LOP: A Novel SRAM-based Architecture for LOw Power Packet Classification

Xin He, Jorgen Peddersen, Sri Parameswaran
School of Computer Science and Engineering, National ICT Australia
The University of New South Wales, Sydney, NSW 2052, Australia
{xinhe,jorgenp,sridevan}@cse.unsw.edu.au

UNSW-CSE-TR-0907
April 2009

School of Computer Science and Engineering
The University of New South Wales
Sydney, NSW 2052, Australia

Abstract

Performance of packet classification algorithms is an important area of concern in modern networks. Algorithms for matching incoming packets from the network to pre-defined rules, have been proposed by a number of researchers. Current software-based packet classification techniques have low performance, so many researchers have moved their focus to new architectures encompassing both software and hardware components. Some of the newer hardware architectures exclusively utilize Ternary Content Addressable Memory (TCAM) to improve performance of rule matching. However, this results in systems with very high power consumption. TCAM consumes a high amount of power as the entire memory array is read during any given access, much of which may not be necessary. In this paper, we propose a novel SRAM-based (named *LOP*) architecture where incoming packets are compared against parts of all rules simultaneously until a single matching rule is found for the compared bits in the packets, significantly reducing power consumption (i.e., only a segment of the memory is compared to the incoming packet). This comes with a penalty of time to match a single packet, but multiple packets can be compared in parallel to improve throughput beyond the levels of TCAM approaches.

Nine different benchmarks were tested in with two classification systems, with results showing that *LOP* architectures provide high lookup rates and throughput, and consume low power. Compared with a low power commercial TCAM approach, *LOP* achieves power reduction of more than 60% with equivalent throughput, and a power reduction of about 20% with high throughput (220 million searches per second (Msps) compared to 66Msps). Furthermore, energy can be reduced by up to 75% compared with commercial TCAMs in 0.18μm CMOS technology.

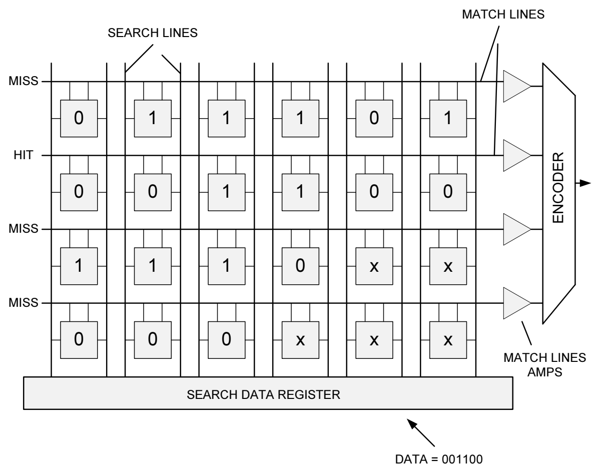


Figure 1: Four Entries with Six Cells TCAM Architecture

1 Introduction

Packet Classification methods serve to detect network intrusion, enable the deployment of Quality of Service (QoS) techniques, and facilitate the use of firewalls in large networks. Incoming packets in the network contain information that determines how they should be handled (such as IP addresses and TCP ports). Packets Classification methods match this information to a pre-defined set of rules. The rules that match the packets indicate the tasks which need to be performed. The information field can be refereed to as *decision bits*. As speeds of networks increase and requirements of users grow, rules get more complex, resulting in multifarious packet matching algorithms and architectures which aim to speed up the process. Typical parameters matched are a combination of IP addresses, TCP ports, and QoS parameters.

Recently, researchers have focussed on building novel architectures for packet classification based upon algorithms which exhibit high performance and low power consumption [24]. These algorithms were categorized in [12] into basic search algorithms, geometric algorithms, heuristic algorithms, and hardware-specific search algorithms. The first three groups are software-based algorithms and are usually slow and memory inefficient, but are flexible and easy to implement. The final group of hardware specific algorithms require special hardware with specialized architectures, but result in faster implementations. This paper deals with one such high speed hardware architecture.

Typically, these hardware algorithms are implemented using either Content Addressable Memory (CAM) or Ternary Content Addressable Memory (TCAM). CAMs and TCAMs are fully associative memories that allow searching for a particular data value. The search value is input to the CAM/TCAM and if a matching entry is found, the address of that entry is output. Each CAM cell stores a binary '1' or binary '0'; and each TCAM cell stores a '1', '0' or a 'don't care'. '0' and '1' values match only their specified value in the corresponding bit of the search input, while the *don't care* value, often shown as 'x', will match either of the possible search values (and hence always matches). Both CAM and TCAM are able to process one packet[1] per clock cycle. Typically hundreds (up to 128K) of 96-bit to 144-bit wide rules are stored into a CAM or TCAM. Figure 1 illustrates the basic architecture of TCAM-based packet classification system. Data in the search register is compared to every cell along the search columns, with each cell reporting whether it matches or mismatches. The match lines are enabled if and only if all the cells in the row indicate a match. The encoder calculates the address of the activated line on a match.

However, CAM and TCAM are costly to implement and consume enormous amounts of power. A commercial CAM cell is normally made up of 10 transistors (equal to an SRAM cell plus one XOR gate) and a TCAM cell typically uses 16 transistors (equal to two SRAM cells plus one XOR gate) [22]. Note that an SRAM cell only requires 4-6 transistors. CAMs/TCAMs are less dense (cells per unit area) than SRAMs since each CAM/TCAM cell is made up of a significantly greater number of transistors. Since, in CAM- and TCAM-based classification systems, all rules are simultaneously checked against an incoming packet, such systems exercise every single cell during every comparison, resulting in increased consumption of energy and power. For a typical usage pattern, an average TCAM cell consumes approximately 100 times the power of an SRAM cell [26]. Such TCAM-based systems in routers consume 30% - 40% of the total line card power necessitating a costly cooling system for routers. For example, SibreCore's low power SCT2000 TCAM consumes 1.7μW/cell (without power management), but has a clock cycle width of 15ns in 0.18μm CMOS technology [7]. Some TCAM vendors allow for higher clock speeds at the cost of an increase in power consumption. For instance, Analog Bits provides a 512 * 144 TCAM which has a 1.25ns clock period, but consumes 13.5μW/cell at a supply voltage 2.5V [7].

In this paper we propose a multiple-stream SRAM-based architecture called *LOP* which achieves high performance and consumes less power (and energy). By initially matching the first bit of a packet to the first field[2] of all the rules, a number of non-matching rules are eliminated. Then the second bit of the incoming packet is matched against the second field of all rules and further rules are eliminated. The process continues until only one rule remains. By carefully arranging the fields of the rules, it is possible to match the packets without accessing all fields of all the

[1]The term *packet* is used to refer only to the search information being used for classification of a typical packet.

[2]The term *field* in this work is used to refer to the equivalent information of a TCAM cell. The three possible values of a TCAM cell require two standard SRAM bits to encode (e.g., "00" for '0', "01" for '1' and "10" or "11" for 'don't care'). Thus, the *LOP* SRAM will have the same number of fields as an equivalent TCAM has cells.

rules. By creating an architecture which simultaneously matches many packets (multiple-streams) at the same time, and by carefully crafting the architecture to save power, it is possible to exceed the throughput of a CAM/TCAM based system while consuming less energy.

The remainder of the paper is organized as follows. A summary of related work is presented in Section 2. In Section 3, the *LOP* approach and architecture are described. Section 4 describes two packet classification systems. Section 5 shows the experimental setup and results. Finally, the paper is concluded in Section 6.

2 Related Work

In the past few years, researchers have proposed several algorithms, both in software and hardware for packet classification. Some of the software-based packet classification methods include linear searching [12], grid-of-tries [28], HiCuts [11], HyperCuts [23], tuple space search [27], and recursive flow classification [10]. However, none of these existing software-based packet classification methods are capable of meeting the ultra high performance requirements of modern networks. Thus, researchers have increasingly moved their focus from the software domain to hardware-based packet classification methods. Typically such hardware schemes are implemented using CAMs or TCAMs. In particular, TCAMs have been widely employed in high performance network devices, such as routers, for packet classification [15, 17, 25, 26]. There are mainly three different categories of methods to reduce power consumption without sacrificing the throughput of the system. The first group of methods reduce CAM/TCAM power at the circuit level; the second group of methods reduce power by partitioning rules at the system level (thus, only a selection of the rules are searched, reducing power); and the third group of methods build novel hardware architectures which replace CAM/TCAM with other type of memories.

Examples of the first category of methods are described in [2, 5, 20, 21, 30]. The authors of [30] proposed a low power CAM using pulsed NAND-NOR match-line which reduced the match-line power. By using a charge-recycling search line driver, the authors further reduced the search line power. In [20, 21], the authors proposed a pipelined hierarchical CAM architecture. This architecture reduces match-line power by breaking CAM into several segments to support pipelined matching, and reduces search-line power by using a hierarchical search-line scheme. The proposed architecture [21] can reduce power consumption by 60% when compared to a non-pipelined commercial CAM architecture, with a 7ns clock cycle in 0.18μm CMOS technology. Mohan [18] introduced three cell-level design techniques for TCAMs to reduce power consumption. The first technique reduces the cell-leakage power by using smaller supply voltage; the second technique reduces the leakage of TCAM by removing two access transistors and the third technique reduces the match-line capacitance by modifying the comparison logic. The results show a 25% reduction in energy.

The second category of methods, that of partitioning rules at the system-level, is orthogonal to designing a new type of CAM/TCAM-based packet classification system. Spitznagel et al. [26], Zane et al. [32] and Basci et al. [8] cope with the high power consumption problem of CAM/TCAM at the system-level by carefully partitioning rules. The authors in [32] present a power-efficient TCAM-based forwarding engine called CoolCAM. The core idea of CoolCAM is to divide the routing table into several sub-tables. Each sub-table is stored into one or more TCAM blocks. Several selected bits of the incoming packet are used to hash to one of the sub-tables in the first-stage of lookup. The authors provided two architectures which differ in the first-stage lookup. The first architecture utilizes several destination address bits as index and the second architecture used a small trie to form a prefix of the destination address. By performing a partial lookup in the first stage, only one part of TCAM needs to be searched in the second stage, thus reducing power and energy consumption. The authors in [26] proposed an architecture, called the Extended TCAM, which exploited a similar idea as [32]. In [26], the authors partitioned the rules into several TCAMs and each TCAM is associated with an index which is stored in a separate index TCAM. One heuristic algorithm was used to process the rules and return a set of indices and subsets of rules which were then populated into the respective TCAMs. Extended TCAM architectures can handle 100 million searches per second. The authors in [19] introduced a TCAM-based single-cycle multi-match (the previous methods usually match a single rule) packet classification architecture, which saved power by carefully partitioning the rules.

The third category of methods do not use CAMs or TCAMs, but use other types of memories instead. Kaxiras et al. [13, 14] proposed a set-associative memory based (SRAM-based) IP-lookup approach called IPStash which saves up to 64% power consumption compared to commercial TCAMs. IPStash functions as a set-associative cache and the routing prefixes, are inserted in this structure. An iterative longest prefix match approach was proposed to find the most appropriate indices for different prefixes to achieve a minimum level of required associativity. This pre-processing heuristic allows insertion of route prefixes of different lengths in IPStash into an efficient way. Furthermore, they proposed a class partitioning scheme that speeds up the longest prefix match approach. Due to the cheaper price and low power consumption of SRAM-based IP-lookup architecture, IPStash is competitive with CAM/TCAM-based IP-lookup for a routing system. However, IPStash cannot be easily developed to support matching against multiple domains (e.g. matching TCP source and destination port) due to the complexity of rules. Index selection will be one issue which limits the performance of this architecture. Further limitation are that IPStash cannot support 'don't care' values because single SRAM cells can only contain '0' and '1' without encoding logic and TCP ranges cannot be simply mapped to prefixes as TCAM does. Therefore, to support modern routing systems (which need to classify packets in multiple domains), IPStash needs to be modified by adding extra logic to make it feasible.

In this paper, we propose a low power SRAM-based packet classification architecture which can be used as a replacement of TCAM-based packet classification for modern routing systems.

2.1 Contributions and Limitations

The main contributions of this paper can be summarized as follows:

- A novel SRAM-based architecture for packet classification which is capable of supporting ”don’t care” values within rules.
- A novel packet matching scheme which is both parallelizable, and terminates upon a match without having to search the entire rule space.
- Exploration of the design space of the SRAM-based architecture.

The limitations of this work can be outlined as follows:

- *LOP* relies on heuristic pre-processing of rules and thus has limited support for runtime updates of the rules. Therefore, *LOP* should be used in systems which infrequently adjust their rules.
- Area comparisons are not given in this paper. This is due to the non-availability of comparative data for other implementations.
- *LOP* performs poorly when using classification systems that utilize large amounts of overlapping rules within isolated rule sets (such as random rule sets).

3 Methodology and Hardware Design

3.1 LOP Scheme

To achieve high lookup rates, TCAM-based packet classification schemes utilize a fully-parallel architecture which matches every bit of an incoming packet with every bit of rules in parallel. Due to this parallel matching architecture, TCAM-based packet classifications can achieve a constant high lookup rate of one packet per clock cycle. Equivalent lookup rate to this TCAM-based approach can be achieved by matching S different packets within an average of S clock cycles. Our LOP scheme utilizes this alternate throughput method by matching many packets simultaneously to only a selective part (i.e., several fields) of all rules in each clock cycle. This regains similar per-cycle throughput to TCAM-based approaches, while a higher clock rate is possible and only a fraction of the power is consumed. Table 1 provides a sample set of rules that we will use to demonstrate the *LOP* packet

Field Rule	0	1	2	3	4	5	6	7	8	9	10	11
0	0	x	x	1	0	0	0	1	1	1	0	1
1	1	0	x	0	0	x	0	1	x	0	x	x
2	1	1	0	0	x	0	1	0	x	x	1	x
3	x	1	0	0	0	1	x	0	0	1	1	1
4	1	1	1	0	1	x	x	1	0	0	1	1
5	1	x	x	0	0	x	1	0	x	1	1	1
6	1	0	1	0	1	1	1	0	0	x	x	1
7	0	1	1	0	1	x	0	x	x	1	0	0
8	1	1	x	x	x	0	0	x	1	1	1	1
9	0	1	1	0	1	0	0	0	0	1	1	1

Table 1: A sample rule table with 10 rules of 12 decision fields

Bit Packet	0	1	2	3	4	5	6	7	8	9	10	11
1	1	1	0	0	1	0	1	0	0	1	1	1
2	1	0	1	0	0	1	1	0	1	0	1	1
3	0	1	1	0	1	1	0	1	0	1	0	0

Table 2: A sample packet table with 3 packets of 12 bits

matching algorithm. For ease of understanding, the rule table is simplified to match packets with twelve decision bits to one of ten rules. Table 1 shows the values of the twelve decision fields for each rule, one rule per row, with the columns showing the field numbering from left to right. Each field of a rule can have three values with '0' in the rule only matching '0' in the packet, '1' only matching '1' and 'x', the don't care value, matching either a '0' or '1' in the packet. Table 2 lists some packets we will use as input to demonstrate the *LOP* methodology.

For simplicity, we will first show how a single packet can be matched to a rule by only testing a subset of the rule fields. Table 3 shows the running result at each step of the algorithm operating on packet 1 from Table 2 with the value "110010100111". One field from each rule is compared to the corresponding packet bit at each step (the number of the matched bit is shown in column 2 of Table 3). In the first step, field 0 of each rule is compared to bit 0 of the packet in parallel (bit 0 of the packet has the value '1', matching rules with '1' or 'x' at field 0). The result of this comparison is listed in columns 3-12 of Table 3, one column for each rule with a '1' representing a match and a '0' representing a mismatch. The row for step 1 tells us that rules 1, 2, 3, 4, 5, 6 and 8 match at bit 0. In the second step, we compare field 1 of the rules with bit 1 of the packet (i.e., '0') and perform a logical AND with the results from step 1 to provide the result of step 2. This result represents all rules matching both bits 0 and 1 from packet (i.e., rules

Step	Matched Bit	Matching Rules									
		0	1	2	3	4	5	6	7	8	9
1	0	0	1	1	1	1	1	1	0	1	0
2	1	0	0	1	1	1	1	0	0	1	0
3	2	0	0	1	1	0	1	0	0	1	0
4	3	0	0	1	1	0	1	0	0	1	0
5	4	0	0	1	0	0	0	0	0	1	0
6	5	0	0	1	0	0	0	0	0	1	0
7	6	0	0	1	0	0	0	0	0	0	0

Table 3: Matching a single packet (1) one bit at a time.

2, 3, 4, 5 and 8). This process continues with each step performing a comparison masked by the previous step's result, until a case where there is only a single or no matches left, as shown in the remainder of Table 3. For our example, this happens at the seventh step, when it is determined that rule 2 is the only remaining rule that matches (having compared with bits 0-6 of the packet). At this point, the remainder of rule 2 can be read from memory and compared with the packet to ensure its untested bits match as well. Note that the number of bits compared is much less than that used by TCAM approaches, as the comparison against all rules stops as soon as a single rule entry is found. This example only tested one packet with one bit per step. We now expand this procedure to operate on multiple packets simultaneously, and to compare multiple bits from the packets in each step. Allowing for multiple simultaneous packets requires multiple result vectors, one for each simultaneous packet. Comparing multiple fields from the rules could (for two rules per step) compare fields 0 and 6 with the corresponding packet bits within the first step, then fields 1 and 7 in the next step and so on. Table 4 shows how these expansion techniques can be employed. For this example, we will look at matching two packets simultaneously, comparing two sparse bits (separated halfway through the rule length) at a time. Each step now provides two results vectors for the two simultaneous packets respectively. The packet number (from Table 2) of the two packets currently being matched by each step are provided in column 3 of Table 4. The packet number is highlighted in column 3 whenever a new packet is begun. Columns 4-13 list the matched rules and a row is highlighted if a single packet match is found (meaning we are done with that packet and a new packet can start in the next step).

Step 1 of Table 4 shows the result at the first stage. In this case, we match both fields 0 and 6 of each rule to the corresponding bits in both packets. The result of the comparison with packet 1 is shown in the upper part of the step 1 row and the result for packet 2 is listed below. Step 2 operates as it did for the previous example, comparing bits 1 and 7 and masking against the previous step's results. In step 3, we find our first single rule match for packet 2. The only rule it can match

Step	Matched Bits	Packet No.	Matching Rules 0	1	2	3	4	5	6	7	8	9
1	0 & 6	1	0	0	1	1	1	1	1	0	0	0
		2	0	0	1	1	1	1	1	0	0	0
2	1 & 7	1	0	0	1	1	0	1	0	0	0	0
		2	0	0	0	0	0	1	1	0	0	0
3	2 & 8	1	0	0	1	1	0	1	0	0	0	0
		2	0	0	0	0	0	1	0	0	0	0
4	3 & 9	1	0	0	1	1	0	1	0	0	0	0
		3	0	0	1	1	0	1	1	1	1	1
5	4 & 10	1	0	0	1	0	0	0	0	0	0	0
		3	0	0	0	0	0	0	1	1	0	0

Table 4: Matching multiple packets (1-3) two bits at a time.

is rule 5, so the remainder of that rule is all that needs to be compared. This also allows the lower result vector to be used for a new packet (packet 3). We can see in step 4 the result of this first comparison on the new packet. The upper section of step 4 shows that rules 2, 3 and 5 still match packet 1, for which we have tested bits 0-3 and 6-9. The lower section of step 4 shows that rules 2, 3, 5, 6, 7 and 9 match packet 3, for which we have tested only bits 3 and 9. Step 5 provides a match for packet 1 and a new packet could start matching in the next step. Packet 3 does not have a match yet, but inspection shows that a single match will be found in the next step.

It should be noted in this technique that new packets can start comparisons at any location, depending on how many bits it took to match the previous packet. The fields being compared will continue, wrapping around to the start of the packet if necessary until all rules mismatch, a single match is found, or all field have been compared. The simultaneous comparison units allow high throughput as they are always fully utilized. The translation of these methods to algorithms and physical hardware is found in the following sections.

Algorithm 1 provides a pseudocode representation of the system described in the previous examples. Each iteration of the main loop that starts on line 4 represents one step of the algorithm as shown in Tables 3 and 4. The array, $match$, stores the results for each stage of the system, as per Table 4. C is the number of simultaneous bits checked during a step, M is the number of steps required to complete matching of an entire packet (i.e., the total number of decision bits divided by C). N is the number of rules and S is the number of simultaneous packets to be processed in parallel, which will henceforth be called the number of stages. The data variables used in the algorithm and their purpose are listed below. Array size is shown within square parentheses as necessary.

- $FieldAddr$: Counts from 0 to M-1 and represents the location of the least significant bit being compared this step (the lower number in column 2 of

Table 4.

- $idle[S]$: Indicates whether the referenced stage is idle. If so, it is ready for a new packet.
- $RuleFields[N * C]$: Stores one row of the rule table which will be further described in Section 3.2. This reads all the required fields from all the rules for the current step.
- $packet[S][C*M]$: Stores the packet decision bits during matching of a stage.
- $match[S][N]$: Indicates which rules continue to match the packet, as shown

Algorithm 1 *LOP* Algorithm

```
 1: $FieldAddr = 0$
 2: for $i = 0$ to $S - 1$ do
 3:     $idle[i] = true$
 4: loop
    // Read field FieldAddr of all segments of all rules
 5:     $RuleFields = ReadMem(FieldAddr)$
 6:     for $i = 0$ to $S - 1$ do
 7:         if $idle[i]$ then // If stage is idle, get a new packet
 8:             $packet[i] = getNewPacket()$
 9:             $match[i] =$ “111111”
10:             $startField[i] = FieldAddr$
11:             $idle[i] = false$
            // Compare field FieldAddr of the rule and mask
12:         for $j = 0$ to $N - 1$ do
13:             for $k = 0$ to $C - 1$ do
14:                 if $packet[i][j * C + k] \neq RuleFields[j * C + k]$ then
15:                     $match[i][j] =$‘0’
        // Increment FieldAddr
16:     if $FieldAddr = M - 1$ then
17:         $FieldAddr = 0$
18:     else
19:         $FieldAddr = FieldAddr + 1$
        // Check for end conditions
20:     for $i = 0$ to $S - 1$ do
21:         if $match[i] = 0$ then
22:             report $NoMatch$
23:             $idle[i] = true$
24:         else if $match[i]$ has only one ‘1’ then
25:             report $Match$
26:             $idle[i] = true$
27:         else if $startField[i] = FieldAddr$ then
28:             report $MultiMatch$
29:             $idle[i] = true$
```

in Table 4.

- $startField$: Remembers which bit was the first bit compared in the packet. Used to break algorithm if multiple rules match the packet.

Lines 1-3 of Algorithm 1 perform initialization of the system indicating that all stages are idle and setting the start location to field 0. The loop that starts on line 4 performs one step per iteration. Only one read of the rule memory is needed per step, as shown in line 5. Each stage performs identical operation, all using this single read as input. The loop from lines 6-15 handles calculation of the $match$ array for each stage. If a stage is initially idle, a new packet is fetched and the variables reset on lines 8-11. Then, the loops from lines 12-15 perform all the necessary matching for the current step. The fields read from the memory are compared against the packet bits and if there is a mismatch, the match bit for that rule (j) in the appropriate stage (i) is cleared. Hence, those rules where all the appropriate bits match remain as '1' in the $match$ array. Lines 16-19 increment the FieldAddr value to its next number, wrapping back to the start when necessary. The loop from line 20-29 checks the end conditions that could have occurred and reports these so that action can be taken. If all bits within the match vector for a stage are '0', then all rules mismatch packet. The system reports the case of $NoMatch$ and prepares for a new packet during the next step. If only one rule matches, the rule number is reported with a $Match$ condition and the system will check the remainder of the rule to ensure it is a complete match. Finally, if neither previous case holds, and the updated FieldAddr matches the original $startBit$, then we have already compared every field in the rule table and still have 2 or more matches, indicating multiple rule matches. The matching rules are reported and a new packet can start on the next step.

3.2 Architecture Template

The *LOP* packet classification scheme is divided into two architectures, *LOP_A* and *LOP_B*. The difference between these two architectures is the method used to detect the end case of a single rule match. SRAM is used to hold the pre-defined rules, and other peripheral logic units are used to match incoming packets to the rules. In both architectures, S number of packets are matched in parallel using S stages of *Feedback XNOR units* (FXUs) and *One Hot Subtractors* (OHSs).

Figure 2 represents the architecture of *LOP_A* and with a single stage T shown. Figure 3 expands upon the architecture to also show how multiple stages are implemented. The SRAM is structured in a unique way. Typically, entries within the memory would be rules and these would be read separately. In *LOP*, each entry of SRAM stores a subset of the fields for all rules. These fields are arranged so that only one entry within the SRAM is read on any given step of the algorithm. An example of the bit mapping for rule 1 is shown in Figure 2. In this case, the rule has been separated into three parts, or *segments*. Thus, with the memory size being

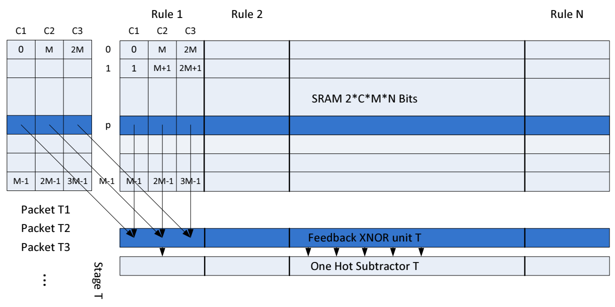


Figure 2: Stage T of LOP_A

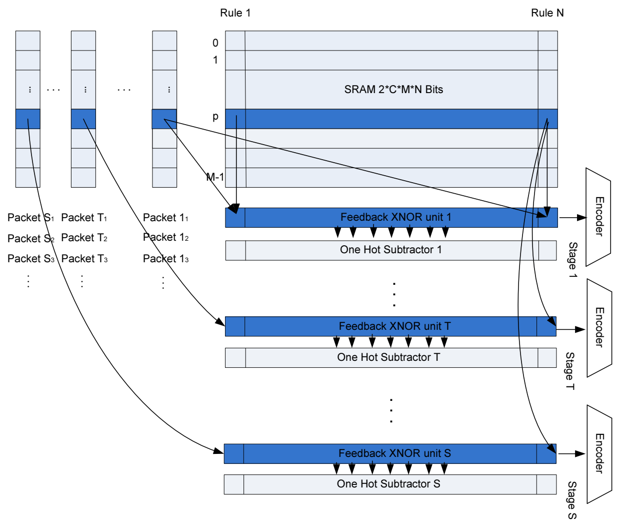


Figure 3: LOP_A

M entries, the fields located in entry 0 of the memory are fields 0, M and $2M$ of each rule. Similarly, entry p contains fields p, $p + M$ and $p + 2M$ of every rule. With memory arranged in this fashion, three fields of each rule will be compared each cycle. C is used to refer to the number of segments, which each rule is split into (in this case, C=3). N is the number of rules and M is the number of entries in the memory. Each comparison field in the SRAM has one of three possible values: '0', '1' or 'x'. Thus, each field can be represented by 2 SRAM bits and the total size of the SRAM will be $2.C.M.N$ bits arranged in a $M \times 2.C.N$ manner.

In every clock cycle, one read of a single entry of SRAM is performed, and each stage compares the fields against its appropriate packet bits to determine which rules continue to match. This comparison is performed by the boxes labelled Feedback XNOR units shown in Figures (2 and 3). In hardware, these are made from logical circuits as shown in Figure 4 and there is one of these circuits for each rule

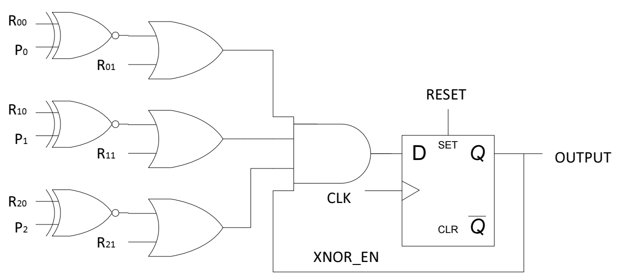


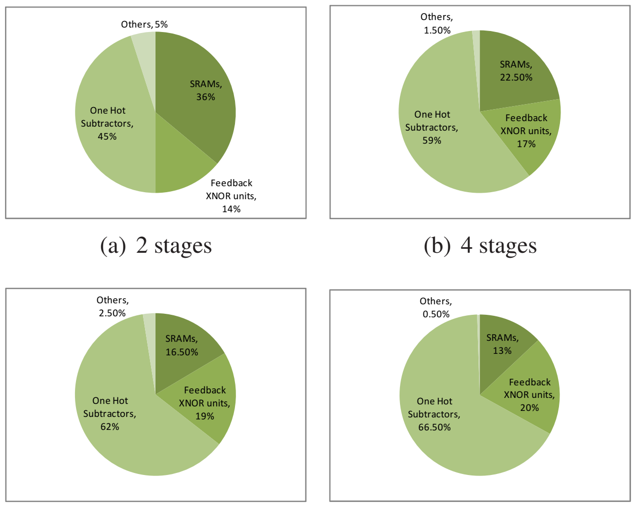
Figure 4: *Single Feedback XNOR Circuit*

at each stage.

Figure 4 specifically shows the implementation of a single Feedback XNOR circuit when $C = 3$. Thus, 3 fields are compared each cycle. Field i will compare bit i of packet P which is P_i with the two corresponding field bits, R_{i0} and R_{i1}. If R_{i1} is '1', the rule field represents the 'don't care' condition and will always match. Otherwise, R_{i0} provides the value that P_i must match. Each of these three comparisons is performed via an OR and XNOR gate combination as shown and a logical AND operation checks all three packet bits matched the rule fields and masks with the previous value of the rule to ensure all previous bits up until the current step have also matched. When the stage is made idle, the RESET input of the flip-flop is triggered to prepare it for a new packet. Increasing C is possible by increasing the number of XNOR+OR comparators and adding extra inputs to the AND gate.

The result of the *Feedback XNOR units* provides an implementation of the $match$ array used in Algorithm 1. The remainder of the algorithm is to determine if there is only one matching rule remaining and this is where the difference between *LOP_A* and *LOP_B* lies. In *LOP_A*, each stage's Feedback XNOR units are input into an *OHS* (i.e., *One Hot Subtractor T* for stage T in Figure 2) which detects whether the result for that stage is either zero (hence no rules matched) or there is only one '1' (only one rule matches). The OHS performs this task by calculating $match[T] AND (match[T] - 1)$ which results in a vector of all zeros only in the cases where there is one or zero '1's in $match[T]$. The size of the subtractor used to implement the *OHS* increases with the number of rules.

In order to match S number of packets in parallel, more stages must be added as shown in Figure 3. There are S stages of *FXU* and *OHS* to handle S packets respectively. Despite the addition of more packets and comparison units, there is still only the need for one read from SRAM in any cycle. All stages perform comparisons with their respective packet bits and store their own result. The worst case here is if the system needs to compare all $C * M$ deciding bits from the packet. This would cause power consumption of the system to be high due to the large amount of additional circuitry required. However, from the simulation results (see Section 5), we can see that the average number of stages needed is typically



(a) 2 stages (b) 4 stages

(c) 6 stages (d) 8 stages

Figure 5: Power Consumption Proportion of *LOP_A* in different stages

very low.

To handle the results of multiple stages, encoders (or priority encoders) are added to handle packet classification decisions provided by the Feedback XNOR and OHS units. Figure 3 shows each stage having its own encoder. However, it is also possible to allow multiple stages to share an encoder up to the point of using one encoder for all stages. Multiple stages are serviced by priority, as only one stage can be serviced at a time. This results in a loss of lookup rate (and throughput) when several stages using the same encoder complete simultaneously, as the result of the *Feedback XNOR units* cannot be utilized while waiting to be serviced by the encoder (the impact is shown in section 5).

An encoder will check all its managed stages to see if there is a result. If there is no match (result vector is all zero) or there are multiple matches (every field in the RAM has been compared), the result can be passed to the controlling system for appropriate handling. If there is a single match, the encoder reads an alternate SRAM storing the rules in the typical orientation (addressed by the rule number). The matched rule is read from this alternate SRAM (not shown) and compared to the packet to ensure the bits that have not yet been checked match the remainder of the rule. The result of this final check determines if the packet fully matches the rule and can be passed to the controlling system for processing.

In *LOP_A*, the *OHS* is a critical path in the architecture. Figure 5 shows that the *OHSs* contribute 45%, 59%, 62% and 66% of the total power consumption with 2, 4, 6 and 8 stages respectively. An increase in the number of rules causes the size of the *OHSs* to increase, causing higher power consumption.

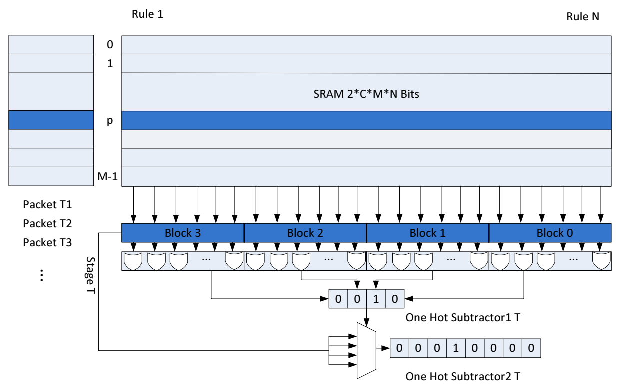


Figure 6: Stage T of *LOP_B*

To reduce power consumption and latency, a modified *OHS* architecture, *LOP_B*, is proposed that splits the *OHS* into two parts, *One Hot Subtractor1* (*OHS1*) and *One Hot Subtractor2* (*OHS2*). Figure 6 shows the structure of a single stage of *LOP_B*. *LOP_B* differs from LOP_A by using two smaller OHSs compared to one large one. The result of the *Feedback XNOR units* is divided into B blocks of equal size. Figure 6 shows the split when $B = 4$ and $N = 32$. A logical OR is applied over all bits in each block. The result of the OR operations form a new vector of size B which is tested by an *OHS* (labeled One Hot Subtractor1 T). This One-Hot comparison determines if there is only one block that contains '1's. If so, a multiplexor is used to select the bits of the only block which contains '1's (Block 1 is selected in Figure 6). This small block of size $\frac{N}{B}$ is input into a second OHS (One Hot Subtractor 2) and its result is treated like the result of the single *OHS* used in *LOP_A*.

Note that to achieve optimal timing for *LOP_B*, the size of *OHS1* and *OHS2* should be close or equal (i.e., $B = \sqrt{N}$). For example, for a system with 1024 rules, N will be 1024 and B should be set to 32 so that both *OHSs* will be the same size. This alteration does not affect the lookup rate of *LOP_B* from that of *LOP_A* (for the same M, N and S).

Note that *LOP_A* can be interpreted as *LOP_B* with one block. For consistency, in Section 5, *LOP_A* is named *LOP_1B*, *LOP_B* with 4 blocks is named *LOP_4B* and *LOP_B* with 8 blocks is named *LOP_8B*.

4 Packet Classification System

This section shows how the *LOP* architecture can be applied to two common types of packet matching methods. The first is *High Priority based Packet Matching (HPPM)*, and the second, *Multiple Packet Matching (MPM)*. *HPPM* uses several sets of rules at different priority level; the rule sets are searched in order of prior-

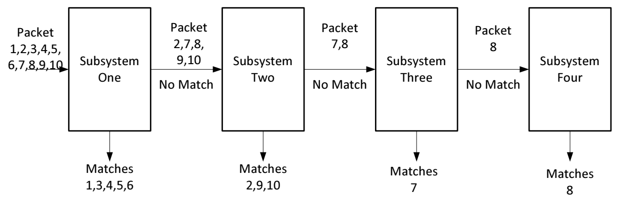


Figure 7: Highest Priority based Packet Classification

ity until a match is found. *MPM* uses separate rulesets in parallel where a packet may match several rules, but only one in each set. *HPPM* is useful for forwarding of certain packets to implement IP route lookup and to enable Internet quality of service. To enable the matching of the highest priority rule to the incoming packet, rules are stored according to their priorities (the entries in the forwarding table or the classifier database are stored in decreasing order of prefix lengths or priority of the rules). *MPM* is effective for enforcing security restrictions, monitoring traffic, network intrusion detection systems and other applications which need further inspections.

4.1 High Priority Packet Matching

A sample of the system used for *HPPM* is shown in Figure 7. The rules are divided into four blocks, assuming that there are four different priorities for the entire ruleset. Rules in *Subsystem One* have the highest priority and rules in *Subsystem Two* have the second highest priority and so on. An incoming packet, $Packet$ 1, is received into *Subsystem One* and is compared to all rules with the highest priority. If *Subsystem One* reports a $match$, with $Packet$ 1, then the search is halted, otherwise $Packet$ 1 continues to *Subsystem Two* and is compared to all rules within the second highest priority block and so on. From Figure 7, it can be seen that the whole system is implemented in a pipelined fashion, and only $Packet$ 8 go through the entire ruleset. Since only a part of the ruleset for each packet is matched, this highest priority based packet matching system can save a considerable amount of energy when compared to a non-partitioned system.

Both TCAM-based and the *LOP* SRAM-based packet classification architecture can be used in *HPPM*. They both share similar controlling logic for each pipelined stage. Since a TCAM-based architecture can match a single packet every clock cycle, it is possible to pipeline this system using sub-TCAMs. Since the time of matching of blocks will be unequal for the *LOP* architecture, it is necessary to add queues between blocks. For example, one packet A might need 8 clock cycles to be processed in *Subsystem Two* while another incoming packet B might need only 5 clock cycles in *Subsystem One*. Thus, packet B has to be stored somewhere temporarily until *Subsystem Two* finishes processing packet A. Queues between stages have to be added to map the *LOP* architecture to *HPPM*. Figure 8 illustrates

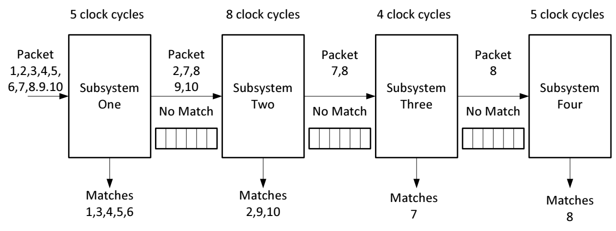


Figure 8: Modified Highest Priority based Packet Classification

the modified system with queues which prevents the dropping of packets. The sizes of queues are decided by the critical pipeline stage which takes the longest processing time, the position of the critical stage, and the ratio of the number of clock cycles of other stages to number of clock cycles of critical stage.

We configure the queues to be the same size between each of the block to simplify calculations. The size of the queue can be calculated using two parameters: maximum number of clock cycle (*Max*)(in the critical stage) and minimum number of clock cycle (*Min*) in the system.

$$\textit{Total size of queues} = (K-1) * S * \frac{\textit{Max}}{\textit{Min}} * \textit{packet size} \tag{1}$$

K is the number of subsystems and S is the number of stages (number of parallel streams of packets). In the worst case, when the range is between 1 cycle and 32 cycles (96 fields divided into 3 segments), then the total queue size will be 15KByte (if K is 4 and S is 4) [3]. Queues are typically implemented using Flip-Flops. The 15kB queues consume $19.23mW$ of power which was tested using Synopsys provided $PrimePower$ [6] in 0.18μm technology. A typical TCAM-based packet classification architecture (18Mbit) consumes up to 15 Watts of power when all the entries are searched [32]. Adopting the TCAM-based architecture into the *HPPM* system, the minimum power consumption will be 3.57 Watt (if K is 4 and every incoming packet matches within *Subsystem One*.) In this case, the power of the queues will maximally take 0.54% of the entire power consumption of the system.

4.2 Multiple Packet Matching

As stated previously, some emerging network applications require a packet classification architecture which supports multiple matches. In order to enable such multiple matches, researchers have proposed several different schemes and architectures [15, 19, 31]. Figure 9 shows an example system architecture which is used

[3]Note that the number of priorities of rules vary from 13 to 55 as shown in [25]. However, we don't need to partition rules into this many subsystems. The rules in subsystems need to be stored in decreasing order of priority.

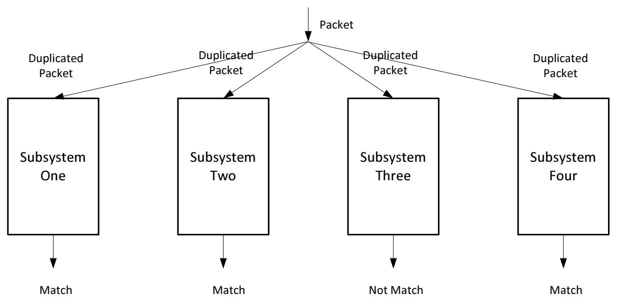


Figure 9: Multiple-Matches based Packet Classification

for the multiple matches based packet classification. This parallel system (similar to [19]) needs to partition the entire ruleset into different blocks. The rules are partitioned offline, such that no rules overlap (i.e., a packet will match at most only one rule within a single block). For example, the ruleset is partitioned into four sub-rulesets in Figure 9. One incoming packet is copied four times, and a copy is sent to each of these subsystems. The matched result shows, *Subsystem One*, *Subsystem Two* and *Subsystem Four* in Figure 9, three of the subsystems contain rule that matches the incoming packet, resulting in three matches to the single packet. Both a TCAM-based packet classification and our *LOP* SRAM-based packet classification architecture can be mapped to this system. A TCAM-based architecture finds all matches within one single cycle (since the lookup rate is always one), while the *LOP* architecture report matches over a number of clock cycles, since a match in each subset will be found after a different number of clock cycles. To reduce the impact on the throughput, it is possible to use queues in the *LOP* architecture in a similar manner to the *HPPM* system. Figure 10 shows the modified system with queues. The size of queues are determined by the maximum number of clock cycles (Max is 32 in our setup).

$$Total\ size\ of\ queues = 2 * K * S * Max * packet\ size \tag{2}$$

K is the number of subsystems and S is the number of stages (number of parallel streams of packets). In the worst case, a 40kB queue can satisfy the requirements of the system (if K = 4 and S = 4).

5 Experimental Setup and Results

The *LOP* architecture with various configurations (2, 4, 6 and 8 stages; 3, 4, and 6 segments; 1-8 priority encoders; and block size of 1, 4 and 8) have been implemented using VHDL and synthesized using Synopsys Design Compiler [6] with the Tower 0.18μm process library. We examined two main aspects: one, the comparison of power consumption between TCAM and *LOP* designs described in the

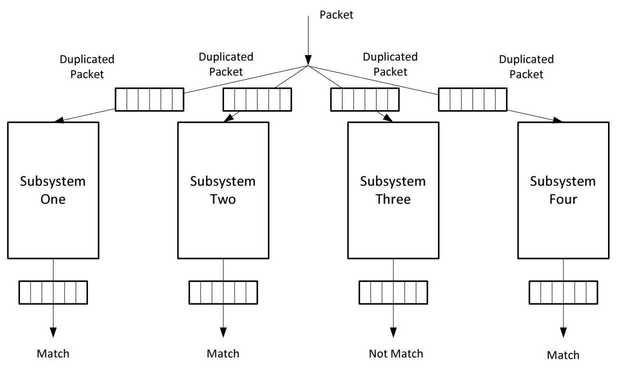


Figure 10: Modified Multiple-Matches based Packet Classification

paper; and two, the lookup rate and throughput of the *LOP* scheme when used with different benchmarks. Nine rulesets and corresponding packet traces were generated using the ClassBench tool [3] to create benchmarks. Note that to overcome the shortage of publicly available rulesets, Taylor et al. [29] presented a suite of tools called ClassBench which generates benchmarks for packet classification algorithms. ClassBench includes a Filter Set generator (which produces rulesets that exhibit characteristics of real rulesets) and a Trace generator (which generates a series of packet headers for testing purposes). $PrimePower$, part of the Synopsys tool suite, was used to estimate the power consumption of the circuits used in this paper. ModelSim SE 6.0 [4] was used to simulate the design under the Linux environment.

5.1 Field Selection and Segment Selection

In our experiments, each rule has 96 fields in four domains: IP destination address (IPdst = 32 fields), IP source address(IPsrc = 32 fields), TCP destination port (TCPdst = 16 fields) and TCP source port (TCPsrc = 16 fields). The number of segments (C1 to Cn) was varied to compare the affect on latency and power consumption. There are various ways to split the rules, e.g., in Figure 11 (a), one rule can be divided into three segments where segment one contains IPdst, segment two contains IPsrc and segment three contains TCPdst and TCPsrc. The number of segments and fields allocated to each segment affect the power consumption and the lookup rate of the system. The following experiments compare several methods to determine the best method for splitting rule fields into segments.

5.1.1 Field Selection

In this section, we examine whether the selected fields should be chosen from different domains, or it is better to choose fields from same domain. For example,

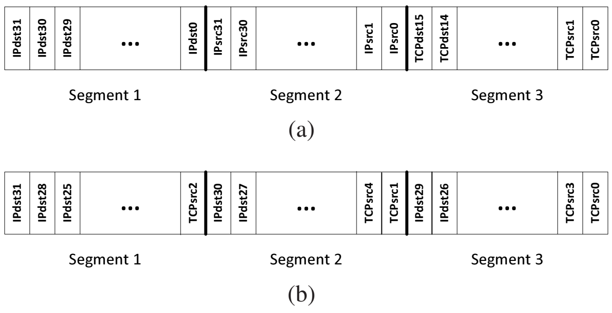


Figure 11: Two ways for allocation of fields to segments

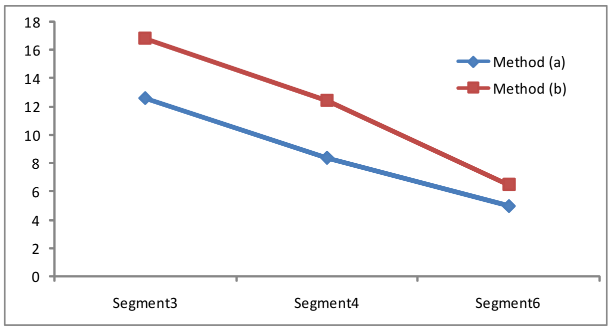


Figure 12: Required Number of Clock Cycles per Match

if the IPdst, IPsrc and TCP ports are separated into three segments, the selected fields (from each segment) will be from different domains.

Figure 11 shows one example of dividing rule into three segments, with each segment containing 32 fields. Method (a) maps it sequentially with the first 32 fields (IPdst in this case) into segment one, second 32 fields into segment two and last 32 fields into segment three. Method (b) maps in a non-sequential manner. Every clock cycle, three fields (one from each segment) need to be processed. Thus, in Method (a), fields IPdst0, IPsrc0 and TCPsrc0 are matched against the bits IPdst0, IPsrc0 and TCPsrc0 of an incoming packet in the first clock cycle; fields IPdst1, IPsrc1 and TCPsrc1 are matched against bits IPdst1, IPsrc1 and TCPsrc1 of the packet in the next clock cycle and so on. In Method (b), fields TCPsrc0, TCPsrc1, TCPsrc2 are matched against the bits TCPsrc0, TCPsrc1, TCPsrc2 of an incoming packet in the first clock cycle; fields TCPsrc3, TCPsrc4, TCPsrc5 are matched against bits TCPsrc3, TCPsrc4, TCPsrc5 and so on.

Figure 12 shows the average required number of clock cycles per match varying number of segments between 3, 4 and 6. Method (a) is shown to require fewer clock cycles to finish matching in all cases. This is likely because selecting several fields from different domains provides a higher chance of detecting non-matched rules in the early steps. Thus, mapping method (a) is used in the remainder of the experiments. It is a good, but non-optimal field mapping, which is outside the scope of this paper.

5.1.2 Segment Selection

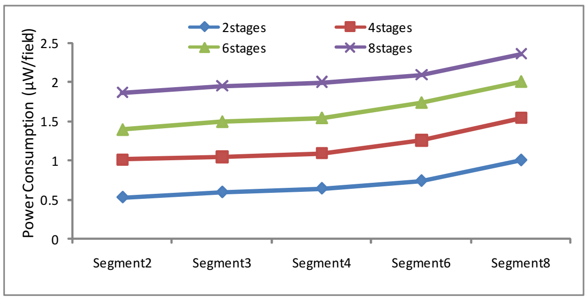
The number of segments influences power consumption and lookup rate of the *LOP* architecture. In this experiment, the number of segments is varied between 2, 3, 4, 6 and 8. Figure 13 illustrates the variation of power and lookup rate for different segments for the *LOP_8B* architecture, configured with 2 to 8 stages with fully shared encoders. The results show that lookup rate and power consumption increase with increasing number of segments. Figure 13(a) shows the power consumption; Figure 13(b) shows the lookup rate; and Figure 13(c) shows the ratio of power to lookup rate. Since the lookup rate of 1.26 with segment size of six (at 200MHz clock cycle) reaches throughput of 250 million searches per second (Msps) (250Msps is considered fast for a low power system [26]), we do not consider the system with segment size of eight (since the power consumption is higher). In further sections, we examine segment sizes three, four and six more closely.

5.2 Power Consumption Comparison

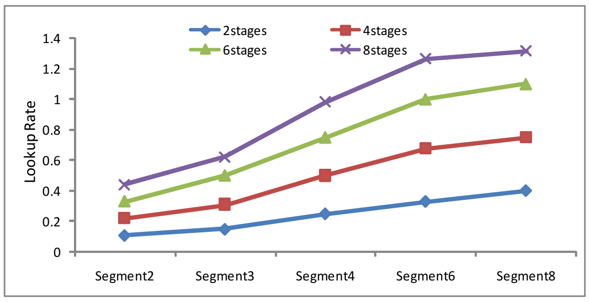
We calculated the power consumption of the architecture while varying various parameters. The number of segments C varied between 3, 4 and 6 (3C, 4C and 6C) and various number of stages S from 2 to 8 (2S, 4S, 6S and 8S) and block sizes in *LOP_1B*, *LOP_4B* and *LOP_8B* [4]. Table 5 shows the power consumption of *LOP_1B*, *LOP_4B* and *LOP_8B* respectively. The estimated power of the implemented hardware includes SRAMs (both the *LOP* SRAM shown in Figure 2, 3 and 6 and the additional SRAMs used by the encoders to complete the rule checking of partially matched rules), *Feedback XNOR units*, *One Hot Substrctors*, priority encoders and glue logic. To easily compare with TCAM-based architecture, power consumption is shown in Table 5 in μW/field. The total power consumption can be calculated by μW/field * (number of rules) * (number of fields per rule).

In Table 5, the first row shows the architectures *LOP_1B*, *LOP_4B* and *LOP_8B*. In the first major column, the sizes of segments and the number of encoders are shown in two sub-columns. The second major column shows the power consumption per field in *LOP_1B* with two stages (1B_2S), four stages (1B_4S), six stages (1B_6S) and eight stages (1B_8S). Similarly, the third and fourth major columns depict the power consumption in *LOP_4B* and *LOP_8B*. As discussed before, it can be seen that sharing encoders in these architectures can save power by reducing

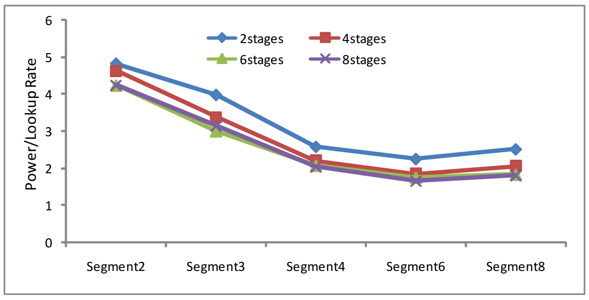
[4]Note that, due to the enormous amounts of time needed for power simulations, we did not consider rule sets N larger than 1024. However, we can estimate the power consumption of the proposed architecture for larger rulesets using these smaller models.



(a) Power Consumption



(b) Lookup Rate



(c) Power/Lookup Rate

Figure 13: The Selection of Number of Segments

the number of extra encoders and SRAM blocks (used by encoders). The power consumption of our architecture (except SRAMs) is estimated using Synopsys provided PrimePower. The power consumption of the SRAM array was obtained (for CMOS 0.18μm technology, at 200MHz) from [9], which was 69nW/bit (low-power SRAM only consumes 13.4nW/bit [1], at 250MHz, in CMOS 0.18μm). The *LOP_8B* always provides the lowest power consumption compared with *LOP_1B* and *LOP_4B*. Hence, block size of eight is a more appropriate solution than size four and one for *LOP* architectures due to the reduced power consumption. Block

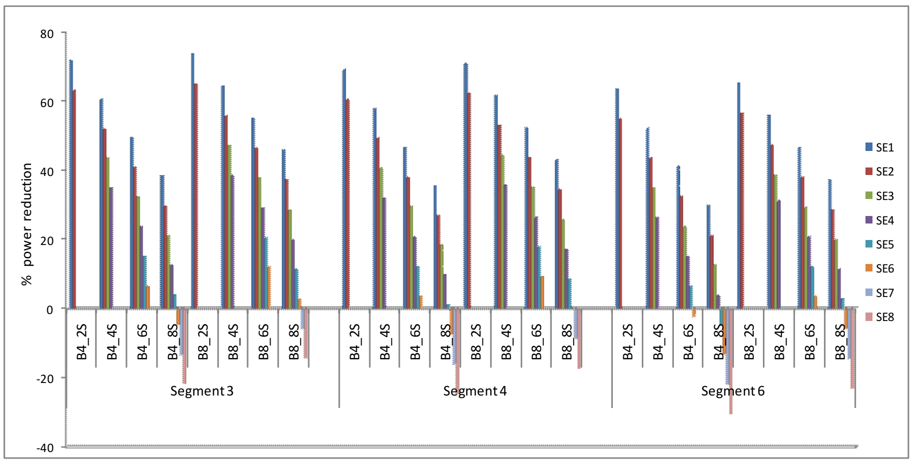


Figure 14: Power Reduction

sizes beyond eight were not considered, as the critical path increased to an extent which compromised our planned 200MHz clock cycle speed. Table 5 also shows the variation in power consumption when *LOP* shares different number of encoders and corresponding SRAMs.

Figure 14 shows the power reduction of *LOP_4B* and *LOP_8B* compared with a commercial TCAM, SibreCore's SCT2000 (a low power TCAM consumes 1.7μW/ field with a throughput of 66 million searches per second (66Msps) in 0.18μm CMOS technology [7]). The horizontal axis labels the *LOP_4B* and *LOP_8B* with 2 to 8 stages with three, four, and six segments. The vertical axis shows the power reduction of *LOP_4B* and *LOP_8B* compared to SibreCore's SCT2000. The *LOP_4B* and *LOP_8B* architectures with less than 8 stages and sharing less than 4 encoders consume less power than a SibreCore's SCT2000.

5.3 Performance

Nine different benchmarks (acl1, acl2, ...,ipc2) and their corresponding trace packets from ClassBench [29] are pre-processed to form the rulesets for *HPPM* and *MPM*. The benchmarks correspond to the three rule set formats generated by ClassBench for access control lists (acl), firewalls (fw) and IP chains (ipc). Each rule contains IPdst (IP destination address), IPsrc (IP source address), TCPdst (TCP destination port) and TCPsrc (TCP source port) in all of these benchmarks. TCP ranges in the rules are converted to several rules using don't care prefixes. Three different sizes of segments (C = 3, 4 and 6) are used in this experiment.

Architecture		LOP_1B (μW/field)				LOP_4B (μW/field)				LOP_8B (μW/field)			
Segments	Encoders	2S	4S	6S	8S	2S	4S	6S	8S	2S	4S	6S	8S
	1	0.809	1.224	1.676	2.131	0.481	0.671	0.859	1.048	0.449	0.607	0.765	0.922
	2	0.981	1.395	1.848	2.303	0.627	0.817	1.006	1.195	0.596	0.754	0.912	1.068
	3	-	1.567	2.019	2.474	-	0.963	1.152	1.341	-	0.899	1.058	1.215
3	4	-	1.739	2.192	2.646	-	1.109	1.299	1.487	-	1.046	1.205	1.361
	5	-	-	2.363	2.818	-	-	1.445	1.634	-	-	1.351	1.508
	6	-	-	2.535	2.989	-	-	1.591	1.780	-	-	1.497	1.654
	7	-	-	-	3.161	-	-	-	1.927	-	-	-	1.801
	8	-	-	-	3.333	-	-	-	2.071	-	-	-	1.945
	1	0.816	1.269	1.724	2.179	0.527	0.717	0.908	1.096	0.496	0.654	0.812	0.969
	2	0.987	1.442	1.895	2.351	0.674	0.863	1.055	1.242	0.642	0.800	0.958	1.116
	3	-	1.613	2.067	2.523	-	1.009	1.201	1.389	-	0.947	1.104	1.263
4	4	-	1.785	2.239	2.695	-	1.156	1.348	1.535	-	1.093	1.251	1.409
	5	-	-	2.410	2.867	-	-	1.494	1.682	-	-	1.397	1.556
	6	-	-	2.582	3.038	-	-	1.641	1.828	-	-	1.544	1.702
	7	-	-	-	3.210	-	-	-	1.975	-	-	-	1.848
	8	-	-	-	3.382	-	-	-	2.121	-	-	-	1.995
	1	0.912	1.367	1.821	2.277	0.623	0.814	1.003	1.194	0.592	0.751	0.909	1.068
	2	1.083	1.539	1.993	2.449	0.769	0.960	1.150	1.341	0.739	0.897	1.055	1.215
	3	-	1.710	2.164	2.621	-	1.106	1.297	1.487	-	1.043	1.202	1.361
6	4	-	1.882	2.336	2.792	-	1.253	1.443	1.634	-	1.189	1.348	1.507
	5	-	-	2.508	2.964	-	-	1.590	1.780	-	-	1.495	1.654
	6	-	-	2.679	3.136	-	-	1.736	1.926	-	-	1.641	1.800
	7	-	-	-	3.307	-	-	-	2.073	-	-	-	1.947
	8	-	-	-	3.479	-	-	-	2.219	-	-	-	2.093

Table 5: Power Consumption per Bit (μW/field)

5.3.1 Lookup Rate and Throughput

We executed the benchmarks in *LOP* with 8 blocks and a varying number of stages from 2 to 8 (*LOP* with 8 blocks provided the best power savings). We use Equations 3 and 4 to calculate lookup rate and throughput respectively. Table 6 shows the throughput of a single *Subsystem* for *HPPM* and *MPM* (due to limited space, only results of interest are shown). The first major column provides the configurations of the *LOP* architecture used. The four sub-columns show the advanced classification system (*HPPM* and *MPM*), the number of segments (3, 4 and 6), the number of stages (2, 4, 6 and 8) and the number of shared encoders respectively. In the second major column, the throughput of nine benchmarks is shown. The throughput varies according to different configurations of *LOP_8B*. Rows 16 and 52, which are highlighted in the Table 6, provide the average throughput 67.6Msps (lookup rate is 0.338 and *LOP* has been implemented under 200MHz). This throughput is equivalent to SibreCore's SCT2000 commercial TCAM (which has throughput of 66Msps). The configuration with 4 segments, 4 stages and 3 encoders, highlighted as rows 17 and 53, demonstrates a configuration that balances both power consumption and throughput with average average throughput of 97Msps (lookup rate is 0.485). A third example, highlighted in rows 35 and 71, provides high throughput of 220Msps (lookup rate is 1.1 searches in one clock cycle). In subsection 5.2, Table 5 show the *LOP_8B* consumes 0.654μW/field, 0.947μW/field and 1.361μW/field with the above corresponding throughput respectively. Comparing with SibreCore's SCT2000 (power consumption is 1.7μW/cell), *LOP_8B* consumes much less power (only 38.5% of the power of SibreCore's SCT2000) with the same throughput.

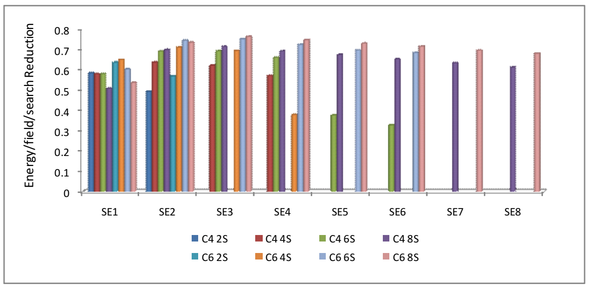
$$Lookup\ Rate = \frac{No.\ of\ searches}{No.\ of\ clock\ cycles} \tag{3}$$

$$Throughput = LookupRate * Clock\ Frequency \tag{4}$$

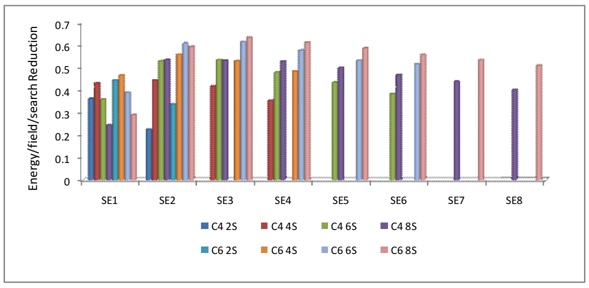
System	Segments	Stages	Encoders	acl1	acl2	acl3	acl4	fw1	fw2	fw3	ipc1	ipc2
HPPM	3	2	1	26	48	30	22	24	24	24	24	48
			2	28	50	34	24	28	28	24	24	50
		4	1	48	80	54	36	50	42	48	36	82
			3	58	96	76	48	52	48	48	56	102
			4	60	98	76	48	56	48	48	56	102
		6	1	68	84	76	44	66	48	64	82	48
			3	80	148	110	60	80	70	74	96	146
			6	92	152	114	72	80	76	74	96	146
		8	1	68	84	80	48	70	48	68	48	82
			3	90	186	138	88	96	78	96	118	188
			6	102	196	190	144	138	128	96	200	128
			8	104	204	168	102	100	104	96	142	200
	4	2	1	40	58	46	38	36	32	36	38	56
			2	50	58	56	42	36	32	36	42	58
		4	1	70	86	80	62	66	56	64	50	84
			3	98	118	98	78	84	76	68	88	116
			4	100	120	102	82	84	76	68	96	116
		6	1	72	86	84	68	74	68	72	84	56
			3	130	166	150	108	108	98	102	148	164
			6	138	176	158	112	108	112	104	156	180
		8	1	72	86	84	70	74	74	82	62	84
			3	150	210	170	136	136	124	168	186	204
			6	166	230	190	144	138	128	172	202	232
			8	170	236	202	148	140	144	180	212	240
	6	2	1	52	66	62	64	54	44	40	76	66
			2	52	66	66	70	56	44	40	76	70
		4	1	78	88	84	92	86	52	90	84	88
			3	122	134	130	134	110	88	138	138	140
			4	122	138	134	142	110	88	142	142	140

		6	1	78	88	84	96	86	62	90	88	84
			3	170	196	194	192	156	128	196	186	226
			6	182	202	200	210	162	136	212	222	200
		8	1	78	88	84	100	86	66	90	84	88
			3	192	230	222	226	200	162	240	242	226
			6	210	266	240	240	218	172	276	262	268
			8	220	272	242	248	222	182	284	262	274
MPM	3	2	1	38	34	26	36	34	52	36	28	22
			2	46	44	32	44	42	54	36	28	22
		4	1	60	54	48	64	58	72	58	42	32
			3	76	66	56	78	72	96	76	56	62
			4	80	86	60	82	72	96	88	62	62
		6	1	84	62	48	72	80	72	68	42	48
			3	110	94	80	98	114	138	110	96	68
			6	144	108	90	102	126	156	116	102	84
		8	1	84	82	52	72	84	82	68	48	48
			3	136	120	110	132	152	172	134	106	106
			6	170	130	118	144	162	188	156	138	116
			8	174	144	124	148	168	202	174	138	118
	4	2	1	60	42	36	66	58	72	52	32	32
			2	60	42	44	72	62	72	52	34	32
		4	1	86	82	56	76	58	82	76	44	46
			3	120	110	82	116	92	142	108	82	62
			4	122	110	84	116	92	148	108	88	68
		6	1	90	88	56	88	90	96	76	48	50
			3	186	144	116	174	126	192	150	116	108
			6	200	150	118	184	128	208	176	136	122
		8	1	90	90	56	88	90	96	78	50	52
			3	218	152	140	208	150	266	172	156	128
			6	260	178	160	242	172	272	192	176	148
			8	262	192	164	252	180	288	206	192	166
	6	2	1	66	46	76	84	58	76	76	68	56
			2	72	50	80	88	70	76	84	72	58
		4	1	92	88	98	94	62	84	84	84	64
			3	136	122	148	152	100	148	164	148	114
			4	144	142	150	156	100	152	164	148	134
		6	1	90	92	90	92	88	102	96	82	86
			3	196	150	196	196	152	196	236	200	178
			6	212	164	220	228	166	216	254	208	210
		8	1	90	92	100	102	88	102	96	86	84
			3	240	182	230	236	188	272	244	232	212
			6	276	198	270	248	202	284	266	252	224
			8	284	200	276	268	218	296	286	256	248

Table 6: Throughput of a single *Subsystem*



(a) *LOP_8B* vs. SibreCore's SCT2000 TCAM



(b) *LOP_8B* vs. Analog Bits's high-speed TCAM

Figure 15: Energy/filed/packet Reduction

5.3.2 Energy per field per search

We use Equation 5 to calculate energy per field per search for *LOP* architecture and energy per cell per search for TCAM-based architectures. Figure 15 shows the energy reduction per field per search of *LOP_8B* with three segment sizes (3C, 4C and 6C) and four different stages (2S, 4S, 6S and 8S) compared with two commercial TCAMs: SibreCore's SCT2000 and Analog Bits's high-speed TCAM with the same ruleset. In Figure 15 (a) and (b), the horizontal axis labels the number of encoders that are used (SE1 to SE8). The vertical axis of Figure 15 (a) shows energy per cell per search reductions compared to SibreCore's SCT2000 TCAM (in 0.18μm CMOS technology). The vertical axis of Figure 15 (b) shows energy per cell per search reductions compared to Analog Bits's high-speed TCAM (in 0.18μm CMOS technology). Figures 15 (a) and (b) show that *LOP_8B* provides the best energy saving when the hardware shares three encoders in almost all of the configurations. *LOP_8B* provides the lowest energy per field per search (6.2fJ/field/search with three segments, eight stages and three encoders). The best energy per field per search saving of *LOP_8B* is 75% compared to SibreCore's SCT2000 TCAM, whereas the maximum energy per field per search reduction is 65% compared to Analog Bits's high-speed TCAM. Note that, like SibreCore's and an the Analog Bits' technologies, *LOP* was implemented at 0.18μm. Noda et al. [18] proposed a cost efficient and high performance TCAM, implemented with 0.13μm CMOS technology, which consumes 1.48μW/cell with 143Msps throughput (10.3fJ/cell/search). Thus, the energy per field per search of *LOP_8B* can be up to 40% lower when compared to Noda's cost efficient and high performance TCAM, despite Noda's smaller implementation technology of 0.13μm. Lin et al. [16] proposed a low power 3D-TCAM which consumes 1.98fJ/cell/search (in 3D-TCAM) or 4.5fJ/cell/search (in 2D-TCAM) with 65nm CMOS technology. The lowest energy per field per search of *LOP_8B* is still higher than these 2D- and 3D-TCAMs. If *LOP* were implemented in 65nm technology and exploited the 3D architecture, we would expect to be significantly below the power consumption of Lin et al.'s TCAMs.

$$Energy\,per\,field\,per\,search = \frac{Power/field\,(cell)}{Throughput} \tag{5}$$

6 Conclusion and Future Work

In this paper, we have proposed a novel low-power *LOP* architecture to handle the packet classification problem. *LOP* packet classification architectures are implemented to compare against existing commercial TCAMs (both power and throughput). Results show that the power of *LOP_8B* provides the best power and energy saving. *LOP_8B* can achieve a lookup rate of more than 1.1 (throughput of 220Msps) with 20% power reduction over SibreCore's TCAM (at 66Msps) with a configuration of six segments, eight stages and three encoders. We also show the best energy saving per search of *LOP_8B* architecture can achieve 65% and 75% compared to two commercial TCAMs implemented in 0.18μm CMOS technology.

The *LOP* architecture is capable of trading low power consumption against high throughput by altering configurations. Thus, the *LOP* architecture is more flexible than TCAM architectures for packet classification applications. *LOP* can be configured in a more optimal way according to different network environments. In the future, we will research partial reconfiguration of *LOP* to achieve better power reduction and speedup.

References

[1] Tft-lcd application specific low power sram using charge-recycling technique. In *the sixth international symposium on quality of electronic design*, 2005.

[2] Analog bits technologies, 2008. Available at: http://www.analogbits.com/.

[3] Classbench tools, 2008. Available at: http://www.arl.wustl.edu/ det3/ClassBench/.

[4] Modelsim - a comprehensive simulation and debug environment for complex asic and fpga designs, 2008. Avalibale at: http://www.model.com/.

[5] Sibercore technologies, 2008. Available at: http://www.sibercore.com/.

[6] Synopsys, 2008. Available at: http://www.synopsys.com/home.aspx/.

[7] B. Agrawal and T. Sherwood. Ternary cam power and delay model: Extensions and uses. *IEEE Transactions on Very Large Scale Intergation*, pages 554–564, 2008.

[8] F. Basci and T. Kocak. Statistically partitioned, low power tcam. In *The 2nd Annual IEEE Northeast Workshop on Circuits and Systems*, 2004.

[9] T. Enomoto and Y. Higuchi. A low-leakage current power 180-nm cmos sram. In *the 2008 conference on Asia and South Pacific design automation*, 2008.

[10] P. Gupta and N. McKeown. Packet classification on multiple fields. In *SIGCOMM 99*, 1999.

[11] P. Gupta and N. Mckeown. Classifying packets with hierarchical intelligent cuttings. *Micro,IEEE*, 20:34–41, 2000.

[12] P. Gupta and N. Mckeown. Algorithms for packet classification. *IEEE Networks*, 15:24–32, 2001.

[13] S. Kaxiras and G. Keramidas. Ipstash: a power-efficient memory architeture for ip-lookup. In *the 36th International Symposium on Microarchitecture (MICRO-36 2003)*, 2003.

[14] S. Kaxiras and G. Keramidas. Ipstash: a set-associative memory approach for efficient ip-lookup. In *INFOCOM'05)*, 2005.

[15] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary. Algorithms for advanced packet classification with ternary cams. In *SIGCOMM*, 2005.

[16] M. Lin, J. Luo, and Y. Ma. A low-power monolithically stacked 3d-tcam. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2008.

[17] H. Liu. Efficient mapping of range classifier into ternary-cam. In *10th Symposium on High Performance Interconnects Hot Interconnects*, 2002.

[18] N. Mohan. *Low-Power High-Performance Ternary Content Addressable Memory Circuits*. PhD in electrical and computer engineering, Electrical and Computer Engineering, University of Waterloo, Canada, 2006.

[19] M. Nourani and M. Faezipour. A single-cycle multi-match packet classification engine using tcams. In *the 14th IEEE Symposium on High-Performance Interconnects (HOTI'06)*, 2006.

[20] K. Pagiamtzis and A. Sheikholeslami. Pipelined match-lines and hierarchical search-lines for low-power content-addressable memories. In *the IEEE 2003 Custom Integrated Circuits Conference*, 2003.

[21] K. Pagiamtzis and A. Sheikholeslami. A low-power content-addressable memory (cam) using pipelind hierarchical search scheme. *IEEE Journal of Solid-State Circuits*, pages 1512– 1519, 2004.

[22] K. Pagiamtzis and A. Sheikholeslami. Content-addressable memory (cam) circuits and architectures: A tutorial and survey. *IEEE Journal of Solid-State Ciruits*, 4, 2006.

[23] S. Singh, F. Baboescu, G. Varghese, and J. Wang. Packet classification using multidimensional cutting. In *ACM SIGCOMM'03*, 2003.

[24] H. Song and J. W. Lockwood. Efficient packet classification for network intrusion detection using fpga. In *FPGA*, 2005.

[25] H. Song and J. Turner. Fast filter updates for packet classification using tcam. In *GLOBECOM*, 2006.

[26] E. Spitznagel, D. Taylor, and J. Turner. Packet classification using extended tcams. In *11th IEEE International Conference on Network Protocols*, 2003.

[27] V. Srinivasan, S. Suri, and G. Varghese. Packet classification using tuple space search. In *SIGCOMM 99*, 1999.

[28] D. E. Taylor. Survey and taxonomy of packet classification techniques. *ACM Computing Surveys*, 37:238–375, 2005.

[29] D. E. Taylor and J. S. Turner. Classbench: A packet classification benchmark. In *IEEE 24th INFOCOM*, 2005.

[30] B.-D. Yang and L.-S. Kim. A low-power cam using pulsed nand-nor match-line and charge-recycling search-line driver. *IEEE Journal of Solid-State Circuits*, pages 1736– 1744, 2005.

[31] F. Yu, T. V. Lakshman, M. A. Motoyama, and R. H. Katz. Efficient multimatch packet classification for network security application. *IEEE Journal on Selected Areas in Communications*, 24:1805–1816, 2006.

[32] F. Zane, G. Narlikar, and A. Basu. Coolcams: Power-efficient tcams for forwarding engines. In *IEEE INFOCOM'03*, 2003.