

Tree-Based Minimization of TCAM Entries for Packet Classification

Yan Sun and Min Sik Kim

School of Electrical Engineering and Computer Science

Washington State University

Pullman, Washington 99164-2752, U.S.A.

Email: {ysun,msk}@eecs.wsu.edu

Abstract—Packet classification is a fundamental task for network devices such as edge routers, firewalls, and intrusion detection systems. Currently, most vendors use Ternary Content Addressable Memories (TCAMs) to achieve high-performance packet classification. TCAMs use parallel hardware to check all rules simultaneously. Despite their high speed, TCAMs have a fundamental in dealing with ranges efficiently. Many packet classification rules contain range specifications, each of which needs to be translated into multiple prefixes to store in TCAMs. Such translation may result in an explosive increase in the number of required TCAM entries. In this paper, we propose a redundancy removal algorithm using a tree representation of rules. The proposed algorithm removes redundant rules and combines overlaying rules to build an equivalent, smaller rule set for a given packet classifier. This equivalent transformation can significantly reduce the number of required TCAM entries. Our experiments show a reduction of 70.9% in the number of TCAM entries. Besides, our algorithm eliminates requirement of priority encoder circuits. It can also be used as a preprocessor, in tandem with other methods, to achieve further performance improvement.

I. INTRODUCTION

There are a number of network services that require packet classification, such as policy-based routing, firewalls, provision of differentiated qualities of service, and traffic billing. In each case, it is necessary to determine which flow an arriving packet belongs to so as to determine, for example, where to forward it, whether to forward or filter it, what class of service it should receive, or how much should be charged for transporting it. As packet classification has been widely deployed on the Internet, demand for efficient packet classification grows. The function of a packet classification system is to map each packet to a decision according to a sequence of rules, which is called a packet classifier. The rules specified in a packet classifier may or may not be mutually exclusive; two rules may overlap in a packet classifier. When it happens with no explicit priorities specified, we follow the convention that a rule closer to the top of the list takes priority. Table I shows a simple packet classifier of four rules.

Perhaps the most popular method for high-speed packet classification in practice is to use Ternary Content Addressable Memory (TCAM) [1]. A TCAM is a memory chip where each entry can store a packet classification rule in ternary form. It stores data patterns in the form of (value, bit mask) pairs. A query key can be simultaneously compared against all the

patterns stored in a TCAM. A key q is said to match a stored pattern (v, m) if $q \& m = v \& m$, where “&” is the bit-wise logical AND operator. Given a packet, the TCAM hardware can compare the packet with all stored rules in parallel and then return the decision of the first rule that the packet matches through a priority encoder. Thus, it takes $O(1)$ time to find the decision for any given packet. Because of their high speed, TCAMs have become the industrial standard for high speed packet classification [2].

Despite their high speed, TCAMs have two major drawbacks when used in packet classifiers. First, they consume a large amount of power and have high hardware cost. Thus, their capacity in packet classifiers is often limited. Second, they are inefficient when applied to packet classifiers with port number ranges, because TCAMs can only store rules in ternary form, which means that port numbers need to be converted to one or more prefixes before being stored in TCAMs. This may lead to a significant increase in the number of TCAM entries needed to encode a rule. For example, 30 prefixes are needed to represent a single range [1, 65534], and 20 prefixes are needed to represent [1, 2046]. Overall, $30 \times 20 = 600$ TCAM entries are required to represent a single rule with these two ranges. We observe that packet classifiers typically have at most one port range in each rule, and rules specifying two port ranges are very rare. However, a small number of such rules can consume most of the TCAM entries and the number of such rules is increasing. Therefore, minimizing the number of required TCAM entries is crucial.

In this paper, we propose a tree-based algorithm to minimize the number of TCAM entries used by a packet classifier. An effective way to reduce the number of TCAM entries is to remove redundant rules and combine overlaying rules, yielding an equivalent set of rules without affecting the classification results. Our algorithm is based on such an equivalent transformation. It can reduce the number of TCAM entries significantly while preserving the behavior of packet classifiers. A key advantage of reducing TCAM entries through this algorithm is that the algorithm can be easily deployed without any modification of existing packet classification systems. In our experiments using the Snort [3] rule sets, we achieve a reduction of 70.9% in the number of TCAM entries by removing redundant rules and combining overlaying rules.

The remainder of the paper is organized as follows. Sec-

TABLE I
A SIMPLE HEADER RULE SET

Rule	Type	Source IP	Destination IP	Source Port	Destination Port	Decision
r_1	TCP	*	192.168.0.0/16	<1024	*	accept
r_2	TCP	*	192.168.14.1	*	139	discard
r_3	UDP	192.168.0.0/16	*	*	700:900	accept
r_4	TCP	*	*	*	*	discard

tion II presents previous work related to this paper. Section III discusses redundancy in packet classifiers, and Section IV proposes our tree-based algorithm to reduce the number of TCAM entries. Then the experimental results are presented in Section V. Finally, we conclude in Section VI.

II. RELATED WORK

Previous work exploring solutions to deal with the range expansion problem falls into two major categories: hardware-based solutions, which require changing TCAM hardware circuits, and software-based solutions, which do not require such changes. Next, we review previous work in these two categories.

a) Hardware-based solutions: The basic idea of hardware-based solutions is to modify TCAM circuits and architecture [2], [4]–[7]. For example, van Lunteren et al. proposed a method of adding comparators at each entry to better accommodate range matching in packet classifiers [6]. While this allows to use TCAMs more efficiently, any solution from this research line has some drawbacks such as the cost of hardware modification.

b) Software-based solutions: Many software-based solutions have been proposed [8]–[12]. Their basic idea is to preprocess ranges that appear in a packet classifier or convert a given packet classifier to another semantically-equivalent packet classifier that requires fewer TCAM entries, and then store the new rule set in a TCAM. Hence, the TCAM circuits need not be modified to implement range storage, but the preprocessing is required. Although these methods can efficiently reduce the redundancy in the rule set, they may still miss some redundancy. Our work falls into this category and further optimizes existing approaches. Software-based solutions are more likely to be adopted by networking vendors and ISPs because they do not require changing TCAM hardware or existing packet classification systems.

III. REDUNDANCY IN PACKET CLASSIFIERS

Packet classifiers usually check the following five fields in each packet header: protocol type, source port number, destination port number, source IP address, and destination IP address. The lengths of these fields are 8 (bits), 16, 16, 32, and 32, respectively. When a TCAM is used to implement a packet classifier, all rules stored in TCAM entries must be represented as exact binary values or binary values with wildcard bits (e.g., suffix digits in an IP address prefix). However, in a typical packet classification rule, some fields such as source and destination port numbers are represented as integer ranges rather than exact values or prefix values. Thus, we need to

convert a rule with fields represented as integer ranges into one or more prefix rules, which may lead to range expansion. During the process of range expansion, each field of a rule should be expanded separately. For example, if a 3-bit field of a rule is [1, 6], the corresponding minimum set of prefixes covering the range are 001, 01*, 10*, and 110. The worst-case range expansion of a w -bit integer range yields $2w - 2$ prefixes [13]. The next step is to compute the cross-product of obtained prefix sets, resulting in an exponential increase of number of prefixes needed to replace a single rule.

A sequence of rules (r_1, r_2, \dots, r_n) is *complete* if and only if for any packet p , there is at least one matching rule in the sequence. To guarantee completeness, the last rule is typically configured to cover all the ranges, so that each packet matches at least one rule. Two rules in a packet classifier may overlap, which means that one packet may match two or more rules. Besides, two rules in a packet classifier may conflict with each other. In other words, two overlapping rules may have different decisions. Packet classifiers typically resolve conflicts by employing the first match, which has higher priority. For firewalls, typical decisions include “accept,” “discard,” “accept with logging,” and “discard with logging.”

There are four types of redundancy in a rule set:

- *Backward redundancy:* A rule r in a packet classifier is *backward redundant* if and only if there exists another rule r' listed above r such that all packets that match r also match r' .
- *Forward redundancy:* A rule r in a packet classifier is *forward redundant* if and only if there exists another rule r' listed below r such that the following three conditions hold: (i) all packets that match r also match r' , (ii) r and r' have the same decision, and (iii) for each rule r'' listed between r and r' , either r and r'' have the same decision, or no packet matches both r and r'' .
- *Multi-part redundancy:* A rule r in a packet classifier is *multi-part redundant* if and only if r can be replaced with a set of new rules, where some of them are backward redundant and the rest are forward redundant.
- *Overlapping redundancy:* A rule r in a packet classifier has *overlapping redundancy* if and only if there exists another rule r' listed above r such that their ranges overlap with each other and r is not a redundant rule. This type of redundancy is different from the three aforementioned types, for only the overlapping portion of the rule is redundant.

For example, a simple packet classifier with three rules is shown in Fig. 1. The rule r_2 is forward redundant, and r_3

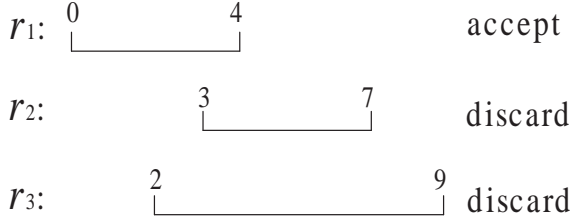


Fig. 1. A simple packet classifier

- $r_1 : F_1 \in [0, 4] \wedge F_2 \in [0, 9] \rightarrow \text{accept}$
- $r_2 : F_1 \in [0, 4] \wedge F_2 \in [4, 9] \rightarrow \text{accept}$
- $r_3 : F_1 \in [5, 9] \wedge F_2 \in [7, 9] \rightarrow \text{accept}$
- $r_4 : F_1 \in [5, 9] \wedge F_2 \in [0, 2] \rightarrow \text{discard}$
- $r_5 : F_1 \in [0, 9] \wedge F_2 \in [0, 9] \rightarrow \text{discard}$

Fig. 2. A simple packet classifier

has overlapping redundancy because of [2, 4].

IV. MINIMUM RANGE TREE

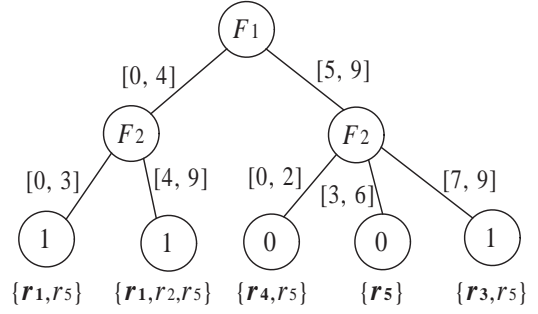
Our goal is reducing redundancy in a given packet classifier, including redundant rules and overlapping parts. To achieve the goal, we build a minimum range tree as follows.

A range tree T for a packet classifier $f: (r_1, r_2, \dots, r_n)$ over fields F_1, \dots, F_d is a tree that has the following properties:

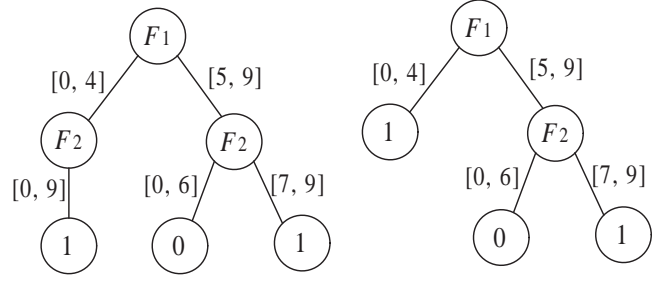
- The height of the tree is equal to the number of fields in the packet classifier.
- Edges of each depth of the tree store the ranges of the corresponding field. All edges in the same depth cover the whole range of the field, and there is no overlap between any pair of them.
- A directed path from a leaf node to the root is called a *decision path*. For a given packet, the tree has exactly one matched decision path.
- Each leaf node is labeled with decision that is associated with the corresponding decision path.

Fig. 3(a) shows a range tree for the simple packet classifier in Fig. 2. In this example, we assume every packet has only two fields, F_1 and F_2 , and the domain of each field is [0, 9].

We use number 1 as a shorthand for “accept” and number 0 as a shorthand for “discard” in labeling leaf nodes in Fig. 3. In Fig. 3, each edge represents a range in the corresponding field. We first build a tree as in Fig. 3(a) according to the packet classifier in Fig. 2, and the rules below each leaf node represent which rules satisfy the decision path. They are not a part of the tree. Then we combine two neighboring leaf nodes if they have the same decision and share the same parent node. The result is shown in Fig. 3(b). Last, we move a leaf node to its parent node if this leaf node is the only child node as shown in Fig. 3(c). As a result, we get the minimum range tree in Fig. 3(c). It corresponds to three new rules in Fig. 4, which are equivalent to the rules in Fig. 2.



(a)



(b)

(c)

Fig. 3. Constructing a minimum TCAM entry tree for the packet classifier in Fig. 2

- $r'_1 : F_1 \in [0, 4] \rightarrow \text{accept}$
- $r'_2 : F_1 \in [5, 9] \wedge F_2 \in [0, 6] \rightarrow \text{discard}$
- $r'_3 : F_1 \in [5, 9] \wedge F_2 \in [7, 9] \rightarrow \text{accept}$

Fig. 4. Three new rules according to Fig. 3(c)

In this example, we remove all redundancies in the leaf level. However, there may be redundancies among internal nodes. Therefore, we also need to remove such redundancies. After removing redundancies in leaf nodes, some of their parents may become leaf nodes. We apply the same algorithm recursively from leaf nodes to the root to check the redundancy for them.

There is another type of redundancy that is not removed by the previous algorithm. After applying the previous algorithm, it is obvious that if two leaf nodes still have the same decision and have the same field range, then these two nodes must have different parent nodes. If these two nodes have the same grandparent node, it implies that there must be redundancy in their parent nodes, which we call parent redundancy. However, we cannot reduce this kind of redundancy through the algorithm we discussed above; we need to search for parent redundancy in a separate step. Fig. 5 shows a simple example of removing parent redundancy.

In Fig. 5(a), the first leaf node and the third leaf node have the same decision (0), the same field range ([0, 2]), and the same grandparent node (F_1). Therefore, there is redundancy between their parent nodes (two F_2). In this case, we delete the

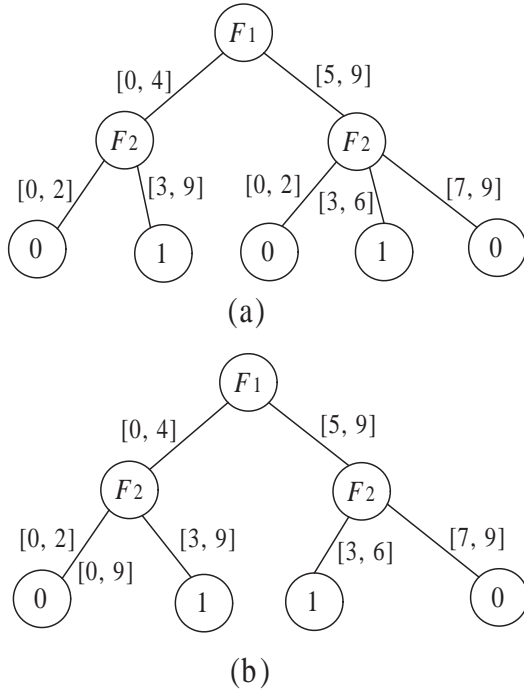


Fig. 5. An example of removing parent redundancy

second leaf node and combine their parent nodes' field ranges ([0, 4] and [5, 9]) so that it becomes the second field range of the first leaf node as shown in Fig. 5(b). This new range overrides its parent's field range [0, 4], yielding the following decision path:

$$F_2 \in [0, 2] \wedge F_1 \in [0, 9] \rightarrow \text{discard}. \quad (1)$$

From the discussion above, in order to reduce TCAM entries, we have the following three strategies:

- 1) Reduce redundant rules,
- 2) Reduce overlapping between rules, and
- 3) Combine rules even there is no overlapping.

These three strategies are based on the following observations:

- The number of prefixes increases as the range length increases.
- If two ranges, $[a, b]$ and $[c, d]$, are overlapping or have no gap between them, the number of prefixes needed by them is larger than the number of prefixes needed by the combined range $[a, d]$.

Table II shows the number of prefixes needed by ranges with different lengths varying from 1 to 10. We can see in the second column that ranges with the same length may need different numbers of prefixes. We compute the frequency of each number of prefixes in the third column, assuming that ranges are uniformly distributed. Using the frequency as weight, we obtain the average number of prefixes for each range length in the last column. The average number of prefixes increases as the range length increases. Note that

TABLE II
NUMBER OF PREFIXES NEEDED BY A RANGE

Range length	Possible numbers of prefixes required to represent the range	Frequency ratio of required prefix numbers	Average number of required prefixes
1	1	1	1
2	1, 2	1:1	1.5
3	2	1	2
4	1, 2, 3	1:1:2	2.25
5	2, 3	1:1	2.5
6	2, 3, 4	2:1:1	2.75
7	3	1	3
8	1, 2, 3, 4	1:1:2:4	3.125
9	2, 3, 4	1:1:2	3.25
10	2, 3, 4, 5	2:3:1:2	3.375

TABLE III
ANALYSIS OF SNORT RULES

Field	Number of distinct values	Most frequently used values	Number of rules
Protocol type	4	TCP	7550
		UDP	490
		ICMP	135
		IP	39
Destination port	314	445	1574
		\$HTTP_PORTS	1568
		any	1564
		139	1464
		\$ORACLE_PORTS	291
Source port	196	any	7056
		\$HTTP_PORTS	737
		1024	43
Destination IP	15	\$HOME_NET	5519
		\$EXTERNAL_NET	1220
		\$HTTP_SERVERS	959
Source IP	11	\$EXTERNAL_NET	6952
		\$HOME_NET	1198
		any	28

the sum of two average lengths is larger than the average length of the sum of their range lengths. For example, the average number of prefixes for a range of length 3 is 2 and that for a range of length 4 is 2.25. However, if we combine two ranges into one of length 7, the corresponding average number of prefixes is 3. Thus, the compression ratio becomes $(2 + 2.25 - 3)/(2 + 2.25) = 29\%$. For longer ranges, the average number of prefixes grows logarithmically.

In what order we should use the fields to construct the tree, starting from the root, is another issue we need to address. In order to reduce the redundancy effectively, the complexity of fields should increase as we move from the root to leaf nodes. As a case study, Table III presents the analysis of the complete set of the 8214 Snort rules [14].

While IP addresses have only a few distinct values, port numbers have hundreds of distinct values. In addition, destination IP addresses and port numbers have more distinct values than source IP addresses and port numbers, respectively. Therefore, in order to build the most efficient tree, the protocol type, which is the simplest, should be used in the root level, and source IP address, destination IP address, source port number, and destination port number should follow in that order. The same tendency is observed in other rule sets we

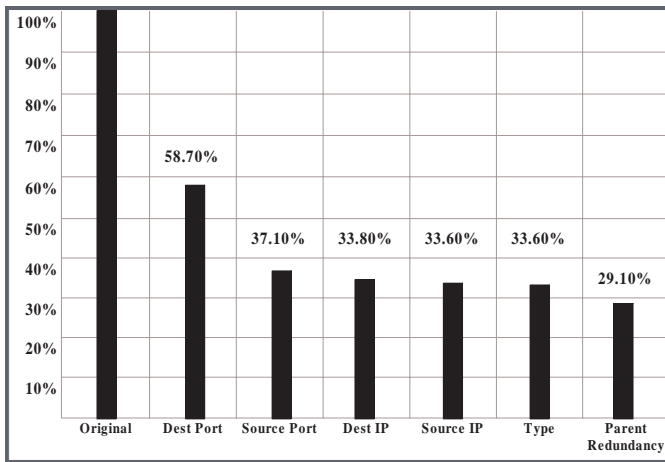


Fig. 6. Ratio of TCAM entries left after redundancy removal

used in experiments.

After building the minimum range tree, we can extract the decision path from each leaf node to the root. The number of decision paths is equal to the number of leaf nodes. Note that these decision paths cover the entire ranges and there is no overlap. Then we convert the ranges along each decision path into prefixes and store them in a TCAM. When a packet enters the TCAM for comparison, the output bits contain only a single 1 indicating a match, and all the other bits are 0s indicating no match. Therefore, priority encoder circuits (PEC) are not required in our algorithm. We use the output of the TCAM as a mask, and perform an AND operation between the mask and the decision vector, which is a binary vector with 1 representing “accept” and 0 representing “discard.” If the result is zero, this packet is discarded; otherwise it is accepted.

Some applications may have very limited TCAM storage to store all the prefixes even after the number of prefixes is reduced by our algorithm. Even in such cases, our algorithm can still be used as a preprocessor, making actual mapping between prefixes and TCAM entries by other methods easier.

V. EXPERIMENTAL RESULTS

For experiments, we collected rules from actual packet classifiers. Because rule sets vary across different applications, we gathered as many rules as we could and then randomly selected one thousand rules from them. We compared the number of TCAM entries used by original rules and the number of TCAM entries used after reducing redundancies with our algorithm. The result is shown in Fig. 6.

After reducing redundancies for each level of the tree, starting from leaf nodes, we get the percentage of TCAM entries left. We can see that the redundancy of the destination port field alone affects the number of required TCAM entries most; 41.3% of TCAM entries are reduced. On the other hand, the IP address fields have only a small impact, and the reduction by the protocol type field is almost negligible. Finally, the parent redundancy removal reduces 4.5% TCAM entries. Overall, our algorithm reduces 70.9% of TCAM entries in this experiment.

For comparison with other existing approaches, we selected one of the existing algorithms, which was proposed by Singh et al [10]. Using the same rules, our approach reduces 19.3% (70.9% – 51.6%) more TCAM entries. Furthermore, our algorithm eliminates priority encoder circuits used to process the TCAM output.

VI. CONCLUSION

In this paper, we proposed a tree-based redundancy removal algorithm to remove redundant rules and combine overlapping rules to build a new rule set in a packet classifier. According to our experiments using randomly selected rules from actual packet classifiers, this equivalent transformation can significantly reduce the number of TCAM entries needed by a packet classifier. The experiment shows a reduction of 70.9% in the number of TCAM entries. Furthermore, priority encoder circuits used in other TCAM-based methods are not required because exactly one entry matches in our algorithm. Our algorithm can also be used as a preprocessor, in tandem with other methods, to achieve further performance improvement.

REFERENCES

- [1] K. Pagiamtzis and A. Sheikholeslami, “Content-addressable memory (CAM) circuits and architectures: A tutorial and survey,” *IEEE Journal of Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, Mar. 2006.
- [2] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary, “Algorithms for advanced packet classification with ternary CAMs,” in *Proceedings of ACM SIGCOMM ’05*, Aug. 2005, pp. 193–204.
- [3] *Snort User Manual 2.8.5*, The Snort Project, Sep. 2009, available as http://www.snort.org/assets/120/snort_manual.pdf.
- [4] H. Liu, “Efficient mapping of range classifier into ternary-CAM,” in *Proceedings of the 10th Symposium on High Performance Interconnects*, Aug. 2002, pp. 95–100.
- [5] J. van Lunteren, T. Engbersen, and S. Member, “Fast and scalable packet classification,” *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 4, pp. 560–571, May 2003.
- [6] E. Spitznagel, D. Taylor, and J. Turner, “Packet classification using extended TCAMs,” in *Proceedings of the 11th IEEE International Conference on Network Protocols*, Nov. 2003, pp. 120–131.
- [7] M. Faezipour and M. Nourani, “CAM01-1: A customized TCAM architecture for multi-match packet classification,” in *IEEE Global Telecommunications Conference*, Nov. 2006, pp. 1–5.
- [8] F. Baboescu, S. Singh, and G. Varghese, “Packet classification for core routers: Is there an alternative to cams?” in *Twenty-Second Annual Joint Conference of the IEEE Computer and Communications*, Mar. 2003, pp. 53–63.
- [9] S. Singh, F. Baboescu, G. Varghese, and J. Wang, “Packet classification using multidimensional cutting,” in *Proceedings of ACM SIGCOMM ’03*, Aug. 2003, pp. 213–224.
- [10] A. X. Liu, C. R. Meiners, and Y. Zhou, “All-match based complete redundancy removal for packet classifiers in TCAMs,” in *Proceedings of the 27th IEEE INFOCOM*, Apr. 2008, pp. 111–115.
- [11] D. A. Applegate, G. Calinescu, D. S. Johnson, H. Karloff, K. Ligett, and J. Wang, “Compressing rectilinear pictures and minimizing access control lists,” in *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, Jan. 2007, pp. 1066–1075.
- [12] Q. Dong, S. Banerjee, J. Wang, D. Agrawal, and A. Shukla, “Packet classifiers in ternary CAMs can be smaller,” in *Proceedings of ACM SIGMETRICS/Performance 2006*, Jun. 2006, pp. 311–322.
- [13] P. Gupta and N. Mckeown, “Algorithms for packet classification,” *IEEE Network*, vol. 15, no. 2, pp. 24–32, Mar. 2001.
- [14] A. Yoshioka, S. H. Shaikot, and M. S. Kim, “Rule hashing for efficient packet classification in network intrusion detection,” in *Proceedings of the 17th IEEE International Conference on Computer Communications and Networks*, Aug. 2008.