1$ and $Rule_3$ are swapped to solve the overlap anomaly. This solution guarantees the high-level policy, and not only for packet header spaces.

6.4 Altering route

The alternative route solution is used to find an alternative route that not only can repair the contradiction anomaly between the rules but also guarantee the high-level policy. This solution tries to find an alternative route that does not pass through the faulty switch(S_f) and reaches the next

expected switch. However, the algorithm is not guaranteed to find an alternative if the high-level policy prohibits this possibility.

According to the solution, finding the alternative route is the necessary part that is described as Inference System 7.

Inference System for Findding an Alternative Route:

$$
\begin{aligned}
&Init: \frac{}{(Ds, Sf, Sr, Se, \emptyset)} \quad where \begin{cases} Ds = The\ list\ of\ all\ distinct\ switches \\ Sf = The\ faulty\ switch. \\ Sr = The\ source\ switch\ that\ is\ before\ the\ faulty\ switch. \\ Se = The\ expected\ switch\ to\ be\ reached\ after\ the\ faulty\ switch. \end{cases} \\
&Add: \frac{(sw^{alt} :: Ds, Sf, Sr, Se, D)}{(Ds, Sf, Sr, sw^{alt}, \{sw^{alt}\} \cup D)} \quad if \begin{cases} P \models Reach\left(c_i^{Sf}, sw^{alt}\right) \wedge Forward\left(c_i^{Sf}, sw^{alt}\right) \\ Sr \neq sw^{alt} \end{cases} \\
&\qquad where \begin{cases} \left\{(sw^{alt}, r_i^{sw^{alt}}), (Sf, r_i^{Sf})\right\} \subseteq Rl \\ r_i^{sw^{alt}} = \left(i, c_i^{sw^{alt}}, a_i^{sw^{alt}}\right) \\ r_i^{Sf} = \left(i, c_i^{Sf}, a_i^{Sf}\right) \end{cases} \\
&Success: \frac{(sw^{alt} :: Ds, Sf, Sr, Se, D)}{(Sr, Se, \{sw^{alt}\} \cup D)} \quad if \begin{cases} P \models Reach\left(c_i^{Sf}, sw^{alt}\right) \wedge Forward\left(c_i^{Sf}, sw^{alt}\right) \\ Sr = sw^{alt} \end{cases} \\
&\qquad where \begin{cases} (sw^{alt}, r_i^{sw^{alt}}) \in Rl \\ r_i^{sw^{alt}} = \left(i, c_i^{sw^{alt}}, a_i^{sw^{alt}}\right) \end{cases} \\
&Skip: \frac{(sw^{alt} :: Ds, Sf, Sr, Se, D)}{(Ds, Sf, Sr, Se, D)} \quad if P \not\models Reach\left(c_i^{Sf}, sw^{alt}\right) \wedge Forward\left(c_i^{Sf}, sw^{alt}\right) \\
&\qquad where \begin{cases} (sw^{alt}, r_i^{sw^{alt}}) \in Rl \\ r_i^{sw^{alt}} = \left(i, c_i^{sw^{alt}}, a_i^{sw^{alt}}\right) \end{cases} \\
&Failure: \frac{(\emptyset, Sf, Sr, Se, D)}{Fail} \quad if\ \ no\ other\ rules\ apply
\end{aligned}
\tag{7}
$$

The inference system is initiated via four variables, a list of distinct switches Ds, source switch Sr, expected switch Se, and the empty set D as an alternative route. Source switch (Sr) refers to the switch that is one hop before the faulty switch (sw^f). The expected switch (Se) refers to the switch that is one hop after the faulty switch.

Every sw^{alt} in Ds for which the condition of the *Add* rules is correct is considered as a candidate that can reach Se. Then the rule will add the sw^{alt} to D and replace the Se with sw^{alt}. Therefore, in the next iteration, the system will look for a switch that can reach the new expected switch Se. This iteration and the adding will be continued until the $Sr = sw^{alt}$. This means that an alternative route from Sr to Se has been found and the system will terminate.

If the sw^{alt} does not reach the Se, the *Skip* rule will be applied. The rule removes the candidate from the list.

If there is no candidate in Ds and no alternative route is found, then the rule *Fail* will be applied and the system will terminate. At this juncture, we should provide a theorem that shows the correctness of our inference system when adding an alternate route.

Theorem: (Correctness)

Correctness of Success: Let Sr be the source switch and Se the expected switch. If following is true:

$$\left(Ds, Sf, Sr, Se, \emptyset\right) \vdash_I \cdots \vdash_I (Sr, Se, D)$$

then D is the list of switches along the alternative route with respect to P from Sr to Se.

Proof To prove the correctness of success of Inference System 7, we must prove that D is a valid route from Sr to Se. This is obvious since, after the application of the inference rule *Add*, the list D contains the list of switches that, with respect to the P, gives a route from Sr to Se. If there is no valid route then $D \neq \emptyset$ and the inference system returns failure.

Correctness of failure: Let Sr be the source switch and Se the expected switch. If the following is true:

$$\left(Ds, Sf, Sr, Se, \emptyset\right) \vdash_I \cdots \vdash_I failure$$

then there is no D as a valid alternative route from Sr to Se with respect to P.

Proof To prove the correctness of failure of the Inference System 7, we must prove that all possible candidates in Ds have been checked and that there is no valid alternative route. This is obvious since if there is a valid alternative route then:

$$\left(D \neq \emptyset\right) \wedge (\exists sw \in D, sw = Sr)$$

Thus, the inference rule Success should be applied. Therefore, when the inference rule failure is applied it means there is no valid alternative route.

According to the above theorem, we can define a *lemma* as follows:

Remark 1 f Inference System 7 returns D and $D \neq \emptyset$, then D is an alternative route. Moreover, if $D = \emptyset$ it means there is no alternative route that exists for the given faulty route.

Proof Since the *lemma 1* is derived from the Inference System 7, the proof of the correctness of success and the correctness of failure of Inference System 7 is valid for this *lemma 1*.

According to the lemma 1 and Inference System 7, the inference system for finding a new route is presented. The formal expression of the solution is described in Inference System 8.

Inference System for Altering Route:

$$Init: \frac{}{(Dr, sw^f, sw^{src}, sw^{exp}, \emptyset)} \quad where \begin{cases} Dr & = The\ list\ of\ all\ distinct\ directions\ (route)\ of\ the\ network \\ sw^f & = The\ faulty\ switch. \\ sw^{src} & = The\ source\ switch\ that\ is\ before\ the\ faulty\ switch. \\ sw^{exp} & = The\ expected\ switch\ to\ be\ reached\ after\ the\ faulty\ switch. \end{cases}$$

$$Add: \frac{(d_i :: Dr, sw^f, sw^{src}, sw^{exp}, \{d_i\} \cup D)}{(Dr, sw^f, sw^{src}, sw^{exp}, \{d_i'\} \cup D)} \quad if \begin{cases} P \models Reach\left(c_i^{sw^f}, sw^{exp}\right) \wedge \neg Forward\left(c_i^{sw^f}, sw^f\right) \\ AR \neq \emptyset \end{cases}$$

$$where \begin{cases} \left\{(sw^{src}, r_i^{sw^{src}}), (sw^{exp}, r_i^{sw^{exp}}), (sw^f, r_i^{sw^f})\right\} \subseteq Rl \\ r_i^{sw^{src}} = \left(i, c_i^{sw^{src}}, a_i^{sw^{src}}\right) \\ r_i^{sw^{exp}} = \left(i, c_i^{sw^{exp}}, a_i^{sw^{exp}}\right) \\ r_i^{sw^f} = \left(i, c_i^{sw^f}, a_i^{sw^f}\right) \\ AR = alterRoute\left(sw^f, sw^{src}, sw^{exp}\right) \\ d_i' \in AR \end{cases}$$

$$Failure: \frac{(d_i :: Dr, sw^f, sw^{src}, sw^{exp}, D)}{Fail} \quad if \begin{cases} P \models Reach\left(c_i^{sw^f}, sw^{exp}\right) \wedge \neg Forward\left(c_i^{sw^f}, sw^f\right) \\ AR = \emptyset \end{cases}$$

$$where \begin{cases} \left\{(sw^{exp}, r_i^{sw^{exp}}), (sw^f, r_i^{sw^f})\right\} \subseteq Rl \\ r_i^{sw^{src}} = \left(i, c_i^{sw^{src}}, a_i^{sw^{src}}\right) \\ r_i^{sw^{exp}} = \left(i, c_i^{sw^{exp}}, a_i^{sw^{exp}}\right) \\ r_i^{sw^f} = \left(i, c_i^{sw^f}, a_i^{sw^f}\right) \end{cases}$$

$$Skip: \frac{(d_i :: Dr, sw^f, sw^{src}, sw^{exp}, D)}{(Dr, sw^f, sw^{src}, sw^{exp}, D)} \quad if\ \ no\ other\ rules\ apply$$

$$Success: \frac{(\emptyset, sw^f, sw^{src}, sw^{exp}, D)}{D} \tag{8}$$

The system is initiated via the list of distinct routes Dr and the empty set of the new route D.

For each route that verifies the condition of the inference rule *Add*, or where there is a contradiction anomaly along the route, the inference rule tries to find an alternative route and add it to the D route list. The procedure of finding the alternative rule is handled by Inference System 7. If there is no alternative route for route d the inference rule *Fail* will be applied. According to the *lemma 1*, if there is a contradiction anomaly but there is no solution for it, the system terminates in failure. Moreover, if the conditions do not match with the route d, then there is no anomaly and the route will be removed from Dr. In the end, if the candidate list Dr is empty, the inference rule *Success* is applied. This means that there are no more routes to be checked and the system terminates successfully. Therefore, the system is correct and the correctness theorem is explained as follows.

Theorem: (Correctness)

Correctness of success: Let D be the list of switches along the route. If the following is true:

$$\left(Dr, sw^f, sw^{src}, sw^{exp}, \emptyset\right) \vdash_I \cdots \vdash_I (sw^{src}, sw^{exp}, D')$$

then D' is a set of switches along the alternative route that connect the sw^{src} to the sw^{exp} with respect to P. D' is a valid alternative route that does not pass through sw^f.

Proof To prove the correctness of success of the Inference System 8, we must prove that D' is valid. This is obvious since after the application of the inference rule *Add*, if there is no valid alternative route then $AR = \emptyset$ and the inference system returns a failure.

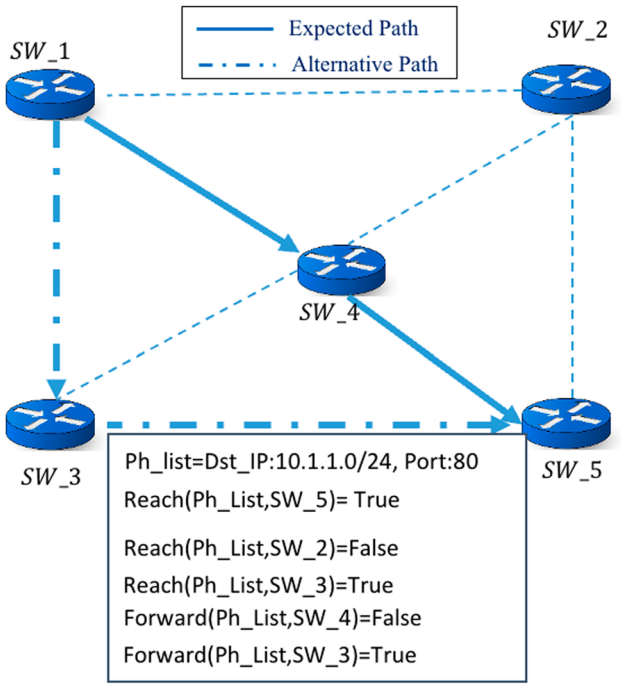


Fig. 12 Example of repairing the contradiction anomaly.

Correctness of failure: Let D be a list of switches along the route. If the following is true:

$$\left(Dr, sw^f, sw^{src}, sw^{exp}, \emptyset\right) \vdash_I \cdots \vdash_I failure$$

then there is no D' as a valid alternative route.

Proof To prove the correctness of failure of Inference System 8, we must prove that there is no valid alternative route. This is obvious since if there is a valid alternative route then $D' \neq \emptyset$ and $AR \neq \emptyset$. So the inference rule add should be applied. Therefore, when the inference rule failure is applied it means there is no valid alternative route.

For example, in Fig. 12 and regarding the high-level policies (Fig. 9a, b), according to the high-level policy in Fig. 9a b, there is an alternative route from SW_3 that enables the packet header space (DHS_1) to reach the SW_5.

7 Evaluation

In this section, we evaluate the practical value of the proposed method by implementing the inference systems. The implementation uses C++, and Ubuntu 20.04 as the operating system. The simulation is run on a MacBook Pro via 2.7 GHz Dual-Core Intel core i5 processor and 4 GB memory. To simulate a real-life scenario we have used a topology

based on a real ISP configuration from the Rocketfuel dataset [37] that contains more than 17,000 routing policies and Stanford topology [38] with 12 OpenFlow switches and more than 750,000 rules. The Stanford topology is run on the Mininet [39] emulator via the Ryu [40] controller. Snapshots of the flow tables were then used for experiments. Five different scenarios are designed for assessment that contains switches with 250 rules in its flow tables. Five scenarios each include a different number of faulty rules, which are 1, 5, 10, and 20 erroneous rules.

7.1 Evaluation metrics

We consider the processing time of creating a new rule, swapping rules, and finding the alternative route as metrics for assessing the performance of the implemented inference systems. Moreover, since the overlap and misordered anomalies take place in a single switch, there is a possibility to apply a multi-thread approach to correct the anomalies of separate switches. Therefore, the processing time of the single-thread approach is compared with the parallel approach as an evaluation metric.

Different positions for the target rules are considered to evaluate the effect of priority fields on performance. We also compare the effect of the inter- and intra-anomalies on the method performance.The algorithms are run 30 times for each distinctive synthetic flow table. We report the average execution time with the confidence interval 95%. In addition, we compare the overhead of each inference system in the network in order to assess the side effects of the methods have on the network performance. Finally, the downtime values are compared with the different approaches to find out which type of anomalies can be resolved faster.

7.2 Evaluation results

In this section, the effect of the number of faulty rules on different correcting methods is presented. Moreover, the performance of the detection method is compared with the single-thread and multi-thread approaches to demonstrate the parallelization capability of the method. The performance evaluation of each correcting method is discussed.

7.2.1 Adding a new rule

As mentioned in Sect. 5, if two rules have an overlap anomaly then the system should calculate the overlap section. As a next step, it creates a new rule based on the intersect part and inserts it into the flow table with a higher priority. In this section, the processing time of adding multiple rules is evaluated for both single-thread and multi-thread approaches. The result is sketched in Fig. 13.

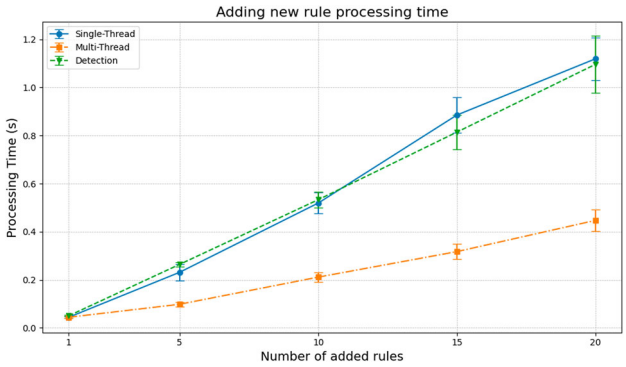


Fig. 13 Adding a new rule inference system processing time

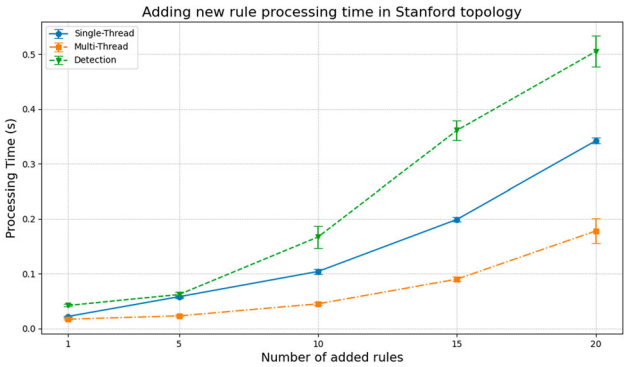


Fig. 14 Adding a new rule inference system processing time in Stanford topology

According to Fig. 13, the processing time for the single-thread approach is longer than the multi-thread approach and has almost linear behavior. However, since there are some overhead processes in the multi-thread method, its processing time does not follow a linear pattern. Moreover, since the detection method has an almost linear pattern, the processing time of the correcting method is closer to that of the single-thread approach. According to Fig. 14, we get the same pattern in our experiments with the Stanford topology.

7.2.2 Swapping two rules

The swapping action is applied to a pair of rules with a misordered anomaly. In this section, the performance of the swapping of multiple rules is evaluated for both single-thread and multi-thread approaches. The result of the assessment is presented in Fig. 15.

According to Fig. 15, the swapping process is not time-intensive. For this reason, there is not a meaningful difference between the single-thread and multi-thread approaches.

According to Fig. 16, in Stanford topology the swapping process with a multi-thread approach has more overhead than the single-thread approach. We believe that the swapping process is a small process to which the parallelization process adds more overhead to it. As presented in the chart, the swap-

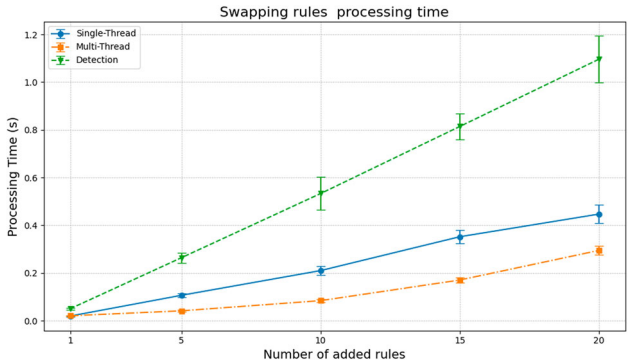


Fig. 15 Swapping rules inference system processing time

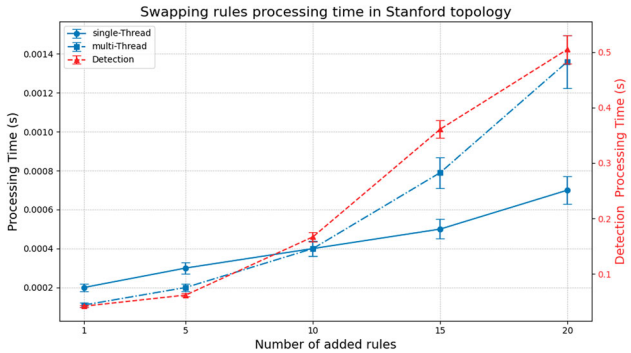


Fig. 16 Swapping rules inference system processing time in Stanford topology

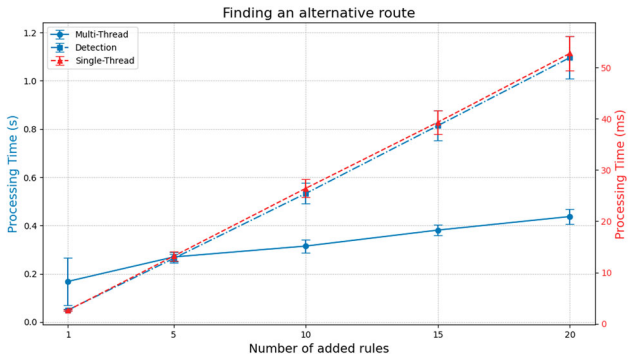


Fig. 17 Finding an alternative route inference system processing time

ping process is significantly faster than the detection process when the number of faulty rules are increased.

7.2.3 Finding an alternative route

According to the definition of the contradiction anomaly, which is discussed in Sect. 5, searching for a possible valid route may resolve the conflict. To this end, the two inference systems, alter route and add a new route, try to find the possible valid route in accordance with the high-level policies. In this section the performance of the this process is evaluated and the results are shown in Fig. 17.

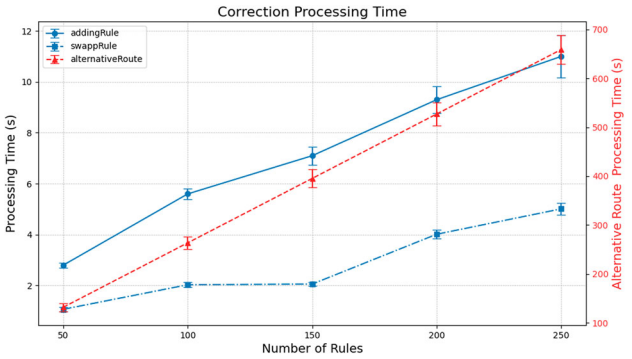


Fig. 18 Comparing the effect of the target rule position on the performance of the correction methods

As sketched in Fig. 17, the single-thread processing time has an almost linear pattern. If the faulty switches are not same, then we can use the multi-thread approach to find a possible alternative route. In this scenario, the processing time of finding a solution is drastically reduced. The detection process and finding the root cause of the failure has a linear pattern and it is slower than the multi-thread correcting approach.

7.3 Analysis of the rule position

In this section, we try to understand how the position of the target rule affects correction performance. When we talk about the position, we mean the number of rules existing between the target rule and the wrong rules. The rules that are between the target and the wrong rules should be checked with the target rule. If there is an overlap between them, the correction process should also be applied.

In this section, we describe five scenarios with a different number of rules between the target and wrong rules. The scenarios try to simulate the worst case scenario. This means that all rules between the target rule and the wrong rules have a conflict with the target rule and the correction method should be applied. The result is presented in Fig. 18.

As sketched in Fig. 18, the different positions of the target in a worst case scenario have a direct effect on the processing time of the methods. According to the result, all methods have a linear processing time, but finding an alternative route has the slowest process.

7.4 Overhead analysis

The overhead of the resolving methods on network performance is one of the important metrics for evaluation. In this section, the overhead of adding a new rule, swapping rules, and finding an alternative route in the network is analyzed and compared. When we talk about overhead, we mean the

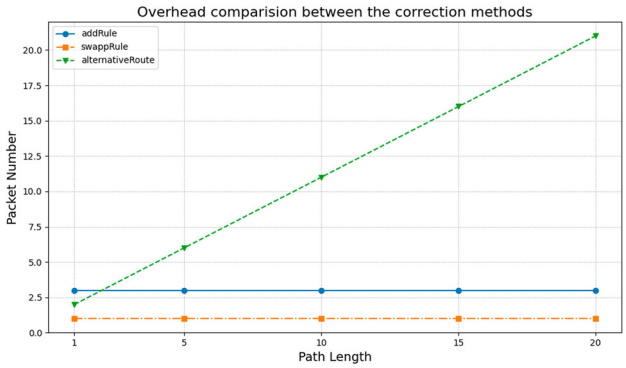


Fig. 19 Comparing the overhead of correcting methods

number of packets that are generated for either adding a new rule, swapping rules or finding an alternative route.

As shown in Fig. 19, adding rules and swapping rules have a constant overhead for different path lengths. The add rule method generates three packets for each correction procedure. Swapping rules generates only one packet for each correction action. However, finding an alternative method has a linear pattern when generating the extra packets for each correcting procedure.

According to the overhead evaluation result, finding an alternate method with 2 extra packets has less overhead than the adding rule method for correcting a rule in one node. The swapping rule has the least overhead via one packet.

7.5 Downtime analysis

In this section, the downtime duration of different numbers of faulty rules are compared and analyzed. The downtime duration refers to the detection processing time for discovering the root causes of the failure, and the processing time of the corresponding correction method.

The downtime duration of the correcting methods can be seen in Table 2. According to the Table 2, the anomalies with contradiction root causes have the longest downtime. The process is slow since the altering route algorithm must check all possible routes to the target switch and compares them via the high-level policies. However, the multi-thread approach checks all possible routes simultaneously, and thus, the downtime is significantly reduced. The downtime of the misordering failures is the shortest when using both single-thread and multi-thread approaches. Since the misordering failures are solved via swapping the rules, the correction process is fast. Since the overlapped failures require the addition of a new rule, they have a longer correcting process than the misordering failures.

The summary of the comparison of the Net Auto-Solver approach with Anime and other related work is presented in Table 3. All anomalies in Table 3 refers to the anomalies in flow tables that cause packets to pass through routes, which are not expected regarding the high-level policies. Most importantly, we observe that Net Auto-Solver is the only approach that not only detects all types of anomalies but also corrects them via an automatic proved method, which makes it superior to the rest of the approaches.

The column "Formally Proved" refers to the soundness and completeness of the proposed method. As opposed to related work, the soundness and completeness of our method have been formally proven.

Since the lower downtime is an important criterion, we compare the complexity of the Net Auto-solver with relevant methods, which validate the rule tables and solve the conflicts automatically. The result is presented in Table 4.

Table 2 Correction method downtime comparison

	Overlapped		Mis-Ordered		Contradiction	
Failure No.	Single-Thread(s)	Multi-Thread(s)	Single-Thread(s)	Multi-Thread(s)	Single-Thread(s)	Multi-Thread(s)
1	0.094	0.094	0.069	0.070	2.685	0.218
5	0.495	0.362	0.370	0.305	13.456	0.533
10	1.052	0.744	0.743	0.617	26.960	0.848
15	1.699	1.131	1.165	0.984	40.140	1.195
20	2.215	1.543	1.542	1.390	53.863	1.533

Table 3 Comparisons of OpenFlow anomaly correction methods

	Detecting all types of anomalies	Automatic Correction	Formally Proved
Net Auto-Solver	✓	✓	✓
Anime [15]	✗	✗	✗
NSOA [16]	✗	✗	✗
Yazidi et al. [9]	✗	✗	✗
NEAt [10]	✓	✓	✗

Table 4 Comparisons of the complexity of the relevant methods

	Net Auto-solver	ROCA [13]	EPE [28]	Jinjing [14]
Complexity	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(2^n)$

According to the Table 4, Jinjing has the lowest complexity and Net Auto-solver has the same complexity as ROCA and EPE. However, it should be acknowledged that the Jinjing method works on BGP rules. Moreover, contrary to the Net Auto-solver, there is no clue regarding the accuracy of the other methods in Table 4.

8 Conclusion

Computer networks play a fundamental role in our lives and conflicts between network rules can be very costly to us. Therefore, anomaly detection and correction will surely be a hot topic in the coming years. There are a significant number of research papers that present the detection of conflicts between firewall rules or forwarding devices in SDN. However, only a few papers focus on trying to correct the anomalies without human intervention.

This paper proposes the *Net Auto-Solver* method to resolve conflicts automatically. The Net Auto-Solver uses a formal expression to model the high-level policy. Furthermore, by defining inference systems, it tries to detect and correct the anomalies automatically by resorting to the high-level policies. To evaluate the performance of Net Auto-Solver, we designed a real-life scenario. The single-thread and multi-thread approaches of the correction method are implemented and assessed, and the result is promising. According to the experimental results, Net Auto-Solver can correct 20 different faulty rules between 400 ms and 1.1 seconds, which is a feasible time on this scale.

As future work, we would like to apply an incremental approach to improve the performance of the correction process. Since the data plane rules are updated frequently, the correction performance is an essential factor in the efficiency of the method. In addition, by expanding the functionality of the inference systems, we can try to solve more problems in the network administration area.

Funding Open access funding provided by University of Oslo (incl Oslo University Hospital)

Data availibility Data sharing is not applicable to this article as no datasets were generated or analyzed during the current study.

Declarations

Conflict of interest All potential conflicts of interest related to the funding source were disclosed and managed in accordance with the policies of OsloMet University. All authors declare that they have no conflicts of interest related to this research.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Aryan, R., Yazidi, A., Bouhoula, A., Engelstad, P.E.: Net auto-solver: a formal approach for automatic resolution of openflow anomalies. In: 2020 IEEE 45th Conference on Local Computer Networks (LCN), pp. 357–360. IEEE (2020)
2. Aryan, R., Yazidi, A., Engelstad, P.E., Kure, Ø.: A general formalism for defining and detecting openflow rule anomalies. In: 42nd IEEE Conference on Local Computer Networks. Institute of Electrical and Electronics Engineers (IEEE) (2017)
3. Shaghaghi, A., Kaafar, M.A., Buyya, R., Jha, S.: Software-defined network (sdn) data plane security: issues, solutions, and future directions. In: Principles and Paradigms, Handbook of Computer Networks and Cyber Security, pp. 341–387 (2020)
4. Deb, R., Roy, S.: A comprehensive survey of vulnerability and information security in sdn. Comput. Netw. **206**, 108802 (2022)
5. Aryan, R., Yazidi, A., Brattensborg, F., Kure, Ø., Engelstad, P.E.: Sdn spotlight: a real-time openflow troubleshooting framework. Future Gener. Comput. Syst. **133**, 364–377 (2022)
6. Kazemian, P., Varghese, G., McKeown, N.: Header space analysis: static checking for networks. In: Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12), pp. 113–126 (2012)
7. Kazemian, P., Chang, M., Zeng, H., Varghese, G., McKeown, N., Whyte, S.: Real time network policy checking using header space analysis. In: Proceedings of 10th USENIX Conference in Networked System Design Implementation, pp. 99–112 (2013)
8. Khurshid, A., Zou, X., Zhou, W., Caesar, M., Godfrey, P.B.: Veriflow: verifying network-wide invariants in real time. In: Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13), pp. 15–27 (2013)
9. Yazidi, A., Bouhoula, A.: On assisted packet filter conflicts resolution: an iterative relaxed approach. In: 2016 IEEE 41st Conference on Local Computer Networks (LCN), pp. 35–42 (2016)
10. Zhou, W., Croft, J., Liu, B., Ang, E., Caesar, M.: Automatically correcting networks with neat. In: 15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18), pp. 595–608 (2018)
11. Prakash, C., Lee, J., Turner, Y., Kang, J.-M., Akella, A., Banerjee, S., Clark, C., Ma, Y., Sharma, P., Zhang, Y.: Pga: using graphs to

express and automatically reconcile network policies. ACM SIGCOMM Comput. Commun. Rev. **45**(4), 29–42 (2015)
12. Al-Shaer, E.S., Hamed, H.H.: Firewall policy advisor for anomaly discovery and rule editing. In: IFIP/IEEE Eighth International Symposium on Integrated Network Management, pp. 17–30. IEEE (2003)
13. Asif, A.B., Imran, M., Shah, N., Afzal, M., Khurshid, H.: Roca: auto-resolving overlapping and conflicts in access control list policies for software defined networking. Int. J. Commun Syst **34**(9), e4815 (2021)
14. Tian, B., Zhang, X., Zhai, E., Liu, H.H., Ye, Q., Wang, C., Wu, X., Ji, Z., Sang, Y., Zhang, M., et al.: Safely and automatically updating in-network acl configurations with intent language. In: Proceedings of the ACM Special Interest Group on Data Communication, pp. 214–226 (2019)
15. Kheradmand, A.: Automatic inference of high-level network intents by mining forwarding patterns. In: Proceedings of the Symposium on SDN Research, pp. 27–33 (2020)
16. Fan, Z., Wu, H., Xu, J., Tang, Y.: An optimization algorithm for spatial information network self-healing based on software defined network. In: 2017 12th International Conference on Computer Science and Education (ICCSE), pp. 369–374. IEEE (2017)
17. Băluţă, A., Soare, R.M., Rughiniş, R., Turcanu, D.: Geckonet-self-healing sdn framework. In: 2024 23rd RoEduNet Conference: Networking in Education and Research (RoEduNet), pp. 1–6. IEEE (2024)
18. Prometheus monitoring. https://grafana.com/oss/prometheus/. Accessed 10 Jan 2025
19. Grafana monitoring. https://grafana.com/oss/grafana/. Accessed 10 Jan 2025
20. Gember, A., Shrestha, R., Sun, X.: Localizing router configuration errors using minimal correction sets. arXiv:2204.10785 (2022)
21. Yao, J., Jiang, Z., Zou, K., Weng, S., Li, Y., Li, D., Li, Y., Cao, X.: Fast verification of network configuration updates. Comput. Mater. Contin. **75**, 1 (2023)
22. Bringhenti, D., Marchetto, G., Sisto, R., Valenza, F., Yusupov, J.: Automated firewall configuration in virtual networks. IEEE Trans. Depend. Secure Comput. **20**(2), 1559–1576 (2023)
23. Shahriyar, M.M., Saha, G., Bhattacharjee, B., Reaz, R.: Deft: distributed, elastic, and fault-tolerant state management of network functions. In: 2023 19th International Conference on Network and Service Management (CNSM), pp. 1–7. IEEE (2023)
24. Sekar, H., Vasudevan, S.K.: Openflow groups based fast failover mechanism for software defined networks. Int. J. Adv. Intell. Paradigms **28**(3–4), 238–252 (2024)
25. Menaceur, A., Drid, H., Rahouti, M.: Fault tolerance and failure recovery techniques in software-defined networking: a comprehensive approach. J. Netw. Syst. Manag. **31**(4), 83 (2023)
26. Shi, L., Wang, Y., Alur, R., Loo, B.T.: Netrep: automatic repair for network programs (2021). arXiv preprint arXiv:2110.06303
27. Saâdaoui, A., Souayeh, N., Bouhoula, A.: Fare: Fdd-based firewall anomalies resolution tool. J. Comput. Sci. **23**, 181–191 (2017)
28. Hussain, M., Shah, N.: Automatic rule installation in case of policy change in software defined networks. Telecommun. Syst. **68**(3), 461–477 (2018)
29. Aryan, R., Yazidi, A., Engelstad, P.E.: An incremental approach for swift openflow anomaly detection. In: 2018 IEEE 43rd Conference on Local Computer Networks (LCN), pp. 502–510. IEEE (2018)
30. Perešíni, P., Kuźniar, M., and Kostić, D.: Monocle: dynamic, fine-grained data plane monitoring. In: Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies, p. 32. ACM (2015)
31. Ibrar, M., Wang, L., Muntean, G.-M., Akbar, A., Shah, N., Malik, K.R.: Prepass-flow: a machine learning based technique to minimize acl policy violation due to links failure in hybrid sdn. Comput. Netw. **184**, 107706 (2021)
32. He, H., Yang, S., Zhou, X., Wang, J.: Smwt: a universal and lightweight sdn-based meter table watermarking traceback scheme. In: 2023 24st Asia-Pacific Network Operations and Management Symposium (APNOMS), pp. 71–76. IEEE (2023)
33. Yan, B., Liu, Q., Shen, J., Liang, D.: Flowlet-level multipath routing based on graph neural network in openflow-based sdn. Future Gener. Comput. Syst. **134**, 140–153 (2022)
34. Attarian, R., Mohammadi, E., Wang, T., Beni, E.H.: Mixflow: assessing mixnets anonymity with contrastive architectures and semantic network information. Cryptology ePrint Archive (2023)
35. Piotrowska, A.M., Hayes, J., Elahi, T., Meiser, S., Danezis, G.: The loopix anonymity system. In: 26th Usenix Security Symposium (usenix security 17), pp. 1199–1216 (2017)
36. Hu, H., Ahn, G.-J., Kulkarni, K.: Detecting and resolving firewall policy anomalies. IEEE Trans. Depend. Secure Comput. **9**(3), 318–331 (2012)
37. Rocketfuel. http://research.cs.washington.edu/networking/rocketfuel//. Accessed 01 Jan 2025
38. Stanford backbone. https://github.com/wuyangjack/standford-backbone. Accessed 25 Feb 2025
39. Mininet. http://mininet.org/. Accessed 01 Jan 2025
40. Ryu. https://ryu-sdn.org/. Accessed 01 Feb 2025

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.