Research Article

Integrated Packet Classification to Support Multiple Security Policies for Robust and Low Delay V2X Services

Jaehyeong Wee and Wooguil Pak

Department of Computer Engineering, Keimyung University, Daegu 704-701, Republic of Korea

Correspondence should be addressed to Wooguil Pak; wooguilpak@kmu.ac.kr

Received 19 June 2018; Revised 28 September 2018; Accepted 3 October 2018; Published 1 November 2018

Guest Editor: Safdar H. Bouk

Copyright © 2018 Jaehyeong Wee and Wooguil Pak. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

One of the key applications in the 5G system is Vehicle-to-Everything (V2X). Ultra-low delay communication is essential for the safety of users and pedestrians in V2X. However, as sophisticated and various cyberattacks are increasing, it becomes hard to satisfy low delay constraints. To protect networks from such attacks, even single network security equipment provides multiple security functions, resulting in the inevitable additive delay in packet processing. In this paper, we suggest a new packet classification paradigm to resolve this issue. The proposed algorithm integrates multiple policy rule-sets into a single rule-set and classifies incoming packets using the integrated rule-set. Thus, it has a unique feature providing high classification performance regardless of the number of security policies. Through extensive performance evaluations, we confirm that the performance improvement is also increased with the total rule-set number increasing without the significant overhead of memory cost. We expect that it will mitigate the delay issue of existing network equipment for upcoming services such as V2X.

1. Introduction

Vehicle-to-Everything (V2X) service is one of the most promising applications in the 5G system. It frequently exchanges information between drivers, pedestrians, vehicles, and transportation infrasystems [1–6], and the information should be delivered with low delay and high reliability for the safety of involved persons.

Modern cyberattacks have become more sophisticated and diverse, and as a result, security functions installed in modern security equipment also become more complex and various. For protecting networks from various cyberattacks, single multifunction network equipment has been introduced [7]. For example, unified threat management (UTM) supports multiple rule-sets using multiple policy tables as shown in Figure 1. Such integrated network equipment has advantages in security but disadvantages in strict delay requirements of V2X. Each security policy is implemented by complicate packet classification that searches a matching rule with the highest priority by comparing each field of every rule with the incoming packet header. As the integrated equipment should independently perform packet classification for each policy rule-set, the classification cost increases as the number of rule-sets increases [8–14]. Multiple classifications are a bottleneck of network performance, especially in terms of the delay [15–20]. Therefore, the high performance and scalable packet classification is essential for supporting V2X.

In this paper, we propose a new packet classification algorithm that has a distinct feature against other competitors. Although most existing classification algorithms suffer from deteriorated performance as the total rule-set number increases, the proposed algorithm can achieve high classification performance regardless of the total rule-set number. It can effectively support reliable and low delay V2X services. Figure 2 shows the overall architecture of the software-defined networking (SDN) for V2X. Packet classification is a basic function of OpenFlow controller and SDN switch.

To increase the performance of packet classification, high-end SDN switches adopt the expensive hardware-based solution. However, the OpenFlow controller usually adopts software-based packet classification since hardware solution cannot achieve high flexibility to support various security requirements from customers. This algorithm targets OpenFlow controllers and software-based SDN switches to reduce the burden of packet classification. It can play a very
important role in them to provide high performance and high security, simultaneously.

The remainder of this paper is organized as follows: Section 2 briefly presents related work, and the motivation of this research is explained in Section 3. In Section 4, the proposed algorithm is described in detail. The performance evaluation results are compared with those of competitors in Section 5. Finally, Section 6 concludes.

2. Related Work

Although many factors can be used to evaluate the performance of packet classification algorithms, packet classification speed and memory requirements are most important factors. However, most algorithms cannot support high classification speed with low memory requirement.

Packet classification is classified into hardware- and software-based approaches [21–27]. Hardware-based packet classification can achieve very high classification speed that is impossible for the software-based one. Most modern network equipment adopts hardware packet accelerators to provide 100 Gbps performance with multiple rule-sets. However, the hardware should be redesigned to satisfy the various requirements of users such as adding a new field in the rule structure. Moreover, hardware-based solutions usually adopt expensive memory called ternary-content addressable memory (T-CAM) for classification. Since supported rule-set size is determined by the size of T-CAM, it costs very high to support large rule-sets.

The strongest advantage in the software-based approach is flexibility. If the structure of field should be changed, it can be easily supported by modifying software. Another merit of the software-based approach is cost. If larger rule-set is needed, the user can increase the rule-set capacity of the network equipment by just adding much cheaper dynamic random-access memory (DRAM) compared to T-CAM.

Well-known algorithms belonging to the software-based approach are exhaustive search, cross-producing-based classification, tuple space search, and decision tree-based algorithms [21]. Now, we will briefly describe each software-based algorithm.

Exhaustive search linearly compares each rule with keys from highest to lowest priority to find the matching rule. Due to the searching procedure, the packet classification performance is degraded as the rule-set size increases. However, it requires smallest memory among all packet classifications and supports very fast update. Most of all, it can be easily implementable. As a result, it is suitable for a system with a small rule-set.

Cross-producing-based classification independently performs searching for each field, and it merges intermediate results [28–33]. This procedure is repeated until the final matching rule is found. It is one of fastest classification algorithms but it requires a huge amount of memory and time to build a classification table. Since it cannot support incremental update, it needs to rebuild entire table whenever a rule-set is updated. Although it has such critical weaknesses, it can support the classification performance almost similar to that of the hardware-based approach. Therefore, a lot of research is still going on to improve the weaknesses.

Tuple space search probes each sub-rule-set called tuple space to find the matching rule [34–37]. A tuple is defined by combination of each prefix length for five tuples, and the set of tuples are called tuple space. Since each rule of a rule-set belongs to only one of the tuples, tuple space has a good scalability in terms of a rule-set size. Although it achieves a moderate classification performance, it supports fast update, i.e., inserting or deleting a rule. Therefore, it has been adopted in Open vSwitch [38]. However, the classification performance is decreased proportional to the number of tuples, thus requiring further research to improve the performance.

Decision tree-based algorithm recursively chooses a child node according to the predefined policy on decision tree built based on a rule-set until it reaches a leaf node [39–47]. If it reaches, it searches the matching rule with the highest priority among all rules stored in the leaf node. The overall classification performance is known to have log complexity in terms of the rule-set size.

Decision tree-based algorithm provides a comparable classification performance with that of cross-producing-based classification algorithm but requires much smaller memory size. Thus, the decision tree-based algorithm is one of the most actively researched algorithms at present. When a decision tree-based algorithm partitions a rule-set into multiple sub-rule-sets, partitioning criteria is controlled by two factors: space factor, the maximum ratio of the sum of all rules belong to all sub-rule-sets to the original rule-set size, and binth, the maximum allowed rule size in the leaf node. Hence, the classification performance and the table size can be adjusted according to the requirements of applications.

Large space factor increases partitioning number but decreases the height of a decision tree, resulting in fast classification performance. However, the total number of duplicate rules is increased, and therefore, generating a large decision tree. On the other hand, large binth reduces partitioning number, so the tree size is decreased but the searching cost in the leaf node increases, and thus, providing low classification performance.

Well-known algorithms belonging to the decision tree-based approach are HiCuts and HyperCuts [39, 40]. Although they provide high classification performance, they still suffer from a large decision tree due to significant rule duplications. Recently, EffiCuts was introduced to decreasing rule duplications [41]. EffiCuts is based on HyperCuts but groups rules by fields with wildcard and generates a separate tree for each group. This approach significantly reduces rule duplications, so the total tree size is greatly decreased. However, separate tree deteriorates the classification performance. As a mitigation, trees with similar wildcard characteristics can be merged to increase classification performance while the overall tree
size is almost the same. EffiCuts is known to support fast updating [48, 49].

We will describe the operation of EffiCuts in detail. At first, EffiCuts splits the total rule-set into some predefined categories according to how many wildcard field each rule contains, where wildcard field is a field on which the rule has a large matching range, typically at least 50% of the total range of the field. For 5-tuple rule-set, we have four cases as follows:

(i) Category 1: four wildcard field rules
(ii) Category 2: three wildcard field rules
(iii) Category 3: two wildcard field rules
(iv) Category 4: one or zero wildcard field rules

For example, assuming that matching ranges of a rule for source IP, destination IP, source port, destination port, and protocol are ANY, ANY, 0 to 32768, 80, 0 to 128, it has four wildcard fields except for destination port, and therefore belonging to Category 1. Since each category contains similar rules only, EffiCuts builds a decision tree for sub-rule-set belonging to the same category and reduces replicated rules during building a decision tree. Although EffiCuts generates multiple decision trees, the total tree size is very small compared to the original HyperCuts. However, the number of decision tree affects the total classification performance. To reduce the number of trees, EffiCuts merges similar categories. This tree merging process increases the total tree size but it can still avoid excessive replication of rules. By doing so, EffiCuts achieves high classification performance and low memory requirement, simultaneously.

Table 1 summarizes each feature of packet classification algorithms.

3. Motivation
As shown in Table 1, the software-based approach consumes much memory to achieve high classification performance. However, high complexity of memory requirement results in low scalability in terms of rule-set size. Although decision tree-based algorithms have a high complexity of memory requirement, i.e., $O(N^D)$, latest decision tree algorithms show very low memory requirements, where $N$ and $D$ denotes the total dimension number and the rule-set size, respectively.

To verify the memory requirement, we performed the following experiment. We synthesized multiple firewall rule-sets whose size is from 20K to 100K using ClassBench [50]. Then, we built the total decision tree and calculated the ratio of the tree size to the rule-set size for each rule-set, where space factor and binth were configured to the best values. Figure 3 shows the experimental results obtained from EffiCuts. EffiCuts shows almost the same ratio regardless of the rule-set size, which means EffiCuts achieves almost $O(N)$ for memory requirement. Thus, it can decrease the decision tree size by 100 times for 100,000 rules compared to HiCuts or HyperCuts [41].

Figure 4 shows the ratio of the average memory access number and the rule-set size on the same configuration.
We synthesized the packet data using each rule-set and searched the decision tree to find every packet in the data. We counted the total number of memory accesses during searching process and obtained the ratio of the total number and the total packet number. As the rule-set size increases, the number of memory access for EffiCuts also increases in Figure 4. However, the ratio of the access number to the rule-set size decreases as shown in Figure 4. From Figures 3 and 4, we can finally find two characteristics as follows:

**Characteristic 1**: \(M(R_1) + M(R_2) > M(R_1 + R_2)\), where \(M(R)\) is the average memory access number for rule-set \(R\).

**Characteristic 2**: \(S(R_1) + S(R_2) \equiv S(R_1 + R_2)\), where \(S(R)\) is the size of decision tree for rule-set \(R\).

For example, we can see that \(M(T_{20K} + M(T_{40K}) > M(T_{20K}) + T_{40K}) \equiv M(T_{60K})\) from Figure 4 and \(S(T_{20K} + S(T_{40K}) \equiv S(T_{20K} + T_{40K}) \sim S(T_{60K})\) from Figure 3, respectively, where \(T_n\) denotes the testing rule-set with a size of \(n\) used in the experiment and where \(K\) means 1,000.

Until now, existing research studies for packet classification focus on classification with a single rule-set. However, network systems with multiple rule-sets become popular, and fast classification algorithm oriented on a single rule-set has limitation to achieve high performance for multiple rule-sets. Thereby, it is required to consider multiple rule-sets for designing high performance classification algorithms. Hence, Characteristics 1 and 2 suggest a new guideline for developing packet classification algorithms. According to Characteristic 1, if a system has multiple rule-sets, it is advantageous to integrate them into one rule-set to construct a decision tree for improving classification speed. Characteristic 2 also implies that the size of the decision tree for integrated rule-sets is not larger than the sum of sizes for each decision tree for all rule-sets.

We finally conclude that packet classification algorithm based on integrated rule-sets has many advantages and suggest a new classification algorithm utilizing the features of integrated rule-sets.

### 4. Proposed Algorithm

The proposed algorithm performs packet classification using an integrated rule-set that combines all rule-sets in a system. At first, we briefly show the features of the proposed algorithm, and then, describe the algorithm in detail. For simple explanation, we assume that the rule consists of five fields but it can be easily extended to more field cases.

#### 4.1. Features of the Proposed Algorithm

**4.1.1. Minimized Classification Cost.** The proposed algorithm can complete total packet classification for all rule-sets with one search. Therefore, it can minimize the increased overhead due to the repetitive classification. Since it can maintain the high packet classification performance regardless of the number of rule-sets, it is very important feature of the proposed algorithm.

**4.1.2. Early Packet Drop.** Integrated rule-set has not only advantage to decrease classifying overhead but also to remove unnecessary classifying. For example, Figure 5 shows existing and proposed packet classifications. Assume that an incoming packet is allowed by rule-sets 0 to \(k-1\), but it is rejected by rule-set \(k\). In this case, packet classifications for rule-sets 0 to \(k-1\) are eventually unnecessary since the packet cannot be forwarded due to rule-set \(k\). However, packet classification for each rule-set is performed in sequence, so it cannot avoid
the unnecessary classifications for rule-sets 0 to \( k - 1 \) for existing classification. For the proposed classification algorithm, all rule-sets are integrated into one larger rule-set, making almost the same effect as searching multiple rule-sets, simultaneously. Thereby, the problem of existing classification is mitigated in the proposed one.

4.2. Building Decision Tree. The proposed algorithm builds a decision tree using EffiCuts after merging each rule-set into a large rule-set. However, it needs unique procedure called “fast rule skipping” and “early drop marking” in each leaf node for improving the searching performance.

4.2.1. Fast Rule Skipping. The proposed algorithm requires an additional table called “rule-set starting index table” to store all indexes of the first rule in each rule-set. If we reach a leaf node during traversing the tree, we should find matching rules for each rule-set. Original EffiCuts linearly searches matching rules, so it will take a long time. To increase searching performance, we need to skip unvisited rules in rule-set \( k \) and go to the next rule-set \( k + 1 \) when we find matching rule in the rule-set \( k \). It is called “rule skipping.” For example, if we find a matching rule \( r_3 \) for firewall rule-set in the leaf node as shown in Figure 6, we do not need to check rules \( r_1 \) and \( r_5 \) anymore. In this case, we can find the index number for firewall rule-set, i.e., 4, and skip \( r_3 \) and \( r_5 \). Thus, we can directly start searching the matching rule for firewall rule-set.

4.2.2. Early Packet Drop Marking. Assume that we build a node of a decision tree. Each node corresponds to disjoint hypercube searching space. Let us define some notations for describing “early packet drop marking”:

\[
\begin{align*}
(i) \Delta_v &: \text{the searching space for node } v \\
(ii) k &: \text{the total rule-set number} \\
(iii) n^p_v &: \text{the total number of rules for rule-set } p \text{ belong to node } v \\
(iv) r^p_v[i] &: i\text{th rule of rule-set } p \text{ belonging to node } v \text{ when the rules are sorted in the order of decreasing priority} \\
(v) S(\cdot) &: \text{a set of all matching keys with given rule} \\
(vi) S^p_v[i] &: S(r^p_v[i]) \cap \Delta_v
\end{align*}
\]

We define \( D^p_v(j) \) as a set of all keys matching with action “drop” from first to \( i \)th rules of the rule-set \( p \) belonging to node \( v \) when the rules are sorted in the order of decreasing priority. Then, it is recursively defined as

\[
D^p_v(j) = \begin{cases} 
D^p_v(j - 1) \cup S^p_v[j] & \text{if action of } r^p_v[i] \text{ is "drop"}, \\
D^p_v(j - 1) - S^p_v[j] & \text{otherwise},
\end{cases}
\]

where \( D^p_v(0) = \emptyset \).
Assume that an incoming packet is, respectively, matched with $r_{p}^{i}[i]$ and $r_{p}^{j}[j]$ for rule-sets $i$ and $j$, where the actions of $r_{p}^{i}[i]$ and $r_{p}^{j}[j]$ are “allow” and “drop.” In this case, the packet should be dropped by the rule-set $j$. If all packets matching with $r_{p}^{i}[i]$ are always matched with rules in other rule-set with action “drop,” it will be very helpful to know that the packet will be dropped for increasing classification performance. This idea can be generalized as follows.

If $S_{p}^{i}[i] \subset D_{p}^{i}(n_{p}^{i})$, any packet matched with $r_{p}^{i}[i]$ in node $v$ is dropped, where $p \neq q$. Thus, while building a node $v$, the proposed algorithm finds any rule $r_{p}^{i}[i]$ s.t. $S_{p}^{i}[i] \subset D_{p}^{i}(n_{p}^{i})$, where $p < q$, and mark $r_{p}^{j}[j]$ with “early packet drop.” If a packet matches with a rule that has a mark “early packet drop” during searching, the searching procedure is finished and the packet is dropped. This “early packet drop marking” significantly increases the performance.

4.3. Proposed Packet Classification Performance Analysis.
The proposed algorithm merges multiple rule-sets into an integrated one and constructs a decision tree. Now, we will show numerical analysis results for our algorithm. Assume that rules are homogeneous, and the decision tree is perfectly balanced $B$-tree for easy analysis. Let us define some notations as follows:

(i) $k$: the total rule-set number.
(ii) $B$: binth, the maximum allowed rule size in the leaf node.
(iii) $c$: the child number of each node. For easy analysis, we assume that $c$ is fixed.
(iv) $s$: space factor. The maximum ratio of the sum of all rules belongs to all sub-rule-sets to the original rule-set size.
(v) $N$: the total rule number for each rule-set size. We also assume that $N$ is fixed.

4.3.1. Total Packet Classification Cost Analysis. Assume that EffiCuts has $N$ rules in a root node. If it has $c$ child nodes and the space factor is $s$, the first level child node has at most $sN/c$ rules. In a similar way, we can calculate the rule number in the leaf node as follows:

$$\frac{s^{h}N}{ch} \leq B, \quad (2)$$

and it should be equal to or less than $B$, where the height of the decision tree is $h_{\text{EffiCuts}}$. From (2), we can find the height as follows:

$$h_{\text{EffiCuts}} \equiv \log_{(c/s)} \left(\frac{N}{B}\right). \quad (3)$$

For the proposed algorithm, we can similarly obtain the height as

$$h_{\text{proposed}} \equiv \log_{(c/s)} kN/B. \quad (4)$$

Thus, the total packet classification cost for EffiCuts is approximated as follows:

$$kh_{\text{EffiCuts}} \equiv k \cdot \log_{(c/s)} \frac{N}{B} \quad (5)$$

Now, we calculate the difference between two costs:

$$kh_{\text{EffiCuts}} - h_{\text{proposed}} = k \cdot \log_{(c/s)} \frac{N}{B} - \log_{(c/s)} kN/B,$$

$$= (k - 1) \cdot \log_{(c/s)} \frac{N}{B} - \log_{(c/s)} k > 0 \cdot k > 1 \quad (6)$$

Then, we can conclude that the proposed algorithm can always provide higher classification performance than EffiCuts.

4.3.2. Total Decision Tree Size Analysis. Since we assume that the decision tree is a perfectly balanced $B$-tree, EffiCuts requires at most $2^{h_{\text{EffiCuts}}}/(k - 1)$ nodes for one rule-set, so the total number of nodes is $k(2^{h_{\text{EffiCuts}}}/(k - 1))$. Similarly, the proposed algorithm requires $2^{h_{\text{proposed}}}/(k - 1)$. If we calculate the difference between two node sizes,

$$2^{k \log_{(c/s)} \frac{N}{B}} - 2^{k \log_{(c/s)} \frac{N}{B} - 1} = 2^{k \log_{(c/s)} \frac{N}{B} - 1} - \left(2^{k \log_{(c/s)} \frac{N}{B} - 1} - 1\right)$$

$$= 2^{k \log_{(c/s)} \frac{N}{B} - 1} - 2^{k \log_{(c/s)} \frac{N}{B} - 1} - \left(2^{k \log_{(c/s)} \frac{N}{B} - 1} - 1\right)$$

$$= 2^{k \log_{(c/s)} \frac{N}{B} - 1} - 2^{k \log_{(c/s)} \frac{N}{B} - 1}.$$

Since $2^{k \log_{(c/s)} \frac{N}{B}} - k > 0$, if $1 < (c/s) < 2$, the proposed algorithm creates larger tree than EffiCuts, where $1 < (c/s) < 2$. However, we found that $c \gg s$ for most nodes in a decision tree. It means that the proposed algorithm builds a tree which size is not significantly large compared to EffiCuts.

5. Performance Evaluation
We compared the performance of the proposed algorithm with EffiCuts. Since EffiCuts is almost an unique decision tree-based packet classification algorithm to support fast classification and large rule-set size simultaneously, we choose it as a competitor. We measured average and worst case classification memory access numbers, and decision tree size using the optimal bucket size and space factor for each evaluation. Since the average classification memory access number defines the overall performance of the network equipment, it is the most important metric. The worst case classification memory access number represents the maximum queuing delay required to guarantee in-order packet forwarding. Last, the total decision tree size is also a critical factor to represent the scalability in terms of rule-set size. Considering modern network traffic increases...
exponentially and rule-set becomes larger and more complicated to support various services, we choose these three metrics for performance evaluation.

For evaluating performance of the proposed algorithm, multiple rule-sets are needed. Thus, three rule-set types such as FW, ACL, and IPC were generated using Classbench [50]. Each rule consists of five tuples, and the rule-set size was set to 20K to 100K increasing by 20K where K means 1,000.

Thus, the integrated rule-set size was to 60K to 300K. For each evaluation, binth and space factor were set to the optimal values, i.e., 30 and 2, respectively.

Figure 7 shows the average classification performance in terms of the average number of memory accesses according to the size of the total integrated rule-set. The proposed algorithm achieves about 2.5 times lower memory access number regardless of the rule-set size compared to EffiCuts. It is almost similar to the memory access number of each rule-set. It confirms that integrated rule-set has many benefits to increase the classification performance.

Figure 8 shows results for the worst case packet classification performance. The proposed algorithm decreases the memory access number by 2.2 times regardless of the total rule-set size compared to competitor. Although the improvement is slightly smaller than that for average memory access number, it also confirms that the proposed algorithm is very effective to increase the packet classification performance for the worst case.

The worst case performance actually affects the packet processing delay since most network equipment should guarantee that the packets are processed in sequence, keeping that the orders of incoming and outgoing packets are the same. As the worst classification performance is improved, it can efficiently provide in-order packet forwarding while minimizing packet queuing delay.

Figure 9 shows the comparison results between proposed and EffiCuts for decision tree size. As mentioned earlier as Characteristic 2, the proposed algorithm generates a decision tree whose size about 20% is larger than that of EffiCuts for 300K rules. Therefore, the proposed algorithm does not suffer from significantly increased tree size caused by rule-set integration.

Although we used three rule-sets for most performance evaluations, it is also important to investigate the performance as the rule-set number increases for evaluating scalability in terms of the rule-set number. Figure 10 shows the ratio of the results of EffiCuts to those of the proposed algorithm for the memory access and the decision tree sizes as the rule-set size increases from 1 to 10.

As shown in Figure 10, the decision tree size of the proposed algorithm is almost the same to that of EffiCuts.
regardless of rule-set size but the memory access size is decreased fast compared to EffiCuts. For 10 rule-sets, the proposed algorithm achieves 3 times higher classification performance while the decision tree size is just increased by 10%. Thus, we can see that our proposed algorithm can provide high classification performance without any cost of decision tree size.

6. Conclusions

In this paper, we proposed a new packet classification algorithm to achieve high packet classification performance without significant increasing of memory requirement. It can be adopted in modern high performance network equipment that use various classification rule-sets such as routing, switching, QoS, and other rule-sets. Existing network equipment with multiple rule-sets independently perform classification for each rule-set, thus resulting in deteriorated performance as the rule-set number increases. Our algorithm combines each rule-set and achieves high performance that cannot be provided by existing algorithms. We expect that it will help to enable robust and low delay V2X services in modern networks.

Data Availability

The source code data used to support the findings of this study are currently under embargo while the research findings are commercialized. Requests for data, 12 months after publication of this article, will be considered by the corresponding author.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

References


