2009 IEEE 34th Conference on Local Computer Networks (LCN 2009)
Zürich, Switzerland; 20-23 October 2009

LOP_RE: Range Encoding for Low Power Packet Classification

Xin He
School of Computer Science and Engineering
University of New South Wales and
National ICT Australia (NICTA)
Sydney, Australia
Email: xinhe@cse.unsw.edu.au

Jorgen Peddersen
School of Computer Science and Engineering
University of New South Wales and
National ICT Australia (NICTA)
Sydney, Australia
Email: jorgenp@cse.unsw.edu.au

Sri Parameswaran
School of Computer Science and Engineering
University of New South Wales and
National ICT Australia (NICTA)
Sydney, Australia
Email: sridevan@cse.unsw.edu.au

***Abstract*—State-of-the-art hardware based techniques achieve high performance and maximize efficiency of packet classification applications. The predominant example of these, Ternary Content Addressable Memory (TCAM) based packet classification systems can achieve much higher throughput than software-based techniques. However, they suffer from high power consumption due to the highly parallel architecture and lack high-throughput range encoding techniques. In this paper, we propose a novel SRAM-based packet classification architecture with packet-side search key range encoding units, significantly reducing energy consumption without reducing the throughput from that of TCAM and additionally allowing range matching at wire speed.**

***LOP_RE* is a flexible packet classification system which can be customized to the requirement of the application. Ten different benchmarks were tested, with results showing that *LOP_RE* architectures provide high lookup rates and throughput, and consume low power and energy. Compared with a TCAM-based packet classification system (without range encoding) implemented in 65nm CMOS technology, *LOP_RE* can save 65% energy consumption for the same rule set over these benchmarks.**

KEYWORDS

Packet Classification, Range Encoding, Packet-Side Key Encoding

I. INTRODUCTION

Packet classification is a topic of great importance in such area as network monitoring, security, routing, and quality of service. To classify packets accurately, one must match parts of the header of a packet with an existing table (containing rule information for routing etc.). Pure software-based packet classification approaches are not able to satisfy the speed requirements [1], [2] of modern network. Thus, hardware specific algorithms have been created to achieve high performance packet classification. Typically, these hardware algorithms are implemented using Ternary Content Addressable Memory (TCAM). TCAMs are fully associative memories that allow searching for a particular data value. Each TCAM cell stores a '1', '0' or a 'don't care'. Due to the parallel architecture, TCAMs provide high lookup rates that each packet can be searched within one clock cycle. However, TCAMs suffer from (1), high power consumption as TCAM-based classification systems simultaneously check all rules against an incoming packet. Thus, such systems exercise every single memory cell during every comparison; and (2), inefficiency in storing ranges which can also lead to high power consumption [3].

Typically, a range of TCP port numbers will require particular treatment. For example, assume that all packets with TCP source ports greater than 1023 are given a lower quality of service. Thus the ruleset will have to store a rule (or rules) which match the incoming packets with TCP source port greater than 1023. While it is possible to have 1000's of individual rules, it is best to have a few rules with ranges attached to them. Without range encoding, a TCP source range greater than 1023 (up to 65535), will requires at least six different prefixes [2]. Note that the part of the packet header which is matched is called a packet search key. A packet search key can be made up of a number of selectors (such as source TCP port number, destination IP address etc). Thus the packet search key from the incoming packet is matched against the standard prefixes in the ruleset.

Any range can be converted into multiple prefixes. In the worst case, for TCP destination and source ports, there are two domains, thus there can be a maximum of 900 entries, since each port is represented by 16 bits. Researchers have proposed rule-side range encoding algorithms to improve storage efficiency [4], [5], [6]. For the example in the previous paragraph, with six prefixes (assuming that both source and destination have the same range), there need to be 36 entries in the rule set. If there are only two distinct ranges in the entire rule set, we can encode the ranges [0,1023] equal to '0' and [1024,65535] equal to '1'. Thus one bit can represent each TCP port address range and only a single entry is required instead of 36. While such encoding is an efficient way of representing the rules, the conversion of a packet header into an encoded packet search key (to form a corresponding value to match against the encoded rule set), can be difficult at wire speed [3]. To achieve high performance, and reduce power consumption, it is imperative, that a feasible way is created to convert packet headers to encoded packet keys against encoded rule sets. Figure 1 provides an example of an entire packet

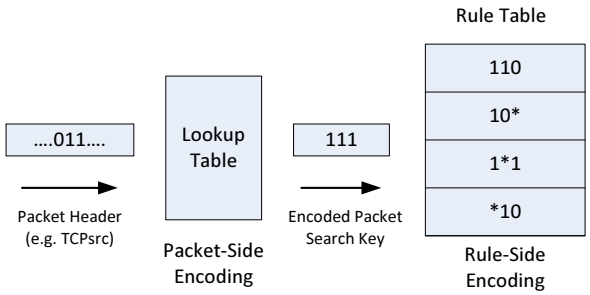

Fig. 1: Packet Classification System with One-Dimensional Range Encoding

classification system with range encoding. The *Rule Table* contains the encoded ranges and the *Lookup Table* translates the packet header information to encoded packet search key. For example, the TCP source value (16bits) from packet is converted to "111" after packet-side encoding lookup table and will match against the encoded rules (the third entry of the *Rule Table* is the reported match).

TCAM-based packet classification requires pre-processing units to form the search key for any incoming packet. This encoding of the packet search key is a non-trivial task and requires time to calculate. Although many rule-side encoding techniques have been proposed [7], [5], barely any of the authors have analyzed the packet-side encoding hardware to see whether packets can be encoded without slowing the clock speed of the system [3]. To the best of our knowledge, no TCAM-based packet classification with range encoding technique has been proposed that does not sacrifice throughput of the classification system.

In this paper for the very first time, we propose a multiple-stream SRAM-based architecture with packet-side search key encoding called *LOP-RE* which achieves high performance and consumes low power (and energy). By initially matching the first bit of a packet search key (non-encoded or encoded) to the first field[1] of all the rules, a number of non-matching rules are eliminated. Subsequently, the second bit of the packet search key is matched against the second field of all rules and further rules are eliminated. The process continues until only one rule that continues to match remains. In addition, two packet-side search key encoding circuits (to be able to handle TCP destination port range and TCP source port range) are integrated. Each packet-side search key encoding unit generates one encoded bit per clock cycle to match against the corresponding field of the 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, we demonstrate in this paper that it is possible to exceed the throughput of TCAM based systems while consuming less energy.

[1]The term *field* in this work is used to refer to the equivalent information of a single 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.

The remainder of the paper is organized as follows. A summary of related work is presented in Section II. In Section III, the *LOP_RE* approach and architecture are described. Section IV shows the experimental setup and results. Finally, the paper is concluded in Section V.

II. Related Work

Over the last few years, several algorithms were proposed for packet classification in both software and hardware [2]. Some of the software-based packet classification methods include linear searching [1], grid-of-tries [2], HiCuts [8], HyperCuts [9], tuple space search [10], and recursive flow classification [11]. 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 deployed in high performance network devices, such as routers, for packet classification [7], [12], [13], [14].

There are three main sets of methods to reduce power consumption without sacrificing the throughput of a hardware-based packet classification system. The first group of methods reduces TCAM power at the circuit level [15], [16], [17], [18], [19]; the second group of methods reduces power by partitioning rules at the system level (thus, only a selection of the rules are searched at one time, reducing power) [20], [14], [21]; and the third group of methods builds novel hardware architectures which replace CAM/TCAM with other type of memories [22], [23]. The first category of methods mainly focus on designing efficient gate-level architecture to reduce search-line and match-line power consumption of TCAM. The second category of methods, that of partitioning rules at the system-level, is orthogonal to designing a new type of TCAM-based packet classification system. Methods from the third category do not use TCAMs, but use other types of memories, and can utilize the rule partition schemes to further reduce power consumption.

Supporting rules with ranges without efficient encoding methods or specific range checking circuits can result in storage inefficiency. As stated earlier, in the worst case, 900 entries are needed in a TCAM-based packet classification system to convert a range on both source and destination TCP ports. Che et al. [3], [24] claimed that there are a great number of rules with ranges and the TCAM storage efficiency will be at most 16% in real-world rule databases.

There are mainly two different categories of methods to improve the storage efficiency for packet classification system. The first group of methods adds customized circuits for

range comparisons to TCAMs [25], [14]. The second group of methods utilizes rule-side encoding algorithms to reduce the expansion of storage entries and form an encoded search key from each packet using pre-processing units [26], [4], [3], [7], [27], [6], [24]. Spitznagel et al. [14] proposed an architecture, called the Extended TCAM, which adds extra comparison logic incorporated into the TCAM array for range checking. Since the authors handle port ranges by extending the TCAM functionality with embedded range comparison circuits directly, the extra logic increased the area of the TCAM array by 22%, which could lead to a high power consumption due to the parallel architecture of extended TCAM. However, the authors did not provide detailed analysis of power consumption. Hwang et al. [25] proposed a TCAM chip which contains range matching, each of which represent one range in the TCAM system. The range matching devices will increase the clock width of the TCAM chip, reducing throughput (the authors did not provide throughput or power consumption results). Other works focus on utilizing range encoding algorithms to improve storage efficiency. The bit mapping based range encoding algorithms were shown in [12], [5]. Lunteren et al. [6] proposed P^2C-based encoding schemes, which divide overlapping ranges into different layers. Thus, the system is able to encode rules using fewer bits. Chang et al. [4] proposed a Gray Code based encoding algorithm which consumes less TCAM entries than the bit mapping scheme and elementary interval-based range encoding algorithms [6]. However, the scheme proposed in [6] consumes more SRAM storage (which is used for translating input information to the encoded packet search key). Other encoding schemes such as [26], [27] work well in one-dimensional packet classification problems and are hard to be extended to multiple domains.

A key challenge for the range encoding based packet classification scheme is that the packet search keys need to be encoded at wire speed [3]. To achieve high throughput, (for example, 500 million searches per second (500Msps)), the searching key units need to be able to work within two nanoseconds. The authors in [28], [5], [6] assumed that multiple processors and multiple memories are available for packet-side search key encoding. However, not all the network processing units are able to implement the packet search key encoding [3] at wire speed. Other range encoding approaches, such as [7] did not consider the key searching problem, only focusing on the size of the rule set. Che et al. [3] used a TCAM coprocessor to assist range encoding. Using several TCAM lookups for search key encoding provided an immediate solution but reducing throughput by a factor of 2 [3]. To the best of our knowledge, no TCAM based approach has been shown that demonstrates the ability to process packets using range encoding at wire speed (with high throughput).

In this paper, we propose a low power SRAM-based packet classification architecture with range encoding (key encoding units) which can be used as a replacement of TCAM-based packet classification for modern routing systems. Two elementary interval-based range encoding schemes using Buddy Code [4] and Gray Code are mapped in this system and the corresponding packet-side search key encoding hardware is proposed which can be integrated into a novel packet classification architecture without sacrificing throughput. Unlike all previous TCAM approaches, the proposed technique implements packet classification including range encoding at wire speed.

A. 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 packet-side encoding at wire speed.
- A flexible packet matching scheme which can be configured to achieve superior power consumption and throughput according to the different network environment.
- Exploration of the design space of the SRAM-based architecture.

The limitations of this work can be outlined as follows:

- *LOP_RE* relies on heuristic pre-processing of rules and thus has limited support for runtime updates of the rules. Therefore, *LOP_RE* 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 [4], [29].

III. Methodology and Hardware Design

Consider a system with a packet classification module implemented using TCAM. This TCAM-based packet classification module can save storage and energy by using (rule-side) range encoding schemes. However, it is not feasible to implement packet-side search key encoding at wire speed without sacrificing throughput [3]. The *LOP* architecture can serve as a replacement of The TCAM-based packet classification system which has beeen explained in [30]. We propose an extension of *LOP* scheme, called *LOP_RE* which can support range encoding (with packet-side key encoding units) in addition to packet matching. Thus, a *LOP_RE*-based module is feasible to integrate range encoding in both rule-side and packet-side of packet classification system.

A. LOP_RE Scheme

In general, the storage efficiency and power consumption of the hardware-based packet classification system is affected by the rules with ranges. Using range encoding methods could reduce the storage expansion. Increased storage is the primary reason for high cost and increased power consumption. Chang et al. [4] provided a detailed comparison between several different (rule-side) range encoding methods (e.g., range-to-prefix, bitmapping, elementary interval-based schemes etc.) in terms

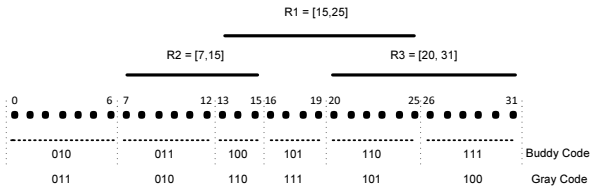

Fig. 2: An Example of Elementary Intervals Set

Input Packet Values (TCPdst = 17)		Output Encoded Values	
AddressRegions	*BeginValue*	Buddy Code	Gray Code
000	0	0-0-0	0-0-0
001	0	0-0-1	0-0-1
010	0	0-1-0	0-1-1
011	7	0-1-1	0-1-0
100	13	1-0-0	1-1-0
101	16	1-0-1	1-1-1
110	20	1-1-0	1-0-1
111	26	1-1-1	1-0-0

TABLE I: A sample packet-side encoding table (encoding lookup table) with 8 regions

of TCAM storage requirements in both the number of TCAM entries and TCAM lengths. In addition, the corresponding SRAM storage requirements for each range encoding approach for translating TCP port addresses of incoming packets to the encoded packet key value (which can be matched with the encoded rules) has been compared. The authors in [4] reported that the elementary interval-based scheme using Buddy Code (EIEBC) and the elementary interval-based scheme using Binary Reflected Gray Code (EIEGC) provide reasonably small amount of TCAM storage, which counted both the number of TCAM entries and the length of each entry. In addition, EIEBC and EIEGC consume much less SRAM space compared with other schemes such as Bitmapping. In this paper, we do not propose new rule-side range encoding methods. Instead, EIEBC and EIEGC are used to encode rule ranges in this paper. Here, we will focus on designing corresponding packet-side encoding hardware units (to form an encoded packet key), and show how they can be integrated with *LOP* at wire speed. The system developed here is called *LOP_RE*. The *LOP-RE* scheme is able to extract information from packets and form the required intermediate values (encoded packet search keys) without sacrificing throughput of the *LOP*-based packet classification system.

1) Elementary Interval based encoding schemes: A range can be represented by a set of *elementary intervals*. An *elementary interval* is the region between any two end-points of a range. Elementary intervals cannot be overlapped and each elementary interval shares immediately adjacent end-points. For example, assume there are three ranges $R1$, $R2$ and $R3$ which divide a range G [0,31] into six elementary intervals as shown in Figure 2. Each elementary interval can be represented using three bits (Buddy Code or Gray Code) for this case. Thus, $R1$ is converted into two prefixes (encoded ranges) 10* and 110, $R2$ into 011 and 100, and $R3$ into 11* using Buddy Code (asterisks represent don't care). Similarly, $R1$ can be converted into 1*1 and 110 (or, 11* and 101), $R2$ into *10, $R3$ into 10* using Gray Code. In this case, there are a total of five 3-bits encoded ranges, which need to be stored in rule table of the packet classification system using EIEBC scheme, and four 3-bits encoded ranges need to be stored using EIEGC scheme.

2) Corresponding Packet-Side Encoding Units: For illustrative purposes, we will show an example to demonstrate how one packet-side encoding unit works. Assume that the six elementary intervals described above are used. Totally, eight regions can be represented by three bits which include the six elementary intervals and another two invalid regions denoted by 000 and 001 in both Buddy Code and Gray Code. The first major column in Table I is the input of the elementary intervals (which will be compared with the incoming packet), which is divided into two sub-columns. The first sub-column shows the address of regions from 000 - 111, and the second sub-column stores the lower ends of the ranges (*BeginValue*) of each elementary interval (0 for invalid regions). The second major column is the output of the encoded packet search keys and it is divided into two sub-columns, which represent the Buddy Code based key addresses and Gray Code based key addresses respectively.

In this example, Successive Approximation via Binary Search is used, and in, every step (clock cycle), one address from the encoding table is predicted and the referenced *BeginValue* will be compared to the required information extracted from incoming packet (e.g., TCP destination port (TCPdst) and TCP source port address (TCPsrc)). The prediction direction of the binary search is determined by the output from the previous steps. Assume there is an incoming packet with the TCPdst value of 17. In the first step, this is compared with the midpoint of the table (address 100) which has the value of 13. The result of the comparison (in this case '1' as 17 $>$13) will be used for further comparisons. Thus, the second cycle will check against the *BeginValue* at address 110 which is 20 (if the first comparison had resulted in a '0', we would test against address 010 instead). Since 17 $<$ 20, the second bit of the encoded key address is formed ('0' in Buddy Code and '1' in Gray Code). In the third step, the *BeginValue* 16 under 101 will be compared with 17 and since 16 $>$ 13, the third bit of encoded key address is reported ('1' in Buddy Code and '0' in Gray Code). Therefore, the entire encoded key addresses are 101 (1-0-1) and 111 (1-1-1) for Buddy Code and Gray Code respectively. If the total number of regions is E (E = 8 in this case), then the encoding unit takes $\log E$ clock cycles (the complexity of binary search is $\log E$) to finish a formation of an encoded packet search key address. In a real network, TCP destination port address (TCPdst) and TCP source port address (TCPsrc) need to be encoded and

two packet-side encoding tables are required for each of them respectively. The *LOP*-based packet classification system is not affected by the number of elementary intervals because *LOP_RE* only requires one bit of the encoded key to be matched with corresponding encoded fields of rules per step (clock cycle). Therefore, *LOP_RE* can handle even large packet-side encoding tables without sacrificing throughput.

Algorithm 1 provides a pseudo code representation of the encoding procedure described in the previous examples. Let W be the width of the bits representing the addresses of the packet-side encoding table ($AddressRegion$ (Ar)) and S 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 brackets as necessary.

- $idle[S]$: Indicates whether the referenced stage is idle. If so, it is ready for a new packet.
- $Ar[S]$: The address of the current region to be examined in the referenced encoding lookup table in a certain stage.
- $PreviousAdd[S]$: Stores the previous reported bits of address of referenced encoding lookup table.
- $BeginValue[S][Ar]$: The beginning values of each elementary ranges, indicates by Ar[s], as shown in Table I.
- $KeyAddress\_BC[S](W)$: Reports the calculated value of the Buddy Coded key address for corresponding packet processed in referenced stage.
- $KeyAddress\_GC[S](W)$: Reports the calculated value of the Gray Coded key address for corresponding packet processed in referenced stage.

Lines 1-2 of Algorithm 1 perform initialization of the system indicating that all stages are idle. The loop that starts on line 3 performs one step per stage per iteration; the information required to be encoded will be extracted (e.g., TCPdst value). The loop from lines 4-29 handles calculation of the output of the encoded packet key $KeyAddress\_BC[I]$ and $KeyAddress\_GC[I]$ for each stage. If a stage is initially idle, a new packet is fetched and the variables are reset on lines 6-9. Then, the loops from lines 10-29 perform the packet search key encoding for the current packet value.

At each step, the packet is compared with the chosen element from the *BeginValue* table. If the value from the packet is higher or equal to the respective *BeginValue* lookup value from packet-side encoding table, then the system reports region address bit '1', otherwise reports '0'. Additionally, the corresponding bit of the Buddy Code based key address and Gray Code based key address are calculated based on this result and stored. The Buddy Code encoding bit will be equal to the result of the comparison. The Gray Code encoding requires the previous address to calculate the correct value for the current encoded bit. Line 11 provides an interrupt that if the referenced stage is idle, the encoding procedure is stopped. Note that, the stage status (idle or not) is determined by whether there are $matches$ reported in *LOP*-based system. When the matched rules are reported such as *NoMatch*, *Match* or *MultMatch*, then a new packet can start on the next step and the status of the corresponding stage is idle.

B. Architecture Template

In this section, we will describe the architecture of *LOP_RE*. *LOP* is a basic architecture without range encoding logic, whereas, *LOP_RE* extends *LOP* to support range encoding. LOP_RE adds packet-side search key encoding units to form the encoded packet search key at wire speed. SRAM is used to hold the pre-defined rules (shown in 3). Encoding Lookup Tables contain the *BeginValues* (e.g. for TCPdst and TCPsrc) and perform the packet-side key encoding processing. In addition, 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 Detectors* (OHDs).

Figure 3 represents the architecture of *LOP_RE* with a single stage T. Figure 4 expands upon the architecture to also show how multiple stages are implemented [2]. *LOP_RE* has one larger SRAM called *SRAM_A* which stores the non-encoded fields (such as IPdst and IPsrc). Whereas, the encoded fields (such as

[2]We set the *LOP*-based systems to compare six bits of the key at once (C = 6).

Algorithm 1 Key Encoding Algorithm

```
 1: for i = 0 to S − 1 do
 2:     idle[i] = true
 3: loop
    // read required information from packets
 4:     for i = 0 to S − 1 do
 5:         if idle[i] = true then // If stage is idle, get a new packet
 6:             packet[i] = getNewPacket();
 7:             Ar[i] = 2^(W−1)
 8:             idle[i] = false
 9:         PreviousAdd[i] = '0'
10:         for j = W − 1 to 0 do
11:             if idle[i] = true then
                    Break
12:             if packet[i] ≥ BeginValue[i][Ar] then
13:                 KeyAddress_BC[i](j) = '1'
14:                 if j>0 then
15:                     Ar[i] = Ar[i] + 2^(j−1)
16:                 if PreviousAdd[i] = '0' then
17:                     KeyAddress_GC[i](j) = '1'
18:                 else
19:                     KeyAddress_GC[i](j) = '0'
20:                 PreviousAdd[i] = '1'
21:             else
22:                 KeyAddress_BC[i](j) = '0'
23:                 if j>0 then
24:                     Ar[i] = Ar[i] − 2^(j−1)
25:                 if PreviousAdd[i] = '0' then
26:                     KeyAddress_GC[i](j) = '0'
27:                 else
28:                     KeyAddress_GC[i](j) = '1'
29:                 PreviousAdd[i] = '0'
```

encoded value of TCPdst and TCPsrc) are duplicated and stored into S number of smaller size of SRAMs called *SRAM_Bs* in *LOP_RE*. These fields are arranged so that only one entry within the SRAMs is read on any given step of the *LOP*-based algorithm (shown in Figure [30]). Each rule is split into C segments. N is the number of rules and M (or W) is the number of entries in the memory. Each comparison field in the SRAMs have 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 $2.C.M.N$. An example of the bit mapping for rule 1 is shown in Figure 3 for the *LOP_RE* architecture. Each rule has also been separated into six *segments*. However, four *segments* are stored in SRAM_A (C1-C4, and C=4) and the other two segments (encoded values) are stored in SRAM_B (C5-C6 and C=2).

The rule set is divided into two parts stored into *SRAM_A* and *SRAM_B* in *LOP_RE*. Each entry is read, one per clock cycle, from *SRAM_A* and compared with corresponding bits of the non-encoded packet key. If a new packet comes, *SRAM_A* does not need to be reset, the fields from the current entry continue to compare with the corresponding bits from the next packet key. Similarly, each entry is read, one per clock cycle from the *SRAM_B* of each stage and matched against the corresponding bits of the encoded packet key (generated from packet-side encoding lookup table). For each new packet, the address in the *SRAM_B* of that stage is reset to the start (the first entry contains the first encoded fields from TCPdst and TCPsrc of all rules) to guarantee that correct fields will be compared with the corresponding bits of the encoded packet key. The detailed description of *FXUs* and *OHDs* is in [30].

More stages must be added as shown in Figure 4 in order to match S number of packets in parallel. Despite the addition of more packets and comparison units, there is only one read from *SRAM_A*. However, encoded fields for each stage from corresponding *SRAM_B* need to be read separately. All stages perform comparisons with their respective packet key bits or encoded packet key bits and store their own result. To handle the results of multiple stages, encoders (or priority encoders) are added to handle packet classification decisions provided by the *FXUs* and *OHDs*. Each stage has one encoder shown in Figure 4.

IV. Experimental Setup and Results

The *LOP* and *LOP_RE* architectures with various configurations (2, 4, 6 and 8 stages and 1-8 priority encoders) have been implemented using VHDL and synthesized using Synopsys Design Compiler [31] with the TSMC 65nm process library. Three main aspects are explored: one, the relative storage expansion ratio between non-encoded rule set and encoded rule set; two, the power consumption per cell (field) of *LOP* and *LOP_RE* designs; three, the lookup rate and throughput of the *LOP_RE* scheme with different benchmarks and the energy comparison between TCAM, *LOP*, and *LOP_RE*. Ten

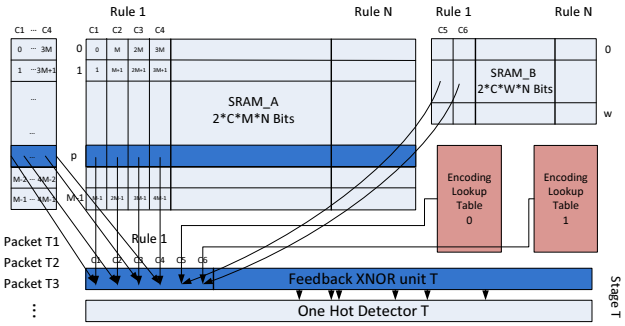

Fig. 3: *LOP* for single stage T

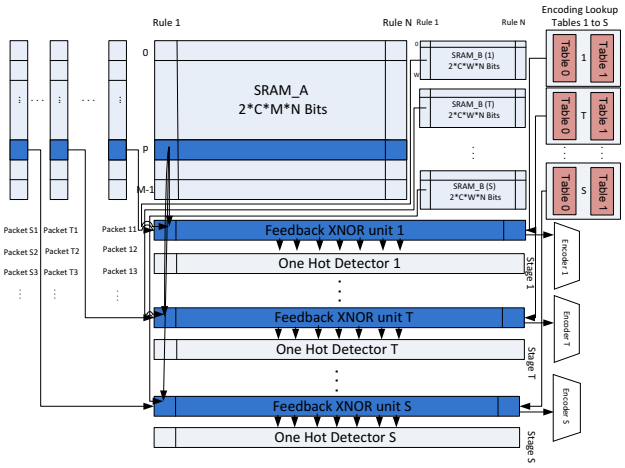

Fig. 4: *LOP_RE* for multiple stages

rulesets and corresponding packet traces were generated using the ClassBench tool [32] to create benchmarks. *PrimeTime-PX*, part of the Synopsys tool suite, was used to estimate the power consumption of the circuits used in this paper. ModelSim SE 6.0 [33] was used to simulate the design under the Linux environment.

A. Storage Expansion Ratio

Table II shows the relative storage expansion ratio between non-encoded rules (directly translating ranges into prefixes) and encoded rules in different benchmark using EIEBC and EIEGC encoding. The storage expansion ratio α is defined in Equation 1.

$$\alpha = \frac{the\ number\ of\ expanded\ rules\ after\ converting\ ranges}{the\ number\ of\ (original)\ rules} \quad (1)$$

For example, Table II shows that a non-encoded range system requires, on average, 3.25 times more rules than *LOP_RE* using EIEBC encoding scheme for benchmark acl1.).

Encoding Schemes	acl1	acl2	acl3	acl4	fw1	fw2	fw3	fw4	ipc1	ipc2
EIEBC	3.25	6.87	1.69	2.48	6.00	3.48	13.2	5.67	2.14	2.31
EIEGC	3.52	6.87	1.71	2.56	6.00	3.54	13.2	5.75	2.14	2.48

TABLE II: Relative Storage Expansion ratio α between non-encoded rules and encoded rules

Architecture	LOP (μW/field)				LOP_RE (μW/field)			
Encoders	2S	4S	6S	8S	2S	4S	6S	8S
1	0.697	0.791	0.885	0.979	0.891	1.353	1.815	2.276
2	0.778	0.872	0.967	1.061	0.973	1.434	1.896	2.358
3	-	0.957	1.048	1.143	-	1.516	1.978	2.439
4	-	1.035	1.129	1.224	-	1.597	2.059	2.521
5	-	-	1.211	1.305	-	-	2.140	2.602
6	-	-	1.292	1.387	-	-	2.221	2.683
7	-	-	-	1.468	-	-	-	2.765
8	-	-	-	1.549	-	-	-	2.846

TABLE III: Power Consumption per Bit (μW/field)[5]

B. Power Consumption Comparison

Table III shows the power consumption of LOP and *LOP_RE* in different configurations. The estimated power of the hardware included SRAMs (SRAM in *LOP*, SRAM_A and SRAM_Bs in *LOP_RE* and additional SRAMs used to complete the rule checking after priority encoders (or encoders)), the *Encoding Lookup Tables* (each table contains 128 entries in this experiment), *FXUs*, *OHDs*, *encoders* and glue logic. To be able to compare with TCAM-based packet classification system, power consumption in Table III is presented in μW/field. The total power consumption can be calculated by μW/field $\times$ (number of rules) $\times$ (number of fields per rule).

In Table III, the first major column shows the architectures share various number of encoders (to be able to save power by reducing extra encoders and corresponding SRAM blocks). The second major column shows the power consumption (per field) in *LOP* for different configurations and the third major column shows the power consumption in *LOP_RE*. The power consumption is less than 1.6μW/field in *LOP* and less than 2.9μW/field in *LOP_RE* for all the configurations [3].

C. Throughput and Energy

Ten benchmarks have been tested in *LOP_RE* with different configurations shown in Table IV. In the first major column, two sub-columns show the number of stages and the number of shared encoders respectively. The second to eleventh columns shows the throughput which varies in different benchmarks. The throughput is increasing with the number of stages and the number of shared encoders added. We test *LOP_RE* using clock speed of 1.85ns and the throughput can easily achieve 540Msps (540 million searches per second, when lookup rate = 1) shown in Table IV.

Stages	Encoders	acl1	acl2	acl3	acl4	fw1	fw2	fw3	fw4	ipc1	ipc2
2	1	92	113	167	157	151	205	140	151	157	178
	2	92	130	178	162	157	205	146	151	157	189
4	1	232	194	227	221	238	227	243	243	232	238
	2	232	205	354	324	308	227	292	302	232	243
	3	270	232	351	351	319	400	302	308	281	378
	4	308	259	362	351	319	410	308	308	286	383
6	1	238	221	237	227	238	275	243	243	243	232
	2	308	383	388	383	459	529	416	448	383	302
	3	378	383	481	523	475	556	448	475	383	497
	4	378	389	497	529	475	556	448	475	464	502
	5	416	486	502	540	481	578	464	486	475	540
	6	416	513	518	594	481	583	470	486	486	540
8	1	238	221	238	227	238	275	243	243	243	232
	2	362	432	459	448	457	545	535	454	464	454
	3	497	594	616	599	599	653	583	589	621	610
	4	513	594	670	599	632	686	589	599	632	621
	5	540	616	675	637	637	686	626	605	648	632
	6	540	616	680	648	643	686	626	605	648	713
	7	567	616	680	648	643	707	643	610	653	713
	8	567	637	686	700	648	745	664	616	653	740

TABLE IV: Throughput of *LOP_RE* packet classification. Note that, the throughput of 2-D-TCAM [29] is 495Msps with the lowest energy consumption of 3.38f/cell/search.

$$Lookup\ Rate = \frac{No.\ of\ searches}{No.\ of\ clock\ cycles} \quad (2)$$

$$Throughput = Lookup\ Rate * Clock\ Frequency \quad (3)$$

$$Energy\ per\ field\ per\ search = \frac{Power/field\ (cell)}{Throughput} \quad (4)$$

Equation 4 is used to calculate energy per field per search of *LOP* and *LOP_RE* and energy per cell per search of TCAM. Note that, *LOP* and TCAM systems have not been integrated with packet-side key encoding units. The ranges of rule set is directly converted to prefixes in *LOP* and TCAM systems. Whereas, *LOP_RE* is able to handle range encoding using EIEBC and EIEGC schemes with corresponding packet-side key encoding units. The total energy consumption is equal to the product of *Energy per field (cell) per search*, *number of (original) rules*, α and *fields (cells) per rule*. Note that, in our experiment, the fields (cell) per rule for TCAM, *LOP* and *LOP_RE* is all 96 (includes IPdst, IPsrc, TCPdst and TCPsrc). Note that, we did not compare the energy of *LOP_RE* with TCAM with packet-side range encoding units (*TCAM_RE*)

[3]Note that, due to the very high runtime needed for power simulations, we did not consider rule sets larger than 1024. However, we can estimate the power consumption of the proposed architecture for larger rulesets using the smaller models.

[4]Note that, for the same ruleset, *LOP_RE* needs much less storage compared with *LOP*. See Table II.

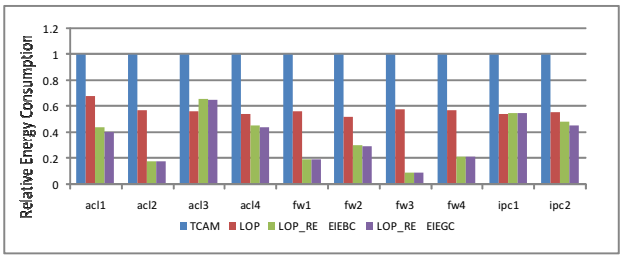

Fig. 5: Relative Energy Consumption between TCAM, *LOP*, and *LOP_RE* packet classification systems

because all current TCAM-based packet classification systems that utilise range encoding, such as [3] did not provide enough power and throughput information for a useful comparison to be made.

Figure 5 plots the comparison of energy consumption among 2-D-TCAM [29] (3.38fJ/cell/search when throughput is 495Msps in 65nm CMOS technology), *LOP* architecture (1.92fJ/field/search when average throughput is 596Msps in 65nm CMOS technology), and *LOP_RE* architecture (4.02fJ/field/search when average throughput is 599Msps in 65nm CMOS technology) with the same ruleset. In Figure 5, the horizontal axis labels the different benchmarks, and the vertical axis shows the relative energy consumption of each architecture (TCAM is set to be 1). We can see that on average, *LOP* consumes 57% of the energy compared that TCAM consumes and the energy consumption of *LOP_RE* is only 36% and 35% of the energy of TCAM (using EIEBC and EIEGC encoding schemes respectively).

V. Conclusion

In this paper, we have proposed a novel low-power *LOP_RE* architecture to handle line-rate, packet-side range encoding for packet classification. *LOP*-based packet classification architectures are implemented to compare against the latest TCAM-bsed approaches in 65nm CMOS technology (both power and throughput). Results show the best energy saving compared to the TCAM-based system is 65% for the same rule set.

The *LOP_RE* architecture is more flexible than TCAM-based architectures and is capable of trading off low power consumption against high throughput by altering configurations. More optimal *LOP_RE* architectures can be achieved according to different network environments. A hybrid *LOP*-based packet classification can be designed according to the characteristics of the rule set (the percentage of the rules which have ranges). Thus, designing such optimal application specific systems can further reduce power consumption and obtain high throughput.

References

[1] P. Gupta and N. Mckeown, "Algorithms for packet classification," *IEEE Networks*, vol. 15, pp. 24–32, 2001.

[2] D. E. Taylor, "Survey and taxonomy of packet classification techniques," *ACM Computing Surveys*, vol. 37, pp. 238–375, 2005.

[3] H. Che, Z. Wang, K. Zheng, and B. Liu, "Dres: Dynamic range encoding scheme for tcam coprocessors," *IEEE Transactions on Computers*, pp. 902–915, 2008.

[4] Y.-K. Chang and C.-C. Su, "Efficient tcam encoding schemes for packet classification using gray code," in *GLOBECOM'07*, 2007.

[5] T.V.Lakshman and D.Stitiadis, "High-speed policy-based packet forwarding using efficient multi-dimensional range matching," in *SIGCOMM*, 1998.

[6] J. van Lunteren and T. Engbersen, "Dynamic multi-field packet classification," in *GLOBECOME'02*, 2002.

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

[8] P. Gupta and N. Mckeown, "Classifying packets with hierarchical intelligent cuttings," *Micro,IEEE*, vol. 20, pp. 34–41, 2000.

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

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

[11] P. Gupta and N. McKeown, "Packet classification on multiple fields," in *SIGCOMM 99*, 1999.

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

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

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

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

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

[17] 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.

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

[19] 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*, pp. 1736– 1744, 2005.

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

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

[22] 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.

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

[24] K. Zheng, H. Che, and Z. W. andand Xin Zhang, "Dppc-re: Tcam-based distributed parallel packet classification with range encoding," *IEEE Transactions on Computers*, pp. 947–962, 2006.

[25] H. Hwang, K. Yamamoto, S. Ata, K. Inoue, and M. Murata, "Minimization of acl storage by adding minimal hardware of range matching and logical gates to tcam," in *International Conference on High Performance Switching and Routing*, 2008.

[26] Y.-K. Chang, "A 2-level tcam architecture for ranges," *IEEE Transactions on Computers*, pp. 1614–1629, 2006.

[27] R. Panigrahy and S. Sharma, "Sorting and searching using ternary cams," *IEEE Micro*, vol. 23, 2004.

[28] J. V. Lunteren and T. Engbersen, "Fast and scalable pacekt classificaton," *IEEE Journal on Selected Areas in Communication*, vol. 21, pp. 560–571, 2003.

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

[30] X. He, J. Peddersen, and S. Parameswaran, "*LOP*: A novel sram-based architecture for low power and high throughput packet classification," in *CODES-ISSS'09*, 2009.

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

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

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