Wire-Speed TCAM-Based Architectures for Multimatch Packet Classification

Miad Faezipour, *Student Member*, *IEEE*, and Mehrdad Nourani, *Senior Member*, *IEEE*

Abstract—Most conventional packet classifiers find only the highest priority filter that matches the arriving packet. However, new networking applications such as network intrusion detection systems and load balancers require all (or the first few) matching filters during classification. In this paper, two TCAM-based architectures for multimatch search are introduced. The first one is a renovated TCAM design that can find all or the first r matches in a packet filter set. The second architecture is a novel partitioning scheme based on filter intersection properties allowing us to use off-the-shelf TCAMs for multimatch packet classification. Our classifier engine finds all matches in exactly one conventional TCAM cycle while reducing the power consumption by at least two orders of magnitude, which is far better than the existing hardware-based designs.

Index Terms—Ternary content addressable memory, multimatch, packet classification, prioritizer, network intrusion detection system, maximum-minimum intersection partitioning, contention resolver.

1 Introduction

1.1 Background

Packet classification, in general, refers to finding the best matching filter containing multiple fields in a *filter* (also called *rule*) set for a given packet. The standard five-tuple fields include the source address, destination address, protocol, source port, and destination port [1]. Among these fields, source and destination address fields are prefixes and often require the longest prefix match (LPM) methods. Protocol field can be wildcards or exact values. Source and destination port numbers are typically introduced as ranges. Packet classification is a multidimensional (multifield) search in contrast to packet forwarding that only involves search in one dimension, (i.e., the destination IP address).

Packet classification performs searching the table of filters to assign a flow identifier for the highest priority filter that matches the packet in all fields. The returning flow ID indicates the action that is next applied to the packet. An example of a packet filter set is shown in Table 1. The standard five-tuple fields are shown in separate columns for the purpose of clarity. This table illustrates a small sample packet filter set with a very few bits for simplicity. Filters are usually sorted in the order of priority in the filter set. In this table, x indicates wildcards inserted in any location of the fields. When a packet arrives, the first (which is generally the best) matching filter in the set is to be found in packet classifiers. The matching filter should match the filter in all five fields. The address (index) of the matching filter is used to point to the action that needs to be applied to the packet. Most packet classifiers store the index to indicate the action that is going to be processed on the packet afterward. For example, if a packet consisting source address of 0101, destination address of 0011, source port of 4, destination port of 6, and protocol field of TCP is received, the packet classifier should report the second filter as the matching filter and, hence, would forward the packet to output port 5. It is obvious that arriving packets may result in multiple filters matching the packet in the set. In this example, the arriving packet matches the seventh filter in addition to the second one. The general packet classification process only reports one filter (which is the highest priority filter) in case of multiple matches. However, we elaborate why some networking applications require finding multiple matches in the packet filter set.

Filter fields are combination of prefixes, wildcards, and exact values. Hence, Ternary Content Addressable Memories (TCAMs) that have the ability to store *don't-care* values in addition to 1's and 0's are often utilized to store filters and perform the parallel search in packet classification. A traditional packet classifier assigns one TCAM entry for each filter and finds the index of the highest priority matching filter in the database (Fig. 1). Range fields are often translated to multiple entries [1], [2]. Overall, each filter f_i $(0 \leq i \leq n-1)$ contains multiple fields (e.g., in the standard five-tuple), and also, the filter database often may have up to 100,000 filters. Therefore, wide TCAM devices both in terms of bits and entries are used for packet classification applications.

1.2 Importance of Multimatch Packet Classification

New emerging networking applications such as Network Intrusion Detection Systems (NIDSs) and load balancers require finding all or the first few matching filters in packet classification. Malicious intrusions and denial-of-service attacks, which are expected to grow rapidly, can be monitored and detected by NIDS. Once all the matching filter headers are found, a detection system such as Snort [3]

• *The authors are with the Department of Electrical Engineering, University of Texas at Dallas, Richardson, TX 75083. E-mail: {mxf042000, nourani}@utdallas.edu.*

Manuscript received 6 June 2007; revised 23 June 2008; accepted 16 July 2008; published online 22 Aug. 2008.
Recommended for acceptance by S. Nikoletseas.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2007-06-0208.
Digital Object Identifier no. 10.1109/TC.2008.159.

TABLE 1
Example of a Packet Filter Set

Filter	Src. Addr.	Dest. Addr.	Src. Port	Dest. Port	Prot.	Action
1	1001	01*xx*	2-4	7	TCP	Forward 3
2	01*xx*	00*xx*	3-9	2-6	TCP	Forward 5
3	110*x*	10*xx*	1-7	4-6	UDP	Accept
4	1101	101*x*	*x*	*x*	ICMP	Queue
5	00*xx*	010*x*	4	5	UDP	Accept
6	111*x*	01*xx*	*x*	*x*	*x*	Drop
7	0101	*xxxx*	*x*	*x*	*x*	Forward 4
8	1*xxx*	0*xxx*	*x*	*x*	*x*	Drop

scans the packet payload for existing worms. The concept of Multimatch Classification (MMC) for NIDS is becoming a major stream of research in the near future, since there is a great demand for network worm detections [2], [4].

Packet level accounting, transparent monitoring, and the Programmable Network Element (PNE) are other networking applications that demand MMC. Specifically, PNE is the general platform for packet processing in layers 2-4 in the edge. Packets entering PNEs are classified to identify the relevant functions. Multimatching classification can be utilized to support multiple functions in PNEs [4].

Both single and multimatch packet classifications should be ideally performed at the wire data rate. Pure software solutions suffer from low speed, since they often require several instructions and as a result several memory accesses to find a single or multiple matches. To achieve wire-speed classification, researchers in industry and academia offered architectural solutions mostly using TCAM [5]. TCAMs are well suited for performing high-speed parallel searches on database with ternary entries, since they provide the match results with deterministic throughput (i.e., one search per cycle) and deterministic capacity. Hence, TCAM has become quite popular for packet classification tasks [1], [6], [7], [8]. While TCAMs perform packet classification at high speed, they cannot directly report all possible matches in a database. This is due to the native structure of a TCAM cell design, which consists of a priority encoder, generating only the highest priority match in each round. Other drawbacks of using TCAM are high cost and high power consumption.

1.3 Main Contribution

Complexity of conventional (software-based) classification techniques linearly grows with number of filters. Our main contribution is twofold. First, we propose a multimatching packet classifier by modifying the prioritizer (PZ) circuit of conventional TCAM. Since the TCAM entries (cells) remain unchanged in our design, our multimatching hardware can be easily adopted for IPv6 where the bit width of the TCAM entries highly increases [9], [10]. Our system finds r (predefined by a user) matches in at most r cycles regardless of total number of filters. This approach has zero-management (fixed-update) time and is highly efficient when filters are updated regularly. Such properties significantly improve the performance by one to two orders

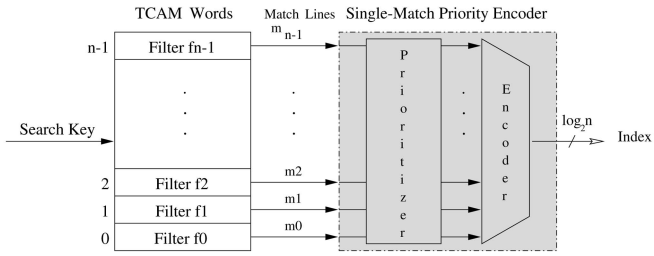


Fig. 1. TCAM structure as a single-match classifier.

of magnitude for large filter set. Second, we propose a parallel architecture for multimatching packet classification by efficiently partitioning the entire packet filter set into disjoint subsets. Each subset is mapped to a relatively small TCAM, which produces a match in one cycle. In this technique, off-the-shelf TCAMs are used. Our system finds r matches in exactly one cycle regardless of the total number of filters and matches. Our partitioning scheme can also be employed as a low-power solution to the conventional single-match packet classification in general and multimatch packet classification in particular. Each of these two architectures, i.e., customized TCAM and partitioned structure, can work as a multimatch classifier engine independently. However, we will show that a tightly coupled system that employs both architectures would achieve the maximum performance.

1.4 Paper Organization

The rest of this paper is organized as follows: In Section 2, we take a glance at prior work related to TCAM-based multimatch packet classification. In Section 3, we first derive the optimized logic equations and then elaborate on our TCAM customized for multimatch tasks. Two configurations (i.e., cascaded and parallel) for a scalable design are also introduced in this section. Section 4 describes our novel intersection-based partitioning schemes. In Section 5, the structural features and advantages of our system are discussed. We summarize our experimental results in Section 6. Finally, concluding remarks are given in Section 7.

2 Prior Work

Some recent work focused on multimatch packet classification using TCAMs. The Entry-invalidation scheme, described in [2], is one of the earliest and simplest schemes. In this method, a valid bit in addition to the header fields is associated with each TCAM entry. Initially, all entries have their valid bits set to 1. Searches are performed multiple times to find all matching entries. Each time a match is found, the valid bit is set to 0 for that matching entry. The same search key is applied again until all matches are found. This method, however, does not support the multithreading feature, which requires multiple packet processing threads to have access to the TCAM device at the same time.

The authors in [4] reorganize the TCAM entry filters in a compatible order to report all matches. The authors use a geometric intersection scheme to remove the overlaps and negation among the intersecting filters placed in the TCAM.

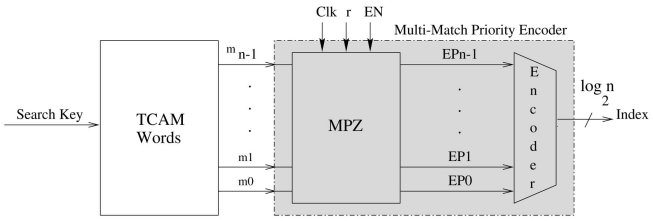


Fig. 2. Conceptual block diagram of our design.

This approach may not be suitable for large databases since it cannot be easily scaled to large tables.

The Set Splitting Algorithm (SSA) introduced in [11] and [12] splits the filter set into two groups to remove at least half of the intersections among filters. It then performs the search on multiple groups in parallel. This method is based on minimum intersections among filters; however, it adds filters for partially overlapped filters in one set. In addition, it performs the search on all sets generated; hence, it increases the search time and power consumption.

The authors in [2] address the problem of finding multiple matches in a TCAM by proposing the multimatch using discriminators (MUD) algorithm. In this algorithm, the extra bits per TCAM entry are used for the required encoding. The MUD algorithm provides multiple matches at high speed. However, it deploys sophisticated encoding on TCAM entry databases, making it difficult to decode the data to their original values.

The BV-TCAM architecture introduced in [8] combines the TCAM and the Bit Vector (BV) algorithm to address the problem of packet classification for network intrusion detection. While this approach improves the search mechanism and the cost by compressing the data representation, it does not effectively differentiate the multiple matches found.

3 MMC Using Renovated TCAMs

A TCAM includes a priority encoder. One widely used implementation of the priority encoder is shown in Fig. 1 by splitting it into a PZ and a conventional encoder [13]. A valid $\log_2 n$-bit address out of the encoder is generated only if at most one of its n inputs is high at a time. Thus, the PZ unit, which provides the encoder input lines, should be designed carefully. Our main idea is to modify the single-match PZ unit to a Multimatch PZ (MPZ), as shown in Fig. 2 so that the encoder would generate all matching indices, one per cycle.

A power-optimized priority encoder cell introduced in [13] is used as our reference model for the PZ unit. The logic equation[1] for the PZ circuit can be written as

$$EP_i = \begin{cases} en \cdot D_i, & i = 0, \\ en \cdot \left(\prod_{k=0}^{i-1} \overline{D_k}\right) \cdot D_i, & 1 \le i \le n-1. \end{cases} \quad (1)$$

Equation (1) indicates that the PZ circuit has n inputs and n outputs, where EP_i denotes the ith output, D_i's are the input lines, and en is the enable line.

1. Throughout this section, symbols $+$, $\cdot$, $\prod$, and $\sum$ stand for Boolean notations.

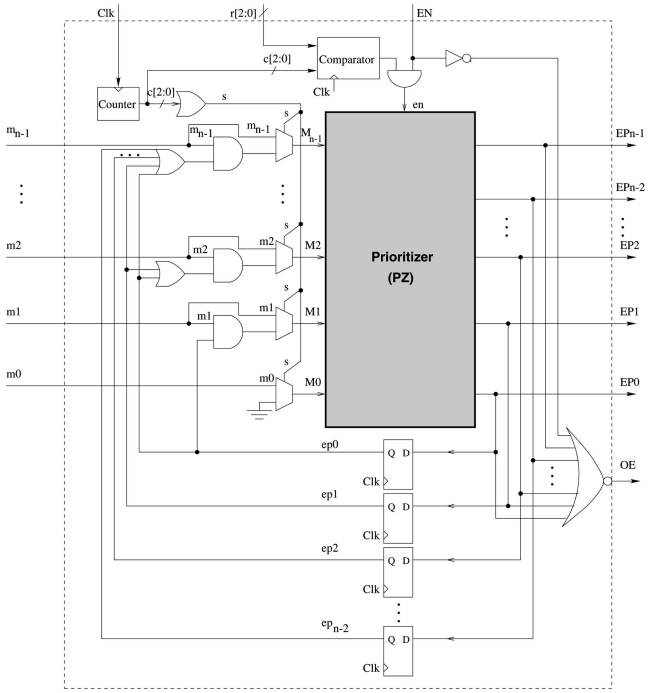


Fig. 3. The MPZ architectural design.

3.1 MPZ Structure

We add a control logic circuitry to the PZ circuit to report all matches in a prioritized sequence. The MPZ circuit, shown in Fig. 3, functions in response to a counter that counts from 0 to n, where n is the highest possible number of matches. In other words, n can be assumed as the number of inputs in the worst case. On the first clock cycle, the MPZ should function as a single-PZ unit, reporting the highest priority match. On the next clock cycle, the next highest priority match should be provided at the output. This procedure should be followed in all other clock cycles until the counter has reached counting up to n. In each clock cycle, a function of the original inputs and the higher priority outputs of the PZ circuit in the previous clock cycle should be fed through the PZ circuit. In each clock cycle, a new set of inputs should be fed through the PZ. These inputs are based on a function of the original inputs and the higher priority PZ outputs in the previous clock cycle. Let m_i denote the original input lines (i.e., match lines from TCAM words), ep_i denote the EP_i outputs of the PZ after one clock cycle, M_i be the set of inputs that should be given to the PZ circuit, and EN be the enable line. The logic equation for the MPZ circuit can be derived as follows:

$$EP_i = \begin{cases} EN \cdot M_i, & i = 0, \\ EN \cdot \left(\prod_{k=0}^{i-1} \overline{M_k}\right) \cdot M_i, & 1 \le i \le n-1, \end{cases} \quad (2)$$

where M_i in (2) can be computed as

$$M_i = \begin{cases} \overline{s} \cdot m_i, & i = 0, \\ \overline{s} \cdot m_i + s \cdot m_i \cdot ep_{i-1}, & i = 1, \\ \overline{s} \cdot m_i + s \cdot \left(m_i \cdot \sum_{k=0}^{i-1} ep_k\right), & 2 \le i \le n-1. \end{cases} \quad (3)$$

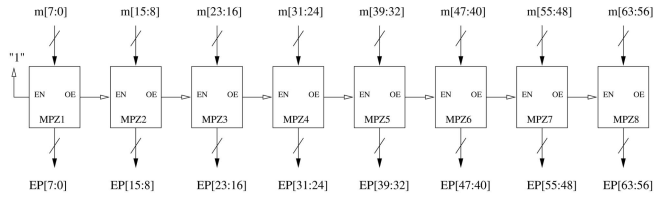


Fig. 4. Cascaded MPZ architecture for a 64-bit MPZ design.

Signal $s = \sum_{k=0}^{log_2 n-1} c_k$ is the select line of the multiplexers that control which data should be chosen for the corresponding M_i. This select line should be low for the first clock cycle and high for the rest. Thus, s can be implemented by simply ORing all the counter outputs c_i. The MPZ unit functions in an efficient manner; in essence, it reports all r matches in exactly r cycles. This implies that in case of the need to report the first r matches instead of all possible matches, a comparator unit could be added to the MPZ design wherein the count value c and the r value are compared. Once the count exceeds r, the enable line EN is set to zero, hence disabling the MPZ unit.

3.2 Scalability

The MPZ unit can be designed for any number of n inputs. However, the synthesized MPZ design for a real database that requires up to thousands of TCAM entries would be very costly. Hence, a modular design that could scale up to the required n inputs is essential.

IPv6 has much longer fields in the header of packets. Our design does not alter the filters placed in the TCAM. Moreover, the MPZ approach processes the match lines coming from the TCAM words. Therefore, it can efficiently be scaled to IPv6 classifiers that use large TCAMs or multiple parallel TCAMs for packet classification.

3.2.1 Cascaded MPZ Architecture

To achieve a modular design with cascaded blocks, we define an *output enable* (OE) line, which indicates when all the matches have been provided at the output [14]. This signal is activated when any number of matches are found at the output, and deactivated when all the matching results have been provided. It also highly depends on the EN line. The OE line is an active low signal used for activating lower priority MPZ units. The OE line in an MPZ can be expressed as follows:

$$OE = \overline{\overline{EN} + \sum_{i=0}^{n-1} EP_i}. \quad (4)$$

Similar to the multilevel look-ahead architectures [15], [16], we cascade v w-bit MPZ units to design an $n = v \times w$ bit MPZ block. Fig. 4 shows the concept of cascading eight 8-bit MPZ modules to design a 64-bit MPZ. In this figure, higher priority stages are placed at the left. By connecting the OE line of each stage to the EN (enable line) of the next stage, we assure that each block would be enabled only if all the higher priority blocks have completed reporting their matches at the output. The cascaded design would have at most two additional clock cycle delays for any mismatching MPZ unit. The authors in [2] claimed that the number of multimatch results could reach up to 153 when considering the *two* header fields, source and destination IP addresses only. It is clear that when considering all *five* header fields, the number of multimatch results would drop dramatically. The authors of the SSA paper stated that a packet can match up to only 12 unique filters for the SNORT rule sets in the worst case [11]. Moreover, as authors in [2] stated, the maximum degree of matches (number of matches) often requested in real-world ACL filters including router databases and SNORT rule sets is statistically around 8. Considering these facts, an 8-bit MPZ unit seems to be an optimum size choice for a basic MPZ cell. Therefore, the counter used in any MPZ unit can be designed to count up to 8. This implies that only $\log_2 8 = 3$ lines are required for the counter, and the comparator can be designed to compare the count value and $r = 8$. The modularity of our design allows the user to easily redesign MPZ for $r > 8$ when needed.

3.2.2 Parallel MPZ Architecture

Due to the complexity of the connections among the cascaded cells in the multilevel folding architecture [15], we have not scaled the design using this method. Furthermore, the parallel priority look-ahead architecture discussed in [13] would not be directly applicable for the multimatch design. This is because each MPZ unit in the second stage should remain enabled for at least eight clock cycles to report multiple matches at the output. Multiple matches may occur at any MPZ cell in the second stage. The ORed output of the corresponding set of inputs would cause the initial MPZ unit outputs to remain high for only one clock cycle, hence enabling the matching unit for only one clock period. This would result in reporting only one match from that unit and not other matches. However, by using a slower clock for the counter of the first stage MPZ comparing to the second stage MPZ units, finding multiple matches would become possible. The first stage MPZ unit would maintain each matching output for a longer period of time, enabling the second stage MPZ units to report all the matches.

In case of a 64-bit multimatch design, the first MPZ unit should have a clock period of at least eight times slower than the eight MPZ units in the second stage. In addition to the frequency of Clk that is eight times faster than Clk', the rising edge of Clk should be delayed by one $T_{Clk}/2$ for proper functionality. This is because the second stage MPZ units should see their enable lines at the rising edge of the clock Clk. The first stage MPZ unit provides its outputs at the rising edge of the clock Clk'; therefore, a delay between the rising edge of the two clocks is needed to ensure proper functionality. This delay does not affect the overall performance. Fig. 5 shows the scalable multimatching design using the concept of the parallel priority look-ahead architecture.

3.2.3 Cascaded versus Parallel Architecture

In the parallel architecture, each matching MPZ unit in the second stage would add a latency of $8 \times T_{Clk}$ to report matches in the other matching MPZ units. However, if there are r matches only in the last eight inputs, the parallel

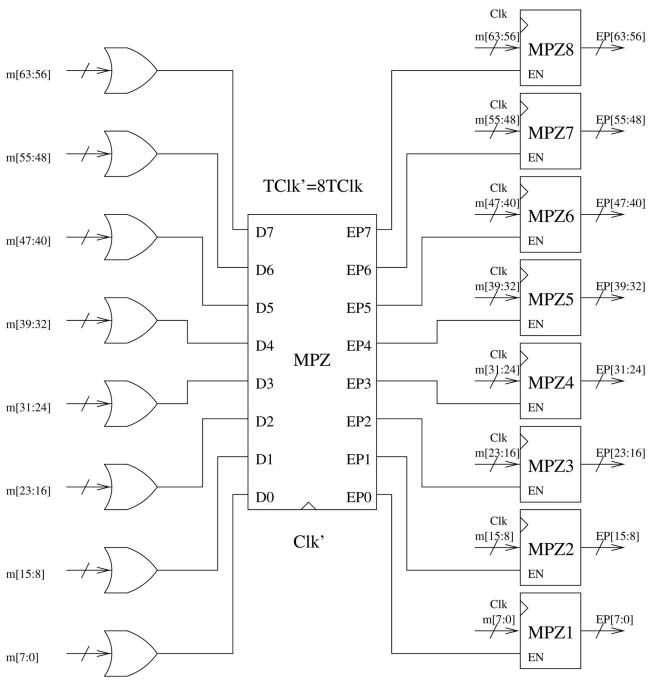


Fig. 5. A 64-bit multimatching design using parallel architecture.

design would have a latency of only $8 \times T_{Clk}$ to enable the last MPZ unit in the second stage and would take another $r-1$ clock cycles to report the matches at the output. In this particular case, the parallel architecture performs faster compared to the 8-bit cascaded architecture mentioned earlier that would have a latency of $(2 \times 8 - 1) = 15$ clock cycles to enable the last MPZ stage. The clock period of the cascaded architecture is assumed to be the same clock period of the MPZ units in the second stage of the parallel architecture.

The speed of our scaled multimatch circuits highly depends on the location of the matches. The cascaded architecture would perform more efficiently if matches are distributed uniformly among MPZ units. The parallel configuration would perform faster if the matches are concentrated within one MPZ unit at lower priority locations.

Area is another concern for the two scalable configurations. The parallel architecture would have an additional MPZ unit plus a few OR gates, compared to the cascaded architecture. More details on cascaded and parallel architectures can be found in [17].

3.2.4 Applications and Limitations of MPZ

One main advantage of the MPZ architecture is that it has fixed update complexity. This is due to the fact that no filter management (e.g., sort, split, duplicate) is required when filters are updated in the set (e.g., added or deleted). Our customized PZ is a hardware engine capable of finding all matches, regardless of the location of matching filters. This feature makes the MPZ architecture highly efficient for applications where frequent updates are needed in MMC.

As for limitation of MPZ, performance of the scaled MPZ in large filter databases depends on the locations of the matches (see Section 3.2.3). A sorting mechanism to rearrange filters based on their priority would be costly. Instead, partitioning schemes that reorganize the filters based on matches and intersecting filters are introduced in the next section. Efficient partitioning can further improve the MPZ performance in terms of speed, since it allows potential matching filters to be gathered in small TCAMs. This technique may, however, require larger update time due to partitioning. The partitioning schemes are quite cost-effective, as they can be complemented to the MPZ unit and other architectures for finding all matching filters in only one cycle.

In addition, TCAMs are complex devices and architectural changes that modify TCAMs involve large amount of investment and long development time [2]. Hence, algorithmic approaches that utilize off-the-shelf TCAMs to solve the problem may be preferred.

4 MMC Using Off-the-Shelf TCAMs

The methods explained in [2], [4], and [11] are schemes that utilize off-the-shelf TCAMs for multimatching packet classification. Our strategy would be to design a hardware engine consisting a filter processing unit and the conventional TCAM cell for reporting multiple matches. As we proceed further, we will also see that the regular (instead of priority) encoder can be used. Replacing the priority encoder is a huge benefit in terms of cost, delay, and power.

4.1 Partitioning Rule

Intersection among filters in the database mainly results in multiple matches. Informally speaking, intersection is defined as having filters that are subset of one another, such that some filters completely overlap others. The filter processing unit applies the partitioning schemes on the packet filter set and conventional TCAMs are used to accommodate the filters of each partition [18].

Let $f_i[w-1:0]$ and $f_j[w-1:0]$ denote two filters of bit-width w. We define the term *distance* between the two filters as

$$d_{i,j} = \sum_{k=0}^{w-1} f_i[k] \otimes f_j[k], \tag{5}$$

where $\otimes$ in (5) is a three-valued operation, defined using XOR operation, as follows:

$$a \otimes b = \begin{cases} 0, & \text{if at least one of } a \text{ or } b \text{ is a } \textit{don't-care}, \\ a \oplus b, & \text{otherwise.} \end{cases} \tag{6}$$

In the partitioning schemes that follow, performing the TCAM search on one partition can significantly improve performance.

4.2 First-Level (Maximum) Intersection Partitioning

The Maximum Intersection Partitioning (MXIP) scheme, which is explained in this section, partitions the filters in the database such that each partition would hold the maximum number of intersections among its filters. This way all possible matches for a packet will be concentrated within one partition only. In addition, partitions will be disjoint,

```
MXIP ( )
01: m = 0;
02: P_temp = {};
03: while (F ≠ {})
04: {
05:     m ⟵ m + 1;
06:     P_m ⟵ {f_1};
07:     F ⟵ F − {f_1};
08:     for (all f_i ∈ F)
09:     {
10:         if (∃ f_j ∈ P_m such that d_{i,j} = 0)
11:         {
12:             P_m ⟵ P_m ∪ {f_i};
13:             F ⟵ F − {f_i};
14:         }
15:     }
16:     if (| P_m |= 1)
17:     {
18:         P_temp ⟵ P_temp ∪ {f_1};
19:         m ⟵ m − 1;
20:     }
21:     n_m ⟵ | P_m |;
22: }
23: N_p ⟵ m;
24: if (P_temp ≠ {})
25: {
26:     N_p ⟵ N_p + 1;
27:     P_Np ⟵ P_temp;
28: }
```

Fig. 6. Pseudocode for MXIP.

i.e., any pair of partitions do not have any overlap in the filters that they contain. Since there would always be a number of filters that do not have any intersection with any other filter, one last partition is needed in which all these *distinct* filters can be placed.

The pseudocode for generating the partitions based on the concept of maximum intersection is shown in Fig. 6. In this pseudocode, F refers to the set of all filters, and P_m denotes the mth partition. Since distance computation has commutative and associative properties, any filter could be the seed of partition. For simplicity, we chose the seed to be the first element remaining in the original set F (i.e., f_1) in every iteration. The *for* loop grows the partition around the seed based on the MXIP heuristic. Lines 16-20 indicate that all distinct filters (that make no intersection with other filters) will be chosen to be assigned to a separate partition. N_p would be the total number of partitions generated based on the MXIP scheme. In line 21, n_m is the total number of filters in partition P_m. Hence, N_p partitions $(P_1, P_2, \ldots, P_m, \ldots, P_{Np})$ will be formed, from which the last partition (P_{Np}) is a collection of all distinct filters.

We apply the main loop of MXIP on the entire filter set (F) until the set is empty. The main loop is shown on lines 8 to 15. We choose a filter from F (the first filter) and add all zero distance filters from F to the first partition. Inside the partition generated, we add all zero distance filters from F to any filter in the partition generated. All filters that are placed in partitions are removed from the initial F set. After set F is emptied, and filters are placed in partitions, if any partition only contains one filter (it had no zero distance with any other filter), that filter is placed in partition P_{temp}. All distinct filters are placed in P_{temp}. N_p would be the number of partitions generated by MXIP, and since the distinct filter set is the last partition, P_{Np} would be the last partition.

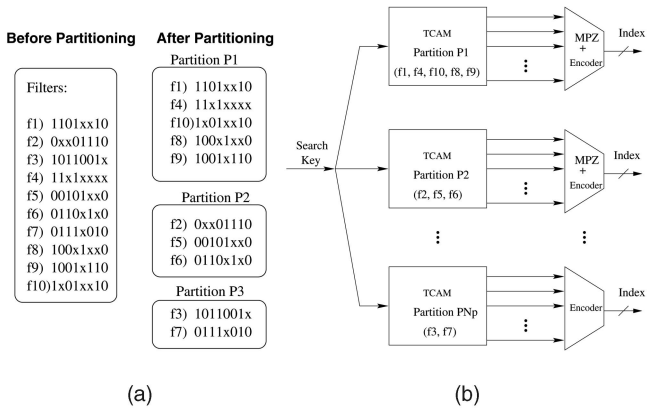


Fig. 7. Application of MXIP to a small example. (a) Partitioning. (b) The classifier engine.

Fig. 7a illustrates a small example of a set of 10 filters partitioned based on maximum intersections. Filters are assumed to be 8 bits long for simplicity. Filters f_1, f_4, and f_{10} have zero distance, hence they can form one partition, i.e., P_1. Also, filters f_8 and f_9 have zero distance with filter f_{10}; therefore, they are also placed in P_1. Filters f_5 and f_6 make a zero distance with filter f_2 and form partition P_2. Finally, filters f_3 and f_7, which have no zero distance with any other filters, form the distinct filter collection and are placed in a separate partition (P_3).

MXIP would ensure that all possible matches for a given search key are located in one partition. One possible architecture of a multimatch packet classifier using the MXIP approach is illustrated in Fig. 7b. All filters in each partition are placed in one TCAM module. The single-match priority encoder unit is replaced by an MPZ circuit along with an address encoder. The MPZ unit (as described in Section 3.1) is a customized PZ circuit that gives all the match lines in a prioritized sequence. The MPZ and encoder circuit connected to the TCAM provide the addresses of the r matches in at most r cycles. Performance of MPZ structure is not adversely affected by the location of matching filters as discussed in Section 3.2, since potential matching filters are grouped as closely as possible by the partitioning scheme. Note that the last partition (P_{Np}) does not need an MPZ unit since it would result in at most one match, and therefore, an address encoder is sufficient. No contention resolver (CR) in this architecture is required because all partitions can be searched in parallel. However, as we discuss in Section 5, having such resolver will significantly reduce the power consumption. In this example, if the search key "11010010" arrives, partition P_1 contains the matches, and the system would provide the three matching results f_1, f_4, and f_{10} in exactly three clock cycles.

4.3 Second-Level (Minimum) Intersection Partitioning

A classifier engine using MXIP finds all r matches in r cycles. By further partitioning the maximum intersected

```
MNIP ( )
01: for (m = 1; N_p − 1; m++)
02: {
03      z = 0;
04:     while (P_m ≠ {})
05:     {
06:         z ⟵ z + 1;
07:         P_{m,z} ⟵ {f_1};
08:         P_m ⟵ P_m − {f_1};
09:         for (all f_i ∈ P_m)
10:         {
11:             if (for all f_j ∈ P_{m,z}: d_{i,j} ≠ 0)
12:             {
13:                 P_{m,z} ⟵ P_{m,z} ∪ {f_i};
14:                 P_m ⟵ P_m − {f_i};
15:             }
16:         }
17:     }
18:     s_m ⟵ z;
19: }
```

Fig. 8. Pseudocode for MNIP.

filters intelligently, all r matching addresses can be found in only one cycle. The second level of partitioning is based on *minimum intersections* among the filters in each partition. This time, subpartitions are generated for as many number of completely overlapping filters. Filters in each subpartition should have a distance greater than zero among each other. This indicates that after minimum intersection partitioning (MNIP) there are no two filters that have a 100 percent overlapping in each subpartition, unlike the MXIP where overlaps were concentrated in one partition.

The pseudocode for generating minimum-intersected partitions is shown in Fig. 8. We assume that N_p number of partitions are generated by the MXIP method. P_m denotes the mth partition based on MXIP. Therefore, $P_{m,z}$ refers to the zth subpartition in P_m generated by the MNIP method. In line 18, s_m has the number of subpartitions in each partition.

The main loop of MNIP is applied to every MXIP partition. The main loop is shown on lines 9 to 16. $|P_m|$ is the number of filters in Partition m generated by the MXIP scheme. All zero distance filters are placed in separate partitions. This indicates that if all filters in the MXIP partition had zero distance with one another, the MNIP would generate $|P_m|$ partitions in the worst case.

If all subpartitions generated by the MNIP approach are placed in separate TCAMs, MNIP would result in finding all matches in one cycle. This is because of the fact that if the entire filter set is partitioned in such a way that each partition would result in at most one match for a given packet, since all partitions are searched in parallel, all matching addresses are ensured to be provided in only one cycle.

5 MX-MN-IP and Its Structural Advantages

The maximum-minimum intersection partitioning (MX-MN-IP) approach is obtained by first applying the MXIP and then the MNIP to the filters in the set. Subpartitions are placed in separate TCAMs and matching indices are provided at the output. A resolver unit is used in MX-MN-IP to perform the TCAM search on one partition only.

In this section, we summarize the main advantages of MX-MN-IP, i.e., low power, high speed, efficient update, and scalability for parallelism.

5.1 Power Saving

Performing the TCAM search on only a small portion of the entire database can significantly save power consumption due to the frequent charging and discharging of the highly capacitive match line. Power consumption is directly proportional to the number of entries being searched in parallel in a TCAM. The MXIP method effectively partitions the database, hence for each packet, only a small portion is being searched, while all others remain idle.

Since each subpartition (TCAM) would result in only one match, the PZ would not be required anymore. In addition, the last filter collection would result in only one match and would not need a PZ unit. This not only reduces the cost but also further improves the power saving. The priority encoder unit, which is a power hungry unit in a conventional TCAM structure, would be removed. Instead, a conventional address encoder can be used to provide the addresses of the matching results.

The MX-MN-IP approach can also be used in packet processors where the conventional single-match packet classification is desired. Traditionally, TCAMs perform the search and provide the index of the highest priority match in one cycle. However, our approach performs the TCAM search on a small fraction of the entire filter set and, thus, reduces the power consumption by at least one order of magnitude.

To achieve the power savings mentioned by partitioning, a *CR* unit should be designed so that the search mechanism would result in enabling the TCAM search on one partition, while disabling others. An identification code for each partition (a representative of the filters in that partition) can be defined to facilitate choosing the right partition. The following equation expresses how the ID is encoded for each partition:

$$ID_m[k] = \begin{cases} 0, & \text{if all } f_i[k]'s \text{ are } 0 \ (0 \le i \le n_m - 1), \\ 1, & \text{if all } f_i[k]'s \text{ are } 1 \ (0 \le i \le n_m - 1), \\ x, & \text{otherwise.} \end{cases} \quad (7)$$

In (7), $ID_m[k]$ is the bit position k for the ID code of a particular partition (i.e., P_m), where $0 \le k \le w-1$, $1 \le m \le N_p - 1$, and n_m ($|P_m|$) denotes the number of filters in the mth partition generated by MXIP. The ID code indicates how the filters in one partition are intersected. Hence, if for each partition a unique ID code is generated, when the packet arrives, an initial search based on searching these ID codes in parallel would result in one ID. All the ID codes can be placed in a small TCAM, as shown in Fig. 9. The arriving packet is compared against these IDs, and the match line of the matching ID entry would enable the partition it represents. Therefore, only one partition (TCAM) performs the search, while others remain idle. Since there is one partition (the last one) containing distinct filters with no zero distance among them, the ID

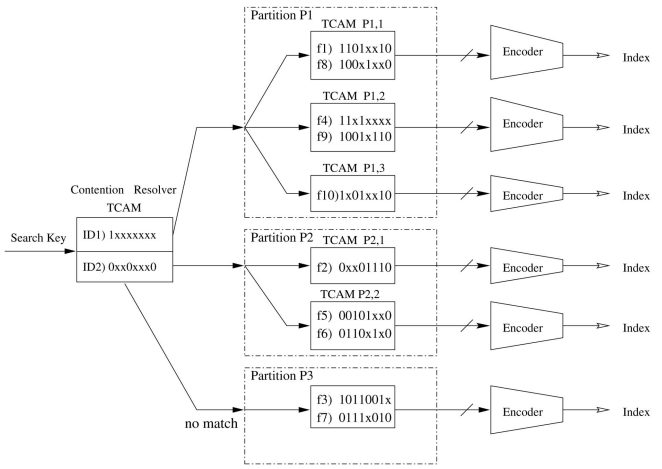


Fig. 9. Classifier engine using the MX-MN-IP approach.

encoding cannot be applied to this partition. In other words, for this partition no unique ID can be defined. As a result, this partition should be searched only if there is no match found at other partitions.

To be more clear, Fig. 9 shows the minimum-intersected partitions for the example and how they are placed in TCAMs. Fig. 9 also shows the ID codes for the partitions generated for the example. Note that partition P_3 is the set of distinct filters and, thus, does not hold any unique ID. The ID codes for partitions P_1 and P_2 are placed in a TCAM unit to perform the initial search. As shown, ID codes are generated based on the union of all maximum intersected filters in one partition. If packet "11010010" arrives, the initial TCAM (CR TCAM) would enable the first maximum-intersected partition, and the search key would be sent to all TCAMs generated based on the second partitioning heuristic. This packet matches filters f_1, f_4, and f_{10}, and their addresses are provided in one cycle.

It is important to point out that we can always come up with unique ID codes for each partition except the distinct filter collection that does not have one. However, the ID codes may not always have nonzero distance with one another. In other words, if two ID codes overlap (have zero distance), a certain search key may activate both partitions, while the result(s) will be found in only one partition. This issue can theoretically degrade the power performance of the system. However, the overlapping ID code cases are very unlikely to happen. The number of bits used for filter representation is relatively large (around 150 bits), making the ID codes very unlikely to intersect. In the worst case, if all ID codes overlap, an MPZ unit can be used after the CR TCAM to activate the search on multiple partitions sequentially to save power consumption. In such rare cases, our scheme would still perform better in terms of power than other systems (e.g., the SSA [11], which consumes power proportional to the total number of TCAM entries in every set that is accessed in each round of TCAM lookup). This relationship also shows how MX-MN-IP is coupled with the MPZ structure to maintain low power consumption. The overall speed of our system would also be limited to more than one TCAM lookup, due to performing multiple TCAM searches sequentially.

5.2 Updating Filters

Updating filters in MX-MN-IP deals with the following cases:

- **Filter Deletion.** Updating the packet classification database may result in the same partitions when one filter is deleted. However, if a filter that provided zero distance with a few sets of filters is deleted, partitions may alter. In this case, if those sets of filters have at least one nonzero distance with each other, that partition should be split into two separate partitions. This is due to the fact that the filter that combined some filters together no longer exists. Hence, some sets of filters do not have any zero distance with the other sets in that partition. This leads to creating a new partition, where the sets of filters are separated. In our running example, if filter f_{10} is deleted from the P_1 set, filters f_1 and f_4 do not have any intersection with filters f_8 and f_9 anymore. Therefore, partition P_1 would be divided into two partitions; one containing filters f_1 and f_4, and the other containing filters f_8 and f_9.
- **Filter Addition.** In most cases, adding a new filter does not change the partitions very much. This filter must be compared against the filters of each partition, to see where a zero distance is generated. The new filter should be added to the partition in which it makes a zero distance. However, in case this filter has a zero distance with more than one partition, all such partitions are combined into one partition.
- **ID Update.** Updating may also alter the ID codes in the CR. The ID code for each partition can be generated only after all filters in that partition are known. Based on the ID encoding, the insertion or deletion of a filter may cause a bit position in the ID code to change from 0 or 1 to x or vice versa as reflected in (7).

The filter partitioning would not add to the delay of the system. This is due to the fact that filter partitioning is assumed to be done only once when the system is assembled. The performance of the system, thus, depends on the components used to assemble it and not on the way filters are stored.

Updating prefixes or filters is an important issue in packet forwarding and classification systems, as it may require time-consuming processing and thus large delay in the system. As observed from Fig. 6, adding or deleting filters would lead to a maximum of two inner loops of processing filters for the MXIP scheme. Therefore, partitioning would have the complexity of $O(|F|^2)$. Updating filters would not really add to the delay, since updating generally does not occur as often as it does in packet forwarding.

As for updating, N newly inserted filters would generate $N \cdot O(|F|^2)$ update cost. However, this is valid when we only consider the software cost to partition the filter set, and it does not account for the hardware partition update cost. Insertion and deletion can cause partitions to be merged

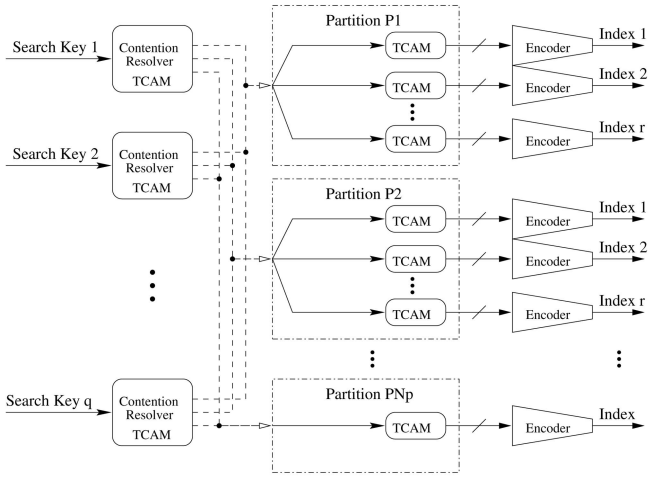


Fig. 10. Usage of multiple CR units to enhance parallelism.

and split, yielding changes to the tree structure and changes to the ID codes for the CR. The previous MMC schemes may lead to high update costs at extreme cases as well. When updating is a design concern, the MPZ design scheme alone can be employed, as it has the lowest update complexity.

5.3 Parallel Processing

Having a small and efficient CR is a critical feature for further performance increase. Multiple CR units can be utilized in the system such that multiple search keys can be processed in parallel. Such property makes our system even more attractive. Fig. 10 shows how multiple CRs can be used in the design to achieve high speedup. If any arriving packets do not require accessing the same partition, $q\,(q > 1)$ packets can be classified in at most one cycle. However, multiple packets may require accessing the same partition(s). Probabilistic analysis shows that the expected number of packets that can be processed simultaneously is approximately $q/2$ [17].

6 Experimental Results

6.1 Simulation of MPZ

An 8-bit MPZ unit was designed and implemented using Synopsys tools [19]. The inputs to the MPZ unit were assumed to be the TCAM output match lines. Assuming $m[7:0]$ to be "01011000" ($58H$) for the MPZ input lines, there would be three matches. Output results $EP[7:0]$ were observed to be "00001000" ($08H$), "00010000" ($10H$), and "01000000" ($40H$) on the first three clock cycles, respectively. The fourth clock cycle would provide all zeros at the output, indicating that no more matches were found. We assumed the enable line EN to be high until all matches are reported at the output. The encoder then receives $EP[7:0]$ from the PZ and reports all three matching addresses in three clock cycles.

Various size MPZ units were also designed and implemented. The maximum throughput of a single MPZ unit would be the smallest clock period for the circuit to function correctly. Table 2 compares the statistics for four MPZ units that we have implemented. As n grows, the delay and the cost of MPZ both increases. However, note carefully that growth of critical path delay, shown in the second column, is not an indication of improvement. To be more clear, the third column shows the estimated speedup (S_n) in each case compared to a software-based classification approach. We assumed that the host running the classification software needs at least one memory access and one comparison instruction per entry. This means the speedup will be $S_n \geq \frac{r \cdot n \cdot (t_{mem}+t_{cmp})}{r \cdot T_{Clk\_n}}$, where t_{mem} and t_{cmp} refer to the corresponding instructions, and T_{Clk_n} is the duration of the clock driving an n-bit MPZ; obviously, T_{Clk_n} should be larger than the delay values given in Table 2. Note that in practical cases $r \ll n$, and thus, as n grows the speedup will be even larger than 40-73 range given in the table because in our approach the overall performance depends on r and not n. Such property makes our hardware engine quite attractive in applications that require fast multimatch packet classification.

TABLE 2
Simulation Results for n-Bit MPZ Units

n	Delay [ns]	Area [NAND]	Speedup S_n
8	9.92	206	40.32
16	17.92	437	44.64
32	25.09	769	63.77
64	43.87	1373	72.94

Table 3 also summarizes the design specification for a 64-bit MPZ cell by cascading n-bit MPZ stages, for different values of n. The table also indicates increasing and decreasing trends for the worst-case delay (second column) and area cost of cascaded designs (third column), respectively. A designer can do the tradeoff and choose appropriate size MPZs that satisfy the time and area constraints.

Nowadays, packet sizes vary from 40 bytes (TCP acknowledge packets with only header and no payloads) to 1,500 bytes (Ethernet packets). On average, Internet packet size is reported to be 402.7 bytes [11]. On the other hand, typical TCAMs perform one lookup task in 4 ns. Based on the results in Table 2, an 8-bit MPZ could achieve 32-Gbps classification rate for minimum packet size and 324 Gbps for average-sized packets. Overall, MPZ speed is observed to be much higher than other software and hardware schemes.

TABLE 3
Implementing a 64-Bit MPZ
by Cascading Various Size MPZ Units

#× size MPZ	Delay [ns]	Area [NAND]
8×8-*bit*	11.01	2264
4×16-*bit*	19.71	2320
2×32-*bit*	30.11	2121
1×64-*bit*	48.26	1373

TABLE 4
Comparison of Various MPZ Units Implemented on Stratix FPGA

n-bit MPZ	Total # of Logic Elements	Total # of Pins	T_{Clk} [ns]	T_{pd} [ns]
8	33	24	4.239	12.222
16	61	41	7.148	16.818
32	110	74	11.175	20.144
64	208	139	18.291	27.308

6.2 FPGA Prototyping

The 8-bit MPZ design was implemented on the NIOS II Altera board [20]. The design was simulated in Quartus II simulator environment [21]. The MPZ design was implemented on device number EP1S10F780C6 of the Stratix 1s10 FPGA family series. As expected, all matches were found given one clock cycle each.

Various-bit MPZ units were also generated, and the symbol blocks were extracted in Quartus II. Table 4 compares different sizes of MPZ units implemented on the Stratix FPGA in terms of total number of Logic Elements (LEs), total number of pins required, and timing analysis results. The MPZ unit(s) occupied a very small portion of the FPGA. The number of LEs is directly proportional to the cost of the system. Therefore, the small area of the MPZ units is an indication of low costs. In the table, n is the size of MPZ, T_{Clk} indicates the minimum acceptable clock period for each MPZ size, and T_{pd} is the worst-case propagation delay from the least significant input bit to the most significant output bit. As observed, the results are slightly different from the results in Table 2, because of the different libraries used in the Synopsys and Quartus toolsets.

6.3 Simulation of the Partitioning Approach

We have applied our partitioning scheme to various randomly generated filter sets that were designed to have the same characteristics as real filter sets, e.g., numbers, sizes, and structures of filters. The random filter set generation and the partitioning method was simulated using MATLAB [22]. Table 5 shows the average statistics of the partitions generated. The filter sets were assumed to have 5,000 filters with bit-width of 150 each, which is typical for real databases. The second column shows the percentage of filters that generate multimatch results, as expressed by

$$\Delta = \frac{\sum_{m=1}^{N_p-1} |P_m|}{\sum_{m=1}^{N_p} |P_m|}. \quad (8)$$

In Table 5, r_{max} is maximum number of filters that would generate a match in each MXIP partition. The maximum size of partitions is $|P|_{max}$ and $|P|_{avg}$ is the average size of partitions. N_p is the total number of partitions generated by the MXIP scheme. The partitioning statistics indicate that for each set of simulation results, a total of N_p MXIP partitions would be generated, each holding a maximum of $|P|_{max}$ MNIP partitions. The size of the distinct filter collection is $|P_{N_p}|$. The last two columns show the ratios in percent for the largest and average-sized partitions, respectively.

As observed from the table, most partitions have very few entries, and most filters would be positioned in the distinct filter collection. This significantly reduces the power consumption by about two to three orders of magnitude, as in multimatch applications, the last big partition is rarely awakened.

The minimum size of subpartitions $|s|_{min}$ after applying MNIP is also shown in Table 5. The minimum size of subpartitions is 1 or 2 in all cases, which may cause a gap between the minimum and maximum sizes of partitions in real scenarios. However, note that the maximum size of MNIP partitions is around 8. This means that several small TCAM units plus a large TCAM (for the distinct filter collection) are required for the multimatch packet classification task. The size and number of partitions generated by our scheme are quite practical for mapping to available TCAMs. Off-the-shelf TCAM chips are available for various entry sizes [23]. Most TCAM products can be configured to allow independent and uneven portions (called buckets) of the TCAM chip to function in parallel [11], [24]. In addition, customized TCAMs can be always used for small entry sizes needed in MX-MN-IP. Load balancing mechanisms need not be employed as it is not a critical concern in our approach. Hence, in general, MX-MN-IP does not negatively affect TCAM space utilization. Moreover, our classifier engine using MX-MN-IP can ultimately be implemented as an ASIC, where uneven TCAM sizes would not matter.

TABLE 5
Partitioning Statistics

r_{max}	Δ%	$\|P\|_{max}$	$\|P\|_{avg}$	N_p	$\|P_{Np}\|$	$\|s\|_{min}$	ω_1 [%]	ω_2 [%]
12	30	12	9	88	4245	2	84.9	0.18
8	30	8	6	124	4225	2	84.5	0.12
4	30	4	3	263	4090	1	81.8	0.06
8	10	8	6	42	4742	2	94.8	0.12
8	20	8	6	81	4487	2	89.7	0.12
8	50	8	6	202	3705	1	74.1	0.12
8	70	8	6	289	3171	1	63.4	0.12

6.4 MX-MN-IP Speedup

The speed of our partitioning architecture is high compared to other conventional hardware-based designs. The delay time of a conventional TCAM can be written as $T_{TCAM} \approx T_{word} + T_{PE}$, where T_{word} is the delay of the TCAM word, and T_{PE} is the delay of the priority encoder.[2] For large TCAMs, $T_{word} \approx T_{PE} \approx T_{TCAM}/2$ [25]. Our approach removes the need of the priority encoder unit, and hence, T_{PE} does not come into picture at all. The delay of our design can be written as

$$T_{MX\text{-}MN\text{-}IP} \approx T_{CR} + T_{Pm} + T_{Encoder} + T_{P_{Np}}. \tag{9}$$

It can empirically be shown that the search delay of a small TCAM (i.e., CR unit and TCAMs holding subpartitions) is around $T_{TCAM}/4$ compared to large conventional TCAMs, and that the encoder unit would also have a latency of approximately $T_{TCAM}/4$ [17]. In the MX-MN-IP approach, the worst case is assumed to be searching one of the maximum-intersected partitions and searching the set of distinct filters afterward. Hence,

$$\begin{aligned} T_{MX\text{-}MN\text{-}IP} &\approx T_{TCAM}/4 + T_{TCAM}/4 + T_{TCAM}/4 + T_{TCAM}/4 \\ &\approx T_{TCAM}. \end{aligned} \tag{10}$$

A conventional multimatch TCAM-based approach would spend at least $r \cdot (7 \times T_{TCAM})$ cycles for finding r matches [2]. Our approach finds r matches in one conventional TCAM cycle, which is far better than software and hardware approaches. Hence, for finding r matches, we obtain speedup of at least

$$S = \frac{r \cdot (7 \times T_{TCAM})}{T_{TCAM}} = 7r, \tag{11}$$

where r would be the maximum number of possible matches in a filter set for a given search key. For a maximum of eight matches, the MPZ design would gain a speedup of 9.18 and the MX-MN-IP approach would achieve a speedup of 56 compared to the conventional approach, which is more than an order of magnitude higher. Note that, in our architecture, performance does not degrade when the number of matches increases. Scalability feature makes our design attractive for high-speed packet processing. Assuming 4-ns lookup time for TCAM, the MX-MN-IP scheme would achieve at least 80- and 805.4-Gbps classification rates on average. This is well above the highest MMC rate reported in literature, e.g., 20 Gbps/201.35 Gbps for minimum/average packet size under the same conditions in the SSA scheme. It is clear that parallel processing would achieve even higher rates.

6.5 Cost

The cost of a conventional TCAM is approximately $A_{TCAM} \approx A_{word} + A_{PE}$, where A_{word} and A_{PE} are the TCAM word area and the priority encoder area, respectively. The area of our classifier engine can be written as

$$A_{MX\text{-}MN\text{-}IP} \approx A_{word} + A_{Encoder} + A_{CR}. \tag{12}$$

As the priority encoder is a very expensive unit, for any size: $A_{PE} > A_{Encoder} + A_{CR}$. Hence, the overall cost (area) of our classifier is approximately the same as a conventional (off-the-shelf) TCAM-based classifier.

6.6 Power Estimates

For the common case of nonzero distance between any two ID codes, the CR TCAM triggers exactly one MXIP partition for each search key. The overall power consumption per search depends on the size of the partition triggered (i.e., $|P_m|$ size of partition m). Suppose $|F|$ is the size of the entire filter set, and $|P_{max}|$ ($|P_{min}|$) is the maximum (minimum) size of a partition. As power consumption of a TCAM is linearly proportional to its size, power saving (ΔE), when partition m is triggered, falls within the following bounds:

$$\frac{|F| - |P_{max}|}{|F|} \le \Delta E \approx \frac{|F| - |P_m|}{|F|} \le \frac{|F| - |P_{min}|}{|F|}. \tag{13}$$

To elaborate on power savings more accurately, consider a general case when n search keys are being analyzed. Let the total number of search keys be written as $n = n_{single} + n_{multi}$, where n_{single} is the number of classification searches activating the largest partition and resulting in a single match, and n_{multi} represents the number of searches that activate partitions other than the distinct filter collection. On average, power consumption of our system for n search keys is

$$\begin{aligned} E_{MX\text{-}MN\text{-}IP} &= n_{single} \cdot E_{max} + n_{multi} \cdot E_{avg} \\ &\propto n_{single} \cdot |P_{N_p}| + n_{multi} \cdot |P|_{avg}. \end{aligned} \tag{14}$$

In this formula, E_{max} is the maximum power of our system that is generated by triggering the largest partition (i.e., the largest partition $|P_{N_p}|$). Symbol $\propto$ reflects proportionality relationship between power and size. The average power of our system (E_{avg}) for a typical multimatch search is proportional to the number of entries for an average-sized partition ($|P|_{avg} = \lfloor \frac{|F| - |P_{N_p}|}{N_p - 1} \rfloor$). By defining three ratio factors $r_{multi} = \frac{n_{multi}}{n}$, $\omega_1 = \frac{|P_{N_p}|}{|F|}$, and $\omega_2 = \frac{|P|_{avg}}{|F|}$, we conclude

$$E_{MX\text{-}MN\text{-}IP} \propto [(1 - r_{multi})\omega_1 + r_{multi}\omega_2] \cdot n \cdot |F|. \tag{15}$$

Average power consumption for n searches in a TCAM is proportional to the total number of entries, i.e., $E \approx n \cdot |F|$. Therefore, for n search keys, MX-MN-IP power saving (compared to single-TCAM implementation) can be written as

$$\begin{aligned} \Delta E &= 1 - [(1 - r_{multi})\omega_1 + r_{multi}\,\omega_2] \\ &= (1 - \omega_1) + r_{multi} \cdot (\omega_1 - \omega_2). \end{aligned} \tag{16}$$

In order to prove that MX-MN-IP can always achieve power savings compared to the conventional approach, we need to show that $\Delta E > 0$. This relationship is always true, since per definition $0 \le r_{multi}$, ω_1, $\omega_2 \le 1$ and $\omega_1 > \omega_2$ (in fact, in most cases $\omega_1 \gg \omega_2$). The experimentation tabulated

2. Note that, in this paper, we consider T_{TCAM}, i.e., one TCAM lookup cycle, as the main clock cycle for time measurement. The ultimate clock frequency of a running oscillator in the final implementation of classifier will depend on T_{TCAM}, I/O pins, and other cores and interface units.

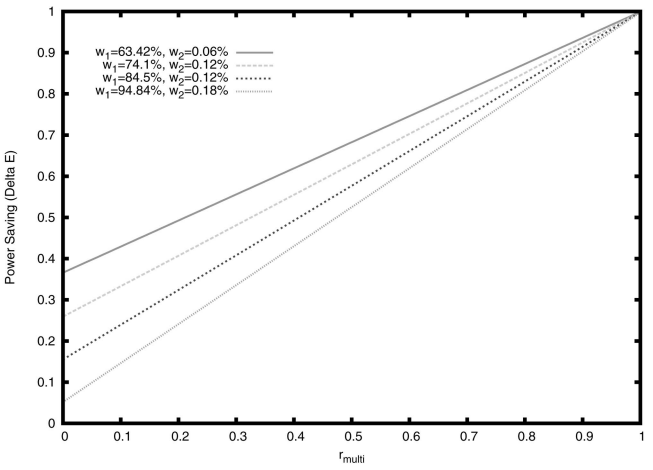


Fig. 11. Power saving versus r_{multi} for different values of ω_1 and ω_2.

in the last two columns in Table 5 confirms these relations clearly.

Equation (16) has been plotted in Fig. 11 to show the power saving ΔE versus r_{multi} for different values of ω_1 and ω_2. Clearly, significant power reduction (5 percent to 99 percent) is achieved compared to conventional (single-partition) TCAM-based approaches. These analyses indicate that our approach is highly efficient for applications in which r_{multi} is large, i.e., when multiple matches (as opposed to a single match) frequently occur.

6.7 Comparing with Other Techniques

Our TCAM-based architectures for multimatch packet classification are superior than other similar approaches in most practical aspects. Table 6 compares various TCAM-based multimatch packet classifiers, including ours. The key metrics such as power reduction, speed, usage of extra memory (extra TCAM entries), and so forth are shown. In the update column, N stands for the newly inserted filters, and f stands for the number of header fields. While the MX-MN-IP approach provides better results in terms of speed and power consumption, the MPZ architecture performs more efficiently in terms of update complexity.

7 Conclusion

We have introduced two multimatch classifier engines. The first one, which is the MPZ architecture, is capable of finding all r matches in exactly r cycles. The MPZ could achieve speedup of 70 and 9 compared to software and conventional TCAM-based approaches, respectively. The MPZ design can operate at wire speed (i.e., approximately three OC-192 using 0.18-μm technology) without performance degradation. Second, intersection-driven partitioning techniques that use the concept of maximum and minimum intersections among filters to efficiently partition the entire filter set were proposed. We achieve significantly higher performance compared to conventional TCAM-based approaches. Power consumption was also reduced by one order of magnitude or more, due to performing the TCAM search on a small portion of the packet filter set. Our design can be employed by IPv4/IPv6 packet classifiers that utilize TCAMs in their architectures.

References

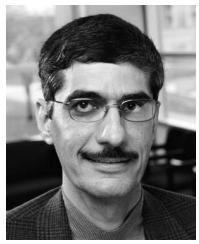
[1] K. Zheng, H. Che, Z. Wang, and B. Liu, "TCAM-Based Distributed Parallel Packet Classification Algorithm with Range-Matching Solution," *Proc. IEEE INFOCOM*, 2005.
[2] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary, "Algorithms for Advanced Packet Classification with Ternary CAMs," *Proc. ACM SIGCOMM '05*, Aug. 2005.
[3] *SNORT Network Intrusion Detection System*, www.snort.org, 2008.
[4] F. Yu, R.H. Katz, and T.V. Lakshman, "Efficient Multimatch Packet Classification and Lookup with TCAM," *Proc. 12th Ann. IEEE Symp. High Performance Interconnects (HOTI '04)*, pp. 28-34, Aug. 2004.
[5] D.E. Taylor and E.W. Spitznagel, "On Using Content Addressable Memory for Packet Classification," Technical Report WUCSE-2005-9, Mar. 2005.
[6] E. Spitznagel, D. Taylor, and J. Turner, "Packet Classification Using Extended TCAMs," *Proc. 11th IEEE Int'l Conf. Network Protocols (ICNP '03)*, pp. 120-131, Nov. 2003.
[7] F. Yu, R.H. Katz, and T.V. Lakshman, "Gigabit Rate Packet Pattern-Matching Using TCAM," *Proc. 12th IEEE Int'l Conf. Network Protocols (ICNP '04)*, pp. 174-183, 2004.
[8] H. Song and J.W. Lockwood, "Efficient Packet Classification for Network Intrusion Detection Using FPGA," *Proc. ACM/SIGDA 13th Int'l Symp. Field-Programmable Gate Arrays (FPGA '05)*, Feb. 2005.

TABLE 6
Comparison of Various TCAM-Based Multimatch Packet Classifiers

Design Scheme	PE in TCAM	Power Reduction	Speed	Extra Memory	Effi cient CR	Multi-thread support	Update Complexity
MPZ Approach	**Required Modifi ed**	**No**	**r-cycles**	**No**	**No**	**Yes**	$\mathbf{O(1)}$
MX-MN-IP Approach	**Not Required**	**Yes signifi cantly**	**1-cycle**	**No**	**Yes**	**Yes**	$\mathbf{N \cdot O(\lvert F \rvert^2)}$
Katz et al. [4]	Not Required	No	1-cycle	Yes too many	NA	Yes	$O(N^f)$
Lockwood et al. [8]	Not Required	Yes not signifi cantly	at least r cycles	No	NA	Yes	$O(N^f)$
SSA Katz et al. [11]	Not Required	Yes not signifi cantly	at least 2-cycles	Yes a few entries	No	Yes	$O(N^f)$
MUD [2]	Required	No	at least r-cycles at most 20 cycles	Yes, bitwidth TCAM entries no	NA	Yes	$O(N)$
Entry Invalidation [2]	Required	No	$7 \times r$ cycles	No	No	No	$O(N)$

[9] N.F. Huang, W.E. Chen, J.Y. Luo, and J.M. Chen, "Design of Multi-Field IPv6 Packet Classifiers Using Ternary CAMs," *Proc. IEEE Conf. Global Telecomm. (GLOBECOM '01),* vol. 3, pp. 1877-1881, Nov. 2001.
[10] N.F. Huang, K.B. Chen, and W.E. Chen, "Fast and Scalable Multi-TCAM Classification Engine for Wide Policy Table Lookup," *Proc. 19th IEEE Int'l Conf. Advanced Information Networking and Applications (AINA '05),* vol. 1, pp. 792-797, Mar. 2005.
[11] F. Yu, T.V. Lakshman, M.A. Motoyama, and R.H. Katz, "SSA: A Power and Memory Efficient Scheme to Multi-Match Packet Classification," *Proc. ACM Symp. Architecture for Networking and Comm. Systems (ANCS '05),* pp. 105-113, Oct. 2005.
[12] F. Yu, T.V. Lakshman, M.A. Motoyama, and R.H. Katz, "Efficient Multimatch Packet Classification for Network Security Applications," *IEEE J. Selected Areas in Comm.,* vol. 24, no. 10, pp. 1805-1816, Oct. 2006.
[13] C. Kun, S. Quan, and A. Mason, "A Power Optimized 64-Bit Priority Encoder Utilizing Parallel Priority Look-Ahead," *Proc. IEEE Int'l Symp. Circuits and Systems (ISCAS '04),* vol. 2, pp. 753-756, May 2004.
[14] M. Faezipour and M. Nourani, "A Customized TCAM Architecture for Multi-Match Packet Classification," *Proc. IEEE Global Telecomm. Conf. (GLOBECOM '06),* pp. 1-5, Nov. 2006.
[15] C.H. Huang, J.S. Wang, and Y.C. Huang, "Design of High-Performance CMOS Priority Encoders and Incrementer/Decrementers Using Multilevel Lookahead and Multilevel Folding Techniques," *IEEE J. Solid-State Circuits,* vol. 37, no. 1, pp. 63-76, Jan. 2002.
[16] J.S. Wang and C.H. Huang, "High-Speed and Low-Power CMOS Priority Encoders," *IEEE J. Solid-State Circuits,* vol. 35, no. 10, pp. 1511-1514, Oct. 2000.
[17] M. Faezipour, "High Speed Multi-Match Packet Classification Using TCAM," master's thesis, UTDEE-11-2006, Nov. 2006.
[18] M. Nourani and M. Faezipour, "A Single-Cycle Multi-Match Packet Classification Engine Using TCAMs," *Proc. 14th IEEE Symp. High-Performance Interconnects (HOTI '06),* pp. 73-78, Aug. 2006.
[19] *User Manuals for SYNOPSYS Toolset Version 2005.06,* Synopsys, 2005.
[20] *User Manuals for NIOS II IDE Version 6.0 Toolset,* ALTERA, 2006.
[21] *User Manuals for Quartus II Version 6.0 Toolset,* ALTERA, 2006.
[22] *User Manuals for Matlab 7.0 Toolset,* MathWorks, 2005.
[23] *IDT: Integrated Device Technology,* www.idt.com, 2008.
[24] Y.-K. Chang, "Power-Efficient TCAM Partitioning for IP Lookups with Incremental Updates," *Proc. Int'l Conf. Information Networking (ICOIN '05),* pp. 531-540, Jan./Feb. 2005.
[25] M.J. Akhbarizadeh, M. Nourani, and C.D. Cantrell, "Segregating the Encompassing Prefixes to Enhance the Performance of Packet Forwarding Engines," *Proc. IEEE Global Telecomm. Conf. (GLOBECOM '04),* pp. 1612-1616, Nov./Dec. 2004.

Miad Faezipour received the BSc degree in electrical engineering from the University of Tehran, Tehran, Iran, in 2002 and the MSc degree in electrical engineering in 2006 from the University of Texas at Dallas (UTD), where she focused on hardware-based architectures for packet classification. She is currently a PhD candidate at the University of Texas at Dallas. Her research interests lie in the broad area of high-speed packet processing in hardware and deep packet inspection architectures. She is a member of the Center for Integrated Circuits and Systems, UTD, and a student member of the IEEE.



Mehrdad Nourani received the BSc and MSc degrees in electrical engineering from the University of Tehran and the PhD degree in computer engineering from Case Western Reserve University. He is currently an associate professor of electrical engineering at the University of Texas at Dallas. His current research interests include system-on-chip testing, signal integrity modeling and test, and high-speed packet processing methodologies and architectures. He has published more than 130 papers in journals and refereed conference proceedings including a Best Paper Award at the 2004 International Conference on Computer Design (ICCD). He is a recipient of the National Science Foundation Career Award (2002) and Cisco Systems URP Award (2004). He is a senior member of the IEEE and a member of the IEEE Computer Society and of the ACM Special Interest Group on Design Automation (SIGDA).