technologies

MDPI

Article

Packet Classification Using TCAM of Narrow Entries

Hsin-Tsung Lin , Wei-Han Pan and Pi-Chung Wang *

Department of Computer Science and Engineering, National Chung Hsing University, Taichung 402202, Taiwan; aries77329@gmail.com (H.-T.L.); de606430y@gmail.com (W.-H.P.)
* Correspondence: pcwang@nchu.edu.tw; Tel.: +88-6-4228-40497

Abstract: Packet classification based on rules of packet header fields is the key technology for enabling software-defined networking (SDN). Ternary content addressable memory (TCAM) is a widely used hardware for packet classification; however, commercially available TCAM chips have only limited storage. As the number of supported header fields in SDN increases, the number of supported rules in a TCAM chip is reduced. In this work, we present a novel scheme to enable packet classification using TCAM with entries that are narrower than rules by storing the most representative field of a ruleset in TCAM. Due to the fact that not all rules can be distinguished using one field, our scheme employs a TCAM-based multimatch packet classification technique to ensure correctness. We further develop approaches to reduce redundant TCAM accesses for multimatch packet classification. Although our scheme requires additional TCAM accesses, it supports packet classification upon long rules with narrow TCAM entries, and drastically reduces the required TCAM storage. Our experimental results show that our scheme requires a moderate number of additional TCAM accesses and consumes much less storage compared to the basic TCAM-based packet classification. Thus, it can provide the required scalability for long rules required by potential applications of SDN.

Keywords: packet classification; ternary content addressable memory; software defined networks; OpenFlow

Citation: Lin, H.-T.; Pan, W.-H.; Wang, P.-C. Packet Classification Using TCAM of Narrow Entries. *Technologies* **2023**, *11*, 147. https://doi.org/10.3390/technologies11050147

Academic Editors: Abdellah Chehri and Valeri Mladenov

Received: 7 April 2023
Revised: 7 October 2023
Accepted: 16 October 2023
Published: 19 October 2023

1. Introduction

Software-defined networking (SDN) [1] implements the control plane of a network in one or more server-based controllers. The controller performs the functions of the control plane and manages a network in a centralized manner. Network administrators can deploy various applications (or services) through controllers. Currently, OpenFlow [2] is the most widely deployed south-bound SDN protocol [1]. The data plane of OpenFlow relies on the operations of packet classification based on packet header fields. As a packet arrives, the packet classifier extracts inspected fields from packet header fields to form a search key for comparison with a list of rules. Each rule is assigned a priority value and has a designated action to process the matching packet. If there is more than one rule matching the search key, the packet classifier selects the rule with the highest priority [3]. While numerous network applications rely on the function of packet classification, both the scalability and performance of packet classifiers are crucial to the development of potential network applications.

With OpenFlow, rules are generated by the applications in controllers and installed on OpenFlow-enabled switches [4]. To increase compatibility and applicability, OpenFlow further expands the number of supported header fields from twelve in OpenFlow 1.0 to more than forty in OpenFlow 1.5 [2], where each rule of IPv4 can have up to 773 bits. The deployment of IPv6 could further increase the maximal rule length to nearly one thousand bits. Because the programmability of OpenFlow is limited by predefined protocol header fields, the programming protocol-independent packet processors (P4) language [5] has been proposed for data plane programmability. Protocol independent switch architecture

(PISA) is the data plane programming model for P4 [6]. Therefore, a packet classifier of PISA must be able to support an arbitrary number of header fields [7].

Ternary content addressable memory (TCAM) has been widely used to achieve high-performance packet classification. It is a viable option for PISA implementation to match wildcard rules at line rate [8–10]. TCAM can store ternary strings because each of its cells has three states: 0, 1, and "don't care". Moreover, TCAM compares all entries in parallel to yield the first matching entry in one access. Thus, it is suitable for performing packet classification by converting rules to ternary strings and storing them in a descending order of priority [11]. Commodity TCAM chips usually support several predefined entry widths, while additional clock cycles are required to access longer TCAM entries. For example, commodity TCAM chips can be configured to have one of the following entry widths: 72, 144, 288, or 576 bits [12], where accessing longer TCAM entries takes additional clock cycles [13]. The storage of long rules in TCAM reduces the number of available TCAM entries. Several drawbacks of TCAM, including limited capacity, high cost, and high power consumption, are directly related to TCAM chip size [14–17]. For example, performing one single access on a large TCAM chip with 72 megabits (Mb) consumes 1047.9 nanojoules (nJ), whereas on a 1 Mb TCAM chip it consumes only 34.5 nJ [17]. Moreover, larger TCAM chips have much longer access latency [17]. Therefore, packet classification for long rules using TCAM is a challenging task.

In this paper, we attempt to improve storage efficiency for TCAM-based packet classification without degrading performance in terms of speed. First, we discuss the challenges of TCAM-based packet classification for rules longer than TCAM entries. Then, we present a novel algorithm for searching long rules using narrow TCAM entries. Our scheme stores the most representative field of a ruleset in TCAM for indexing the complete rules stored in SRAM. It is based on our previous work in [18] involving multimatch packet classification, a technique for yielding all the matching entries in TCAM for any search key, as not all rules can be distinguished using the same field. Although our scheme may require additional TCAM accesses as a tradeoff, it consumes much less storage than the basic implementation of TCAM-based packet classification. Our scheme further reduces the number of TCAM accesses by intelligently selecting fields stored in TCAM and employing supplementary data structures in SRAM. We propose several approaches to eliminate redundant TCAM accesses. The simulation results show that our scheme can effectively search long rules with a moderate number of additional TCAM accesses. Moreover, our scheme reduces the required TCAM storage to enable faster accesses with lower cost and energy consumption. As a result, the scalability of TCAM-based packet classification for long rules is drastically improved.

The remainder of this paper is organized as follows. Section 2 reviews previous work related to our scheme. Section 3 discusses the limitations of conventional packet classification to highlight our idea and the challenges. Our algorithm to enable packet classification using narrow TCAM entries is presented in Section 4, along with storage of the data structure in SRAM to eliminate redundant TCAM accesses. Section 5 presents the performance evaluation results for our scheme. Finally, concluding remarks are provided in Section 6.

2. Related Work

2.1. Many-Field Packet Classification Based on Software

Because OpenFlow supports numerous fields, many-field packet classification is crucial to the feasibility of OpenFlow. The existing packet classification algorithms can be either based on software or hardware. Software-based solutions are attractive due to their flexible implementations. Hsieh and Weng performed many-field packet classification based on selected bits [19]. Another decomposition scheme converts the original field specifications into primitive ranges to be stored in trees to lower the search complexity [20]. Several approaches have aimed to reduce the storage requirement, such as rule reduction [21] and efficient representation [22,23]. TupleMerge reduces the number of hash tables for storing

rules by omitting bits from the rules [7]. PartitionSort divides a ruleset into several sortable partitions, with the rules of each partition are stored in an interval tree [24]. Liang et al. used deep reinforcement learning to improve the efficiency of decision trees [25].

2.2. Many-Field Packet Classification Based on Hardware

Hardware-based solutions can be based on either Field-Programmable Gate Array (FPGA) or TCAM. FPGA has been used to improve the forwarding performance of network devices through its scalability and reconfigurability, and suitable for processing long rules. Qi et al. presented a scheme that can support 50,000 rules [26]. Jiang et al. implemented a packet classification algorithm of multiple decision trees in FPGA [27]. Qu et al. proposed a decomposition-based FPGA algorithm [28] by dividing a rule into multiple range strides. All strides are compared with the corresponding bits of the search key to obtain the matching rule. Chang et al. presented another scheme based on multiple range strides [29].

TCAM is among the most popular hardwares for packet classification. Numerous algorithms have been proposed to solve the issues of TCAM-based packet classification. These issues include range representation [30–33], ruleset compression [3,16,17], and energy consumption [11,14,15,34]. A smaller TCAM has better energy efficiency and shorter access latency [15]. Because OpenFlow switches may update the flow tables frequently, several schemes have been proposed to enable efficient TCAM updates [35–37].

To avoid storing an excessive number of OpenFlow rules in TCAM, several schemes attempt to store a part of the rules in TCAM. SAX-PAC attempts to store only 5–10% of the rules in TCAM [38]. It categorizes rules into different partitions, with the rules in the same partition being mutually disjoint. The rules that cannot be categorized into any partitions are stored in TCAM. The drawback of SAX-PAC is that it must incorporate another algorithmic solution for non-overlapping rules [24]. A number of proposals have used TCAM as a cache to capture both the temporal and spatial localities of flows [39,40]. Several algorithms attempt to reduce the number of TCAM entries by transforming an original ruleset into a semantically equivalent ruleset [41,42]. Because a transformed ruleset is usually not updatable, resulting in high update latency, additional TCAM banks can be employed to mitigate the latency [42]. Our recent work extends the previous work [43] using a decision tree to select the bits of rules to be stored in TCAM [44]. However, this approach requires considerable construction time due to the complex bit selection procedure. Similar issues are addressed in different research areas [45,46].

2.3. Control-Plane Solutions for Many-Field Packet Classification

The storage overhead of TCAM can be alleviated with the support of controllers. Kannan et al. converted the original field specifications to a flow identifier [47]; however, this approach requires switches to implement the flow-identifier table and may lose fine-grained control over the network traffic [48]. Moreover, it has to compare long rules in the first OpenFlow switch of each incoming flow. While controllers can reduce the number of rules by removing redundant rules [49], this approach may not be suitable for dynamic networks because of additional TCAM update cost. For example, the removal of a high-priority rule that covers multiple low-priority rules may result in multiple rule insertions. Moreover, although the previous approaches can consume less TCAM for rulesets, they do not consider the issue of rule length. On the other hand, while rules with length longer than the TCAM word width can only be processed by algorithmic solutions, these are few in number.

2.4. TCAM-Based Multimatch Packet Classification

Although TCAM is designed to yield the first matching entry, algorithmic solutions to produce all matching entries have been proposed as well. Fang et al. presented a geometric intersection scheme [50] that generates pseudo-rules for each unique intersection of any overlapping rules. While both the original and pseudo-rules are stored in TCAM, the pseudo-rules have higher priority, meaning that they can be reported for the search keys

that match these overlapping rules. SSA splits rules into two or more subsets in order to lower the number of overlapping rules [51]. MUD appends a discriminator field to each TCAM entry in order to fulfill multimatch packet classification [52]. The field is used to exclude specific TCAM entries from subsequent accesses. While the discriminator field of MUD is generated based on the sequence of TCAM entries, the discriminator field generated by Chang et al. [18] is a bitmap in which each bit corresponds to a group of non-overlapping rules. For a ruleset of s groups, a s-bit bitmap is appended to each TCAM entry. For a TCAM entry of rule R, only the corresponding bit of the rule group of rule R is *one*, and the other bits are "don't care". The other bits of the bitmap field are set to "don't care", allowing a search key to match TCAM entries of different rule groups simultaneously. In this way, a search key can be designated to yield the matching TCAM entry of a specific rule group by specifying *one* to the bit of the corresponding rule group. Contrarily, the rules of a specific rule group can be excluded from matching a search key if the search key specifies the corresponding bit as *zero*. Therefore, the bitmap of the initial search key has s bits of *one*. When a matching TCAM entry is yielded, the corresponding bit is set to *zero*. As a result, the previously matching TCAM entries can be excluded from the subsequent searches by setting the corresponding bits in search keys to *zero* such that these search keys can yield another matching TCAM entries.

2.5. Summarization

Table 1 summarizs the previous algorithms for many-field packet classification based on TCAM. Currently, these algorithms attempt to reduce the storage overhead of TCAM by either storing selected rules [38–40] or selected bits of rules [44] in TCAM. The algorithms used for selecting rules may have performance issues for rules not stored in TCAM. The algorithm for selecting bits of rules to be stored in TCAM may have high computation overhead owing to the massive number of bit combinations. Currently, there is no efficient algorithm to improve the storage efficiency of many-field packet classification based on TCAM.

Table 1. Summary of algorithms for many-field packet classification based on TCAM.

Reference	Research Goal	Limitation
[38]	Reducing TCAM storage requirements using an algorithmic solution for non-overlapping rules	Unpredictable performance of the algorithmic solution
[39,40]	Using TCAM as cache	Unpredictable cache-miss performance
[44]	Selecting rule bits to be stored in TCAM	High construction time

3. Challenges and Ideas

In this research, we attempt to perform TCAM-based packet classification using narrow TCAM entries. Conventional TCAM-based implementations of packet classification store entire rules in the TCAM chip and input the search key for comparison against all entries. The matchline of each TCAM entry indicates whether the search key matches the stored ternary string. All matchlines are connected to a priority encoder to generate the address of the first matchline with a match state. When a TCAM entry cannot store one entire rule, the classification cannot be performed without algorithmic solutions. Because different rules may specify different fields for various applications, these rules can be categorized into different groups according to their field specifications to enable packet classification for long rules using TCAM.

In our previous work [44], we used a multistage approach in which each stage stores a part of the rule bits in TCAM. The result of each stage can be either a tag for the next stage or a matching rule. However, this scheme may backtrack to previous stages due to TCAM

mismatches in a stage. Moreover, the scheme may consume considerable computation time. Another scheme divides the rules of packet classification into two parts, with each part stored in different TCAM [53]. The result of searching the first TCAM is a tag for the second TCAM. Additional TCAM entries for overlapping rules are inserted into the first TCAM to avoid mismatch. In the worst case, this approach may result in $O(N^2)$ TCAM entries for N rules.

Thus, our motivation is to store one selected field for each rule in TCAM for packet classification. The remaining fields of a rule are stored in SRAM for further comparison if the corresponding field stored in TCAM matches the search key. Although the controller can generate the TCAM entries for the selected field of rules as well as the SRAM entries for the rest fields of rules, transmitting the results to switches requires defining proprietary messages. Notably, although PISA provides the flexibility of packet processing, it does not process rules generated by controllers. Accordingly, we assume that each switch is installed with a software module to preprocess rules from the controller and generate both TCAM and SRAM entries based on our scheme.

This approach is non-trivial for the following reasons. First, multiple rules may share the same or overlapping field specifications. To access these rules, we can either generate one TCAM entry for each rule or store all rules in one SRAM entry. In the former case, multimatch packet classification must be performed to yield all matching entries. As a result, the search performance is affected by the maximum number of rules that share the same field specification. In the latter case, each SRAM entry may store a different number of rules, resulting in memory wastage. Moreover, the number of rules to be further compared is unpredictable. Considering the rules in Table 2, where the former rules have higher priority than the latter ones, each rule has five fields: the source IP address (SIP), destination IP address (DIP), source port (SP), destination port (DP), and protocol (PROT). If only the fourth field of all rules (DP) is stored in TCAM, there could be either eleven TCAM entries which have the same specification [0:65535], or one SRAM entry that stores eleven rules. Both cases can result in unacceptable performance in extreme cases.

Table 2. Example of rules on five fields.

Rule	SIP	DIP	SP	DP	PROT
R1	011∗	1101∗	[0:65535]	[133:133]	[6:6]
R2	011∗	1101∗	[53:53]	[0:65535]	[6:6]
R3	0010∗	1101∗	[0:65535]	[53:53]	[17:17]
R4	0010∗	1101∗	[0:65535]	[25:25]	[17:17]
R5	011∗	0110∗	[161:161]	[0:65535]	[17:17]
R6	011∗	110∗	[5540:5540]	[0:65535]	[17:17]
R7	001∗	11∗	[0:65535]	[0:65535]	[0:255]
R8	00∗	110∗	[0:65535]	[0:65535]	[0:255]
R9	0100∗	∗	[0:65535]	[0:65535]	[0:255]
R10	0101∗	0001∗	[0:65535]	[1433:1433]	[6:6]
R11	011∗	110∗	[22:22]	[0:65535]	[6:6]
R12	001∗	∗	[0:65535]	[0:65535]	[0:255]
R13	∗	0101∗	[0:65535]	[0:65535]	[0:255]
R14	∗	0111∗	[0:65535]	[0:65535]	[0:255]
R15	∗	1001∗	[0:65535]	[0:65535]	[0:255]

Each asterisk indicates the "don't care" bits of a field.

Second, when the fields stored in SRAM do not match the search key, it is necessary to yield the next matching TCAM entry for potential matching rules. The number of

TCAM accesses without yielding any matching rules can become another performance bottleneck. For the example in Table 2, when storing the SIP field in TCAM,the search key $[\text{SIP} : 01101_b, \text{DIP} : 01110_b, \text{SP} : 50, \text{SP} : 23, \text{PROT} : 46]$ matches the TCAM entries of R1, R2, R5, R6, R11, R13, R14, and R15. Most matching TCAM entries cannot yield any matching rules except that of R14. The number of redundant matches should be minimized in order to improve search performance.

Third, the search key may match a rule with a priority that is not the highest among its matching rules. In this case, the classifier must access the next matching TCAM entry for possible matching rules. In the previous example, if we store the distinct specifications of the DIP field in TCAM in order of their first appearance in the ruleset, the search key $[00101_b, 11000_b, 20, 80, 55]$ matches the TCAM entry of R8 first. However, the classification procedure cannot stop, as the search key matches R7 as well, which has a higher priority.

To address these challenges, our scheme intelligently selects a field of each rule to be stored in TCAM to reduce the number of accessed rules stored in SRAM entries. Because one single field may not be able to efficiently identify all rules owing to the rules sharing the same field specifications, our scheme selects different fields for different rules in order to minimize redundant TCAM and SRAM accesses. Moreover, because the rules stored in SRAM are reordered, the highest priority rule can only be determined by yielding all the matching rules. Our scheme employs the multimatch packet classification technique to yield all the matching TCAM entries, allowing all matching rules to be accessed from the corresponding SRAM entries. As a result, all the matching rules with higher priority can be retrieved to ensure the correctness of packet classification. Our previous work used a bitwise approaches to select the bits to be stored in TCAM [43,44]; the new scheme can avoid long construction times by selecting the field of rules to be stored in TCAM.

4. Our Scheme

In this section, we introduce the proposed scheme for employing TCAM with narrow-width entries for packet classification in detail. Our scheme includes three procedures. The first procedure generates both TCAM and SRAM entries from the original rules. The second procedure performs packet classification on the previously generated TCAM and SRAM entries. The last procedure updates the proposed data structures for new or deleted rules.

We present the procedure for generating TCAM and SRAM entries in Section 4.1 and the packet classification procedure in Section 4.3. Section 4.4 describes the techniques for improving the speed performance of our scheme. The update procedure is introduced in Section 4.5.

4.1. Generating TCAM and SRAM Entries

The procedure for generating TCAM and SRAM entries starts by categorizing rules into different groups, where the rules in each group do not overlap with each other. Two rules do not overlap with each other if there exists at least one field in which the specifications of both rules do not overlap with each other. The procedure may consist of multiple iterations, where a field is selected as the index field of each iteration. A good index field can effectively distinguish rules by including more rules in a group. If one index field is not enough to distinguish all rules, the second iteration is conducted, and so forth. Each iteration may select different index fields. The procedure ceases until all rules are categorized into one group with a designated index field. For the rules of each group, their specifications of the corresponding index field are stored in TCAM and the remaining fields are stored in the corresponding SRAM entries. Each search key matches at most one rule and one TCAM entry in a group. Because the search performance is tied to the number of groups, it is desirable to minimize the number of groups by maximizing the number of rules in each group.

Next, we describe the operations for selecting the index field in each iteration. Because our algorithm attempts to select rules based on the specifications of one field, our algorithm generates the complementary graph of the interval graph (hereinafter called the comple-

mentary interval graph) of each field. Assume that there are d fields specified in the ruleset. We convert each field specification as a range. Each range of the ith field corresponds to one node in the complementary interval graph, G_i, where each node corresponds to exact one rule. There are N nodes in the complementary interval graph, where N is the number of rules. The complementary interval graph may consist of multiple nodes corresponding to the same range. Two nodes are connected by an edge if their corresponding rules are disjoint. With the complementary interval graph, we can translate the calculation yielding the maximum number of identified rules as the maximum clique problem. Accordingly, the maximum clique in G_i is equivalent to the maximum set of disjoint rules. By comparing the clique size of each field from 1 to d, the field that can produce the largest clique is selected as the first index field. The rules of the nodes in the largest clique are then removed from all complementary interval graphs. The nodes without storing any rule are then removed as well as their edges. The above procedure is repeated until all the complementary interval graphs are empty. We depict the above procedure using a flowchart, as shown in Figure 1.

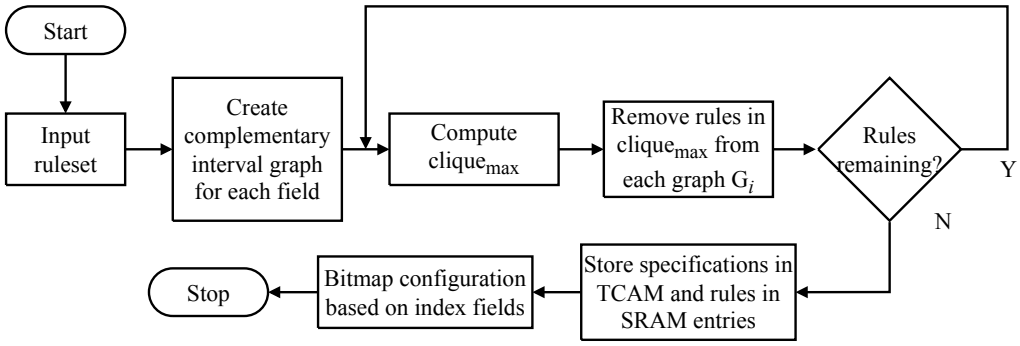

Figure 1. The flowchart of the procedure for generating TCAM and SRAM entries.

Next, we present our heuristic algorithm for the maximum clique problem because the problem is an NP-hard problem [54]. Our algorithm starts by selecting the first node of the clique with the corresponding range that has the smallest higher endpoint among all ranges. If there is more than one range with the same higher endpoint, the node with the rule that has more overlapping rules is selected. If two or more rules have the same maximum number of overlapping rules, the rule priority is the tie-breaker. After selecting the first node, only those nodes connecting the first node are considered as candidates of the second node in the clique. Among these candidates, the selection of the second node is the same as that of the first node. The above steps are repeated until no node can be inserted into the clique.

The reasoning of our heuristic algorithm that selects the node with the range that has the smallest higher endpoint is based on the observation that a smaller range usually overlaps with fewer ranges. Thus, it is possible to include more nodes in a clique by selecting a smaller range. The selection among nodes with the same higher endpoint is related to the cliques generated in later iterations. As mentioned above, the multi-iteration procedure for generating TCAM and SRAM entries may select different fields as index fields. By removing those rules with more overlapping rules in earlier iterations, it is possible to increase the clique sizes of later iterations.

The corresponding pseudocode is listed in Algorithm 1. The algorithm for generating a maximum clique is repeated iteratively until all nodes are removed from each complementary interval graph. Because all complementary interval graphs have the same number of rules, when all rules are removed from G_1, the other graphs must be empty as well. Therefore, the algorithm stops by checking whether G_1 is empty. When a maximum clique is generated, all complementary interval graphs are updated by removing the nodes corresponding to the rules in the clique. The edges of the removed nodes are deleted as well.

Algorithm 1 TCAM entry generation.

Require: $G_i = (V, E)$ for *RuleSet* using the ith field, $1 \leq i \leq d$;
1: $CliqueSet \leftarrow \emptyset$;
2: **while** G_1 *is not empty* **do**
3: **for** $1 \leq i \leq d$ **do**
4: $Clique_{max} \leftarrow \emptyset$;
5: $Clique \leftarrow$ MAXIMUM_CLIQUE_GENERATION(G_i);
6: **if** SIZE($Clique$) > SIZE($Clique_{max}$) **then**
7: $Clique_{max} \leftarrow Clique$;
8: **end if**
9: **end for**
10: $CliqueSet \leftarrow CliqueSet \bigcup Clique_{max}$;
11: **for** $1 \leq i \leq d$ **do**
12: UPDATE(G_i);
13: **end for**
14: **end while**
15: **return** CliqueSet;

16: **function** MAXIMUM_CLIQUE_GENERATION(G)
17: $Clique \leftarrow \emptyset$;
18: Insert the node corresponding to the smallest higher endpoint into *Clique*.
19: Select the node with the smallest higher endpoint among all nodes that connect to all nodes in *Clique*. Repeat this step until no node can be selected.
20: **return** *Clique*;
21: **end function**

We use the SIP field of the rules in Table 2 as an example to illustrate the proposed heuristic algorithm. A complement graph of an interval graph is generated by creating one node for each rule along with its field specification of SIP. Two nodes are connected if their rules are disjoint, as shown in Figure 2. The heuristic algorithm selects the first node from the nodes of R3 and R4, as both nodes have the smallest higher endpoint among all nodes. Because both R3 and R4 overlap with three other rules, the node of R3 is selected as the first clique node because of its higher priority. Next, the nodes of R9 and R10 are inserted into the clique. Then, the five nodes of prefixes $011*$ are the candidates to be inserted. The node of R1 is selected because R1 overlaps with another two rules and the other four rules (R2, R5, R6, and R11) only overlap with at most one rule. After inserting the node of R1 into the clique, the heuristic algorithm stops, as no other nodes can be inserted. As a result, the algorithm generates a clique with four nodes, which are denoted by thick circles in Figure 2 for the SIP field.

The first index field, denoted as I_1, is the one that can generate the largest clique among all fields. We indicate the clique size of I_1 as C_1. Then, all complementary interval graphs, $G_i, 1 \leq i \leq d$, are updated by removing the rules specified in the largest group. As a result, the sizes of all complementary interval graphs are reduced to $(N - C_1)$ nodes. Then, in the second iteration, another index field I_2 is selected, where the clique size of I_2 is C_2. After removing the rules in the clique of I_2, the numbers of nodes in all complementary interval graphs are reduced to $(N - C_1 - C_2)$. Our procedure ends when $N - \sum_{f=1}^{m} C_f = 0$, where m is the total number of iterations. Because m can be larger than d, the index fields of different iterations can be the same header field.

Taking the rules in Table 2 as an example to illustrate the generation of all index fields, the complementary interval graphs for all fields are generated to carry out our heuristic algorithm. The DIP field is selected as the first index field (or I_1) because it leads to the largest clique among five fields. The clique of DIP includes the nodes of R1, R5, R10, R13, R14, and R15. The nodes of these rules are then removed from all graphs. The complementary interval graph of SIP after performing the first iteration of our procedure is shown in Figure 3, where the number of nodes is reduced to nine. In the second iteration, the SIP field is selected as I_2, where its clique includes the nodes of R2, R3, and R9 (the thick

circles in Figure 3). The third index field could be SIP, SP, or PROT, as their cliques have the same size. We select SIP; the reason for this selection is explained in the description of the packet classification procedure. The clique of SIP includes R4 and R11, and SIP is the index field for the next three iterations. As a result, we obtain the following set of index fields: $I = \langle \text{DIP, SIP, SIP, SIP, SIP, SIP} \rangle$.

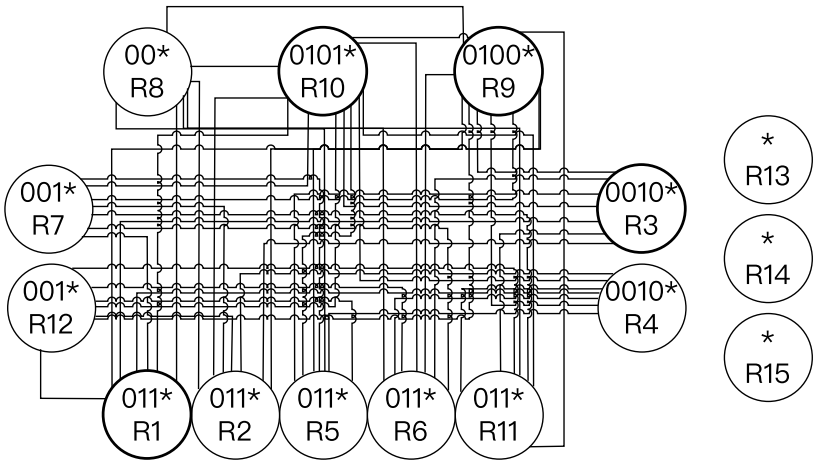

Figure 2. The complementary interval graph of the SIP field in Table 2.

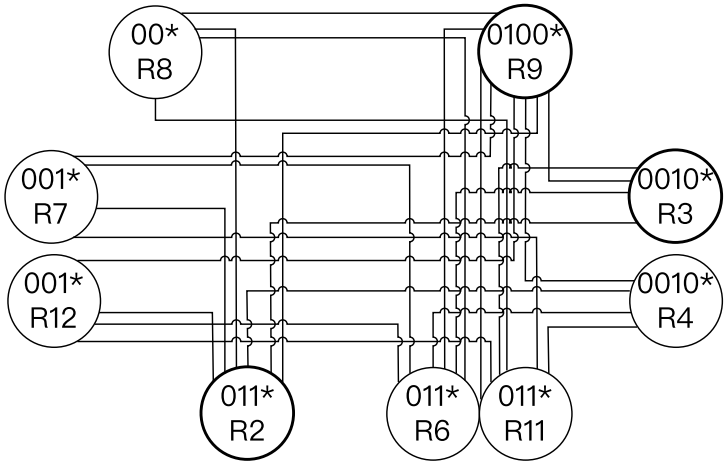

Figure 3. The complementary interval graph of the SIP field in Table 2 after performing the first iteration of our procedure.

4.2. Improvement for Overlapping Rules

For the procedure to generate the TCAM and SRAM entries in Section 4.1, the number of iterations is always higher than or equal to the maximum number of rules that mutually overlap with each other, as these rules must be stored in different cliques. In the previous example, R3, R7, R8, and R12 (or R4, R7, R8, R12) is the largest set of mutually overlapping rules. As a result, the number of overlapping rules may become a performance bottleneck of our scheme for packets with header fields that match these overlapping rules.

This performance bottleneck can be mitigated by increasing the number of rules that map to a node. This improvement aims to reduce the number of TCAM accesses by comparing more rules in each SRAM access. For commodity SRAM chips, in which the width of an SRAM word is 512 bits [55], each SRAM word can store up to three IPv4 rules. For the rules stored in the same SRAM entry, if one of their corresponding TCAM entries matches the search key, all the rules are retrieved from SRAM for further comparisons.

We modify the previous heuristic algorithm for generating a clique to serve the purpose of generating SRAM entries with multiple rules. First, we generate the same

complementary interval graph as the original algorithm. We do not modify the node selection for the clique; rather, an additional operation is carried out after selecting a node. When a node of a complementary interval graph is selected as the first clique node, the operation merges two different nodes with the clique node, where the merged nodes must have the same or an immediately larger higher endpoint than that of the selected node regardless of whether or not they are adjacent. After merging two nodes with the clique node, the range of the clique node is updated to the smallest range that covers the ranges of these nodes, i.e., the smallest enclosure range for these node ranges. The clique stores the rules of the merged nodes. Because the merged nodes are removed from the graph, their edges are removed and the edges of the clique node are updated based on its new range. Then, another clique node is selected and the above operation is performed for the new clique node. The steps in selecting a clique node and merging additional nodes repeat until no clique node can be generated.

We illustrate the modified heuristic algorithm using the example in Figure 2. When the node of R3 is selected as the first clique node, the algorithm merges two nodes, the nodes of R4 and R7, with ranges that have the same or immediately larger higher endpoints. The range of the first clique node is updated to the smallest enclosure range for these nodes, which is 001∗. Next, the node of R9 is selected as the second clique node. Likewise, the nodes of R10 and R1 are merged with the second clique node. Then, the algorithm stops because it cannot generate a new clique node. The updated complementary interval graph is shown in Figure 4. Although the generated clique consists of only two nodes, it includes six rules.

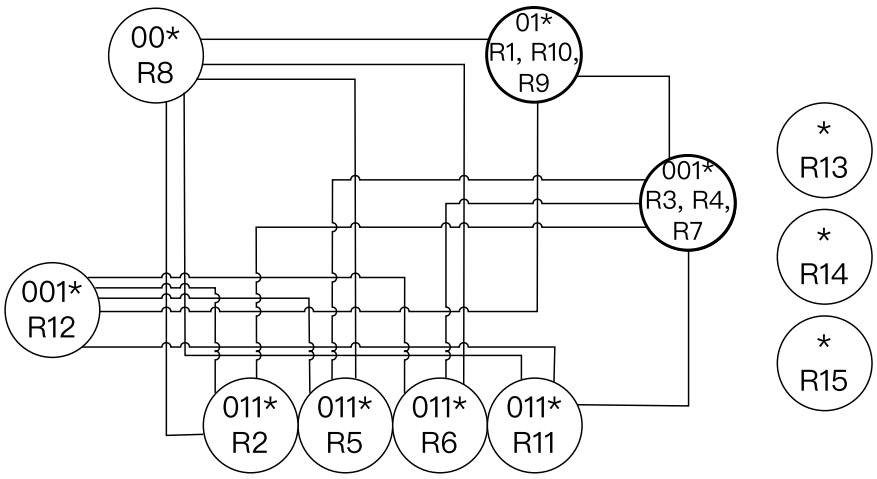

Figure 4. The complementary interval graph of the SIP field in Table 2 with the modified heuristic algorithm.

When an index field is yielded by generating the cliques of all complementary interval graphs, the rules in the clique of the index field are removed from all complementary interval graphs, where the nodes without storing any rule are removed as well. Then, the merged nodes in the complementary interval graphs of all fields must be restored before the next iteration.

With the modified heuristic algorithm, it is possible to remove more rules from the complementary interval graphs. As a result, the number of iterations required by the procedure for generating TCAM and SRAM entries can be reduced. For the rules in Table 2, SIP is the first index field with a clique including two nodes: (001∗/R3, R4, R7) and (01∗/R1, R9, R10). DIP is the second index field to generate another clique with two nodes: (01∗/R5, R13, R14) and (1∗/R2, R6, R15). The remaining rules are stored in the last clique with only one node, (0∗/R8, R11, R12), where I_3 = SIP. Because at most one TCAM access is required for each index field, the modified heuristic algorithm can reduce the maximum number of TCAM accesses from six to three.

4.3. Packet Classification

We proceed to describe the procedure of packet classification based on the TCAM and SRAM entries generated by the above procedure. The packet classification procedure iteratively uses different index fields to access the corresponding TCAM entries, where each TCAM entry stores the index field specification of a rule. The entire rule specifications are then stored in their corresponding SRAM entries. However, for two reasons, only storing the index field specification is not enough for our scheme to yield the highest priority matching rule. First, overlapping rules must be categorized into different groups. Second, a matching TCAM entry does not imply any matching rules, as the entire rule specification is stored in SRAM. As a result, it is necessary to yield all the matching TCAM entries for each unique index field.

Consequently, we present the data structure for TCAM entries. Except for the index field specification, each TCAM entry includes the index field identification *id*, where $1 \leq id \leq d$, in order to specify which field is stored in TCAM. ased on our previous work [18], we further append a bitmap to each TCAM entry to serve as the discriminating field. The bitmap length is equal to the number of rule groups. For each rule in the *i*th group, the *i*th bit of its bitmap is set to one and the other bits are "don't care" (or "$*$"). We show the TCAM entries for the previous example in Table 3, where there are three rule groups. A three-bit bitmap is attached to each TCAM entry. The bitmaps enable the search procedure to perform multimatch TCAM packet classification upon the entries generated by our procedure. Specifically, there are two rule groups of the SIP field, each of which corresponds to the first and third bits of the bitmap field. The TCAM entries of the first rule group, $001*$ and $01*$, have the first bit of the bitmap field as *one*. Likewise, the TCAM entry of the third rule group, $0*$, has the third bit of its bitmap field as *one*. The other bits of the bitmap field are set to "don't care", allowing a search key to match the TCAM entries of different rule groups simultaneously.

Table 3. TCAM and SRAM entries for the rules in Table 2.

	TCAM		**SRAM**
id	Specification	Bitmap	Rules
2	$01*$	$*1*$	R5, R13, R14
1	$0*$	$**1$	R8, R11, R12
2	$1*$	$*1*$	R2, R6, R15
1	$01*$	$1**$	R1, R9, R10
1	$001*$	$1**$	R3, R4, R7

The search procedure starts by extracting the header value of the first index field from the inspected header fields. The search key includes the *id* field and a bitmap with bits that are ones. Because of the *id* field, only TCAM entries with the same index field are compared. If a TCAM entry of the first index field matches the search key, the rules stored in the corresponding SRAM entry are retrieved for further comparison. Assume that the matching entry belongs to the *i*th group. Then, the procedure updates the *i*th bit of the bitmap in the TCAM search key to zero and proceeds to the next TCAM access. The next TCAM access excludes the entries of the *i*th group, as their *i*th bits of the bitmap field are one. The TCAM accesses for an index field are complete when there is no matching entry or all groups of the same index field are excluded. The header value of the second index field is then extracted from the inspected header fields along with the updated *id* field and the previous bitmap to start another iteration of TCAM accesses. The above procedure repeats until all index fields specified in TCAM entries are compared. The highest-priority rule among all the matching rules is the result. Note that the sequence of extracting index fields does not affect the correctness of the search procedure, as all index fields are compared.

We use the example in Table 3 to illustrate the search procedure. For the header fields inspected in an incoming packet (00101_b, 11010_b, 3000, 4000, 20), the specification of the first index field, which is SIP, is extracted for the first TCAM access. The first TCAM search key is $(1, 00101_b, 111_b)$, where the first value is the *id* of the first index field and the last value is a bitmap. By comparing the TCAM entries shown in Table 3, the second TCAM entry matches the search key. The rules in the corresponding SRAM entry, R8, R11, and R12, are fetched for comparing with the inspected header fields, where R8 and R12 are the matching rules. Next, the search procedure updates the bitmap of the TCAM search key by setting the third bit to zero to exclude the TCAM entries of the third group from the following TCAM accesses. In the second TCAM access, the search key is $(1, 00101_b, 110_b)$ and matches the last TCAM entry. Then, the rules, R3, R4, and R7 are fetched for comparisons, where R7 is the matching rule. Because there are only two groups of SIP, the search procedure uses the second index field for the third TCAM access, with a new TCAM search key, $(2, 11010_b, 010_b)$. While the third TCAM entry matches the search key, the corresponding rules, R2, R6, and R15, do not match the inspected header fields. The search procedure stops because all index fields have been examined. As a result, the best-matching rule based on the sequence in the ruleset is R7.

For another set of header fields (10110_b, 01001_b, 161, 23, 17), the first search key is (1, 10110_b, 111_b), which does not match any TCAM entry. Then, the second search key $(2, 01001_b, 010_b)$ is generated to exclude the TCAM entries of the first index field by setting the first and third bits of the bitmap field to *zero*. The second search key matches the first TCAM entry, where the rules in the corresponding SRAM entry are R5, R13, and R14, yielding a matching rule, R5.

In the last example, with the set of header fields (01101_b, 11010_b, 53, 133, 6), the first search key is (1, 01101_b, 111_b), which matches the second TCAM entry to access the rules of the corresponding SRAM entry (R8, R11, and R12). The second search key (1, 01101_b, 110_b) matches the fourth TCAM entry, where R1, R9, and R10 are accessed. The third search key (2, 11010_b, 010_b) matches the third TCAM entry to access R2, R6 and R15. Among the accessed rules, only R1 and R2 match the inspected header fields, and R1 is the highest-priority matching rule.

We note that it is possible that a packet may not include the headers of index fields. For example, when two fields, the destination IP address and VLAN tag, are selected as the index fields, an incoming packet may only specify the destination IP address. In this case, the incoming packet can only match rules for which the VLAN tag field is a wildcard. Accordingly, the search procedure assigns a VLAN tag for the packet, where the VLAN tag only matches those TCAM entries specifying a wildcard. For example, the value of the VLAN tag could be an unused one in the ruleset or one of the reserved VLAN tags (0 or 4095). Notably, the VLAN tag does not cause any compatibility issues, as it is only used in the forwarding engine.

We can eliminate the *id* field of the search key by adaptively setting the bitmap field. By setting all bits which correspond to the groups of an index field in the TCAM search key to zero, the TCAM entries of the index field do not report any match. Consequently, the *id* field can be removed to reduce the length of a TCAM entry. The search procedure is modified by initially setting the bitmap of the TCAM search key to zero except for the bits of the first index field. When the search procedure completes the TCAM accesses for the first index field, the bits of the first index field are set to zero and the bits of the second index field are set to one. The above steps repeat until all index fields are examined.

For the previous example of header fields (00101_b, 11010_b, 3000, 4000, 20), by removing the *id* field from the TCAM entries in Table 3, the TCAM search keys for the same header are $(00101_b, 101_b)$, $(00101_b, 100_b)$ and $(11010_b, 010_b)$. For the first two search keys, the second bit of the bitmap is set to *zero*, meaning that the TCAM entries of the second index field cannot match the search key even without the *id* field. Likewise, the third search key can only match the TCAM entries of the second index field by setting the bits of the bitmap

field corresponding to the first index field *zero*. As a result, the *id* field can be removed from the search key as well as all TCAM entries.

4.4. Speed Performance Refinements

To improve search performance, we use the idea of rule reordering to eliminate redundant TCAM accesses. We observe that there could be multiple TCAM entries sharing the same specification. Thus, it is desirable to reduce the number of accesses for these TCAM entries. We generate one replicated TCAM entry for TCAM entries with the same specification. Then, the SRAM entry corresponding to the replicated TCAM entry stores a set of bitmap masks, with each mask indicating the rule groups that are unlikely to yield any matching rule. To generate the bitmap masks, the rules in the SRAM entries with the same TCAM specification are rearranged by sorting them according to the field with the most unique specifications. Thus, these rules can be assigned to new rule groups. Based on the sorting field of these rules, we can categorize the rules into different subsets with the rules in the same subset in the same subrange. Then, we generate one bitmap mask for each subrange. In the bitmap mask, only the bits corresponding to the rule groups in the same subrange are set to one, while the other bits are zero. The subranges and corresponding masks are stored in the SRAM entry of the replicated TCAM entry. The replicated TCAM entry must be placed before the original TCAM entries in order to improve the performance of packet classification. When the replicated TCAM entry is accessed, the corresponding SRAM entry is used to determine the matching subrange based on the packet header. Then, the mask of the matching subrange is extracted to perform the bitwise **AND** operator with the bitmap of the search key. As a result, only a subset of TCAM entries with the same specification is accessed.

The number of subranges is determined based on the bitmap length to satisfy the requirement that all subranges and the corresponding bitmap masks be storable in one SRAM entry. Moreover, we only generate replicated TCAM entries for the case that three or more TCAM entries have the same specifications. As a result, the number of additional TCAM entries is moderate. Notably, the replicated TCAM entries do not require any additional bits in the bitmap field when assigning one to all bits corresponding to all distinct subranges. Because these entries are placed in front of the original TCAM entries, a search key matching a replicated TCAM entry excludes the rule groups of at least one subrange in order to keep the replicated TCAM entry from matching the search key.

Because there can be multiple rule groups corresponding to one subrange, pointers are used to further reduce the number of TCAM accesses. The pointers are installed in the first SRAM entry of each subrange to indicate the addresses of the subsequent SRAM entries of the same subrange. The employment of the pointer is based on the observation that the access time of TCAM is three times longer than that of SRAM [56]. In a pipeline architecture, imbalanced access times result in a longer latency for packet classification. Thus, we are motivated to retrieve additional SRAM entries in the stage of SRAM access. When the SRAM entry with pointers is retrieved, its pointers are used to access another SRAM entries for rule comparisons. In this way, the TCAM accesses to these accessed SRAM entries can be eliminated.

4.5. Updates

Next, we introduce the update procedure of our scheme. First, we consider the data structure without the proposed performance refinements. There are two common operations for rule updates, namely, insertion and removal. For a new rule, we check whether there exists an SRAM entry to store the rule without altering the corresponding TCAM entry. If the answer is positive, the rule is inserted into the SRAM entry. Otherwise, we check whether there exists a rule group with specifications of the index field that are disjoint to the corresponding field of the new rule. If the answer is positive, one TCAM entry and one SRAM entry are inserted into the group for the new rule. If the rule cannot be stored in any groups of all index fields, a new rule group is created. In this case, one

TCAM entry and one SRAM entry are inserted. Because the bitmap field of all existing TCAM entries stores "don't care" bits except for one bit, allocating one additional bit for the new group does not affect any existing TCAM entries. For our scheme, each TCAM entry insertion does not cause any TCAM entry movements, as TCAM entries can be stored arbitrarily. There are two reasons for this flexibility. First, there is no order relationship between different bits of the bitmap field. Second, the entries of the same group do not overlap with each other. This flexibility can minimize the update cost for TCAM entries.

When a rule is removed, its specification can simply be wiped from the corresponding SRAM entry. If the SRAM entry does not store any rules, it can be removed with its TCAM entry. Because our scheme does not replicate any rules, at most one TCAM entry and one SRAM entry are updated.

We further consider the update procedure with the proposed performance refinements, in which the rules in the SRAM entries of the same TCAM specifications must be sorted according to the specifications of a field. When a new rule is inserted into an existing SRAM entry or a new SRAM is inserted, the other rules of the same TCAM specification can be relocated. The corresponding bitmap masks and pointers are updated as well. Therefore, the maximum number of updated SRAM entries is tied to the number of TCAM entries with the same specification.

In summary, while each rule insertion or removal may generate or remove one TCAM entry, no TCAM movement is required, as our search procedure does not presume rule priority based on the sequence of TCAM entries. The update performance is mainly tied to the number of updated SRAM entries with the same TCAM specification. However, because the proposed algorithm of TCAM entry generation attempts to minimize the number of rule groups, the number of SRAM entries with the same TCAM specification is usually moderate. Notably, incremental updates may degrade the efficiency of the proposed data structures, resulting in additional storage consumption and extra memory accesses. Thus, efficiency can be improved by periodically reconstructing data structures.

4.6. Time and Space Complexity Analysis

We start by analyzing the time complexity of our scheme. First, we consider the procedure for generating TCAM and SRAM entries. The time complexity of the proposed heuristic algorithm for the maximum clique problem is $O(dN \log N)$, as determined by sorting the nodes in a graph according to the higher endpoints of the corresponding field for N d-field rules. The time complexity for the search procedure is $O(MMOR)$, where $MMOR$ is the maximum number of rules that overlap mutually. Because mutually overlapping rules must be categorized into different rule groups, for a packet header matching these rules, the search procedure must access the TCAM entries of these groups. As a result, the time complexity of the search procedure is tied to the maximum number of mutually overlapping rules in a ruleset. The time complexity for updating TCAM entries is $O(1)$ and that for updating SRAM entries is $O(N')$, where N' is the maximum number of rules with the same field specification.

Because our algorithm does not incur any rule replications, the number of TCAM entries is equal to the number of rules in the worst case and the storage requirement of SRAM is identical to the ruleset size. Thus, the space complexity for both TCAM and SRAM is the same: $O(N)$.

5. Performance Evaluation

In this section, we evaluate the performance of our scheme using rulesets of three difference types, namely, access control list (ACL), firewall (FW), and IP chain (IPC). For each type, there is one real ruleset and several synthetic rulesets, with the latter generated by ClassBench [57]. The largest synthetic ruleset has about 10,000 rules. The statistics of these rulesets are listed in Table 4. Although there could be redundant rules in each ruleset, we do not remove them in order to show the performance in the worst case. We assume that range-to-prefix conversion is performed for any entries stored in TCAM, where the

FW rulesets have higher expansion ratios than others due to wide range specifications in both port fields. Each SRAM word stores up to three rules.

Our performance evaluation consists of three parts. In Section 5.1, the storage performance of our scheme is evaluated. Section 5.2 presents the speed performance of our scheme. In Section 5.3, we extend the five-field rules to twelve fields and show the performance of our scheme for twelve-field rules. The last part, Section 5.4, evaluates the update performance.

Table 4. Statistics of real and synthetic rulesets.

Type	ACL		FW		IPC	
	Rules	**Expansion**	**Rules**	**Expansion**	**Rules**	**Expansion**
REAL	683	1471	269	914	1550	2180
1 K	916	1225	791	3306	938	1223
5 K	4415	6148	4653	15,778	4460	5916
10 K	9603	12,947	9311	32,136	9037	12,127

5.1. Storage Performance

First, we show the storage performance of our scheme in terms of TCAM storage. We present the number of required bits for each TCAM entry. Each TCAM entry stores the specification of the index field and the bitmap field. In our experiments, the destination IP address prefix is the longest index field for all rulesets. Therefore, the length of each TCAM entry is equal to the number of groups plus 32, where the gray areas in Figure 5 denote the bits for the bitmap field. In the worst case, each TCAM entry requires 64 bits.

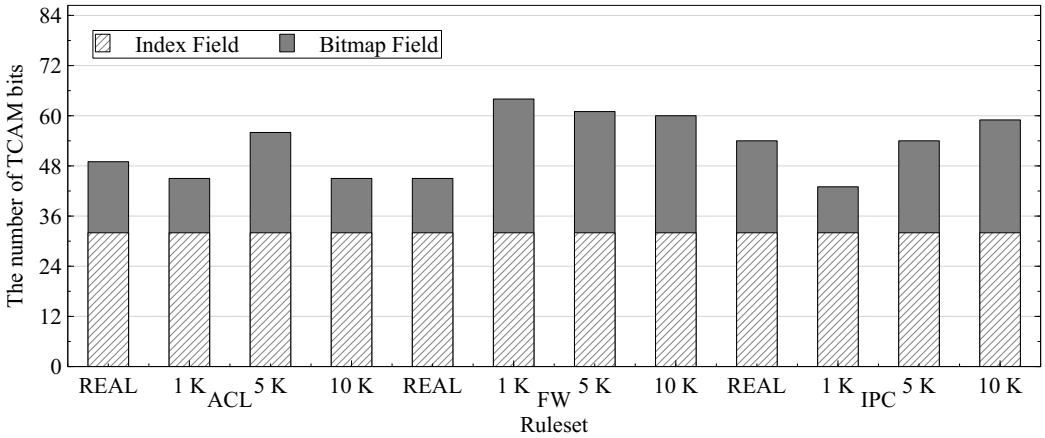

Figure 5. The number of bits for each TCAM entry.

We reveal the number of TCAM entries in Figure 6. Storing the entire rule specifications in TCAM usually has much higher storage penalty because there are two range fields in a rule. The FW rulesets occupy the most TCAM entries among all rulesets because they usually specify wide port ranges. Our scheme can effectively reduce the number of TCAM entries. Although there are additional TCAM entries for performance improvement, these entries are few and can be negligible as compared the number of TCAM entries for storing entire rules. Because our procedure of generating TCAM and SRAM entries may merge different field specifications into one TCAM entry, it can reduce the number of TCAM entries by 50% or more. We also note that our scheme does not suffer from the cost of range-to-prefix conversion because most selected fields are IP address prefixes.

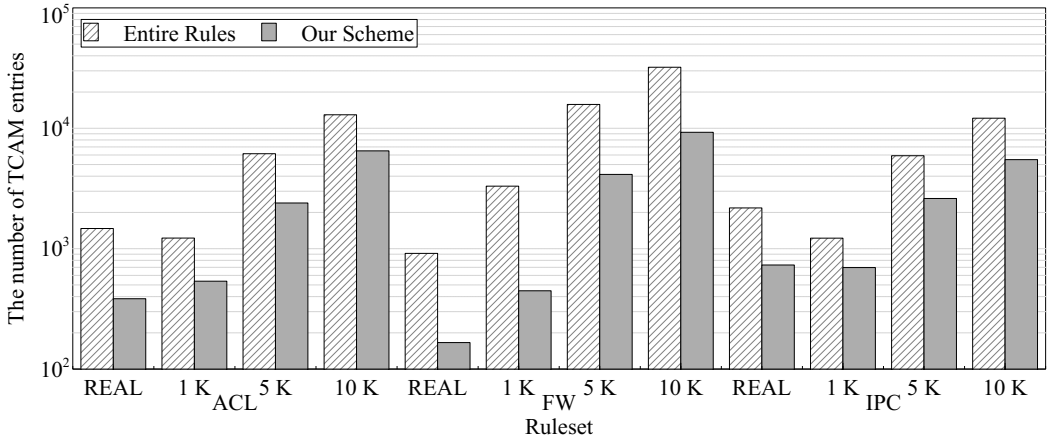

Figure 6. The number of TCAM entries.

The storage requirements of TCAM and SRAM are shown in Figure 7. The required TCAM storage is calculated by multiplying the number of TCAM entries by the entry width of a commodity TCAM chip. As mentioned above, a commodity TCAM chip may allocate 72, 144, 288, or 576 bits for each entry. We will select the smallest entry width that is larger than the required bits of each TCAM entry for our scheme. Since our scheme requires less and narrower TCAM entries, the storage requirements can be effectively reduced to improve the scalability of supporting large OpenFlow flow tables. For our scheme, the number of SRAM entries is equal to that of TCAM entries, where each SRAM entry has 512 bits. As compared to the native implementation of TCAM-based packet classification, our scheme consumes much higher SRAM storage. We believe that this is a reasonable tradeoff since TCAM has higher cost than SRAM [57].

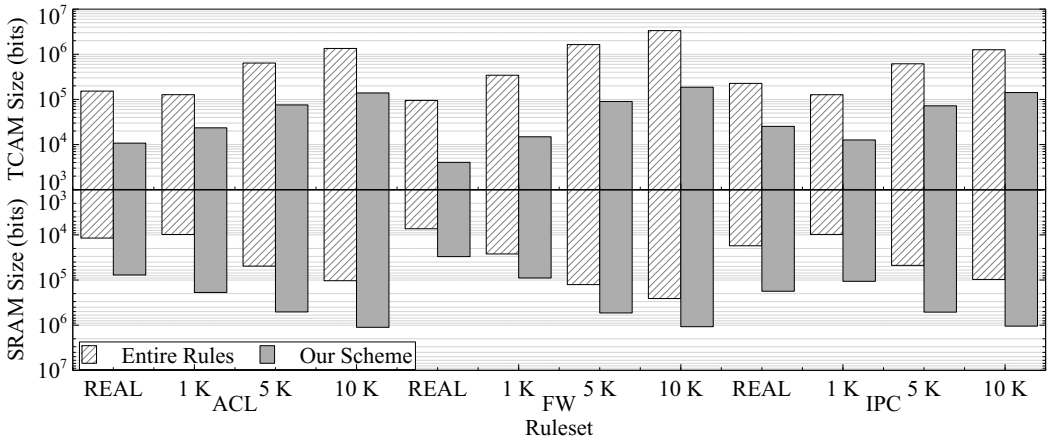

Figure 7. The storage requirements of TCAM and SRAM.

5.2. Speed Performance

Next, we reveal the speed performance of our scheme. Because the speed performance of our scheme is correlated to the number of groups, we show the number of groups in Figure 8. In this figure, we set the threshold value to the number of distinct index fields for generating rule groups, where the maximum threshold value is d. The threshold value is used to limit the selected index fields for rule grouping. When the number of distinct index fields is equal to the threshold value, only a field appeared in the previous index fields can be selected.

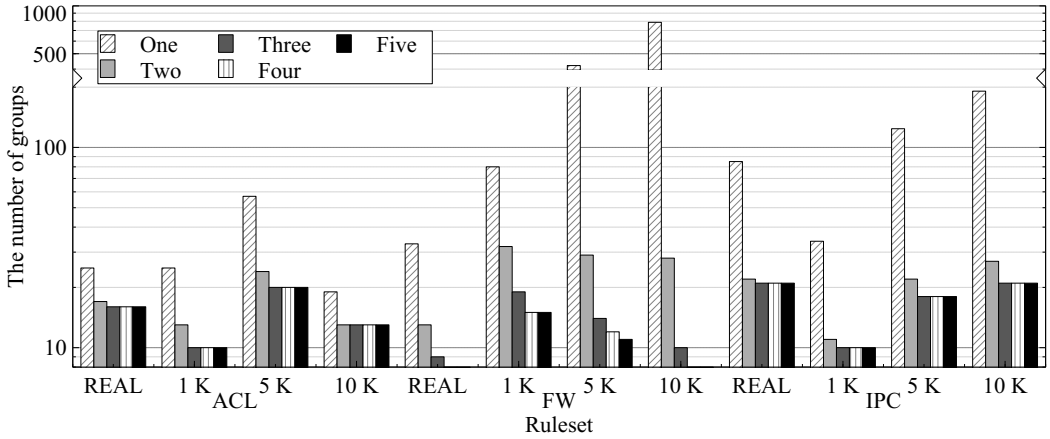

Figure 8. The number of generated groups for different number of unique index fields.

A higher threshold value can provide more flexibility for rule grouping to reduce the number of groups. However, only the rule groups of the same index fields can share the same search key to reduce redundant TCAM accesses. Moreover, the pointers in SRAM entries are only effective for the groups of the same index fields. It is thus desired to increase the number of groups of the same index field. We correlate the threshold value with the number of rule groups in Figure 8. When the threshold value is set to one, near one thousand groups are generated for the 10 K FW ruleset due to numerous rules specifying wildcard in the first index field. One additional distinct index field can significantly reduce the number of rule groups. More distinct index fields can further reduce the number of rule groups, but the improvements are less significant for most rulesets.

We further show the maximum number of TCAM accesses for different threshold values in Figure 9. Although using fewer index fields results in more rule groups, the proposed approach of bitmap masks can greatly reduce the number of TCAM accesses since there are more TCAM entries with the same specifications. As the number of index fields increases, the effectiveness of bitmaps masks is reduced to result in more TCAM accesses. However, additional index fields can reduce the number of rule groups to avoid severe increase of TCAM accesses.

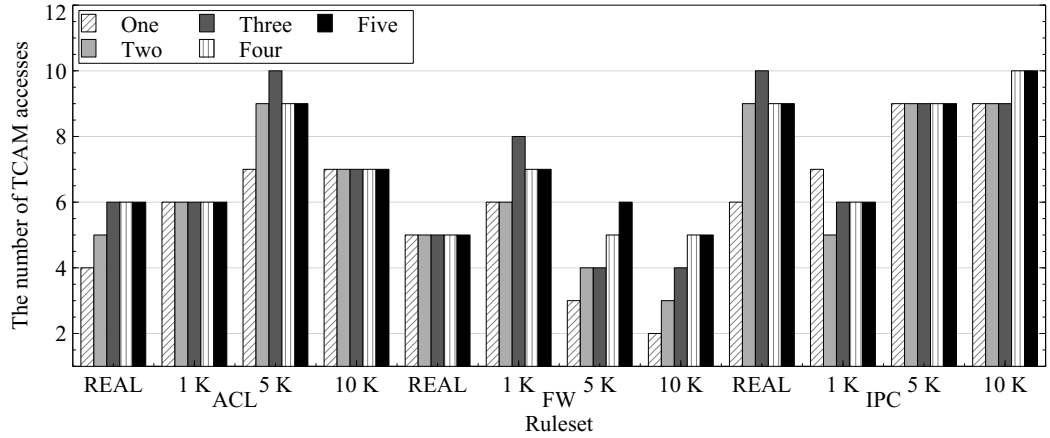

Figure 9. The maximum number of TCAM accesses for different number of unique index fields.

For TCAM entries with two index fields, the performance improvement achieved by employing bitmap masks and pointers is shown in Figure 10. Without the proposed refinements, our scheme may require more TCAM accesses than the number of rule groups because of the TCAM accesses that do not match any TCAM entries. The bitmap masks can effectively eliminate redundant TCAM accesses by excluding rule groups that are unlikely to yield matching rules. The pointers further lower the number of TCAM accesses by

retrieving SRAM entries to be accessed. As a result, the number of TCAM accesses in the worst case can be greatly reduced. We also show the average numbers of TCAM accesses with and without the proposed refinements in Figure 10 by using darker shade on each bar. The result shows that the bitmap masks can also improve the speed performance in the average case, where our search procedure can accomplish one packet classification with four TCAM accesses. Therefore, although our scheme requires additional TCAM accesses for packet classification, we believe that this is a reasonable tradeoff for the scalability and feasibility to support rules with numerous fields.

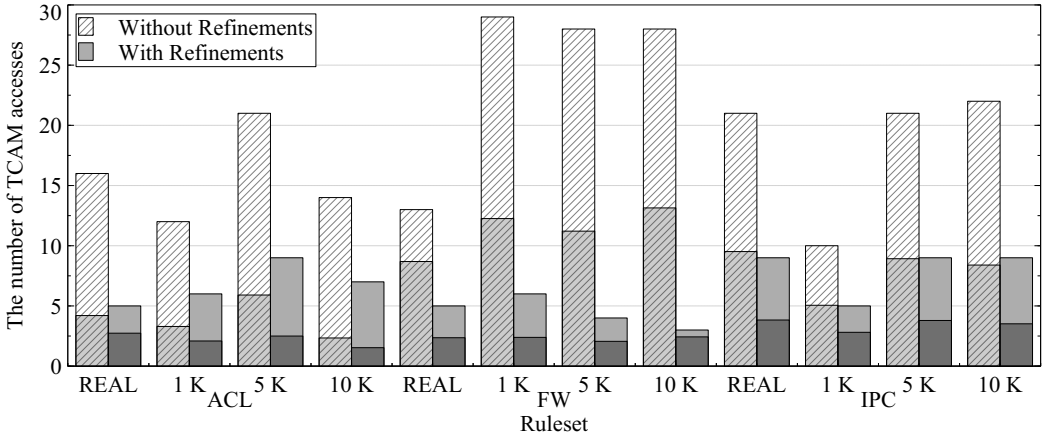

Figure 10. The performance improvement with the proposed refinements.

5.3. Performance for Twelve-Field Rules

In the following experiment, we extend the rules to twelve fields to show the performance difference caused by additional fields. To generate twelve-field rules, we append seven additional fields to each ClassBench rule used in the previous experiments. For each ruleset, we further use three wildcard ratios, 20%, 50% and 80%, to show the performance differences caused by wildcards.

We start from the storage performance of our scheme for different wildcard ratios in Figure 11. First, we consider the required TCAM storage. As the number of fields in a rule increases, the storage requirements for storing entire rules are also expanded because the length of each TCAM entry is extended to 264 bits [58]. In contrast, our scheme can achieve the result that the storage requirements for the twelve-field rules are similar to those for the five-field rules, where the storage saving comes from generating less and narrower TCAM entries. In particular, our scheme requires about 8–40% of the original TCAM entries, and each entry does not require more than 66 bits. As a result, our scheme requires only 10% or less TCAM storage as compared to the basic implementation of TCAM-based packet classification. The storage requirement of SRAM for our scheme is also much higher than that of native implementation as a tradeoff for consuming less TCAM storage.

Then, we investigate the speed performance for the twelve-field rules shown in Figure 12. The average numbers of TCAM accesses are also shown in Figure 12 by using darker shade on each bar. There are nine rulesets, whose speed performance is not degraded by additional fields. The worst case occurs for ACL 5 K and IPC 5 K with 80% wildcards, where the number of TCAM accesses is increased to 10.

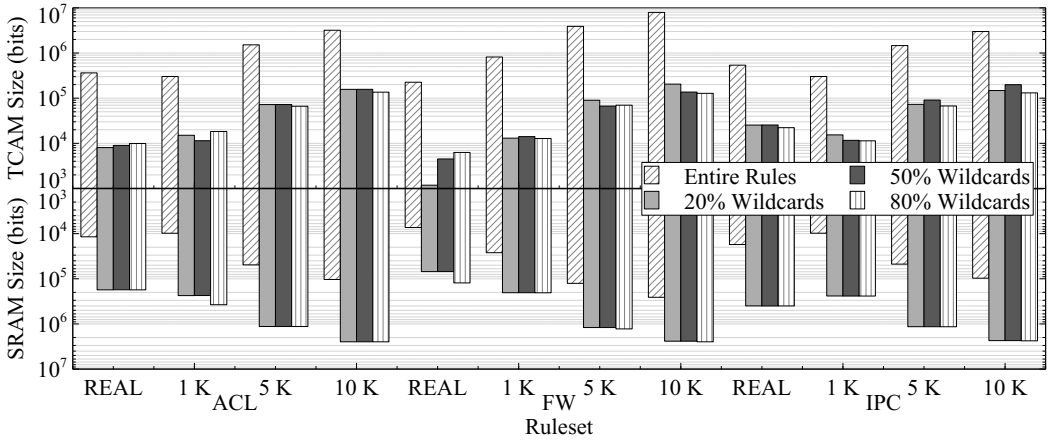

Figure 11. The TCAM/SRAM storage for twelve-field rules with different percentages of wildcards.

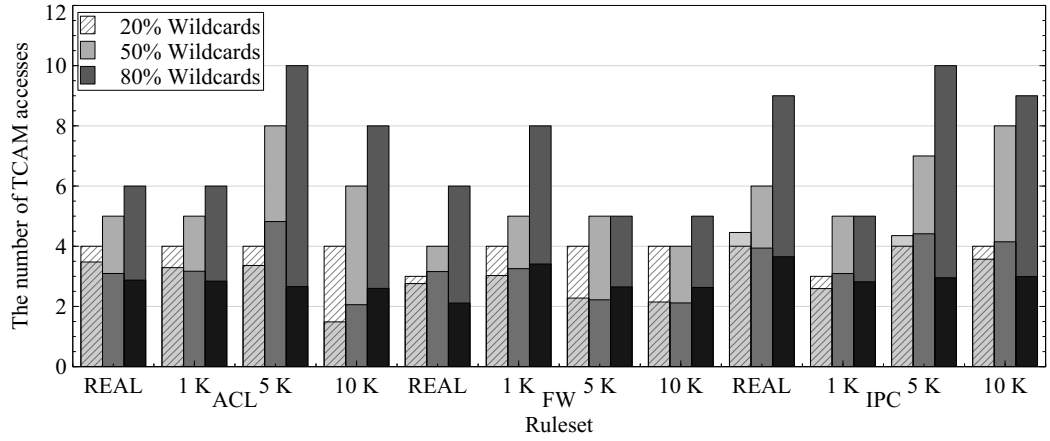

Figure 12. The number of TCAM accesses for twelve-field rules with different percentages of wildcards.

We also reveal the number of accessed SRAM entries and rules in Figure 13. Our algorithm only accesses SRAM entries that could produce matching rules by using bitmap masks. Therefore, it also reduces the number of accessed rules, where the average number of compared rules is less than ten for all rulesets.

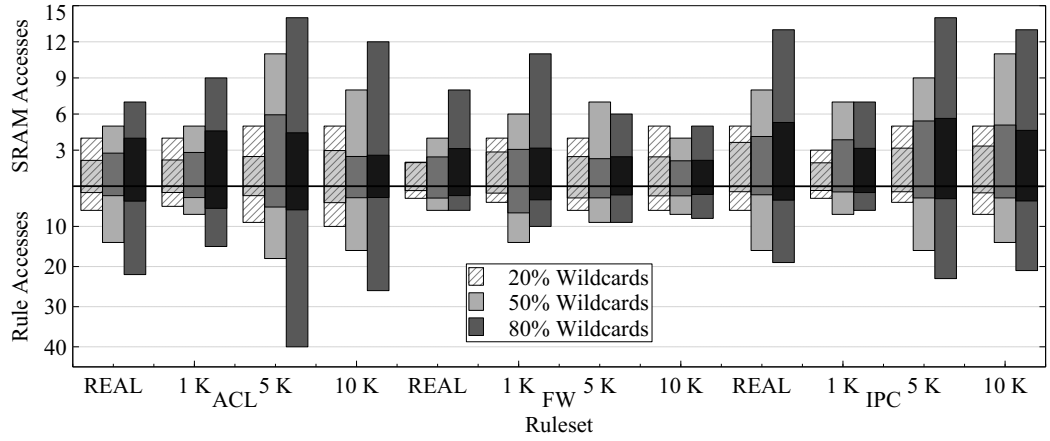

Figure 13. The number of accessed SRAM and rules for twelve-field rules with different percentages of wildcards.

To evaluate the influence of packet throughput caused by additional TCAM accesses, we use a program, TCAM-Model [59], to generate the access latency of TCAM chips. We

first generate the access latency of a TCAM chip which can store entire rules by using the following configurations: 32,768 288-bit rows, 14 nm, and eight sub-banks. The TCAM chip has 9 megabits, where the access latency is 2.49 ns. Another TCAM chip for storing the entries generated by our scheme has 288 kilobits, or 4096 72-bit rows. The estimated access latency is 0.21 ns. Accordingly, the total TCAM access delay of our scheme (2.10 ns) is shorter than that of the basic implementation, which stores entire rules. Since the latency of SRAM accesses can be hidden by TCAM accesses, the packet throughput based on our scheme is not affected by the additional TCAM accesses. With the estimated access latency, our scheme can achieve more than 400 million packets per second to support line rate of 200 Gbps with 64-byte packets.

5.4. Construction Time and Update Performance

We show the construction time of our scheme for both 5-field and 12-field rulesets. Our program is executed on a desktop PC with a 2.1 GHz CPU (Intel Core i7-12700), 16 GB main memory, and Ubuntu operating system. As shown in Figure 14, our scheme consumes less than one-second construction time for all real and 1 K rulesets. As the number of rules increases to 5 K, the construction time is extended to up to two seconds for 5-field rulesets. For 12-field rulesets, the construction time could be as long as eight seconds due to the processing time for additional fields. For 10 K rulesets, the 5-field rulesets require up to eight-second construction time, and the 12-field rulesets may consume up to 18 s. Because our scheme can support incremental updates, the construction time does not affect the performance of packet forwarding. It is feasible to periodically reconstruct the data structures to improve the efficiency of packet forwarding.

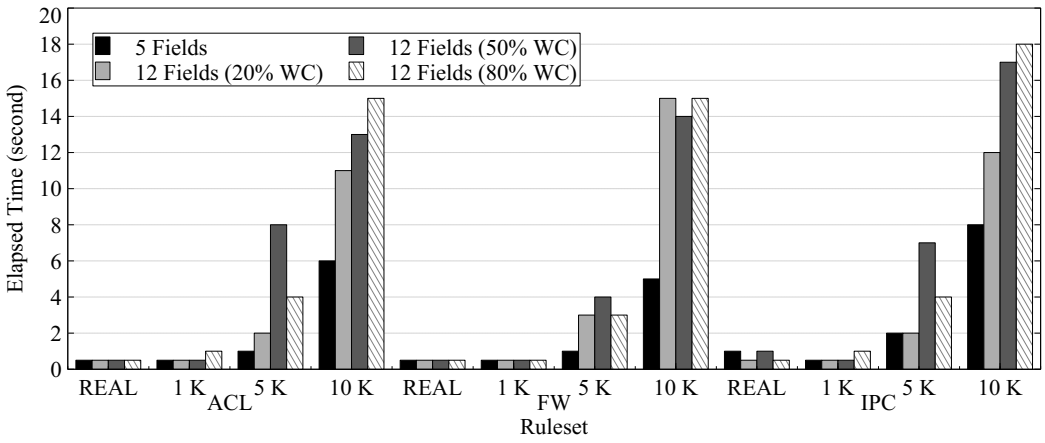

Figure 14. Construction time for different rulesets.

In the last experiment, we further show the update performance of our scheme for both 5- and 12-field rulesets. For each ruleset, we randomly select 20% rules from each ruleset for insertion. The remaining rules are used to create the proposed data structure. The numbers of updated TCAM and SRAM entries for each rule insertion are then calculated, where the worst-case results are shown in Figure 15. The number of updated TCAM entries in the worst case is shown by using darker shade of each bar. Because our scheme does not mandate the order of TCAM entries, only one TCAM entry is updated in the worst case. The number of updated SRAM entries is tied to the number of TCAM entries with the same specification. For rulesets whose all TCAM entries have different specifications, only one SRAM entry is updated. The results show that the number of updated SRAM entries in the worst case is moderate.

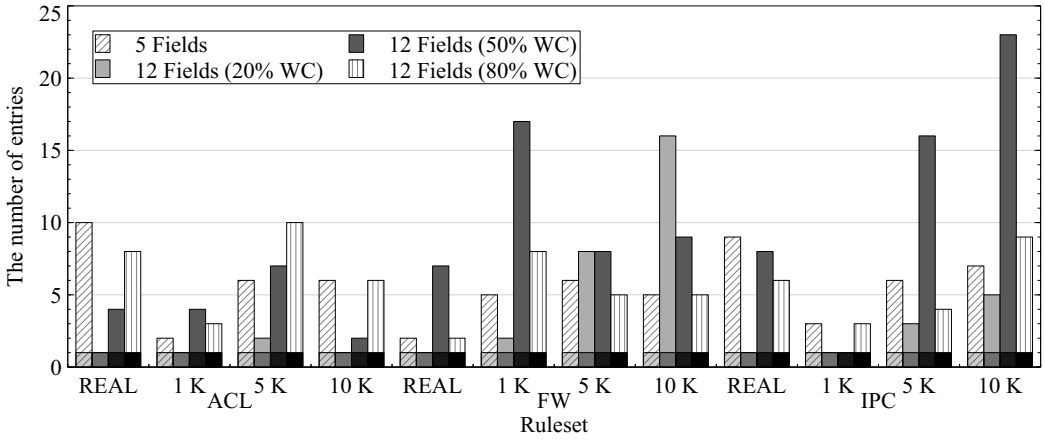

Figure 15. The number of updated TCAM/SRAM entries for different rulesets.

6. Conclusions

In this paper, we improve the storage efficiency of TCAM-based packet classification by using narrow-entry TCAM chips. Our scheme stores only one field of a rule in TCAM to minimize the width of each TCAM entry. The other fields of a rule are stored in the corresponding SRAM entry. When a search key matches the TCAM entry, the rules stored in the corresponding SRAM entry are then retrieved to determine the matching rules. Because the fields stored in SRAM may cause mismatches, our scheme has the tradeoff that the correctness of packet classification is maintained by yielding all the matching TCAM entries. We design an SRAM-based data structure to enable multimatch TCAM packet classification. Our data structure eliminates redundant TCAM accesses by storing pre-computed information. The cost of TCAM updates in our scheme can be minimized, as TCAM entries can remain unchanged for rule updates. The experimental results show that our scheme requires a moderate number of TCAM accesses for each packet classification and that it has superior storage performance for improved scalability in the case of more and longer rules. Because smaller TCAM chips can be employed for storing the entries of our scheme, the TCAM access latency of our scheme can remain shorter than that of basic implementations that store entire rules. We show that our scheme has acceptable update performance for the pre-computed information in SRAM entries. In summary, our scheme provides an efficient and feasible solution for packet classification of long rules by employing a small TCAM chip of narrow entries, and can improve the scalability of TCAM-based packet classification for long rules.

Currently, the speed performance of our scheme is limited by the number of mutually overlapping rules. In future work, we intend to develop a hybrid architecture to improve the overall speed performance by accommodating heavily overlapping rules with an algorithmic solution.

Author Contributions: Conceptualization, W.-H.P. and P.-C.W.; methodology, W.-H.P.; software, W.-H.P. and H.-T.L.; validation, W.-H.P. and H.-T.L.; formal analysis, W.-H.P.; investigation, W.-H.P. and H.-T.L.; resources, P.-C.W.; data curation, H.-T.L.; writing—original draft preparation, W.-H.P. and H.-T.L.; writing—review and editing, P.-C.W.; visualization, W.-H.P.; supervision, H.-T.L.; project administration, P.-C.W.; funding acquisition, P.-C.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by National Science and Technology Council, grant number NSTC 111-2221-E-005-045.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare that they have no known financial conflicts of interest or interpersonal ties that would affect the research presented in this study.

References

1. Chang, Y.C.; Lin, H.T.; Chu, H.M.; Wang, P.C. Efficient topology discovery for software-defined networks. *IEEE Trans. Netw. Serv. Manag.* **2020**, *18*, 1375–1388. [CrossRef]
2. Open Networking Foundation. OpenFlow Switch Specifications Ver. 1.5.1. 2015. Available online: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf (accessed on 1 April 2023).
3. Wang, P.C. Scalable packet classification with controlled cross-producting. *Comput. Netw.* **2009**, *53*, 821–834. [CrossRef]
4. Chang, Y.H.; Wang, P.C. Concise Retrieval of Flow Statistics for Software-Defined Networks. *IEEE Syst. J.* **2022**, *16*, 554–565. [CrossRef]
5. Bosshart, P.; Daly, D.; Gibb, G.; Izzard, M.; McKeown, N.; Rexford, J.; Schlesinger, C.; Talayco, D.; Vahdat, A.; Varghese, G.; et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Comput. Commun. Rev.* **2014**, *44*, 87–95. [CrossRef]
6. Hauser, F.; Häberle, M.; Merling, D.; Lindner, S.; Gurevich, V.; Zeiger, F.; Frank, R.; Menth, M. A survey on data plane programming with p4: Fundamentals, advances, and applied research. *arXiv* **2021**, arXiv:2101.10632.
7. Daly, J.; Bruschi, V.; Linguaglossa, L.; Pontarelli, S.; Rossi, D.; Tollet, J.; Torng, E.; Yourtchenko, A. TupleMerge: Fast Software Packet Processing for Online Packet Classification. *IEEE/ACM Trans. Netw.* **2019**, *27*, 1417–1431. [CrossRef]
8. Malekpourshahraki, M.; Stephens, B.E.; Vamanan, B. ADA: Arithmetic Operations with Adaptive TCAM Population in Programmable Switches. In Proceedings of the 2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS), Bologna, Italy, 10–13 July 2022; pp. 1–11.
9. Michel, O.; Bifulco, R.; Retvari, G.; Schmid, S. The programmable data plane: Abstractions, architectures, algorithms, and applications. *ACM Comput. Surv. (CSUR)* **2021**, *54*, 1–36. [CrossRef]
10. Zolfaghari, H.; Rossi, D.; Cerroni, W.; Okuhara, H.; Raffaelli, C.; Nurmi, J. Flexible software-defined packet processing using low-area hardware. *IEEE Access* **2020**, *8*, 98929–98945. [CrossRef]
11. Banerjee-Mishra, T.; Sahni, S.; Seetharaman, G. PC-DUOS: Fast TCAM lookup and update for packet classifiers. In Proceedings of the IEEE ISCC, Kerkyra, Greece, 28 June–1 July 2011; pp. 265–270.
12. Bremler-Barr, A.; Hay, D.; Hendler, D.; Roth, R.M. PEDS: A parallel error detection scheme for TCAM devices. *IEEE/ACM Trans. Netw.* **2010**, *18*, 1665–1675. [CrossRef]
13. Taylor, D.E.; Spitznagel, E.W. *On Using Content Addressable Memory for Packet Classification*; Technical Report WUCSE-2005-9; Washington University: Saint Louis, MO, USA, 2005.
14. Lu, W.; Sahni, S. Low-Power TCAMs for Very Large Forwarding Tables. *IEEE/ACM Trans. Netw.* **2010**, *18*, 948–959. [CrossRef]
15. Banerjee-Mishra, T.; Sahni, S. PETCAM-A Power Efficient TCAM Architecture for Forwarding Tables. *IEEE Trans. Comput.* **2012**, *61*, 3–17. [CrossRef]
16. Cheng, Y.C.; Wang, P.C. Scalable Multi-Match Packet Classification Using TCAM and SRAM. *IEEE Trans. Comput.* **2016**, *65*, 2257–2269. [CrossRef]
17. Wang, P.C. Scalable packet classification for datacenter networks. *IEEE J. Sel. Areas Commun.* **2014**, *32*, 124–137. [CrossRef]
18. Chang, D.Y.; Wang, P.C. TCAM-Based Multi-Match Packet Classification Using Multidimensional Rule Layering. *IEEE/ACM Trans. Netw.* **2016**, *24*, 1125–1138. [CrossRef]
19. Hsieh, C.L.; Weng, N. Scalable many-field packet classification using multidimensional-cutting via selective bit-concatenation. In Proceedings of the ACM/IEEE ANCS, Oakland, CA, USA, 7–8 May 2015; pp. 187–188.
20. Qu, Y.R.; Zhou, S.; Prasanna, V.K. A decomposition-based approach for scalable many-field packet classification on multi-core processors. *Int. J. Parallel Program.* **2015**, *43*, 965–987. [CrossRef]
21. Demianiuk, V.; Kogan, K.; Nikolenko, S. Approximate Packet Classifiers With Controlled Accuracy. *IEEE/ACM Trans. Netw.* **2021**, *29*, 1141–1154. [CrossRef]
22. Chuprikov, P.; Demianiuk, V.; Gorinsky, S. PREDICAT: Efficient Packet Classification via Prefix Disjointness. In Proceedings of the ICCCN, Athens, Greece, 19–22 July 2021; pp. 1–11.
23. Demianiuk, V.; Kogan, K. How to Deal with Range-Based Packet Classifiers. In Proceedings of the SOSR, San Jose, CA, USA, 3–4 April 2019; pp. 29–35.
24. Yingchareonthawornchai, S.; Daly, J.; Liu, A.X.; Torng, E. A Sorted-Partitioning Approach to Fast and Scalable Dynamic Packet Classification. *IEEE/ACM Trans. Netw.* **2018**, *26*, 1907–1920. [CrossRef]
25. Liang, E.; Zhu, H.; Jin, X.; Stoica, I. Neural Packet Classification. In Proceedings of the ACM SIGCOMM, Beijing, China, 19–24 August 2019; pp. 256–269.
26. Qi, Y.; Fong, J.; Jiang, W.; Xu, B.; Li, J.; Prasanna, V. Multi-dimensional packet classification on FPGA: 100 Gbps and beyond. In Proceedings of the FPT—IEEE, Beijing, China, 8–10 December 2010; pp. 241–248.
27. Jiang, W.; Prasanna, V.K. Scalable packet classification on FPGA. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2012**, *20*, 1668–1680. [CrossRef]
28. Qu, Y.R.; Zhang, H.H.; Zhou, S.; Prasanna, V.K. Optimizing many-field packet classification on FPGA, multi-core general purpose processor, and GPU. In Proceedings of the ACM/IEEE ANCS, Oakland, CA, USA, 7–8 May 2015; pp. 87–98.

29. Chang, Y.K.; Hsueh, C.S. Range-Enhanced Packet Classification Design on FPGA. *IEEE Trans. Emerg. Top. Comput.* **2016**, *4*, 214–224. [CrossRef]
30. Che, H.; Wang, Z.; Zheng, K.; Liu, B. DRES: Dynamic Range Encoding Scheme for TCAM Coprocessors. *IEEE Trans. Comput.* **2008**, *57*, 902–915. [CrossRef]
31. Chang, Y.K.; Lee, C.I.; Su, C.C. Multi-field Range Encoding for Packet Classification in TCAM. In Proceedings of the IEEE INFOCOM, Shanghai, China, 10–15 April 2011; pp. 196–200.
32. Bremler-Barr, A.; Hendler, D. Space-Efficient TCAM-Based Classification Using Gray Coding. In Proceedings of the IEEE INFOCOM, Anchorage, AK, USA, 6–12 May 2007; pp. 1388–1396.
33. Bremler-Barr, A.; Hay, D.; Hendler, D. Layered Interval Codes for TCAM-Based Classification. In Proceedings of the IEEE INFOCOM, Rio de Janeiro, Brazil, 19–25 April 2009; pp. 1305–1313.
34. Banerjee-Mishra, T.; Sahni, S. DUOS—Simple Dual TCAM Architecture for Routing Tables with Incremental Update. In Proceedings of the IEEE ISCC, Riccione, Italy, 22–25 June 2010; pp. 503–508.
35. Wan, Y.; Song, H.; Che, H.; Xu, Y.; Wang, Y.; Zhang, C.; Wang, Z.; Pan, T.; Li, H.; Jiang, H.; et al. FastUp: Fast TCAM Update for SDN Switches in Datacenter Networks. In Proceedings of the IEEE ICDCS, Washington DC, USA, 7–10 July 2021; pp. 887–897.
36. Wan, Y.; Song, H.; Xu, Y.; Zhang, C.; Wang, Y.; Liu, B. Adaptive Batch Update in TCAM: How Collective Optimization Beats Individual Ones. In Proceedings of the IEEE INFOCOM, Vancouver, BC, Canada, 10–13 May 2021; pp. 1–10.
37. Zhao, B.; Li, R.; Zhao, J.; Wolf, T. Efficient and Consistent TCAM Updates. In Proceedings of the IEEE INFOCOM, Beijing, China, 27–30 April 2020; pp. 1241–1250.
38. Kogan, K.; Nikolenko, S.; Rottenstreich, O.; Culhane, W.; Eugster, P. SAX-PAC (Scalable And eXpressive PAcket Classification). In Proceedings of the ACM SIGCOMM, Chicago, IL, USA, 17–22 August 2014; pp. 15–26.
39. Sheu, J.P.; Chuo, Y.C. Wildcard Rules Caching and Cache Replacement Algorithms in Software-Defined Networking. *IEEE Trans. Netw. Serv. Manag.* **2016**, *13*, 19–29. [CrossRef]
40. Wan, Y.; Song, H.; Xu, Y.; Wang, Y.; Pan, T.; Zhang, C.; Liu, B. T-cache: Dependency-free Ternary Rule Cache for Policy-based Forwarding. In Proceedings of the IEEE INFOCOM, Beijing, China, 27–30 April 2020; pp. 536–545.
41. Dong, Q.; Banerjee, S.; Wang, J.; Agrawal, D.; Shukla, A. Packet classifiers in ternary CAMs can be smaller. In Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems, Saint Malo, France, 26–30 June 2006; pp. 311–322.
42. Liu, A.X.; Meiners, C.R.; Torng, E. TCAM Razor: A systematic approach towards minimizing packet classifiers in TCAMs. *IEEE/ACM Trans. Netw.* **2009**, *18*, 490–500. [CrossRef]
43. Lin, H.T.; Wang, P.C. TCAM-Based Packet Classification Using Multi-stage Scheme. In Proceedings of the Fifth ICNCC, Kyoto, Japan, 17–21 December 2016; ACM: Frisco, TX, USA, 2016; pp. 83–87.
44. Lin, H.T.; Wang, P.C. TCAM-based packet classification for many-field rules of SDNs. *Comput. Commun.* **2023**, *203*, 89–98. [CrossRef]
45. Cauteruccio, F.; Stamile, C.; Terracina, G.; Ursino, D.; Sappey-Mariniery, D. An automated string-based approach to white matter fiber-bundles clustering. In Proceedings of the 2015 International Joint Conference on Neural Networks (IJCNN), Killarney, Ireland, 12–17 July 2015; IEEE: New York, NY, USA, 2015; pp. 1–8.
46. Cauteruccio, F.; Terracina, G.; Ursino, D. Generalizing identity-based string comparison metrics: Framework and techniques. *Knowl. Based Syst.* **2020**, *187*, 104820. [CrossRef]
47. Kannan, K.; Banerjee, S. Compact TCAM: Flow entry compaction in TCAM for power aware SDN. In Proceedings of the International Conference on Distributed Computing and Networking, Philadelphia, PA, USA, 8–11 July 2013; Springer: Berlin/Heidelberg, Germany, 2013; pp. 439–444.
48. Alsaeedi, M.; Mohamad, M.M.; Al-Roubaiey, A.A. Toward adaptive and scalable OpenFlow-SDN flow control: A survey. *IEEE Access* **2019**, *7*, 107346–107379. [CrossRef]
49. 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.* **2021**, *34*, e4815. [CrossRef]
50. Yu, F.; Katz, R.H.; Lakshman, T. Efficient Multimatch Packet Classification and Lookup with TCAM. *IEEE Micro* **2005**, *25*, 50–59.
51. Yu, F.; Lakshman, T.V.; Motoyama, M.A.; Katz, R.H. SSA: A power and memory efficient scheme to multi-match packet classification. In Proceedings of the ACM ANCS 2005, Princeton, NJ, USA, 26–28 October 2005; pp. 105–113.
52. Lakshminarayanan, K.; Rangarajan, A.; Venkatachary, S. Algorithms for Advanced Packet Classification with Ternary CAMs. *SIGCOMM Comput. Commun. Rev.* **2005**, *35*, 193–204. [CrossRef]
53. Uga, M.; Shiomoto, K. A novel ultra high-speed multi-layer table lookup method using TCAM for differentiated services in the Internet. In Proceedings of the IEEE HPSR, Dallas, TX, USA, 29–31 May 2001; pp. 240–244.
54. Garey, M.R.; Johnson, D.S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*; W. H. Freeman & Co.: New York, NY, USA, 1990.
55. Banerjee, T.; Sahni, S.; Seetharaman, G. PC-TRIO: A power efficient TCAM architecture for packet classifiers. *IEEE Trans. Comput.* **2015**, *64*, 1104–1118. [CrossRef]
56. Taylor, D.E. Survey and Taxonomy of Packet Classification Techniques. *ACM Comput. Surv.* **2005**, *37*, 238–275. [CrossRef]
57. Taylor, D.E.; Turner, J.S. ClassBench: A Packet Classification Benchmark. In Proceedings of the IEEE INFOCOM, Miami, FL, USA, 13–17 March 2005; Volume 3, pp. 2068–2079.

58. Jouet, S.; Cziva, R.; Pezaros, D.P. Arbitrary packet matching in OpenFlow. In Proceedings of the IEEE 16th HPSR, Budapest, Hungary, 1–4 July 2015; pp. 1–6.
59. Agrawal, B.; Sherwood, T. Ternary CAM Power and Delay Model: Extensions and Uses. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2008**, *16*, 554–564. [CrossRef]