Multi-match Packet Classification Scheme Combining TCAM with an Algorithmic Approach

Hysook Lim*, Nara Lee, and Jungwon Lee

[1] Department of Electronic and Electrical Engineering, Ewha Womans University / Seoul, Korea hlim@ewha.ac.kr
[2] LG electronics Incorporated / Seoul, Korea detlev@hanmail.net
[3] Department of Electronic and Electrical Engineering, Ewha Womans University / Seoul, Korea jungwon0736@gmail.com

* Corresponding Author: Hyesook Lim

Received November 30, 2016; Revised December 28, 2016; Accepted January 10, 2017; Published February 28, 2017

** Regular Paper*

***Abstract*:** Packet classification is one of the essential functionalities of Internet routers in providing quality of service. Since the arrival rate of input packets can be tens-of-millions per second, wire-speed packet classification has become one of the most challenging tasks. While traditional packet classification only reports a single matching result, new network applications require multiple matching results. Ternary content-addressable memory (TCAM) has been adopted to solve the multi-match classification problem due to its ability to perform fast parallel matching. However, TCAM has a fundamental issue: high power dissipation. Since TCAM is designed for a single match, the applicability of TCAM to multi-match classification is limited. In this paper, we propose a cost- and energy-efficient multi-match classification architecture that combines TCAM with a tuple space search algorithm. The proposed solution uses two small TCAM modules and requires a single-cycle TCAM lookup, two SRAM accesses, and several Bloom filter query cycles for multi-match classifications.

***Keywords*:** Packet classification, TCAM, Tuple space search, Power consumption, Bloom filter, Hashing

1. Introduction

In designing an efficient packet-forwarding engine, packet classification is an essential prerequisite for routers when providing the different levels of service required by various Internet applications [1]. Packet classification determines the class of each packet using a set of header fields so that routers process the packets using the service defined for the class of each packet [2-24].

Most traditional applications require longest prefix or highest priority matching. However, the multi-match classification concept is becoming a major research item because of the increasing need for network security from systems such as network intrusion detection or worm detection, or in new application programs, such as load balancing and packet-level accounting [5-14]. As a network intrusion detection system (NIDS) example, a packet may match multiple rule headers, and related rule options for all the matching rule headers need to be identified in order to be checked later. For accounting, multiple counters may need to be updated for a given packet, and hence multi-match classification is necessary in order to identify relevant counters for each packet [11].

To provide wire-speed packet forwarding, IP address lookup and packet classification functions are designed using application-specific integrated circuits (ASICs) with off-chip ternary content-addressable memory (TCAM) storing prefix sets or rule databases [5]. Using TCAM is the best solution for providing single-cycle operation.

However, the applicability of TCAM is restricted by several intrinsic issues. The primary issue is huge power consumption. TCAM consumes 150 times more power per bit than static random access memory (SRAM). TCAM consumes around 30 to 40 percent of the total line card power [6, 7]. When line cards are stacked together, TCAM imposes a high cost on the cooling system. TCAM also costs about 30 times more per bit of storage than double data rate (DDR) SRAM. TCAM's power consumption is directly related to the number of TCAM accesses, and grows linearly with the number of entries searched in parallel [12]. In order to be energy- and cost-efficient, TCAM-based solutions must use an economic TCAM size and perform a limited number of TCAM lookups for each packet. Studies have been conducted into many different solutions, such as reducing TCAM size, reducing the number of entries activated when a lookup is performed, or limiting the number of TCAM lookups required for each operation.

Another issue is related to port ranges in the classification field. A port range has to be converted into multiple prefixes, and hence, multiple TCAM entries are needed to store a rule possessing port-range fields. The third issue is that TCAM is not directly applicable to a multi-match problem, since it was designed for a single best match. In applying TCAM to the multi-match problem, several different approaches have been proposed, but most of these approaches have their own issues.

Various algorithms have been developed to perform packet classification without using TCAM [15-24]. Among these algorithmic approaches, tuple space–based [15, 16] or decision tree–based [17, 18] algorithms provide multi-match packet classification at the same time as single best-match packet classification.

The purpose of this paper is to present a power-efficient multi-match packet classification architecture. We focus on two issues related to the multi-match classification problem. TCAM produces a single best match, and TCAM consumes a great deal of power. It is necessary to have a general solution that works for both single-match and multi-match problems using TCAM. Therefore, changing TCAM hardware to produce multiple matches is not a viable option. Our approach follows the algorithmic approach, combining a tuple space–based algorithm with TCAM. These improvements come at the cost of multiple small SRAM modules that need to be accessed in parallel; however, this cost is inconsequential, given that SRAM is inexpensive and uses very small amounts of power.

This paper is organized as follows: Section 2 defines the packet classification problem. Section 3 presents related work, such as TCAM-based multi-match packet classification structures, tuple space–based algorithms, and Bloom filter theory. Section 4 describes the proposed multi-match packet classification architecture. Section 5 discusses simulation results and a performance analysis, and Section 6 concludes the paper.

2. Packet Classification Problem

Header fields used in packet classification usually include source IP address, destination IP address, source port number, destination port number, and protocol type. Let $\mathcal{R} = \{R_1, R_2, \cdots, R_N\}$ represent a classifier where $R_i\,(i = 1, \quad \cdots \ N)$ is a rule, and N is the number of rules. If each rule, R_i, has α attribute fields and one directive field, then $R_i = (R_i^1, R_i^2, \cdots, R_i^\alpha, A_i)$ where R_i^j (j = $1, \cdots, \alpha$) is the specification on the j th field. Each field, R_i^j, can be a variable-length prefix, a port range, or a fixed protocol value. We often refer to the j th field as the j th dimension. A_i is the directive associated with rule R_i.

When a packet arrives, the header values P^j (j = ,1 $\cdots$, α) from the relevant fields in the packet header are extracted, where P^j is a fixed-length bit string. A packet with header fields $P = (P^1, P^2, \cdots, P^\alpha)$ matches a rule $R_i = (R_i^1, R_i^2, \cdots, R_i^j, A_i)$ if matches R_i^j for all of the fields $j = 1, \cdots, \alpha$.

Each rule in a classifier has a priority index, and they are usually sorted in priority order. This priority index is necessary, because a packet can match more than one rule. In classifier $\mathcal{R}$, it is assumed that R_1 has the highest priority and R_N has the lowest priority.

All matching rules are returned when using multi-match packet classification; the best matching rule (BMR) with the highest priority is returned when using single-match packet classification. Let $\mathcal{M}(P)$ represent the set of rules that packet P matches. Let $I(S)$ represent the smallest priority index in the set of rules, S. For a given packet, $P = (P^1, P^2, \cdots, P^\alpha)$, in the set of rules found in classifier $\mathcal{R} = \{R_1, R_2, \cdots, R_N\}$, the multi-match packet classification problem is to find the set of matching rules, $\mathcal{M}(P)$; the single-match packet classification problem is to find the smallest priority index, $I(\mathcal{M}(P))$.

3. Related Work

3.1 Multi-match Architectures with TCAM

Even though TCAM has been popularly used in packet classification to achieve wire-speed packet forwarding, several issues are still unresolved. In addition to the required TCAM size, the most important issue is the power dissipation problem, which is closely related to the number of TCAM entries activated when a lookup occurs. Another issue is the rule replication problem related to the port number fields. The port number fields can hold a fixed number or a range specified by low and/or high boundaries. If a rule has a field specified by a range, the range has to be converted into multiple prefixes, and the rules with each converted prefix are then stored in TCAM entries. Since a port range can be converted into a maximum of 30 prefixes, if two port fields are specified by ranges, a maximum of 900 TCAM entries are required. Researchers have proposed range-encoding algorithms in order to improve storage efficiency [14].

A critical issue regarding TCAM being applied to the multi-match packet classification problem is that TCAM is designed to produce a single best match. Two different approaches to this issue are possible: modifying the TCAM hardware [8-10] or through algorithms [5-7, 11, 13].

The bit vector TCAM (BV-TCAM) [9] architecture combines a bit vector algorithm [19] to address the multi-match classification problem. Lakshman and Stidialis removed the priority encoder from TCAM to get an -bit vector after the TCAM lookup, where N is the number TCAM entries [19]. Each bit of the bit vector indicates the match to the corresponding rule. However, dealing with the N-bit vector to identify matching results is inefficient, especially when is large and the vector is sparse. The scheme proposed by Deng et al. uses a result encoder, which keeps all of the matching results one by one in exactly one conventional TCAM lookup period [10]. TCAM output without the priority encoder forms a bit vector, and a logic unit monitors the bit vector and locks the matching entry when the corresponding bit has a valid return.

Since TCAM should be used for both single-match and multi-match problems, the algorithmic approach (rather than modifying TCAM hardware) is a better solution. The current industrial solution for using TCAM for the multi-match problem is to use a valid bit for each entry. Every valid bit is set to an initial state. When a given input matches an entry, the valid bit of the entry is reset. The next TCAM lookup cycle will produce another matching result (if any) among the entries with a valid bit set. Therefore, this scheme produces k matching results with TCAM lookups. It requires six cycles to initialize all of the valid bits (when a new packet classification is started) and to reset the valid bit (when a match is produced from TCAM), and hence, this scheme requires a total of 7 cycles for k matching results.

The multi-match using discriminators (MUD) algorithm addresses the multi-match problem in TCAM by utilizing extra unused bits [5]. Each TCAM entry includes a discriminator (an index) value along with the rule fields. If a TCAM lookup cycle produces a single best match for a TCAM entry with a specific discriminator value, the search key of the next cycle lookup is expanded to include a prefix that corresponds to a range greater than the discriminator value. Therefore, the MUD algorithm does not require setting and resetting the valid bits, and produces matching results with k TCAM lookups. Since every entry is accessed for every cycle of a TCAM lookup, the algorithm consumes a large amount of power.

Using a geometric intersection scheme (GIS), Yu et al. [6] generated all of the intersection rules and stored them in TCAM in a compatible order. Using the index of single best matches reported by TCAM, SRAM is accessed to obtain all of the matching rules. Depending on the rule set characteristics, the number of newly created entries for the intersection can be more than 10 times that of the original rule set. This algorithm provides a single TCAM lookup for multi-match classification, but it is an expensive and power-hungry solution because of the newly created intersection entries. The set splitting algorithm (SSA [7, 11]) splits the rule set into multiple groups in order to reduce the number of intersection rules, and performs separate TCAM lookups in these groups in parallel. When a rule set is split into two groups, the removal of at least half of the intersection rules is guaranteed in this scheme. However, it still adds rules for partially overlapped rules in each group, and the pre-processing needed for the set splitting is quite complex.

Faezipour and Nourani [8, 13] proposed the partitioning of a rule set into multiple groups, but their approach is different in that power dissipation is reduced by enabling TCAM to search one partition while disabling the others. Their partitioning approach follows two steps: maximum partitioning and minimum partitioning (MX-MN). In the maximum intersection partitioning step, each partition holds the maximum intersections, and partitions are disjointed from each other. The next partitioning is based on the minimum intersection among the rules found in each partition. Each minimum partition is stored in separate TCAM. A contention resolver activates the maximum partition for each packet; the minimum partitions in the activated maximum partition are then accessed in parallel in order to produce multiple matching results. Activating only a portion of TCAM is a good idea for reducing power, but most partitions have very few entries, and most of the rules are positioned in the distinct rule collection. The required amount of separate TCAM is equal to the total number of partitions, and can be several thousand TCAMs.

3.2 Multi-match Algorithm based on Tuple Space

Since our proposed approach is based on tuple space, a tuple space–based algorithm [15] is described in detail. The tuple space–based algorithm defines a tuple as the length combination of plural fields. The algorithm searches each of the tuples using an efficient search method, such as hashing. However, since the number of tuples that need to be searched for each input packet can be excessive, it is preferable to use a subset of tuples compatible with the individual field matches; this method is called tuple pruning [15, 16]. Tuple-pruning algorithms reduce the number of tuples included in the search space for each given input packet through an individual field operation. A tuple is a combination of the field lengths. Prefix fields have variable lengths. A port range is converted into plural variable-length prefixes. A protocol field has either a length of 0 (the wildcard) or a length of 8 (the exact protocol value).

Let $\ell\left(R_i^j\right)$ be the length of the j th field of rule R_i. Then rule R_i is mapped to a tuple, $T(R_i) = (\ell(R_i^1), \ell(R_i^2), \cdots, \ell(R_i^a))$. The number of possible tuples is the product of the number of possible lengths of all of the fields. Let $\mathcal{U}$ be the universal set of tuples with five attributive fields $(\alpha = 5)$; then, the universal set $\mathcal{U}$ would be $\mathcal{U} = \{(j_1, j_2, j_3, j_4, j_5)|0 \le j_1, j_2 \le 32, 0 \le j_3, j_4 \le 16, j_5 = 0 \text{ or } 8\}$, since the prefix lengths can be 0 through 32, the port ranges can be 0 through 16, and the protocol field has length 0 or 8. However, since the number of distinct lengths tends to be small, the number of distinct combinations will be smaller than the number of all possible tuples. Therefore, for a given classifier, the number of active tuples (to which at least one rule is mapped) will be much smaller than the number of possible tuples.

Let $\mathcal{A}=\{T_1, T_2, \cdots\ T_\mu\}$ represent the set of active tuples of a given classifier, where μ is the number of active tuples. Since every rule mapped to a single tuple has the same combination of field lengths, by concatenating the bits of each field, the rules mapped to a tuple can be stored in a hash table. Let H_k (for $k=1, \quad \cdots \quad \mu$ be the hash table for active tuple T_k. For a given packet, the matching rules are identified by accessing each H_k for $k=1, \quad \cdots \quad \mu$.

However, since it could take a long time to access all of the hash tables for each packet, several optimization techniques have been developed to reduce the search space by reducing the number of tuples that need to be searched for each given input packet. Let $\mathcal{T}(P)$ be the set of tuples determined as the search space for a given packet, P; then, $\mathcal{T}(P) \subset \mathcal{A}$ and $\mathcal{T}(P)=\{T_{p1},\ T_{p2}\ \cdots\ T_{pl}\}$, where $l \le \mu$, and the matching rules will be identified more quickly by accessing the hash tables, H_{pi} for $i=,1 \ \cdots\ l$.

The tuple-pruning algorithm [15] uses individual field operations to reduce the search space for a given packet. Tuple pruning will be beneficial if the reduction in the tuple space afforded by pruning offsets the extra individual field operations. Each field operation is performed against distinct elements of the field, and identifies the matches. The matching lengths are used to identify the subset of tuples that are compatible with the individual matches. The search space of the packet is determined by combining the matching lengths obtained from each field operation.

For header field $P^j\ (j=1, \cdots, \alpha)$ of a given packet, P, let $\mathcal{L}(P^j)$ represent the set of prefix lengths that the header P^j matches. The set $\mathcal{L}(P^j)=\{\ell_1, \cdots\ \ell_v\}$ is identified by performing a 1-D lookup against the distinct elements in R_i^j (for $i=,1 \ \cdots\ N$ and $j=,1 \ \cdots\ \alpha$). Next, by calculating the cross product of all $\mathcal{L}(P^j)$ sets for $j=1, \cdots, \alpha$, the cross product set $\mathcal{C}(P)=\mathcal{L}(P^1) \otimes \cdots \otimes \mathcal{L}(P^\alpha)$ is obtained to compose the tuple space. The search space $\mathcal{T}(P)$ for a given packet P is defined as the intersection of active tuple set $\mathcal{A}$ and the cross product set $\mathcal{C}(P)$, i.e.:

$$\mathcal{T}(P)=\mathcal{A} \cap \mathcal{C}(P) \tag{1}$$

Note that $\mathcal{A}$ is determined by the given rule set $\mathcal{R}$ in the build process, whereas $\mathcal{C}(P)$ is obtained in the search process for each incoming packet, P. Both $\mathcal{A}$ and $\mathcal{C}(P)$ are subsets of $\mathcal{U}$, $\mathcal{T}(P) \subset \mathcal{A} \subset \mathcal{U}$ and $\mathcal{T}(P) \subset \mathcal{C}(P) \subset \mathcal{U}$, respectively.

It is known that using additional fields, such as port numbers and protocol type, does not improve the search performance in the tuple space–based algorithm [15], and therefore, the tuple-pruning approach is described in this paper as using two prefix fields. Table 1 shows a simplified rule set with two prefix fields as an example.

Table 1. The rule set example.

Rule No.	Source Prefix	Destination Prefix	Tuple Space
R_0	0*	1*	(1, 1)
R_1	1*	110*	(1, 3)
R_2	01*	00*	(2, 2)
R_3	0*	111*	(1, 3)
R_4	111*	110*	(3, 3)
R_5	010*	110*	(3, 3)

The given classifier example has an active tuple space of $\mathcal{A}=\{(1,1), (1,3), (2,2), (3,3)\}$; each rule is stored in the hash table of its corresponding tuple. As a simple example with four-bit source and destination addresses, assuming that packet $P=(0100, 1110)$ is given, the search procedure is as follows.

By performing the search for the source prefix field, we have matches in lengths 1, 2, and 3, i.e. $\mathcal{L}(P^1)=\{1 \quad \}$. Similarly, by performing a search for the destination prefix field, $\mathcal{L}(P^2)=\{1 \ \}$. The cross product of $\mathcal{L}(P^1)$ and $\mathcal{L}(P^2)$ produces the cross product set $\mathcal{C}(P)=\{(1,1), (1,3), (2,1), (2,3), (3,1), (3,3)\}$. By the intersection operation given in (1), the search space is determined as $\mathcal{T}(P)=\{(1,1), (1,3), (3,3)\}$.

The matching rules of the given packet are identified by accessing the hash tables of these three tuples.

3.3 Bloom Filter Theory

As a simple bit vector, a Bloom filter is a space-efficient probabilistic data structure that is used to test whether an element is a member of a set [25, 26]. Bloom filter theory supports two different operations: programming and querying. Programming is a procedure that stores membership information in a Bloom filter. Querying is a procedure that tests whether a given input is a member of the set.

Let M be the length of the Bloom filter, and let N be the number of elements programmed into the Bloom filter. The number of hash indices, K, used in programming the Bloom filter is known to be optimum when it has the following relationship [25]:

$$K=\frac{M}{2^{\log_2 N}} \ln 2 \tag{2}$$

Before programming, every bit location of the Bloom filter is initialized to zero. The procedure to program element x into the Bloom filter follows:

Algorithm 1. Bloom Filter Programming

```
1: Bloom_Filter_Programming( $x$ )
2: for ( $i$ = 1 to $K$ )
3:     BF[ $h_i(x)$ ] = 1;
```

For each element x included in a set, K different hashing indices, $h_i(x)$, are obtained in such a way that the resulting hash index m is in the range $0 \le m < M$. All of the bit locations corresponding to the K hash indices are set to 1 in the Bloom filter.

The procedure to query whether input y is a member of the set is as follows:

Algorithm 2. Bloom Filter Querying

```
1: Bloom_Filter_Querying ( $y$ )
2: for ( $i$ = 1 to $K$ )
3:     if (BF[ $h_i(y)$ ] = = 0) return negative;
4: return positive;
```

For input y , K hash indices are generated using the same hash functions that were used to program the filter. The bit locations in the Bloom filter corresponding to the hash indices are checked. If at least one of the locations is 0, then it is absolutely not a member of the set. If it was a member of the set, the bit location corresponding to the hash index would have been set to 1 during programming. This result is negative. If all the bit locations are set to 1, it is positive. However, even if all these bit locations are set to 1, it does not mean that they were set only by the elements under query. It is possible that these locations could have been set by some other elements. This type of positive result is a *false positive*. On the whole, a Bloom filter may produce *false positives*, but does not produce *false negatives*.

4. Proposed Architecture

This section describes our approach to the multi-match packet classification problem. Our strategy is to design an efficient architecture that provides reduced power consumption and a high lookup speed. In regards to reducing the power requirements, TCAM size should be small, and the required number of TCAM lookups for each packet should be limited to one. Our approach combines the tuple space–based algorithm with TCAM.

The proposed architecture is composed of three parts: TCAM for each field, a tuple-pruning unit, and multiple small SRAM modules employing hash tables. In our approach, TCAM includes every distinct value of each field. The longest matching prefix is generated through the individual field search using TCAM. The issues facing us now are how to combine the results from each field search and to identify the matching rules. We use the tuple space approach to combine the individual search results, and we propose using a Bloom filter to remove unnecessary tuples. Hash tables using SRAM are accessed in parallel for the tuples determined as positive by the Bloom filter to produce matching results.

The tuple-pruning unit is composed of prefix vector SRAM (PV-SRAM), a bit-operation block, and a tuple Bloom filter. The prefix vector represents the prefix containments of distinct values of a field in a given rule set [27]; each PV-SRAM entry includes the prefix vector of the corresponding TCAM entry. For example, if there are prefixes 0*, 01*, and 010* in a field, the prefix vectors corresponding to those prefixes are 100, 110, and 111, respectively. The prefix vector 111 means that prefix 010* has nested prefixes of lengths 1, 2, and 3 (itself). The prefix vector is different from the bit vector [9, 19]; the size of a prefix vector is equal to the longest prefix (which is 32 bits for IPv4), whereas the size of a bit vector is equal to the number of rules [19] or the number of TCAM entries [9]. The bit-operation block calculates the cross product of the prefix vectors and the intersection with active tuples, as seen in (1). The tuple Bloom filter removes redundant tuples so that unnecessary SRAM accesses are avoided. Rules included in tuple space are stored in a hash table. A directive for rule treatment is also stored in the hash table along with each rule.

Fig. 1 shows the proposed architecture for the example rule set given in Table 1. For the same example packet, $P = (0100, 1110)$ the Source-TCAM and Destination-TCAM produce the best matching prefixes, 010* and 111*, respectively. The corresponding prefix vectors are 111 and 101, respectively. By calculating the cross product of these two prefix vectors and with the intersection operation, which is represented by a two-dimensional vector operation in the tuple-pruning unit shown in Fig. 1, the search space is determined as $\mathcal{T}(P) = \{(1,1),(1,3),(3,3)\}$.

The tuple-pruning algorithm is efficient in reducing the number of tuples to search; however, it can be further improved. In our example, accessing the hash table of tuple (3, 3) is unnecessary, since no rule in tuple (3, 3) of Table 1 matches the given input. Since the input matches the three-bit prefix of R_5 in the first field and the three-bit prefix of R_3 in the second field, the tuple (3, 3) was not excluded, even though there is no matching rule combining those two prefixes. The tuple (3, 3) in this example is termed a *false tuple*; false tuples are removed through the use of a simple Bloom filter in our proposed architecture. Let $\mathcal{B}(\mathcal{T}(P))$ represent the positive set obtained by applying the tuple Bloom filter to the tuple set, $\mathcal{T}(P)$. Since some false tuples are removed by the tuple Bloom filter, we obtain: $\mathcal{B}(\mathcal{T}(P)) \subset \mathcal{T}(P)$.

Any arbitrary hash generator can be used to program the Bloom filter. The Bloom filter needs to accommodate prefixes of arbitrary lengths in a single Bloom filter, and hence, it requires a hash generator that produces hash indices for variable-length prefixes. A cyclic redundancy check (CRC) generator has excellent characteristics for our purposes [21]. The CRC generator scrambles the bits of a

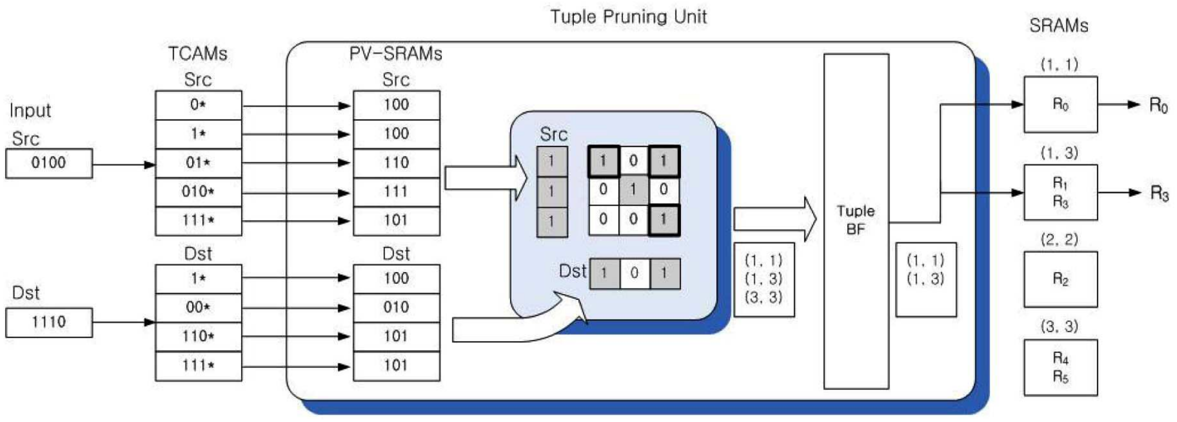

Fig. 1. The proposed multi-match packet classification architecture.

Table 2. The Bloom filter indices for the classifier given in Table 1.

Rule No.	Source Prefix	Destination Prefix	Tuple Space	Hash Key	CRC code	BF Indices
R_0	0*	1*	(1, 1)	01	11010011	26, 20, 19
R_1	1*	110*	(1, 3)	1110	00011010	3, 6, 26
R_2	01*	00*	(2, 2)	0100	11110100	30, 29, 20
R_3	0*	111*	(1, 3)	0111	10011101	19, 7, 29
R_4	111*	110*	(3, 3)	111110	00110010	6, 12, 18
R_5	010*	110*	(3, 3)	010110	10001001	17, 2, 9

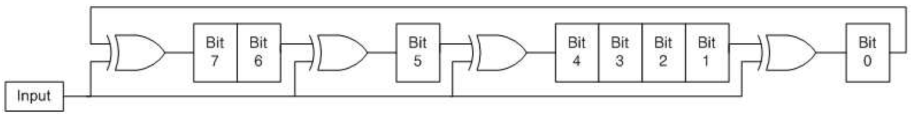

Fig. 2. Eight-bit CRC generator.

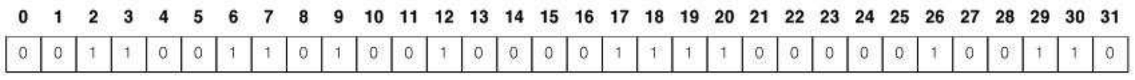

Fig. 3. Tuple Bloom filter programmed by rules in Table 2.

given input and produces a fixed-length binary sequence known as a CRC code, regardless of the prefix length. Any number of hash indices (each of which is used as a pointer to a Bloom filter bit location or to a hash table entry) can be obtained from the generated CRC code by selecting different combinations of bits.

For the classifier given in Table 1, we describe the procedure used to program and query the Bloom filter. An eight-bit CRC generator, as shown in Fig. 2, is used for programming and querying the Bloom filter in this example. Table 2 shows the hash key for the CRC generator, the CRC code corresponding to the hash key, and the Bloom filter indices. Registers of the CRC generator are assumed to be initialized by 10001001 in this example. The Bloom filter size, M, is assumed to be 32 bits, which is four times $2^{\log_2 N_p}$, where N_p is the number of distinct rules in source and destination prefix pairs from Table 1. The number of hash indices programming the Bloom filter is equal to three, according to (2). In this example, we arbitrarily select a set of bits as hash indices. The selected indices are bits [7:3], [6:2], and [4:0] of the CRC code. The Bloom filter programming is completed by setting the bits indicated by the column of Bloom filter indices from Table 2.

Fig. 3 shows the Bloom filter programmed by the indices shown in Table 2 for the example rules. For example packet $P = (0100, 1110)$, a query to the Bloom filter is performed for $\mathcal{T}(P) = \{(1,1),(1,3),(3,3)\}$. The hash key corresponding to the tuple (1, 1) is 01; the CRC code is 11010011. Therefore, the hashing indices for the tuple are 26, 20, and 19. Examining the bits of the Bloom filter located by these indices, we obtain a *positive* result. This means the possibility of a rule with (0*, 1*) in tuple (1, 1), and hence, the hash table needs to be accessed for this tuple. For the next tuple, (1, 3), the key and the corresponding CRC code is 0111 and 10011101, respectively, making the Bloom filter indices 19, 7, and 29. Examining the bits of the Bloom filter located by these indices, we obtain a *positive* result. Lastly, for the tuple (3, 3), the key and the corresponding CRC code are 010111 and 00111000, respectively. The Bloom filter indices are 7, 14, and 24. Examining the bits of the Bloom filter located by these indices, we obtain a *negative* result, since there is at least one zero in the bit locations. Hence, the tuple (3, 3) is identified as a *false tuple*, and it is not necessary to access the hash table for this tuple. In this manner, the search space determined by the proposed algorithm is reduced to $\mathcal{B}(\mathcal{T}(P)) = \{(1,1),(1,3)\}$.

Throughout this example, one false tuple was identified by the Bloom filter, and hence, one SRAM access was eliminated. The only drawback of the Bloom filter is that it can have false positives; the *false positive* rate of a Bloom filter can be controlled by increasing the size and number of indices, as will be shown in the simulation section.

SRAM is accessed in parallel for $\mathcal{B}(\mathcal{T}(P)) = \{(1,1),(1,3)\}$. Two matching rules are found in these two tuples; rule R_0 in tuple (1, 1) and rule R_3 in tuple (1, 3).

The search procedure for the proposed algorithm is summarized in Algorithm 3 for cases where two prefix fields are used for the tuple-space search. The other rule fields are stored in hash tables. Let P^1 and P^2 be the source and destination addresses, respectively, of a given input packet, P. Let $S(P^j, l)$ be the sub-string of the most significant l bits of P^j.

For the source and destination addresses of a given input packet, each TCAM produces an index for the longest matching prefix of each field. This index is used when accessing PV-SRAM. Using the prefix vector, prefix lengths matching each given address, $\mathcal{L}(P^1)$ and $\mathcal{L}(P^2)$, are obtained. By calculating the cross product of the matching lengths, a list of tuples, $\mathcal{C}(P)$, is generated. Bitwise AND the tuple list with the active tuple list of the given rule set to obtain $\mathcal{T}(P)$. For the tuples included in $\mathcal{T}(P)$, the Bloom filter is queried. The tuples determined positive by the tuple Bloom filter are included in $\mathcal{B}(\mathcal{T}(P))$. The hash tables are accessed for tuples in $\mathcal{B}(\mathcal{T}(P))$, and multiple matching rules are identified.

Since TCAM power dissipation is directly related to the number of entries and the number of TCAM lookups needed for a classification, it is ideal to have small TCAM modules and a limited number of TCAM lookups. Since only the distinct values of each field are stored in TCAM, the number of required TCAM entries is much smaller than found in previous approaches. Each TCAM is accessed once for each packet. Rules with entire rule fields are stored in SRAM so that the port ranges of rules are stored as they are, without being converted to multiple prefixes. This results in no rule replication in our proposed approach.

Algorithm 3. Search procedure

```
1: Search ( $P$ ) {
2:      $srcIndex$ = SrcTCAM ( $P^1$ );
3:      $dstIndex$ = DstTCAM( $P^2$ );
4:      $\mathcal{L}(P^1)$ = SrcPV-SRAM($srcIndex$);
5:      $\mathcal{L}(P^2)$ = DstPV-SRAM($dstIndex$);
6:      $\mathcal{C}(P) = \mathcal{L}(P^1) \otimes \mathcal{L}(P^2)$;
7:      $\mathcal{T}(P) = \mathcal{A} \cap \mathcal{C}(P)$;

8:      for all tuples ( $l_1, l_2$ ) $\in \mathcal{T}(P)$ {
9:          $hashKey$ = concat ( $S(P^1, l_1)$  $S(P^2\ l_2)$ ;
10:         for all ( $1 \le i \le K$ ) {
            // $K$ is the optimum number of hash indices
11:             $h_i$ = hash_generator( $i$ , $hashKey$);
                $queryResult$ = BF_query( $h_1, h_2, \cdots\ h_K$ );
12:             if ( $queryResult == positive$)
13:                 Put ( $l_1, l_2$ ) into $\mathcal{B}(\mathcal{T}(P))$;
14:         }
15:     }
16:     for all tuples $(l_1, l_2) \in \mathcal{B}(\mathcal{T}(P))$ {
        // Perform in parallel
17:         $hashKey$ = concat $(S(P^1, l_1)\ \ S(P^2\ \ l_2))$;
18:         $index$ = hash_generator($hashKey$);
19:         $R_i$ = SRAM($index$);
20:         if ( $P$ matches $R_i$ ) return $R_i$ ;
21:     }
22: }
```

5. Simulation Results and Performance Analysis

Classbench [28] has been widely used in generating rule sets and input traces to evaluate the performance of packet classification algorithms. Using Classbench, we have generated three different types of five-dimensional rule sets, an access control list (ACL), an Internet protocol chain (IPC), and a firewall (FW), with about 1000 and 5000 rules. Input traces that have three times the number of rules were also generated. In this simulation, two prefix fields are used for tuple pruning. All the rule information composed of every field was stored in SRAM hash tables; each was compared with a given input when a hash entry was accessed.

5.1 Rule Set Characteristics

Table 3 shows the characteristics of the rule sets. N is the number of rules included in each rule set. N_p is the number of rules that possess distinct source and destination prefix pairs. The FW has the smallest value of N_p; this means that many rules share the same source–destination

Table 3. Rule set characteristics.

Rule Set		N	N_p	$n(\mathcal{A})$	N_s	N_d
1K	ACL	958	570	61	57	361
	IPC	988	925	366	207	454
	FW	871	539	290	143	59
5K	ACL	4660	2453	102	653	907
	IPC	4468	2933	680	128	463
	FW	3067	1274	476	67	505

Table 4. Average number of tuples and rules.

Rule Set		$\mathcal{T}(P)$	*T2_field*	*T5_field*
1K	ACL	4.45	2.87	2.73
	IPC	4.94	1.53	1.19
	FW	7.63	5.41	4.02
5K	ACL	6.87	3.87	3.29
	IPC	15.29	5.41	4.00
	FW	8.16	6.15	5.67

prefix pair in the FW set. Let $n(\mathcal{A})$ represent the number of active tuples. For every rule set, the number of active tuples is significantly smaller than the maximum number of tuples (which is 1089). The ACL has the smallest (and the IPC has the largest) number of active tuples. The IPC has the largest number in both $n(\mathcal{A})$ and N_p. These characteristics affect Bloom filter performance and hash table performance, as will be shown. N_s and N_d are the number of distinct values in the source field and the destination prefix field, respectively, and they are the required number of TCAM entries. They are significantly smaller than N.

Table 4 shows tuple characteristics. The average number of tuples remaining after tuple pruning [15], $\mathcal{T}(P)$, is much smaller than the number of active tuples, $n(\mathcal{A})$ in Table 3. We can see that the tuple-pruning approach effectively prunes the number of tuples, and thereby, effectively reduces the search space. *T2_field* represents the average number of rules matching an input packet in two prefix fields. *T5_field* is the average number of rules matching an input packet in all five fields. These matching rules are returned in the multi-match packet classification.

T2_field is very close to *T5_field*, and hence, tuple pruning using two prefix fields is sufficiently effective and much simpler than using all five fields. When two prefix fields are used for tuple pruning, the difference between $\mathcal{T}(P)$ and *T2_field* is the average number of false tuples. It is large in IPC sets, which is close to 10 for the IPC5K set. In our proposed architecture, a Bloom filter is queried for $\mathcal{T}(P)$, and we will show that the Bloom filter removes the false tuples effectively.

Table 5. SRAM entry structure.

Field	Number of bits
Entry Valid	1
Rule Number	14
Source Prefix Length	6
Source Prefix	32
Destination Prefix Length	6
Destination Prefix	32
Source Port Wild	1
Source Port Start	16
Source Port End	16
Destination Port Wild	1
Destination Port Start	16
Destination Port End	16
Protocol Wild	1
Protocol Type	8
Directive	18
Total	23 bytes

5.2 Entry Structure

As shown in Section 5.1, for sets with $N < 5000$, $N_p < 3000$. The required memory for the tuple Bloom filter is about 4 KB when the size of the Bloom filter is $8N_p^{'}$, where $N_p^{'} = 2^{\lceil \log_2 N_p \rceil}$. This means that the Bloom filter can easily be embedded into a forwarding ASIC.

Table 5 shows the entry structure for SRAM. As shown, the width of the entry is 23 bytes. The total memory for hash tables is less than 200 KB for rule sets with about 5000 rules, which is calculated as $2^{\lceil \log_2 N \rceil}$ entries multiplied by the width of the memory.

The required amount of SRAM is equal to the number of active tuples, assuming they are accessed in parallel. However, all of these hash tables are not necessarily accessed in parallel. In our simulation, the number of simultaneous accesses is 4 to 7 on average, and less than 20 in the worst case. All the rules can be stored in a single hash table without separating them by tuples. Since the required memory amount is small, if we have multiple copies of the hash table for parallel access, then the required amount of SRAM can be reduced to the worst case number of positive tuples, which is 20. Even in this case, the total amount of SRAM is less than 4 MB.

5.3 Comparison with other TCAM-based Multi-match Architectures

Table 6 shows a comparison of our proposed architecture to other algorithmic TCAM-based multi-match packet classification architectures described in Section 3.1. The previous approaches have trade-offs in the amount of TCAM, the number of TCAM entries, or the number of TCAM lookups. Our approach requires two small TCAM modules, and the number of TCAM lookups is only one per packet. Therefore, our approach is power-efficient. These improvements come at the cost of multiple

Table 6. Comparison of the proposed algorithm to other TCAM-based approaches.

Approaches	MUD[5]	GIS[6]	SSA[7]	MX-MN[8]	Proposed
No. of TCAM Modules	1	1	2 or 4	> thousands	2
No. of TCAM Entries (in *N*)	1	> 10 times	> 1.2 times	1	<< 1
No. of TCAM Lookups	about 20	1	1	1	1
No. of SRAM Modules	1	1	1	1	20
Pre-processing Complexity	low	high	very high	very high	low
Update Cost	low	high	medium	low	low
Port Range Problem	Yes	Yes	Yes	Yes	No

Table 7. Decision-tree characteristics of HiCuts [17] and HyperCuts [18].

Rule	*N*	*binth*	HiCuts		HyperCuts 1		HyperCuts 2	
			D_t	*f*	D_t	*f*	D_t	*f*
ACL1k	958	6	66	19.6	17	5.2	35	9.8
ACL5k	4660	13	34	16.2	13	7.0	23	10.3
IPC1k	988	5	70	11.0	22	3.3	30	11.8
IPC5k	4468	15	67	201.8	19	96.8	22	490.5
FW1k	871	11	64	97.5	22	9.2	25	70.3
FW5k	3067	17	63	393.4	18	47.1	19	878.3

small SRAM modules and parallel access to SRAM, but this cost is insignificant, given that SRAM is much less expensive and requires very little power. Several approaches require very complicated pre-processing to store the rules in TCAM, either in identifying the intersections of the rules [6] or in separating them into multiple groups [7, 11], resulting in a high update cost with these approaches. The pre-processing requirements of our approach is to calculate prefix vectors and store them into PV-SRAM, and to program a Bloom filter using two prefix fields of each rule. This process is much simpler than that found in other approaches. All of the other approaches shown in Table 6 have a port range problem, since all of the rule fields are stored in TCAM, whereas our approach does not have this problem, since the port range fields are stored in SRAM.

5.4 Comparison with Other Algorithmic Approaches

Our proposed architecture was compared with other algorithmic approaches that enable multi-match packet classification. Among various algorithms that have been developed for single best–match packet classification, decision tree–based algorithms such as hierarchical intelligent cutting (HiCuts) [17] and multi-dimensional hierarchical cutting (HyperCuts) [18] also provide a multi-match classification capability.

In building HiCuts and HyperCuts, *spfac* was set at 2 in this simulation. Table 7 shows the HiCuts and HyperCuts characteristics for an arbitrary value of *binth*. As shown in the copy factor, *f*, a rule is replicated many times.

Figs. 4 and 5 show a search performance comparison using the average number of memory accesses and the worst-case number of memory accesses, respectively. For a fair comparison with HiCuts and HyperCuts, it was assumed that a single SRAM module is used in our proposed architecture (and the tuple-pruning algorithm [15]). Hence, a number of tuples are serially probed. Rules with the same source and destination prefixes are mapped to the same hash entry, and they are stored in a linked list. These entries are linearly probed. Hence, the number of memory accesses becomes worse than the number in *T2_field*. The search performance of HyperCuts is significantly worse than HiCuts in the FW and IPC sets because of the optimization of common rules pushing upward. The figures show that the proposed architecture provides slightly better search performance than the tuple-pruning algorithm [15] and much better performance than the decision-tree algorithms [17, 18] in most cases.

Table 8 shows the required SRAM. In this simulation, the number of hash entries is four times the number of rules, which is $4\times 2^{\log_2 N}$, in order to reduce the number of rules mapped to the same entry. Each hash entry is 23 bytes, as shown in Table 5. For our proposed algorithm, the memory amount for the PV-SRAM, which is $4\times\left(2^{\log_2 N_s} + {}^{\log_2 N_d}2\right)$ bytes, and the memory amount for the Bloom filter, which is 4000 bytes, are additionally required. The tuple-pruning algorithm and the proposed algorithm require two TCAM modules storing distinct source prefixes and distinct destination prefixes, whereas

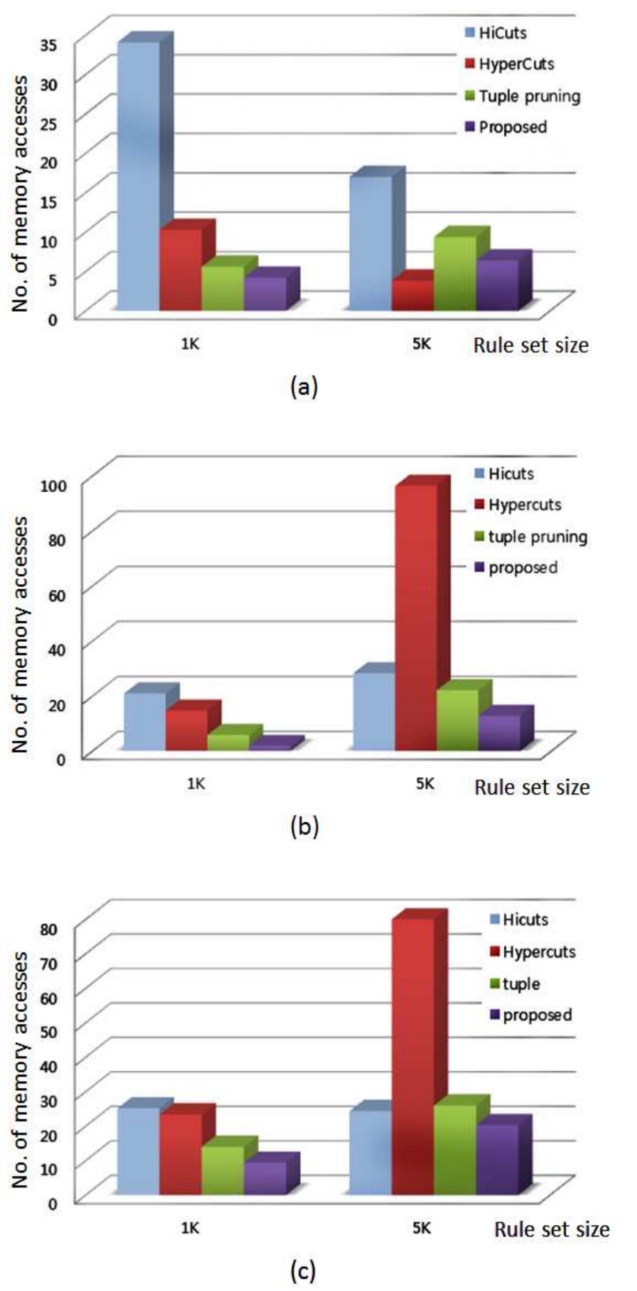

Fig. 4. Comparison of the average number of memory accesses (a) ACL, (b) IPC, (c) FW.

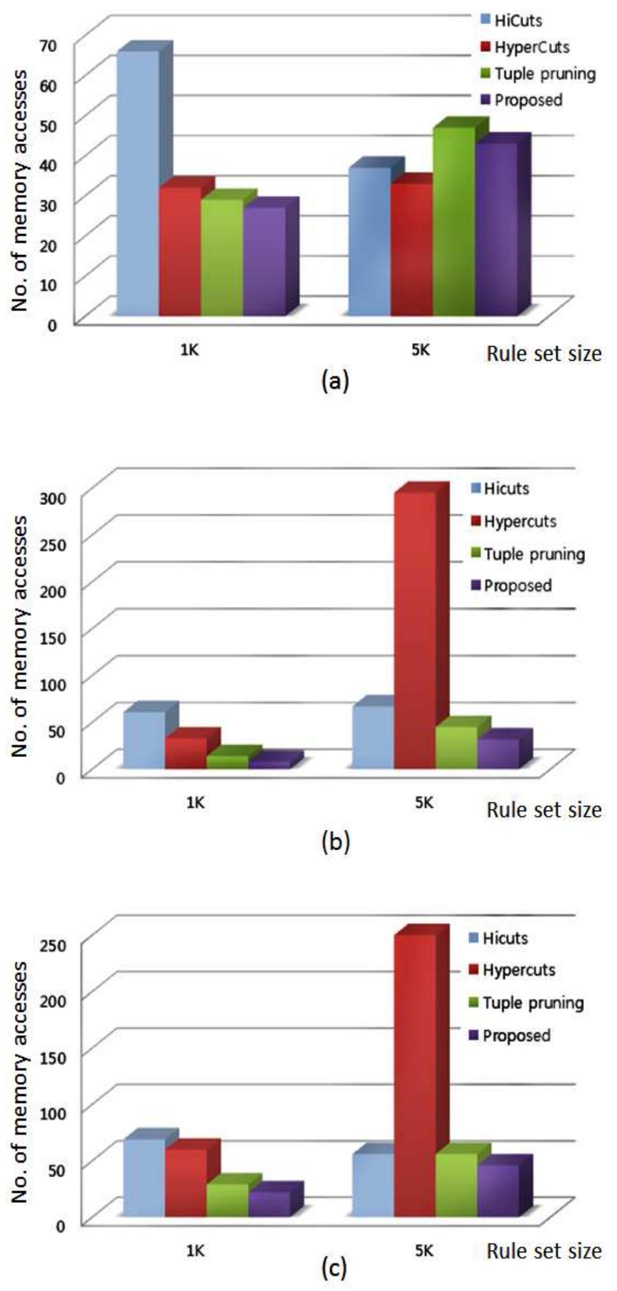

Fig. 5. Comparison of the worst-case number of memory accesses (a) ACL, (b) IPC, (c) FW.

Table 8. Comparison of the required SRAM amount per rule with other algorithmic approaches (MB).

Rule Set		Tuple Pruning [15]	HiCuts [17]	HyperCuts [18]	Proposed
1K	ACL	0.092	0.968	0.172	0.098
	IPC	0.092	0.603	0.119	0.099
	FW	0.092	2.830	0.226	0.097
5K	ACL	0.74	2.190	0.885	0.752
	IPC	0.74	26.58	11.08	0.746
	FW	0.37	36.19	3.680	0.376

decision-tree algorithms do not require TCAM. However, since rules are replicated many times in the decision-tree algorithms, the required SRAM amount is excessive, and hence, SRAM cannot be embedded in an ASIC.

6. Conclusion

Multi-match packet classification is becoming an essential feature that routers need to perform at wire speed for every incoming packet in order to support new emerging applications. This paper proposes a new algorithmic approach to the multi-match classification problem. Our proposed architecture is a power-efficient solution combining TCAM with a tuple-pruning algorithm. TCAM is used for individual field search in our proposed architecture; we propose the use of a TCAM index, which is the single best–matching result for each field, in order to obtain prefix vectors. The prefix vectors for each field are combined to make the list of tuples. Since the list may include false tuples, we also present a way to eliminate them by using a Bloom filter. For the list of tuples deemed positive by the Bloom filter, SRAM is accessed in parallel; the packet-treating directive is obtained by matching rules from SRAM access. Therefore, the proposed multi-match packet classification architecture requires a single TCAM lookup cycle, two SRAM access cycles (one for the prefix vector, and one for the tuple search), and several Bloom filter query cycles. Regarding the power consumption issue, our proposed approach reduces TCAM power by utilizing small amounts of TCAM and by accessing each TCAM entry only once for each packet. Port ranges are stored as they are in SRAM, resulting in no rule replication.

Acknowledgement

This research was supported by the National Research Foundation of Korea, NRF2014R1A2A1A11051762 and NRF- 2015R1A2A1A15054081. This research was also supported by MSIP, Korea, under the ITRC support program (IITP-2016-H8501-15-1007) supervised by IITP.

References

[1] H. J. Chao, "Next generation routers," *Proceedings of the IEEE*, vol. 90, no. 9, pp. 1518–1558, Sep. 2002. Article (CrossRef Link)

[2] K. Vlaeminck, T. Stevens, W. V. D. Meerssche, F. D. Turck, B. Dhoedt, and P. Demeester, "Efficient packet classification on network processors," *International Journal of Communication Systems*, vol. 21, no. 1, pp.51–72, Jan. 2008. Article (CrossRef Link)

[3] P.-C. Wang, "Scalable packet classification using a compound algorithm," *International Journal of Communication Systems*, vol. 23, no. 6, pp.841–860, Jun. 2010. Article (CrossRef Link)

[4] D. Adami, C. Callegari, S. Giordano, M. Pagano, and T. Pepe, "Skype-Hunter: a real-time system for the detection and classification of Skype traffic," *International Journal of Communication Systems*, vol. 25, no. 3, pp.386–403, Mar. 2012. Article (CrossRef Link)

[5] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary,"Algorithms for advanced packet classification with ternary CAMs," *Proc. ACM SIGCOMM*, pp. 193–204, 2005. Article (CrossRef Link)

[6] F. Yu, R. H. Katz, T. V. Lakshman, "Efficient multimatch packet classification and lookup with TCAM," *IEEE Micro*, vol. 25, no. 1, pp. 50–59, Jan/Feb. 2005. Article (CrossRef Link)

[7] F. Yu, T. V. Lakshman, M. A. Motoyama, and R. H. Katz, "Efficient multimatch packet classification for network security applications," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 10, pp. 1805–1816, Oct. 2006. Article (CrossRef Link)

[8] M. Faezipour and M. Nourani, "Wire-speed TCAM-based architectures for multimatch packet classification," *IEEE Trans. on Computers*, vol. 58, no.1, pp. 5–17, Jan. 2009. Article (CrossRef Link)

[9] H. Song and J.W. Lockwood, "Efficient packet classification for network intrusion detection using FPGA," *Proc. ACM SIGDA FPGA*, pp. 238–245, 2005. Article (CrossRef Link)

[10] X. Deng, Z. Huang, S. Su, C. Liu, G. Tang, and Y. Zhang, "A sequence encoding scheme for multi-match packet classification," *Proc. NSWCTC*, pp. 641–644, 2009. Article (CrossRef Link)

[11] F. Yu, T. V. Lakshman, M. A. Motoyama, and R. H. Katz, "SSA: A power and memory efficient scheme to multimatch packet classification," *Proc. ANCS Conf.*, pp. 105–113, 2005. Article (CrossRef Link)

[12] A. X. Liu, C. R. Meiners, and E. Torng, "TCAM razor: a systematic approach towards minimizing packet classifiers in TCAMs," *IEEE/ACM Trans. on Networking*, vol. 18, no. 2, pp. 490–500, Feb. 2010. Article (CrossRef Link)

[13] M. Faezipour and M. Nourani, "CAM01-1: a customized TCAM architecture for multi-match packet classification," *Proc. IEEE Globecom*, pp. 1–5, 2006. Article (CrossRef Link)

[14] H. Che, Z. Wang, K. Zheng, and B. Liu, "DRES: dynamic range encoding scheme for TCAM," *IEEE Trans. on Computers*, vol. 57, no.7, pp. 902–915, Jul. 2008. Article (CrossRef Link)

[15] V. Srinivasan, S. Suri, and G. Varghese, "Packet classification using tuple space search," *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 4, pp. 135–146, 1999. Article (CrossRef Link)

[16] H. Lim and S. Kim, "Tuple pruning using Bloom filters for packet classification," *IEEE Micro*, vol. 30, no.3, pp. 48–58, May/Jun. 2010. Article (CrossRef Link)

[17] P. Gupta and N. Mckeown, "Classification using hierarchical intelligent cuttings," *IEEE Micro*, vol. 20, no. 1, pp. 34-41, Jan./Feb., 2000. Article (CrossRef Link)

[18] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet classification using multidimensional

cutting," *Proc. SIGCOMM*, 2003. Article (CrossRef Link) T. V. Lakshman and D. Stidialis, "High-speed policy-based packet forwarding using efficient multi-dimensional range matching," *Proc. ACM SIGCOMM*, pp. 203–214, Oct. 1998. Article (CrossRef Link)
[19] F. Baboescu, S. Singh, G. Varghese, "Packet classification for core router: is there an alternative to CAMs?," *Proc. IEEE INFOCOM*, vol. 1, pp. 53–63, Mar. 2003. Article (CrossRef Link)
[20] G. Priya and H. Lim, "Hierarchical packet classification using a Bloom filter and rule-priority tries," *Computer Communications*, vol. 33, no. 10, pp. 1215–1226, Jun. 2010. Article (CrossRef Link)
[21] H. Lim, H. Chu, and C. Yim, "Hierarchical binary search tree for packet classification," *IEEE Communications Letters*, vol. 11, no. 8, pp. 689–691, Aug. 2007. Article (CrossRef Link)
[22] H. Lim, M. Kang, and C. Yim, "Two-dimensional packet classification algorithm using a quad-tree," *Computer Communications*, vol. 30, no.6, pp. 1396–1405, Mar. 2007. Article (CrossRef Link)
[23] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast and scalable layer four switching," *Proc. ACM SIGCOMM*, pp. 191–202, 1998. Article (CrossRef Link)
[24] S. Dharmapurikar, P. Krishamurthy, and D. E. Taylor, "Longest prefix matching using bloom filters," *IEEE/ACM Transactions on Networking*, vol. 14, no. 2, pp. 397–409, April 2006. Article (CrossRef Link)
[25] H. Lim, J. Lee, and C. Yim, "Complement Bloom Filter for Identifying True Positiveness of a Bloom Filter," *IEEE Communications Letters*, vol. 19, no. 11, pp. 1905-1908, Nov. 2015. Article (CrossRef Link)
[26] H. Lim, H. Kim, and C. Yim, "IP address lookup for Internet routers using balanced binary search with prefix vector," *IEEE Trans. on Communications*, vol. 57, no. 3, pp. 618–621, Mar. 2009. Article (CrossRef Link)
[27] D. E. Taylor and J. S. Turner, "Classbench: a packet classification benchmark," *IEEE/ACM Trans. on Networking*, vol. 15, no.3, pp. 499–511, June 2007. Article (CrossRef Link)

Hyesook Lim received the B.S. and M.S. degrees at the Department of Control and Instrumentation Engineering in Seoul National University, Seoul, Korea, in 1986 and 1991, respectively. She received the Ph.D. degree at the Electrical and Computer Engineering from the University of Texas at Austin, Texas, in 1996. From 1996 to 2000, she had been employed as a member of technical staff at Bell Labs in Lucent Technologies, Murray Hill, NJ, USA. From 2000 to 2002, she had worked as a hardware engineer for Cisco Systems, San Jose, CA, USA. She is currently a professor in the Department of Electronics Engineering, Ewha Womans University, Seoul, Korea, where she perform research on packet forwarding algorithms such as IP address lookup and packet classification, and in Content Centric Networks. She was awarded Year 2014 Women in Sciences and Technologies by the Ministry of Science, ICT and Future Planning of Korea. She is a senior member of the IEEE.

Nara Lee received the B.S. and M.S. degrees in electronics engineering from Ewha Womans University, Seoul, Korea, in 2009 and 2012, respectively. She is currently a Research Engineer with the IP Technical Team, DTV SoC Department, SIC Lab, LG Electronics, Inc., Seoul, Korea. Her research interests include various network algorithms such as IP address lookup, packet classification, Web caching, and Bloom filter application to various distributed algorithms.

Jungwon Lee received the B.S. degree in Mechatronics Engineering from Korea Polytechnic University, Gyeonggi-do, Korea, in 2011 and M.S. degree at the Department of Electronics Engineering at Ewha Womans University, Seoul, Korea, in 2013. She is currently pursuing a Ph.D. degree from the Department of Electronics Engineering at Ewha Womans University. Her research interests include address lookup, packet classification algorithms, and packet forwarding using Bloom filters at Content-Centric Networks.