Redundant rule Detection for Software-Defined Networking

Jian Su[1], Ruoyu Xu[1], ShiMing Yu[1], BaoWei Wang[1], and Jiuru Wang[2*]

[1]School of Computer science and technology, Nanjing University of Information Science & Technology

NanJing, 210044, China

[2]School of Information science and engineering, Linyi University

Linyi, 276005, China

[e-mail: wangjiuru@lyu.edu.cn]*

* Corresponding author: Jiuru Wang

Received August 30, 2019; revised March 16, 2020; accepted March 31, 2020; published June 30, 2020

Abstract

The emergence of Software Defined Networking (SDN) overcomes the limitations of traditional networking architectures. There are some advantages in SDN which are centralized global network view, programmability, and separation of the data plane and control plane. Due to the limitation of data plane storage capacity in SDN, it is necessary to process the redundancy rules of switch. In this paper, we propose a method for active detection and processing of redundant rules. We use the result generated by the customized probe package to detect redundant rules. And by checking the forwarding behavior of probe packets in the data plane, the redundancy rules are further processed. Furthermore, in order to quickly check the dynamic networks, we propose an incremental algorithms for rapidly evolve the network strategies. We conduct simulation experiments on Matlab to verify the feasibility of the algorithm. The influence of some parameters on the result are discussed.

This work is supported by the Natural Science Foundation of China (No.61802196, 61972207); the Natural Science Foundation of Jiangsu Province (No.BK20180791); the Natural Science Foundation of Jiangsu Higher Education Institution of China (No.17KJB510036); and the Startup Foundation for Introducing Talent of NUIST. This work is also supported in part by Soft Science Program of China Meteorological Administration, Priority Academic Program Development of Jiangsu Higher Education Institutions, and Engineering Research Center of Digital Forensics, Ministry of Education. This work is in part supported by the Natural Science Foundation of Shandong Province (No.ZR2018LF007); Shandong Provincial Key Research and Development Program (SPKR&DP)(No.2019GNC106027).

Keywords: SDN, Redundant rule detection, Missing detection

1. Introduction

SDN is an emerging networking architectures which enable program data plane become an independent-packet forwarding device, while the control logic is implemented on the control plane [1-2]. SDN takes advantage of excellent visibility and flexibility to manage the network. Meanwhile the centralization of the network controls to optimize its performance. In SDN, logic control is centralized in the software-based controllers in SDN. So Appearing of SDN architectures promotes the separation of the control logic of hardware. These features of SDN are in contrast of traditional network architectures, such as vertical integrated solution.

The concept of SDN mainly comes from the OpenFlow project created by McKeown et al. [3]. So far, SDN has been developing continuously. In SDN, the network architecture is divided into programmable data plane and logically centralized control plane. Most of the network control logic (specified by software programming) is put into the control plane of the split architecture, which simplifies the data plane. Therefore, the data plane is only controlled by the forwarding decision on the control plane installation [4-5]. SDN provides powerful programmability for networking and reduces the complexity of network management. Networks implemented using SDN can rapidly evolve to satisfy the rapidly changing needs of network users for network resources, such as in Cloud Computing [6], Network Function Virtualization (NFV) [7], and Internet of Things (IoT) [8-10] .

In traditional networks, in order to extend the network life, Wireless sensor network (WSN) needs to be equipped with a certain number of redundant sensors to balance network energy consumption [11-14]. Similarly, a multi-path routing strategy based on energy efficiency also can greatly extend the lifetime of Wireless Sensor Networks (WSNs) too [15]. However, the rigid and static traditional network architectures has been unable to adapt to the increasing interactive and dynamic multimedia network types. Unlike the traditional network, the network control plane of SDN is no longer distributed, but more centralized intelligence. Therefore, redundant rules in the flow table are unnecessary and need to be cleaned up.

In the past, in addition to adding match-field, priority and action in the flow table, instruction items have been added in OpenFlow1.3 [16]. Instruction items are used to indicate the aging time of the flow table. If the unused time of the rule exceeds the specified time interval, the flow table is dropped to save memory resources. That is the passive deletion mechanism of the aging rule based on the timeout mechanism. The reason for deleting rules is that memory is a relatively scarce resource in the data plane. Based on the fast and efficient performance requirements of data plane, flow table storage in switches usually uses ternary Content Addressable Memory [17].

However, TCAM's hardware circuits are complex and expensive. Furthermore, since TCAM support parallel search of table items, a large number of matching operations increase power consumption. So these characteristics of the TCAM greatly limit the size of the flow meter in the switchgear, but also limit the size of the SDN. For example: Pica8 p-3290 [18] switch supports the storage of 2000 TCAM table flows. Therefore, it is necessary to delete redundant flow tables.

However, as the scale of classified packet flow tables continues to expand, using only the timeout mechanism will result in a large number of useless flow tables stuck in the switch at a fixed time interval (a fixed time interval is a parameter of the timeout mechanism). It will significantly increase the number of flow tables stored in the switch. Therefore, in terms of controllers, we need a mechanism to actively handle redundant rules, which is complementary to passive methods.

This paper presents a novel idea. We propose an algorithm to detect and process redundancy rules for a given set of rules. Redundancy rules are packets that cannot be matched because of the priority problem of the rules themselves and the matching principle of OpenFlow protocols. The existence of redundant rules will take up the already scarce memory resources of the Switch. Based on the above requirements, we propose an algorithm to detect redundancy rules. This algorithm can not only detect redundancy rules, but also detect the absence of high priority rules before deletion. Avoiding the deletion of useful rules and ensures the reliability of the algorithm. Moreover, we can extend the algorithm forwarding strategy by using non-metabolic variants in dynamic rule update. That can be used to dynamically change the flow table.

The organization of the rest of this paper as follows. Section 2 introduces some basic knowledge of SDN and the reason of causing redundancy rules. Section 3 come up with a series of algorithms for solving redundancy rules problem in the whole rule set and incremental rule set. Section 4 carries out simulation experiments for multiple variables and report evaluation results. Finally, Section 5 concludes this paper.

2. Related work

In this section, we first make a preliminary introduction to SDN. And then according to the description in section 1 above and the basic situation of SDN in section 2.1, the current problems to be solved are described in detail.

2.1 Preliminary Introduction of SDN

In a typical SDN architecture, a set of Switches is controlled by controller assigned standard protocols such as OpenFlow. The controller compiles the planned forwarding path into a set of rules and installs it in the flow table of the corresponding Switch. The Switch will process packets from the data plane using the rules of the flow table.

For any entry in the flow table, there are three main components. These include: (1) Flow-Match Header, it defines an entry of flow, (2) Action, it determines how packets are forwarded (i.e., forward to other switch or to another different flow table) and (3) some additional field, it contains priority, statistics, cookie identifier and so on. This is shown in **Fig. 1**.

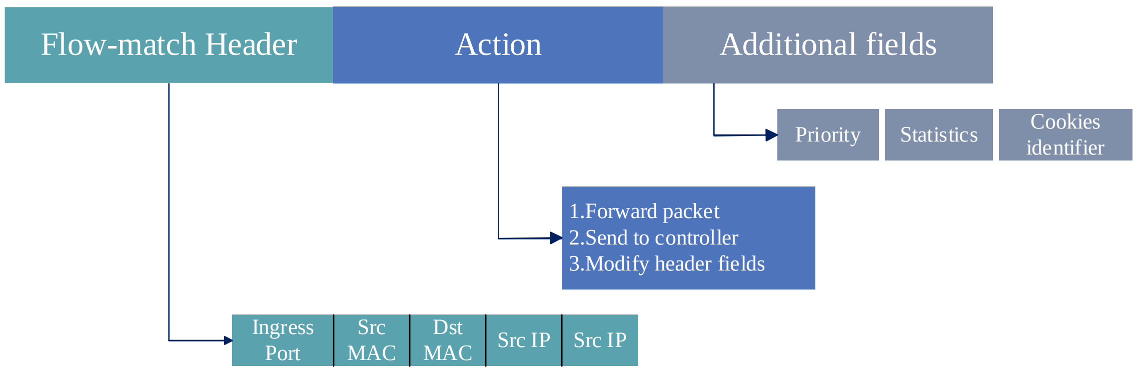


Fig. 1. Example of flow entry

SDN strengthens the network policy by converting the forwarding path into Switch-understandable rules. As shown in **Fig 1**, the role of the Flow-Match header and the rule action regulates the packets to be processed and how to handle them, respectively. Based on SDN traffic-based forwarding rules, packet matching rules can be made by comparing with regular traffic matching heads. Since wildcards (represented by *) can match bits 0 and 1 at the same time, the matching procedure can aggregate multiple traditional exact matching rules. That is, packets from the data plane can match multiple rules in the Switch, which is defined as matching blur. Hence, in order to solve the above problems, priority values SDN be further assigned for each rule. When a packet can match multiple rules, it matches the rule with highest priority.

Based on the basic mentioned above, we will introduce some definitions below for the problem.

Definition 1: $r_i.S$ is the set of all packet that match the Flow-Match Header space.

Definition 2: a pair of rules $<r_i, r_j>$ is said to be overlapping if $r_i.S \cap r_j.S \neq \varnothing$, and $r_i.p < r_j.p$.

Where $r_i.p$ denotes the priority of rule r_i.

2.2 Problems

For matching sets $r_i.S$ and $r_j.S$ of any two rules, there are three different relationship of them. (1) The two sets are independent for each other. (2) They are overlapping partially. (3) One set that contains another set. **Fig. 2** shows above three different cases.

Considering the case shown in **Fig 2(c)**, the packet will never match r_i when the rule that meets the inclusion relation is further qualified. More specifically, a pair of sets $<r_i, r_j>$ are identified as overlapping relationships. Moreover, the size of their matching set satisfies r_i greater than r_j. Then we suppose that r_i has a higher priority than r_j. Under the above conditions, because of the flow table matching mechanism, the packet will match r_i and always ignore r_j. This means that the existence of the r_j is useless. So we have the following Definition 3.

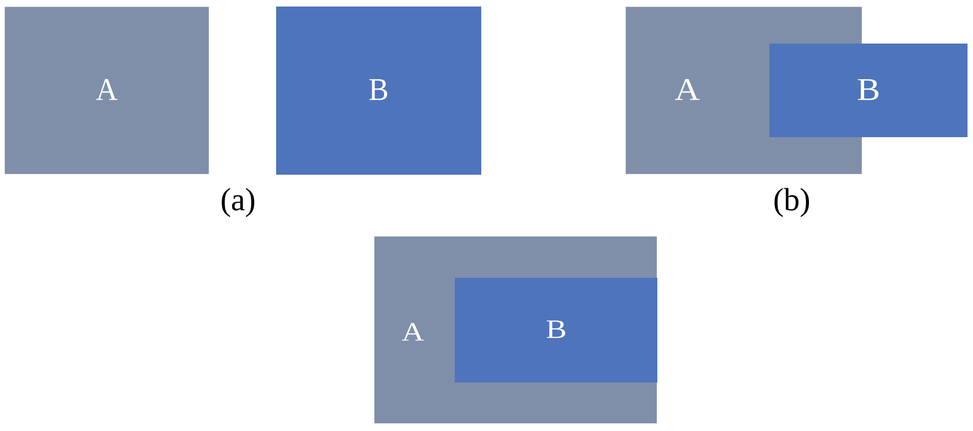


Fig. 2. Veen diagrams of rules' matching set $r_i.S$ and $r_j.S$ in three cases, where A denoted $r_i.S$, and B denoted $r_j.S$.

Definition 3: a rule r_i is said to be redundant if $\exists r_j$, $r_i.S \subseteq r_j.S$, and $r_i.p < r_j.p$.

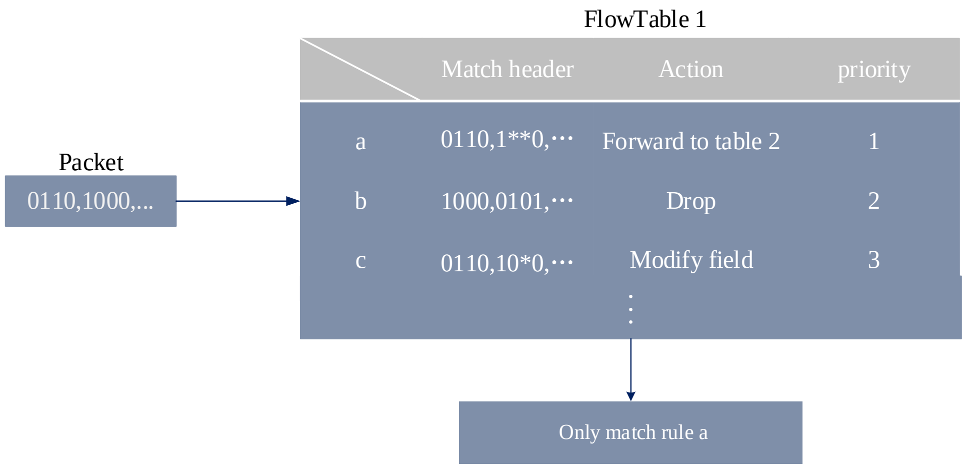


Fig. 3. Problem description

When a rule r_i satisfies Definition 3 above, it can be removed from the Switch. However, before deleting the rule r_i, we should make sure the rule r_j with higher priority than r_i in the Switch are not lost. Otherwise, if the higher priority rule r_j is lost while the rule r_j deleted from the Switch at the same time, the that happens to be transferred in this Switch will have no matching rules. It can lead to packet loss. Therefore, before we delete the redundancy rules, we need to insert a probe to determine whether the higher priority rules are missing.

The existence of redundant rule definitions means that this situation is indeed possible in the SDN. Therefore, you need to add a mechanism to the SDN architecture to solve this problem. However, the existing SDN structure does not take into account this possible problem and does not pay enough attention to the memory problem of the Switch. So we try to study this.

To simplify the process, in this section we will mainly focus on the static rules sets to examine. If rules update happens, we will likely to execute algorithm renewedly on the entire new rule set. Therefore, it is not a wise choice for the dynamic networks with frequent rule updates.

3. Redundancy detection

In this section, we will explore the solutions for redundancy rule detection and processing algorithms. By solving the SAT problem, we can further determine the relationship between the two overlapping rules. SAT results of the problem are also used to construct detectors. It will be injected into the data plane to detect the presence of high priority rules. Based on this, two algorithms are proposed to satisfy the different flow table conditions.

3.1 SAT problem and Probe Generate

There are *l-bit* Flow-Match Header: $r_i.H=(x_{i0},...,x_{ia},...,x_{i(l-1)})$, where $x_{ia}=\{0,1,*\}$. And the section of *a* is [0, *l-1*]. We represent one of r_i's matching field as $r_i.M=(\mathrm{y}_{i0},...\mathrm{y}_{ib},...\mathrm{y}_{i(l-1)})$,. $r_i.H$ is the set of all $r_i.M$. In other word, if $r_i.M$ match with $r_i.H$, we can say the following Eq.1 is true.

$$\bigcap_{0\le a\le l-1} S(x_{ia},y_{ia})=1 \tag{1}$$

where,

$$S(x_{ia},y_{ia})=\begin{cases}1\ ,\ if\ \ x_{ia}=y_{ia}\\ 1\ ,\ if\ \ x_{ia}=*\\ 0\ ,\ if\ \ x_{ia}\neq y_{ia}\end{cases} \tag{2}$$

Furthermore, if $r_i.M$ match with $r_i.H$ we can said the following Eq.3 is true.

$$\bigcap_{0\le a\le l-1} S(x_{ia},y_{ia})=0 \tag{3}$$

That means, there are one or one more $S(x_{ia}, y_{ia})$ whose value are 0.

When it is detected that the rule satisfies Definition 3, we need to inject probe $p=(p_1,...,p_a,...,p_{l-1})$ to judge the missing situation. Before the previous description of $r_i.H$ and $r_i.M$. Combining Eq.1 and Eq.2, we can infer that if probe packet *p* exists that detect missing fault of r_i. The following equation should be satisfied.

$$(\bigcap_{0\le a\le l-1} S(x_{ia},p_a))\bigcap(\neg\bigcap_{0\le a\le l-1} S(x_{ja},p_a))=1 \tag{4}$$

where $r_j\in FT_{sw}$, and has higher priority than r_i.

For any package, if we verify that it satisfies Eq. 4, it is equivalent to verifying that the given true value makes CNF true. The Boolean satisfaction problem (SAT) is called the satisfiability problem of the connection paradigm. Its purpose is to find out if there is a set of assignment of Boolean variables so that the value of the whole conjunction normal form is true. Therefore, a missing-fault detection packet generation problem of polynomial time can be reduced to an SAT problem.

3.2 Static Redundancy Detection Algorithm

Static in the title refers to detecting the storage rules of the controller and Switch without updating the rule set. Static redundancy detection algorithm has two key steps, namely. Redundancy detection and deletion detection.

The key of redundancy detection is to analyze the set that matches the matching domain. The larger the set, the looser the limit on the matching field. (That is, there are more packets that can match it. This leads to the influx of data packets into this rule, thereby resulting in the neglect of other rules). Therefore, redundant rules can be detected by comparing the matching set of each matching field. And the core of deletion detection is to find a specific matching entry as a probe and use its forwarding trajectory to determine whether the deletion operation can be carried out.

To make this easier to understand, we visualize the flow table as a dependency graph G=(V, E). In other word, each rule can be represented as a vertex and overlapping relationship can be represented as edge. For any pair of rules r_i and r_j , if r_i is called overlapping ,and has higher priority than r_j, there is a directed edge $<v_i, v_j>$ in G. There is an example of G that shows as **Fig. 4**. E is an isolated point, there is no other rule that overlap with it.

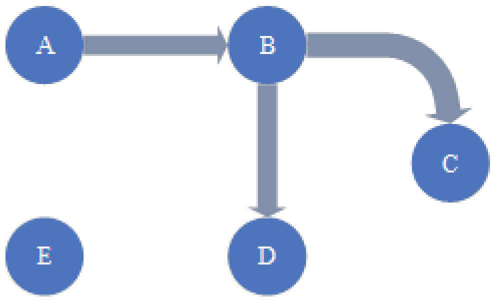


Fig. 4. An example of $G=(V,E)$

There is a defined function $GenerateProbe(r,R)$ that is used to generate probe. A rule r and a set R of rules as input, $GenerateProbe(r,R)$ will output a probe packet p, which matches with rule r but does not match with any rule in R.

Algorithm 1: Static Redundancy Detection Algorithm

Input: Flow table FT_{ctr}

Output: Dependency graph $G_{sw}=<V,E>$ of data plane and Set of redundancy rules R

```
1   $V \leftarrow \{v_i \mid v_i \text{ corresponds to } r_i \in FT_{ctr}\}$
2   Set $S \leftarrow$ any pairs of overlapping rules in $FT_{ctr}$
3   **while** $S \neq \varnothing$ do
4       $(v_i, v_j) \leftarrow$ any overlapping-rule pair in $S$
5       **if** $GenerateProbe(v_i \cup v_j, v_i) = \varnothing$ then
6           $H \leftarrow \{v \mid \text{if } <v, v_i> \text{ and } <v, v_j> \in E\}$
7           $p \leftarrow generateProbe((v_i \cap v_j, H)$
8     Inject p to data plane
9     if p does not follow $v_i$'s action then
10        insert $r_i$ to $FT_{ctr}$
11        $R \leftarrow R \cup \{r_j\}$
12        delete $r_j$ from $FT_{ctr}$ and $FT_{sw}$
13    else
14        delete $r_j$ from $FT_{ctr}$ and $FT_{sw}$
15    $S \leftarrow S - \{< v_i, v_j >\}$
16    $V \leftarrow V - \{v_j\}$
17    $E \leftarrow E - \{< v_m, v_n > | v_m = v_j \ or \ v_n = v_j\}$
```

Algorithm 1 summarizes how the process detects and processes redundancy rules in static rule sets. Firstly, we try to determine whether the relationship between r_i and r_j is consistent with Definition 3(Line 5). r_j can be considered redundant rule when a pair of rules $< r_i , r_j >$ is detected to conform to Definition 3. And next we need to check if the r_i exists. For a pair of rules $< r_i , r_j >$, we can use it to generate a probe packet p, and then p will be injected into the Switch to detect whether the rule is missing (Lines 6-8). If it is detected that the packet is not processed by this rule but by other rules, this means that rule r_i is missing in the Switch. In this case, removing redundancy rules prematurely causes some packets on the switch to be unprocessed. As a result, we need to add r_i to the Switch before remove r_j (Lines 10-12).

On the other hand, rule rj can be removed directly (Lines 14). Finally, the edges and vertices that related to rj in Gsw will be removed will be removed and the redundancy rule is saved as R.

Combining with the above parts, the process indicates that the detection and deletion complexity of the Algorithm 1 is $O(|E|) = O(|V|^2) = O(n^2)$.

3.3 Incremental Redundancy Detection

Incremental redundancy detection is the purpose of redundant detection with the minimum detection packet under the actual premise of continuous addition and deletion of stream tables. Therefore, we use the detection results of static rule sets in the previous section to avoid redundant detection.

Before expounding the algorithm of incremental flow table, we should introduce the concept of transitive closure firstly. For a binary relation *R* on the set *X*, there is a transitive closure which is the minimal transitive relation that includes R on the set X.

Algorithm 2: Incremental Redundancy Detection Algorithm

Input: Flow table FT_{ctr} and the verified dependency graph $G_{ver} = < V_{ver}, E_{ver} >$

Output: Dependency graph $G_{sw} = < V, E >$ of data plane and Set of redundancy rules *R*

```
1   $V \leftarrow V_{ver}$;
2   $E \leftarrow E_{ver}$
3   $S \leftarrow \varnothing$
4   $E_{TC} \leftarrow the\ transitive\ closure\ of\ E$
5   foreach $overlapping\ rule\ pair\ < v_i, v_j >\ in\ FT_{ctr}$ do
6   if $< v_i, v_j >\ or\ < v_j, v_i > \in E_{TC}$ then
7       continue
8   else
9       while $FT_{ctr}$ do
10          $(v_i, v_j) \leftarrow$ any overlapping-rule pair in $S$
11          if $Generate\Pr obe(v_i \cup v_j, v_i) = \varnothing$ then
12              $H \leftarrow \{v \mid \text{if} < v, v_i > \text{ and } < v, v_j > \in E\}$
13              $p \leftarrow generate\Pr obe((v_i \cap v_j, H)$
14              Inject $p$ to data plane
15              if $p\ does\ not\ follow\ v_i\text{'s}\ action$ then
16                  insert $r_i\ to\ FT_{ctr}$
17                  $R \leftarrow R \cup \{r_j\}$
18                  delete $r_j\ from\ FT_{ctr}\ and\ FT_{sw}$
19              else
20                  delete $r_j\ from\ FT_{ctr}\ and\ FT_{sw}$
21              $S \leftarrow S - \{< v_i, v_j >\}$
22              $V \leftarrow V - \{v_j\}$
23              $E \leftarrow E - \{< v_m, v_n > \mid v_m = v_j\ or\ v_n = v_j\}$
```

Algorithm 2 summarizes the process of detection process and deals with redundant rules in the case of incremental update. For each rule $r_i \in FT_{ctr}$, there are four case. (1) Rule r_i can be found corresponding vertices in the verified dependency graph G_{ver}, and a pair of overlapping rules containing rule r_i belongs to E_{TC}. E_{TC} is the transitive closure of E_{ver}. It means that rule r_i has been verified free of missing and redundant faults before incremental updates.(Lines 6-7) (2) The vertices corresponding to rule r_i belongs to the dependency graph G_{ver}, but there are some pairs of overlapping rule containing rule r_i

cannot found in E_{TC}. This suggests that the added rules and the verified rules from some new overlapping pairs, that need to be revalidated. (3) The vertices corresponding to rule r_i belong to the dependency graph G_{ver}, and the pair of overlapping rule containing rule r_i cannot found in E_{TC}. The newly added rules form overlapping pairs with each other, which also need to be revalidated. In the case of 3 and 4, similar as Algorithm 1, not only we should judge whether rule r_i is redundant, but also to confirm that the rule with overlapping relationship with r_i and have priority over it is not missing. (Lines 11-24) (4) The remaining rules, as outliers, are not considered.

The number of probe packets *p* generated by Algorithm 2 is limited by the size of *S*. More specifically, the size of *S* determines the number of iterations of the while-loop that control the number of probes. It is more difficult to estimate the size of *S* directly. But based on the number of redundant rule *k*, a loose upper bound of |*S*| can be estimated, by the calculation $|S| \leq 2kn$.

3.4 Improvement

When multiple interdependent rules are missing, the following condition will be occured. Suppose there are three rules *a,b,c*, and the priority of them are satisfied with the conditions in **Fig. 4(a)**, i.e., *c.p>a.p>b.p*. Meanwhile, the matching set $\{A,B,C \mid A=a.S, B=b.S, C=c.S\}$ that shown as **Fig. 4(b)**, i.e., $\{A,B,C \mid B \subset A,\ A \cap B \cap C \neq \varnothing\}$. When Action of rules *a* and *c* are consistent, it is possible to make errors if the rule is lost simply by determining whether the probe is transmitted to a predetermined port. In other words, according to the principle of rule lose detection, *GenerateProbe(c,a)* outputs a probe packet *p* that matches *c* but does not match rule *a*. But under the limitation of the above rules, if rule *c* is lost, the probe is not be discarded. Instead, it searches down to match the low-priority rule *b*. Then, rule *c* is considered not lost falsely because of the same Action of *a* and *b*. This leads to false negatives.

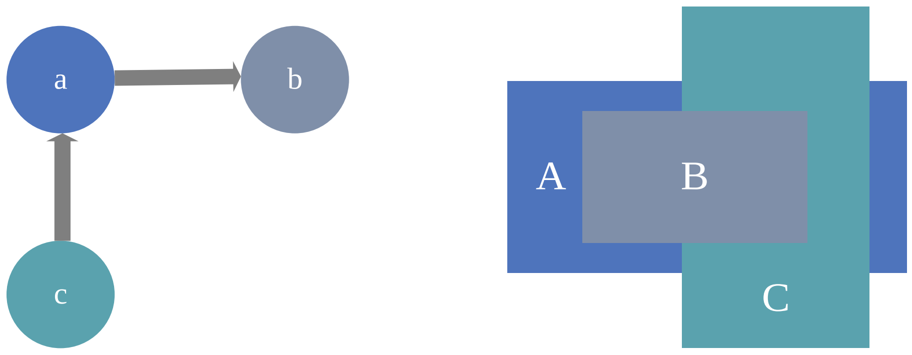


Fig. 5. Veen diagrams of rules

Faulty rules can be detected accurately by Algorithm 1 on the data plane without false negatives or false positives.

Proof: Without loss of generality, we suppose that r_i is the faulty rule to study. There are two cases in this proof, which are divided with whether some rules directly depends on r_i.

When there is no rule that dependent on r_i directly, it means that r_i is located in arc head or is isolated in the dependency graph. If r_i is isolated, there will be no false negative. If r_i is directly dependent on another rule r_j, it means that r_j has a higher priority. The probe that $GenerateProbe(v_i \cup v_j, v_i) = \varnothing$ (line 5) generates will not match r_i due to the higher priority of r_j. Under the premise that r_i can be considered obscure, even if r_i is missing, the false negative problem will not occur.

When some rules directly dependent on r_i, it means that r_i is in arc tail in the dependency graph *G*. The probe *p* generated by function $GenerateProbe(v_i \cap v_j, H)$ (line 7) in algorithm 1 can be matched $r_i \cap r_j$. So even if r_i is missing, it is not going to match with the lower level rules but going to match with r_j.

Algorithm 1 detected error r_i successfully in both cases. So there is no false negatives.

4. Performance evaluation or numerical results

In section 4, the efficacy and efficiency of algorithm 1 and algorithm 2 can be evaluated with MATLAB [19]. We can measure the effectiveness of the algorithm according to its accuracy in detecting the rule errors on the data plane. At the same time, efficiency can be measured by execution time. Comparing with efficacy, efficiency is more important than reporting statistics

4.1 Results on detection with different number of rule set

In this section, we test the algorithms with different number of on-Switch rules to evaluate.

Firstly, a set of matching fields of size 10, 20, 40, 80, 160, 320 and 640 are randomly generated using Matlab. And matching fields contains only 0, 1, and *. This procedure eliminates the possibility of matching fields that duplicate or contain relationships in each collection. Then scale each set of overlapping matching fields and merge the two parts. Through the above process, data sets with dimensions 14, 28, 56, 112, 224, 448 and 896 were obtained. After the dataset is initialized, it can be further processed according to algorithm 1. **Fig, 6** shows evaluation results.

Fig. 6(a): Overall execution time, the total execution time of algorithm 1 with the size of flow table. In section 3.2, when we calculate the time complexity of algorithm 1, we find that the time complexity increases exponentially with the size of the rule set. In the experimental results, this conclusion can be clearly observed with the abrupt change of the curve.

Fig. 6(b): Missing rule overhead: For the large flow tables, the detection time is less than the troubleshooting time. This is because probe packet generation time takes a leap, when the flow table size is over a certain size. When size of flow table increases, time of probe generation increases in all algorithms too. The missing rule detection time in the overall execution time also increases with the size of the flow table. This shows that as the flow table size increases, missing rule detection becomes a major overhead.

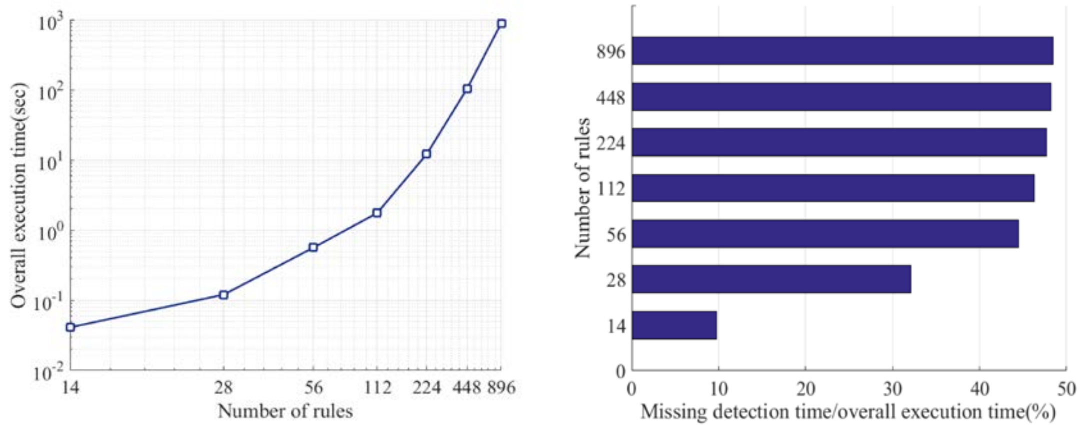


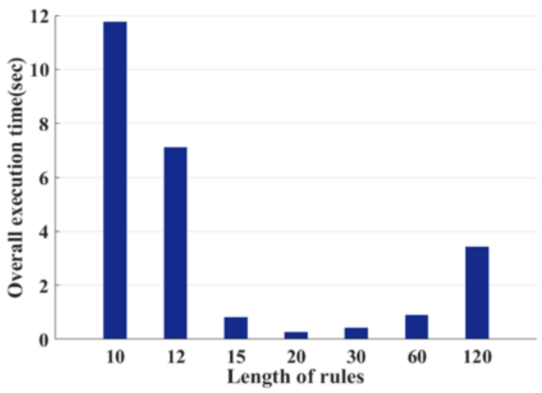
Fig. 6. (a) Overall execution time with different number of rule set

(b) Missing rule overhead/Overall execution time

4.2 Detection with different length of rule

Secondly, we utilize 224-rule flow table which include various length of rules and a numbers of overlapping rules to assess the time efficiency.

The construction of the data set is shown in section 4.1. The difference of the sets in this section is the length of the matching field, which is 10, 12, 15, 20, 30, 60 and 120 respectively. **Fig. 7(a)** and **Fig. 7(b)** respectively report evaluation results under rule length of 10-100 and rule overlap ratio of 14%-42%.



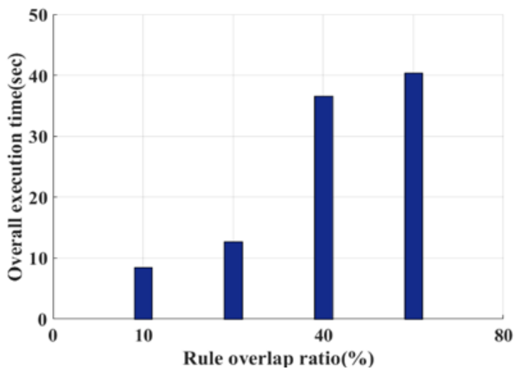


Fig. 7. (a) Overall execution time with different length of rule (b) Overall execution time with different overlap ratio

From the results of the experiment, we can derive three observations. Firstly, with the increase of rule length, the total detection time will first decreases and then increases. As **Table 1** shown, we calculated the ratio of the running time of missingdetection function to the total running time and the number of 1 in the adjacency matrix, which is the number of rules with dependencies. With the increase of the preset rule length, the probability of rule overlap decreases with the rule set of the same size. However, when the length increases to a certain extent, the required matching time will become longer. In OpenFlow protocol, rules and data packets are composed of dozens of fixed structures such as IP address and MAC address. In the subsequent improvement of the protocol, the composition and length of the matching domain can be adjusted appropriately by taking into account both identification and efficiency. Second, when constructing rules, the higher the overlap ratio of the selected rules, the longer the total running time.

Table 1. MissingDetection ratio and number of 1 with different length of rule.

Parameter	Length of rule						
	10	12	15	20	30	60	120
Missingdetetion	95%	94.3%	81.8%	35.6%	44.7%	55.5%	72.9%
Number of 1	2764	1466	511	118	96	96	90

4.3 Comparison

Compared with the original algorithm, the utilization rate of Switch memory is improved after adding the redundancy detection mechanism. **Table 2** shows the average memory optimization efficiency for datasets of different sizes.

Table 2. The average memory optimization efficiency with different number of rules

Number of rules	14	28	56	112	448	896
Ratio (%)	0.170%	0.173%	0.181%	0.181%	2.252%	5.201%

According to **Table 2**, the average memory optimization is more efficient with the increase of the size of rule set. Memory optimization efficiency is obtained by comparing the number of rules obtained through redundancy detection with the number of original rules. Therefore, for a large number of rule sets, this redundancy detection mechanism has a better effect. This is mainly because the probability of rule redundancy increases in a large number of rule sets.

5. Conclusion

In this paper, we analyze the demand for active processing mechanism of redundant rules in SDN and propose a series of algorithms. The algorithm can not only detect redundant rules, but also detect the absence of high-priority rules before deletion. The reliability of the algorithm is guaranteed. Moreover, we extend the algorithms by using variables that do not change when the forwarding policy is changed dynamically so that it can be used for dynamic rule updates. We believe it is very important for building a reliable SDN network to put forward the series of algorithms. The results verify that the proposed solution. However, the addition of this redundancy detection mechanism sacrifices part of the controller's time efficiency. Therefore, our next research work is to improve the detection mechanism and reduces its impact on time efficiency.

Reference

[1] Kreutz D, Ramos F, Verissimo P, et al, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp: 14-76, 2015. Article(CrossRef Link)

[2] Jarraya, Yosr, Taous Madi, and Mourad Debbabi, "A survey and a layered taxonomy of software-defined networking," *IEEE communications surveys & tutorials*, vol. 16, no. 4, pp: 1955-1980, 2014. Article(CrossRef Link)

[3] McKeown, Nick, et al, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp: 69-74, 2008. Article(CrossRef Link)

[4] SDN Architecture, Last Accessed 28 Jun 2018.

[5] Software-defined networking (SDN): Layers and architecture terminology, https://tools.ietf.org/rfc/rfc7426.txt. Accessed: 28 Jun 2018.

[6] Jain, Raj, and Subharthi Paul, "Network virtualization and software defined networking for cloud computing: a survey," *IEEE Communications Magazine*, vol. 51, no. 11, pp: 24-31, 2013. Article(CrossRef Link)

[7] Li, Yong, and Min Chen, "Software-defined network function virtualization: A survey," *IEEE Access*, vol. 3, pp: 2542-2553, 2015. Article(CrossRef Link)

[8] Bizanis, Nikos, and Fernando A. Kuipers, "SDN and virtualization solutions for the Internet of Things: A survey," *IEEE Access*, vol. 4, pp: 5591-5606, 2016. Article(CrossRef Link)

[9] Yin B, Wei X, "Communication-Efficient Data Aggregation Tree Construction for Complex Queries in IoT Applications," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp: 3352-3363, 2018. Article (CrossRef Link)

[10] Li H, Gao G, Chen R, et al, "The influence ranking for testers in bug tracking systems," *International Journal of Software Engineering and Knowledge Engineering*, vol. 29, no. 01, pp: 93-113, 2019. Article(CrossRef Link)

[11] More Avinash, Vijay Raisinghani, "A node failure and battery-aware coverage protocol for wireless sensor networks," *Computers & Electrical Engineering*, vol. 64, pp: 200-219, 2017. Article(CrossRef Link)

[12] Wang, Jin, et al, "An enhanced PEGASIS algorithm with mobile sink support for wireless sensor networks," *Wireless Communications and Mobile Computing*, 2018. Article(CrossRef Link)

[13] Liao, Zhuofan, Junbin Liang, and Chaochao Feng, "Mobile relay deployment in multi-hop relay networks," *Computer Communications*, 112, pp: 14-21, 2017. Article(CrossRef Link)

[14] Cao, Dun, et al, "A robust distance-based relay selection for message dissemination in vehicular network," *Wireless Networks*, pp. 1-17, 2018. Article(CrossRef Link)

[15] Gong, WeiBing, et al, "An adaptive path selection model for WSN multipath routing inspired by metabolism behaviors," *Science China Information Sciences*, vol. 58 no.10 pp: 1-15, 2015. Article(CrossRef Link)

[16] Li W, Meng W, Kwok L F, "A Survey on OpenFlow-based Software Defined Networks: Security Challenges and Countermeasures," *Journal of Network and Computer Applications*, 2016. Article(CrossRef Link)

[17] Ahmed, Ali, Kyungbae Park, Sanghyeon Baeg, "Resource-efficient SRAM-based ternary content addressable memory," *IEEE Transactions on Very Large Scale Integration Systems* vol. 25, no. 4, pp: 1583-1587, 2016. Article(CrossRef Link)

[18] Jia, Xuya, et al, "Intelligent path control for energy-saving in hybrid SDN networks," *Computer Networks*, vol. 131, pp: 65-76 2018. Article(CrossRef Link)

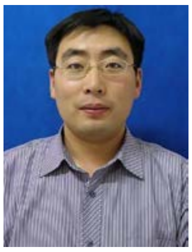
[19] Zhao H, Liu H, Xu J, et al, "Performance prediction using high-order differential mathematical morphology gradient spectrum entropy and extreme learning machine" *IEEE Transactions on Instrumentation and Measurement*, 2019. Article(CrossRef Link)

Jian Su has been a lecturer in the School of Computer and Software at the Nanjing University of Information Science and Technology since 2017. He received his PhD with distinction in communication and information systems at University of Electronic Science and Technology of China in 2016. He holds a B.S. in Electronic and information engineering from Hankou university and an M.S. in electronic circuit and system from Central China Normal University. His current research interests cover Internet of Things, RFID, and Wireless sensors networking. He is a member of IEEE and a member of ACM.

Shiming Yu received the Bachelor of Engineering degree from Binjiang College, Nanjing University of Information Science & Technology, Nanjing, China, in 2019. He is now a master of the School of Computer and Software, Nanjing University of Information Science & Technology. His research interests include Wireless sensor network and Mobile edge computing.

RuoyuXu received the B.E. degree at Nanjing University of information science and technology, Nanjing, China, in 2018. She is currently pursuing the M.S. degree in the Computer Science and Technology from Nanjing University of information science and technology. Her research interests include RFID technology, anti-collision protocols.

Baowei Wang is a professor. He received the Bachelor degree and Ph.D. degree in computer science and technology from Hunan University in 2005 and 2011 respectively. Since 2011, he has been working at Nanjing University of Information Science and Technology. His research interests are mainly in trusted data transaction, Internet of Things, and data security etc.



Jiuru Wang received the M.S. degree from Anhui University of Science and Technology, Huainan, China, in 2009, and Ph.D. degree from Harbin Engineering University, Harbin, Heilongjiang in 2013. Now, He is working at Linyi University. His research interests mainly include information security, key management, and block chain application.