Range Tree-Linked List Hierarchical Search Structure for Packet Classification on FPGAs

Oğuzhan Erdem
Electrical and Electronics Engineering
Trakya University
Edirne, TURKEY 22030
Email: ogerdem@trakya.edu.tr

Aydin Carus
Computer Engineering
Trakya University
Edirne, TURKEY 22030
Email: aydinc@trakya.edu.tr

***Abstract*—Field Programmable Gate Arrays (FPGAs) satisfying the abundant parallelism and high operating frequency demands are the most promising platform to realize SRAM-based pipelined architectures for high-speed packet classification. Due to the restrictions of the state-of-the-art FPGAs on the number of I/O pins and on-chip memory, larger filter databases can hardly be accommodated by the current approaches. Therefore, new data structures which are frugal with the memory are lately in high demand. In this paper, two stage range tree-linked list hierarchical search structure (RLHS) is introduced for packet classification. Our proposed structure comprising range tree in Stage 1 and linked lists in Stage 2, resolves backtracking and memory inefficiency problems in the pipelined hardware implementation of hierarchical search structures. We further present a categorization algorithm that partitions an input ruleset based on the field characteristics of rules to reduce the memory requirement. Each partition has an individual RLHS with specialized node structures free from redundant fields used for storing wildcards and range points. Our design is realized on an SRAM-based parallel and pipelined architecture using FPGAs to achieve high throughput. Utilizing a state-of-the-art FPGA, RLHS architecture can sustain a 404 million packets per second throughput or 129 Gbps (for the minimum packet size of 40 Bytes) while maintaining packet input order and supporting in-place non-blocking rule updates.**

I. Introduction

Due to the fast growth of the Internet, it has become a great challenge to design high performance packet forwarding engines. With the recent advancements in optical networking technology, line speeds go beyond 100 Gbps [1]. To accommodate such high rates, an internet core router needs to process an Internet Protocol (IP) packet in 3.2 ns, i.e. 312 million packet per second (MPPS), for a minimum size (40 bytes) packet. As the demand for high throughput routers increases, data path functions such as packet classification and IP lookup requires further investigations by the research community.

In packet classification, the incoming packets are categorized into flows by comparing multiple fields in a packet header with the corresponding fields of a pre-defined set of filters. The major design metrics in packet classification are throughput, storage space, and dynamic update support. Additionally, the preprocessing complexity, power consumption, implementation cost and the scalability in terms of the size of rulesets are the remaining crucial criterions. To satisfy the high throughput demand in packet classification engines, hardware-based approaches are mostly preferred by router designers. These solutions can be categorized into two: ternary content addressable memory (TCAM)-based and dynamic/static random access memory (DRAM/SRAM)-based. Although TCAM-based engines can retrieve search results in just one clock cycle, they have serious drawbacks comprising low density, high cost, large access time, high power consumption, poor arbitrary range support and poor multiple-match support. On the contrary, an SRAM chip has lower cost, less power consumption, much higher density and speed as against a TCAM [2], [3]. SRAM-based solutions generally utilize tree type data structures and therefore multiple cycles are required to acquire a single search result. To ameliorate the throughput, pipelining techniques are involved in such solutions. Field Programmable Gate Arrays (FPGAs) having unprecedented features such as reconfigurability, vast amount of on-chip logic and abundant parallelism are the most convenient platform to realize these SRAM-based parallel and pipelining architectures. However, due to the restrictions of state-of-the-art FPGAs on the amount of I/O pins and on-chip memory (BRAM), these solutions are unable to support large rulesets. For this reason, memory efficient data structures and resource efficient architectures have lately attracted a great deal of attention from the researchers. This paper makes the following major contributions:

- A ruleset categorization algorithm that partitions a given ruleset into unique sub-rulesets based on the field characteristics of rules (Section III-B).
- A hierarchical structure, named Range Tree-Linked List Hierarchical Search Structure (RLHS) that accomplishes significant memory saving (Section III-C).
- Optimizations on categorization algorithm and RLHS to further ameliorate memory and resource efficiencies while achieving fixed search delay (Section IV).
- A high-throughput multi-pipelined SRAM-based architecture on FPGAs that accommodates the proposed search structure (Section V).

We arranged the rest of the paper as follows; Section II comprises the background and prior work about packet classification. Section III presents the partitioning algorithm and RLHS data structure. Section IV covers the optimizations on categorization algorithm and RLHS. Section V introduces the RLHS architecture. Section VI exhibits the performance evaluation results of proposed structure. Section VII concludes the paper.

II. Background

A. Packet classification overview

In packet classification, Internet Protocol (IP) packets are classified into flows by comparing 5-tuple header fields (Source IP address (SA), Destination IP address (DA), Source Port Number (SP), Destination Port Number (DP), Protocol) with the corresponding fields of rules in a ruleset or filter database. Each rule in a ruleset have multiple fields with their associated values, a priority value, and an action to be taken if matched. Rule fields are specified by mixture of prefixes, ranges, exact values and wildcards. A packet is considered matching a rule if and only if all the header fields matches their corresponding fields within that rule. Table I demonstrates a sample ruleset.

TABLE I. Sample 5-field ruleset

Rule	SA	DA	SP	DP	PRTCL	Priority	Action
R_1	0*	10*	$[80, 80]$	*	TCP	1	Act0
R_2	0*	01*	*	*	UDP	2	Act1
R_3	0*	1*	*	$[44, 44]$	TCP	2	Act2
R_4	00*	1*	$[17, 17]$	*	UDP	3	Act3
R_5	00*	11*	*	$[100, 100]$	TCP	4	Act4
R_6	10*	1*	*	*	UDP	5	Act5
R_7	*	00*	*	*	TCP	5	Act6
R_8	0*	10*	*	$[100, 100]$	TCP	6	Act7
R_9	0*	1*	$[8, 100]$	*	TCP	7	Act8
R_{10}	0*	10*	$[17, 80]$	*	UDP	7	Act9
R_{11}	111*	000*	$[80, 80]$	*	TCP	8	Act10

B. Related work

Packet classification algorithms can be categorized into four groups: (1) exhaustive search [4], [5], (2) decomposition [6]–[8], (3) decision tree [9]–[11], and (4) hierarchical-trie (H-trie) [12]–[16].

Exhaustive search comprises basic linear search and TCAM-based parallel search. In both cases, all the entries in a ruleset are analyzed. However in a linear search, the header of a packet is compared with all the rules sequentially. On the other hand, TCAM searches on entries are performed simultaneously. Decomposition based approaches contain two consecutive phases; (*i*) independent searches on each field and (*ii*) merging results obtained from the first phase. Although these solutions achieve high throughput, vast amount of storage space is required to aggregate individual search results. Decision trees take the geometric view of the packet classification problem. HiCuts [9] and its enhanced version HyperCuts [10] are the most popular algorithms in this group. At each node of the decision tree, the search space is cut into sub-spaces based on the information from one or more fields in the rule. These algorithms are easy to implement in both software and hardware but their performance is much sensitive to the ruleset structure and they have poor incremental update support.

Hierarchical-trie (H-trie) data structure is composed of a single but large source address trie (SA tries) and multiple small destination address tries (DA tries) which are hierarchically connected to that SA trie. SA and DA tries are constructed using SA and DA prefix fields of rules. Each prefix node in a SA trie points to a DA trie in the second stage. Search starts from the SA trie. Once a prefix node in the SA trie is encountered, search passes to the DA trie connected to that prefix node. Even though a match can be found at any node in the DA trie, search has to **backtrack** to the SA trie and continue to find other possible matches and choose the highest priority one. When a leaf node or a null pointer is reached in the SA trie, search ends. Since, the backtracking causes stalling the pipeline in hardware implementations, the realization of H-tries in hardware is not practical.

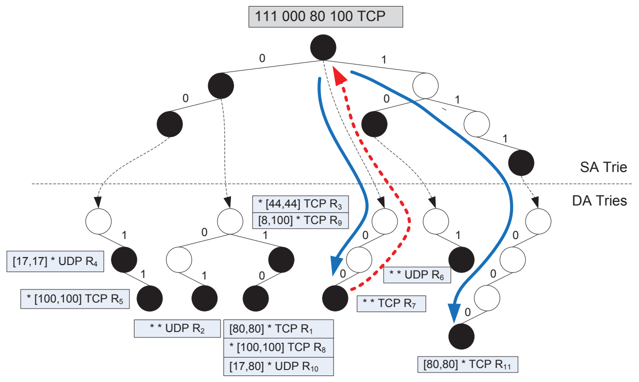


Fig. 1. Backtracking in H-trie structure

Set-pruning trie eliminates the backtracking by replicating the rules [12]. To eliminate the backtracking, Grid-of-tries (GoT) [13] for 2-field packet classification stores each rule into only one node by adding switch pointers to some trie nodes. Extended Grid-of-tries (EGT) [14] improves the previous idea by accommodating multiple fields. Although EGT has good memory performance, high number of worst-case memory accesses decreases the search time performance. To enable hardware implementation, the authors of [15] proposed Clustered Hierarchical Search Structure (CHSS) that eliminates backtracking problem by dividing rulesets into multiple clusters. Each cluster employs non-overlapping rules and implemented using individual three-stage hierarchical data structures. Two-stage hierarchical hybrid search structure that also eliminates backtracking using the similar clustering approach were proposed in [16].

III. Data Structure and Algorithms

A. Motivation

The hardware realization of hierarchical search structures have two issues: (1) backtracking and (2) memory inefficiency.

Fig. 1 illustrates a H-trie structure constructed using the rules in Table I. For this scheme, the backtracking occurs while searching a 5-tuple IP packet header ($SA = 111$, $DA = 000$, $SP = 80$, $DP = 100$, $PRTCL = \text{TCP}$). Even though, the header firstly matches R_7, backtracking is compulsory to search for the other matches. In this figure, the blue lines refer to forwarding search paths while the red dashed line corresponds to a backtracking path. Eventually, R_7 and R_{11} are reported as the matching rules however R_{11} is chosen as the highest priority match.

The memory inefficiency in hardware implementation of H-tries arises from the variety of the number of rules stored in each node. In hardware implementation, the size of trie nodes are initially fixed to a constant value that is determined by the largest node in that trie. In this case, the unused memory spaces in some nodes carrying only a few rules lead memory and

	SA	DA	SP	DP
F^* (wildcard)	*	*	$[SP_{low}, SP_{high}]=[00...0, 11...1]$	$[DP_{low}, DP_{high}]=[00...0, 11...1]$
$F^{!*}$(not wildcard)	!*	!*	$[SP_{low}, SP_{high}]!=[00...0, 11...1]$	$[DP_{low}, DP_{high}]!=[00...0, 11...1]$
$F^=$(point)	-	-	$SP_{low} = SP_{high}$	$DP_{low} = DP_{high}$

(a)

	DA^*	$DA^{!*}$
SA^*	G_1	G_3
$SA^{!*}$	G_2	G_4

(b)

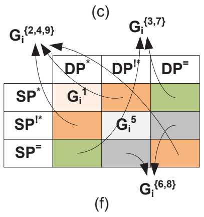
Group	G_1	G_2	G_3	G_4
Range Tree Field	SP or DP	SA	DA	SA or DA

(d)

	DP^*	$DP^{!*}$	$DP^=$
SP^*	G_i^1	G_i^2	G_i^3
$SP^{!*}$	G_i^4	G_i^5	G_i^6
$SP^=$	G_i^7	G_i^8	G_i^9

(c)

	SP_{low} (bits)	SP_{high} (bits)	DP_{low} (bits)	DP_{high} (bits)	Prtcl (bits)	Total (bits)
G_2^1	-	-	-	-	8	**8**
G_2^2	-	-	16	16	8	**40**
G_2^3	-	-	-	16	8	**24**
G_2^4	16	16	-	-	8	**40**
G_2^5	16	16	16	16	8	**72**
G_2^6	16	16	-	16	8	**56**
G_2^7	-	16	-	-	8	**24**
G_2^8	-	16	16	16	8	**56**
G_2^9	-	16	-	16	8	**40**



(f)

(e)

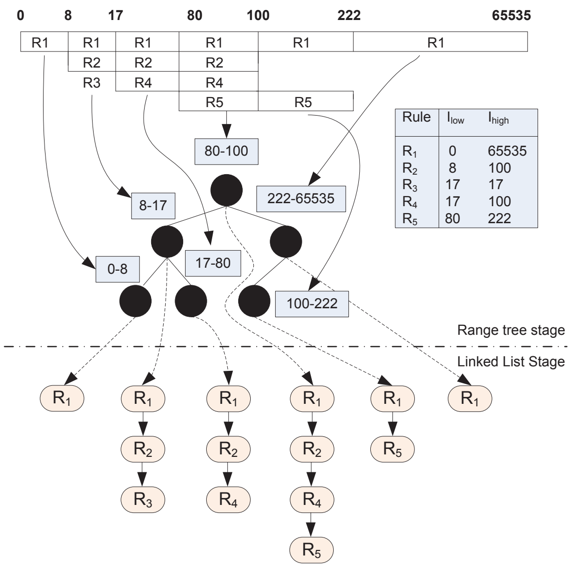
Fig. 2. (a) Field characteristics of rules (b) Step 1 of partitioning (c) Step 2 of partitioning (d) Rule fields used for constructing RTs (e) The number of data bits in a LL node for the sub-rulesets (f) Step 2 optimization of categorization algorithm within G_2

resource inefficiency. As an example, the hardware realization of the H-trie presented in Fig. 1 needs a memory space for 17×3 rules because the largest node has 3 rules and DA tries includes 17 nodes. However, only 21.5% (11 rules) of the total space is used.

In this paper, a hierarchical search structure comprising a range tree (RT) in Stage 1 and linked lists (LL) in Stage 2 is proposed. RT divides the search space into *disjoint* range intervals. Each node of the tree corresponds to an interval and hence at most one match is possible in Stage 1. Thus, once the search quits RT, it does not have to return back. Furthermore, each node in LL carries at most one rule and hence memory inefficiency is naturally solved. To further improve memory efficiency, we also offer a rule categorization algorithm that partitions the ruleset based on the field characteristics of rules. For each partition, an individual data structure with specialized node structures that are purified from redundant fields for wildcards and some special ranges is constructed.

B. Rule Categorization

We categorize the rules in a given ruleset into sub-rulesets based on field characteristics of rules such as wildcards in fields and range properties. As shown in Fig. 2a, a field F in a rule (excluding the protocol field) may have a (1) prefix/range wildcard (F^*) (2) prefix/range data other than wildcard ($F^{!*}$) or (3) point if it is specified as a range ($F^=$). Note that, if a field F stores range data (e.g. SP, DP), this range is specified by using range boundaries (e.g. $[SP_{low}, SP_{high}]$, $[DP_{low}, DP_{high}]$). If a lower and upper boundaries of a given range are composed of all 0's and 1's respectively, then this range covers all the space and denoted as *range wildcard*. On the other hand, if a lower and upper boundaries of a given rule are identical, this range corresponds to a *point*.



Rule	I_{low}	I_{high}
R_1	0	65535
R_2	8	100
R_3	17	17
R_4	17	100
R_5	80	222

Fig. 3. RLHS data structure

Our categorization algorithm inputs a set of rules G and outputs sub-rulesets G_i^j as a result of two partitioning steps. Initially, our algorithm partitions the ruleset into 4 groups based on SA and DA fields of rules, as shown in Fig. 2b. Each group is indexed starting from 1 to 4 and the i^{th} group is represented as G_i. In this scheme, G_1 contains the rules whose SA and DA fields both carry wildcards and G_2 (G_3) is composed of rules having only DA (SA) fields store wildcards. Finally, G_4 includes rules with none of the SA and DA fields stores wildcard information. We further divide these groups into more specific and smaller sub-groups (or sub-rulesets) based on the range properties of SP and DP fields of rules as a second step. Fig. 2c demonstrates the output of a second step of partitioning. Each group G_i in Fig. 2b is now partitioned into 9 sub-groups, recognized as G_i^j. For instance, a sub-ruleset G_i^1 includes the rules which store wildcards in their both SP and DP fields. Similarly, both SP and DP fields of rules in G_i^9 represent a point in a range space.

C. Range Tree-Linked List Hierarchical Structure (RLHS)

As a result of the partitioning, the total number of 36 $(4*9)$ sub-rulesets are obtained. We represent each set using a distinct instance of a Range Tree-Linked List Hierarchical Search Structure (RLHS) which comprises 2 stages:

Stage 1 **(Range tree (RT) Stage)**: A range tree (RT) is constructed for each sub-rulesets. Fig. 3 shows a sample range table and its corresponding range tree. A range tree divides the search space into *disjoint* range intervals and each node of the tree corresponds to a single interval. In an RT, each node comprises: (1) lower bound of interval (I_{low}), (2) upper bound of interval (I_{high}), (3) left pointer (Ad_L), (4) right pointer (Ad_R) and (5) next stage pointer (Ad_N). The left subtree of any node covers ranges whose upper bounds are less than or equal to the lower bound stored in that node. Similarly, the right subtree stores ranges whose lower bounds are greater than the upper bound of that node. The construction of an RT employs two steps; (i) determination of disjoint range intervals (ii) selection of the correct interval (pivot) as root

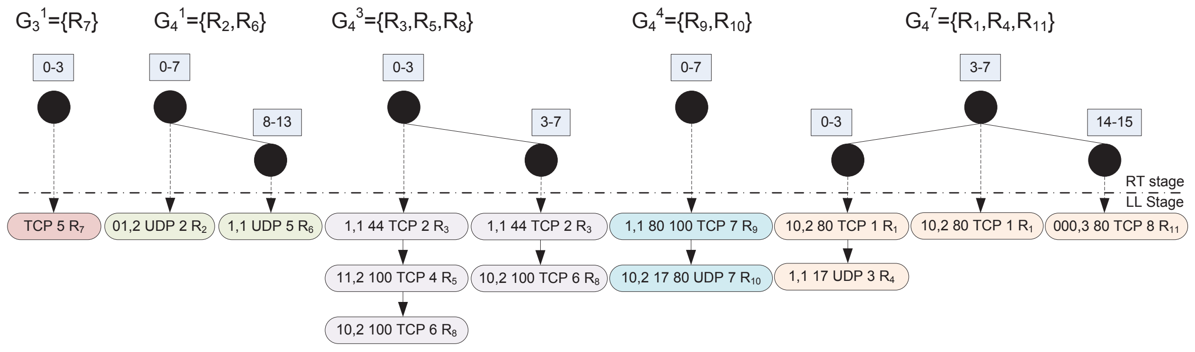


Fig. 4. Range tree-Linked list Hierarchical Search Structure for the ruleset in Table I

and recursively building the left and right subtrees. Note that if an RT is constructed using prefixes rather than ranges, prefix to range conversion is also required before determination of disjoint range intervals. This conversion is simply achieved by appending '0' and '1' bits to a prefix to obtain lower and upper bounds of the interval that is covered by that prefix. For instance, a prefix 01* can be converted to the range $[4(0100), 7(0111)]$ in a 4-bit addressing space.

Stage 2 **(Linked List (LL) Stage)**: Each node in a RT points a linked list data structure in the second stage. Finally, the number of LL instances equals to the number of nodes of the RT in Stage 1. LLs are the most common and basic data structures to store a series of data. Each node in a LL comprises a data field and an address pointer to the next node in the list. Each node in a single list stores a distinct rule as shown in Fig. 3. However, all the rules stored in a single list have the same range value represented by unique node of an RT in Stage 1. The depth of a list depends on the number of rules sharing the same range interval. In Fig. 3, the rules R_1, R_2, R_4 and R_5 share the same range interval ($[80, 100]$) and hence reside in the same list.

An unique RLHS data structure is built per sub-ruleset, G_i^j. However, the rule field to build RT in Stage 1 and the node structure of LL in Stage 2 are individually defined for each sub-rulesets. Fig. 2d gives which fields be used for building RTs for the sub-rulesets within the group G_i. On the other hand, Fig. 2e shows the number of bits required for the data field of a LL node in Stage 2 for the sub-rulesets within G_2. For instance, an RT is built using SA fields of rules in G_2^1 and a LL node stores only protocol information in its data field. Note that, storing only the upper bounds for the range points is sufficient, since the lower bounds are the same. Additionally, no memory space is needed to represent the wildcards in fields in RLHS. For the sub-rulesets within G_4, SA or DA prefix bits and the length of a prefix are also involved in the LL nodes. Due to the space restrictions in this paper, node structure details for the remaining sub-rulesets and update procedures are skipped. Fig. 4 illustrates a RLHS data structure for the given set of rules in Table I.

D. Packet Classification Algorithm

Prior to the search in RLHS, the header fields are extracted from each arriving packet. The resulting frame is then forwarded to all groups and search is performed in parallel in all data structures as follows:

RT search: Search in RT is initiated from the root node and the search frame traverses left or right, based on the comparison result at each node. Once, the input key falls in a specified range of any node, match occurs and search in RT terminates. Next, search proceeds to traverse the connected linked list in the following stage.

LL search: At each LL node, the remaining header fields of the packet other than the one used in RT search are compared with the corresponding fields of the rules stored in that node, in parallel. The input key is considered to be matched with a rule if and only if all the header fields matches the corresponding rule fields. Search terminates, when all the nodes in a LL is visited. Once there is a match in any node and the priority value of the matched rule is higher than that of the previous match, the search result is updated. Finally, the outputs from all groups are fed into the priority encoder which determines the highest priority one among all the matches.

Claim: *Backtracking* is not necessary in the search using the proposed RLHS data structure.

Proof: The proposed RT structure ensures that all the nodes in RT are disjoint. Thus, in Stage 1, at most one match is possible and search does not have to return back to Stage 1, once it quits for Stage 2.

IV. Optimizations

A. Optimization in categorization algorithm

Step 1**:** Based upon our observation on the rule sets provided by class-bench [17], the amount of the rules that fall in the category of G_1 and G_2 are reasonably small when compared to the other groups. Consequently, we suggest merging the groups G_2 and G_4 into a single group $G_{\{2,4\}}$. On the other hand, we suggest canceling Step 2 for all the sub-rulesets within G_1 and represent all using a single RLHS.

Step 2**:** As mentioned in Section III-B, the initial groups in Step 1 are further categorized into 9 sub-groups. However, we again suggest merging some of those sub-groups and reduce the number from 9 to 5 per G_i as demonstrated in Fig. 2f ($G_i^2 \cup G_i^4 \cup G_i^9 \rightarrow G_i^{\{2,4,9\}}$, $G_i^3 \cup G_i^7 \rightarrow G_i^{\{3,7\}}$, $G_i^6 \cup G_i^8 \rightarrow G_i^{\{6,8\}}$). The reason for this merging is the similarity of node structures in some groups which can easily be mapped onto a same data structure by only introducing a few identification bits overhead to the nodes. Finally, the overall number of sub-groups is reduced from 36 to 11 after the optimizations in categorization algorithm.

B. Optimization in RLHS

We propose to set an upper bound for the depth of the LL in Stage 2, named as P_D which also limits the the number of nodes that can be stored in a single LL. Setting an upper bound can simply be achieved by merging some of neighboring range intervals as soon as the final merged interval can contain at most P_D rules. For instance, if the two range intervals $[8, 17]$ and $[17, 80]$ in Fig. 3 are merged, the new merged interval ($[8, 80]$) contains 4 unique rules. However, since the ranges of rules are extended in this optimizations, each node in LL has to keep the original ranges of rules as an overhead.

P_D is a trade-off between the delay and the memory requirement. However, small P_D values may lead rule overflows. In this cases, the overflowing rules that can not be located in RLHS structure are stored in an auxiliary data structure or a TCAM. We call the ratio of the number of rules stored in the auxiliary structure over the total number of rules of the given ruleset α_T. Although this optimization causes to increase the size of LL nodes, it brings the following advantages; (i) the number of disjoint range intervals and hence the number of RT nodes is substantially reduced and memory saving is achieved, (ii) the delay can be optimized to satisfy the service and resource requirements, (iii) the number of rule replications in multiple intervals is reduced and hence the memory efficiency is improved. In the given example above, the merged interval ($[8, 80]$) stores only 4 rules rather than 6 $(3 + 3)$ rules.

V. Architecture and Implementation

Fig. 5a presents an overall block diagram of RLHS architecture that is designed to accommodate the proposed data structure. We used pipelining to attain high throughput. Each sub-group comprises two consecutive pipelines; RT pipeline and LL pipeline. Therefore the number of pipelines equals to the number of sub-groups. The depth of the RTs and LLs determine the number of stages in pipelines. RT structure in Stage 1 is mapped onto the RT pipeline in which each stage carries the nodes of a single level of the RT. A single LL pipeline stores all the linked list structures that are connected to the same RT. Similar to RT pipeline, each stage of a LL pipeline stores a single level of linked lists. The proposed architecture is realized on an FPGA platform.

Fig. 5b and c illustrates the single pipeline stages of RT and LL pipelines respectively. An SRAM module and a match module are involved in a single stage. SRAMs which are used to maintain the nodes of the data structures are implemented using on-chip BRAMs in FPGA board. Match modules employed in both pipelines are used to compare the input keys with the stored rule data in the BRAM. Based on the comparison output, the address of the next node is determined in RT pipeline and the search result is updated if there is match in LL pipeline. To double the throughput of the designed RLHS architecture, dual-ported BRAMs of FPGAs having dual Read/Write ports are utilized in pipeline stages and thus two packets can be processed in a single clock cycle. Additionally, RHLS architecture can be extended with external SRAMs to accommodate much larger rule databases.

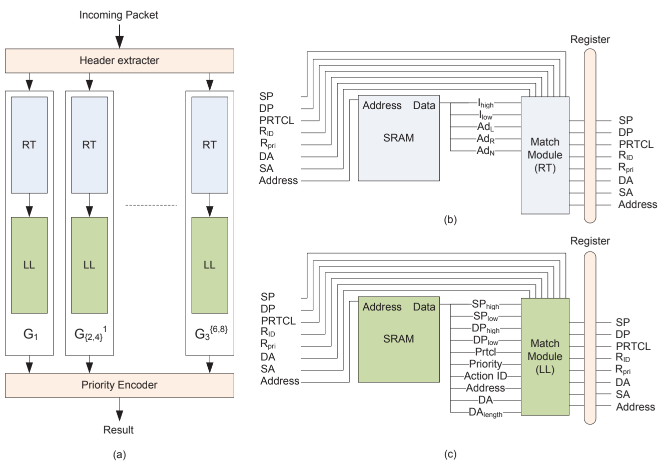


Fig. 5. Multi-pipeline TT_ϵ architecture

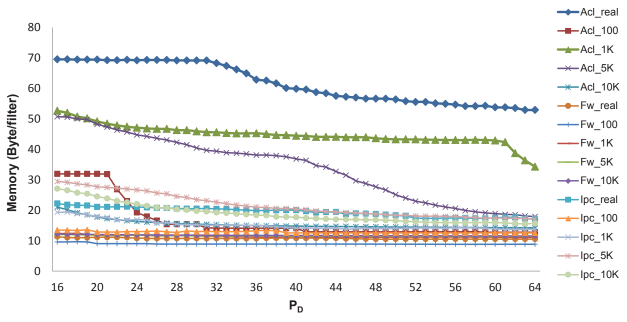


Fig. 6. Memory efficiency for various P_D values

However, the number of external SRAMs that can be used is limited owing to the available I/O pins of FPGA device used.

VI. Performance Evaluation

A. Experimental Setup

The three different types of rule sets (Access Control List (ACL), Firewall (FW), and IP Chain (IPC)) with each has 5 different sized sets (from 100 to 10K rules) provided by class-bench [17] were used in simulations. The size of rulesets are demonstrated in the second column of Table II. All the optimizations proposed for the categorization algorithm and RLHS structure are included in our simulations.

B. Memory requirement

The change of memory efficiency (byte per filter) and α_T value with respect to the P_D value for each rulesets are demonstrated in Fig. 6 and 7 respectively. Note that, $P_D = 40$ ensures for all rulesets that the percentage of the overflowing rules is less than 1% ($\alpha_T \leq 0.01$)

Table II demonstrates the memory efficiency results of the existing approaches for the same rulesets. Column 3 presents the results of RLHS without optimization. The results of the optimized RLHS with $P_D = 64$ in Column 4 guarantees no auxiliary storage is required ($\alpha_T = 0$). Note that all results are presented in the number of bytes per rule. Additionally, the depth of RT and LL structures are also included in Column 1 and 2 using (A - B) format, where A and B indicates the

TABLE II. Memory efficiency (Bytes per rule) for various rulesets

1	2	3	4	5	6	7	8	9	10
Ruleset	N	$RLHS$	$RLHS_{opt}$	$CHSS$ [15]	TT_ϵ [16]	EGT [14]	HyperCuts [10]	BV [8]	Hybrid scheme [11]
ACL	752	45.44 (7 - 59)	52.92 (6 - 64)	25.11	22.08	25.41	32.58	71.80	N/A
ACL100	98	23.17 (5 - 13)	12.79 (1 - 64)	21.54	21.42	27.69	27.78	47.35	24.44
ACL1K	916	35.63 (7 - 49)	34.29 (6 - 64)	22.44	22.20	24.96	38.15	91.63	22.98
ACL5K	4415	37.83 (10 - 96)	17.91 (6 - 64)	23.95	21.44	24.87	59.64	257.23	24.83
ACL10K	9603	31.42 (12 - 24)	14.24 (7 - 64)	22.21	22.62	30.23	54.22	789.22	25.51
FW	269	15.39 (6 - 11)	10.54 (2 - 64)	21.76	10.48	25.31	399.18	40.72	N/A
FW100	92	11.32 (4 - 10)	8.83 (1 - 64)	27.53	10.37	23.42	113.37	27.46	56.63
FW1K	791	15.46 (7 - 26)	11.12 (3 - 64)	23.84	10.41	23.80	6110.58	67.08	215.06
FW5K	4653	20.89 (6 - 13)	11.40 (6 - 64)	35.37	13.27	39.04	16132.65	691.69	255.13
FW10K	9311	22.35 (6 - 13)	11.57 (7 - 64)	41.25	14.51	49.45	12554.18	1582.18	248.54
IPC	1550	18.57 (8 - 59)	17.09 (4 - 64)	21.80	21.64	26.63	128.52	61.57	N/A
IPC100	99	24.80 (6 - 6)	12.55 (1 - 64)	23.65	18.34	31.60	24.57	69.16	23.65
IPC1K	938	29.20 (9 - 20)	13.77 (1 - 64)	22.22	20.05	29.95	61.34	176.03	25.63
IPC5K	4460	27.82 (10 - 74)	17.11 (6 - 64)	21.72	21.56	27.62	406.80	358.61	49.46
IPC10K	9037	28.03 (11 - 70)	15.66 (7 - 64)	22.60	22.81	28.92	2378.35	788.69	43.30

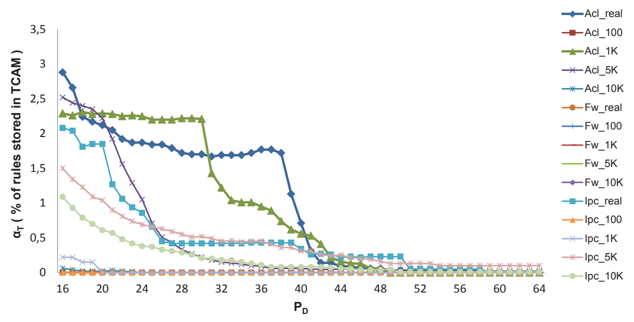


Fig. 7. The percentage (%) of overflowing rules for various P_D values

depth of RT and LL stages respectively. The performance results indicates that our algorithm provides significant memory saving compared to the state of the art algorithms. Moreover, the depth results proves that RLHS optimization reduces the depth of RT where the depth of LL is fixed as 64.

C. Throughput

The proposed architecture was realized in Verilog, utilizing Xilinx ISE 12.4. Xilinx Virtex-6 XC6VSX475T with -2 speed grade was used as the target device. The post place and route results points that our design can run at 202 MHz and is capable of processing packets at the clock period of 4.95 ns. Utilizing dual-ported BRAMs, the design can accommodate 404 million lookups per second (MLPS), or 129 Gbps for the minimum packet size of 40 Bytes (or 320 bits).

VII. Conclusion

In this paper, range tree-linked list hierarchical search structure is introduced. The backtracking problem for hierarchical structures is naturally eliminated in our design. The proposed algorithm accomplishes significant memory saving when compared to the existing algorithms. To accommodate the proposed data structure, we designed and implemented a very high-throughput SRAM-based linear pipelined architecture. Furthermore, several optimizations were conducted to improve the performance of the proposed design. As a future work, we plan to extend the data structure to support packet classification with more number of fields, such as OpenFlow.

References

[1] S. Gringeri, E.B. Basch, and T.J. Xia, "Technical considerations for supporting data rates beyond 100 Gb/s," *IEEE Communications Magazine*, vol. 56, no. 1, pp. 21–30, 2012.

[2] C.R. Meiners, A.X. Liu, and E. Torng, "TCAM SPliT: Optimizing Space, Power, and Throughput for TCAM-based Packet Classification Systems," in *Proc. ANCS*, pp. 200–210, 2011.

[3] H. Song, and J.S. Turner, "Toward Advocacy-Free Evaluation of Packet Classification Algorithms," *IEEE Trans. Comput.*, vol. 60, no. 5, pp. 723–733, 2011.

[4] T.B. Mishra and S. Sahni, "PETCAMA Power Efficient TCAM Architecture for Forwarding Tables," *IEEE Trans. Comput.*, vol. 61, no. 1, pp. 3–17, 2012.

[5] D.E. Taylor, "Survey and taxonomy of packet classification techniques," *ACM Comput. Surv*, vol. 37, no. 1, pp. 238–275, 2005.

[6] L. Sun, H. Le and V.K. Prasanna, "Optimizing Decomposition-based Packet Classification Implementation on FPGAs," in *Proc. ReConFig*, pp. 170–175, 2011.

[7] T. Ganegedara and V.K. Prasanna, "StrideBV: Single Chip 400G+ Packet Classification," in *Proc. HPSR*, pp.1–6, 2012.

[8] T. V. Lakshman and D. Stiliadis. "High-speed policy-based packet forwarding using efficient multi-dimensional range matching," *SIGCOMM Comput. Commun. Rev.*, vol. 28, pp. 203214, 1998.

[9] P. Gupta and N. Mckeown. "Packet classification using hierarchical intelligent cuttings," in *Proc. HOTI*, pp. 34–41, 1999.

[10] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet classification using multidimensional cutting," in *Proc. SIGCOMM*, pp.213–224, 2003.

[11] W. Jiang, and V.K. Prasanna, "Scalable packet classification: Cutting or merging?," in *Proc. ICCCN*, pp. 1–6, 2009.

[12] P. Tsuchiya. "A search algorithm for table entries with non-contiguous wildcarding," *Unpublished report*, Bellcore.

[13] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel. "Fast and scalable layer four switching," *SIGCOMM Comput. Commun. Rev.*, vol. 28, pp. 191202, 1998.

[14] F. Baboescu, S. Singh, and G. Varghese, "Packet classification for core routers: Is there an alternative to cams," in *Proc. INFOCOM*, pp. 53–63, 2003.

[15] O. Erdem, H. Le, and V.K. Prasanna. "Clustered hierarchical search structure for large-scale packet classification on fpga," in *Proc. FPL*, pp.201–206, 2011.

[16] O. Erdem, H. Le, and V.K. Prasanna. "Hierarchical hybrid search structure for high performance packet classification," in *Proc. INFOCOM*, pp.1898–1906, 2012.

[17] D. E. Taylor and J. S. Turner. "Classbench: a packet classification benchmark," *IEEE/ACM Transactions on Networking*, vol. 15, pp. 499–511, 2007.