The CSI Journal on
Computer Science and Engineering
Vol. 9, No. 2 & 4 (b), 2011
Pages 42-51
Regular Paper

High Throughput Multi-Pipeline Packet Classifier on FPGA

Rashid Hatami Mahmood Ahmadi

Department of Computer Engineering, Razi University, Kermanshah, Iran

Abstract

Packet classification is one of the most important functions in the router design. This is to support a variety of network functionalities. Pipeline-based decision tree is one of the best choices for increasing throughput on packet classification. SRAM-based and hardware-based solutions are often used to develop a high speed packet classification engine. However, SRAM-based solutions suffer from throughput degradation, delay variation and memory overflow. Moreover, hardware-based solutions suffer from fast update and non-linear structure. To address these problems, we proposed a FPGA-based multi-pipeline architecture for 5-tuple rules into multiple subsets to build a decision tree with high throughput and fast update. To fit the current largest rule-set in the FPGA device, several optimization techniques are proposed that maximize the resource utilization while sustaining high throughput. In this architecture look up tables (LUTs) are used instead of memory blocks. Partial reconfiguration in Field Programmable Gate Array (FPGA) used to reduce the time that is needed to change the behavior of architecture. The implementation results show that our architecture can store over 10K real-life rules in LUTs of a single Xilinx Virtex-6 FPGA, and sustain over 120 Gbps (i.e. 3× OC-768 rate) throughput for minimum size (40 Bytes) packets.

Keywords: Packet Classification, Pipeline-Based Decision Tree, Rule Set, Field Programmable Gate Array (FPGA).

1. Introduction

Rapid growth in network link rates and development of the Internet poses a strong demand on high speed router to support a variety of network applications, such as quality of service (QoS), virtual private networks (VPN), traffic billing, policy routing, multimedia communications, firewall, and other value added services. In order to provide these services, the router needs to classify the packets into different categories based on a set of predefined rules. The rules contain multiple field values that specify an exact packet header or set of headers. In traditional network applications, packet classification problems usually consider the fixed 5-tuple fields: 32-bit source/destination IP addresses, 16-bit source/destination port numbers, and 8-bit transport layer protocol. Due to the complexity of the search, rapid growth of the Internet traffic, as well as the rule-set size, packet classification is often a performance bottleneck in network infrastructure. For example, the current link rate has been pushed beyond the OC-768 rate (40 Gbps) which requires processing a packet every 8 ns for a minimum size (40 Bytes) packet in the worst case. Such throughput is impossible using any existing software-based solution [1]. Current hardware-based solutions for packet classification can be TCAM-based and Multi-core solutions. Most of the existing works in high-throughput packet classification are based on ternary content addressable memory (TCAM) [2, 3]. Although TCAM-based engines can retrieve lookup results in just one clock, however, TCAMs are not scalable with respect to clock rate, power consumption, or circuit area, and also suffer from range expansion when converting ranges into prefixes, compared to SRAMs [4]. One famous algorithm is HiCuts presented in paper [6], which can process rules well. However, HiCuts algorithm suffers from inefficient resource usage due to variation in the size of the tree nodes. A variety of hashing schemes such as Bloom Filters [8, 9, 10] have become popular due to their O (1) time performance and high memory efficiency [10]. Hashing schemes cannot provide deterministic performance due to potential collision and are inefficient in handling prefix matching [13].

In order to solve the problems mentioned above, in this paper, a FPGA-based architecture is proposed. Our design includes a parallel multi-pipeline architecture with several linear static pipelines. Each linear static pipeline includes 5-stages and each stage of the pipeline consists of several processing units. Our goal is to design a packet classifier engine fast enough for OC768 rate onto parallel architectures such as field-programmable gate array (FPGA).
The contributions of this paper are as follows.

- To the best of our knowledge, revisiting the 5-tuple packet classification solutions and considered decision-tree-based schemes which are considered among the most scalable.
- Presenting techniques to reduce rule-set complexity, which leads to minimize the hardware overhead and optimizing problem of rule duplication when building the decision tree.
- Proposal of a tree-based multi-pipeline architecture for 5-tuple rules into multiple subsets so that each subset uses of header fields to build a decision tree with bounded depth.
- For having flexible architecture and reduce update time, we exploit the partial reconfiguration provided in state-of-the-art FPGAs to achieve a high throughput.
- Our design will process one packet per clock cycle. Implementation results show that our architecture can store over 10K 5-tuple rules in a single Xilinx Virtex-6 FPGA, and sustain 123 Gbps throughput for matching minimum size (40 Bytes) packets.

The remainder of this paper is organized as follows: Section 2 states the problem and summarizes several representative packet classification algorithms. Section 3 reviews the related work on FPGA-based multi-field packet classification engines and pipeline architecture. Section 4 introduces our techniques for partitioning the rule-set and building the optimized decision tree. Section 5 presents proposed architecture and the tree-to-pipeline mapping scheme. Section 6 evaluates the performance of the architecture. Section 7 concludes this paper.

2. Background

2.1. Problem Statment

The packet classification refers to a network-layer activity in which rules are used to divide packets into classes according to the contents of one or more header fields and then a specified action related to that class is performed. In 5-tuple packet classification, an IP packet is classified based on the 5 fields in the packet header: the 32-bit source/destination IP addresses (to forbid or provide different service to source networks), 16-bit source/destination port numbers (to discriminate between traffic types), and 8-bit transport layer protocol (to distinguish between say externally and internally initiated connections) [13]. Each field in a rule is allowed three kinds of matches: prefix match, range match and exact match [14]. In a prefix match, the header field should be a prefix of the header field. This is useful for source/destination IP addresses field. In a range match, the header field should be a range specified of the header field. Range match is useful for source/destination port numbers field. In an exact match, the header field of the packet should exactly match the rule field. This is useful for protocol field. table 1 shows an example of the 5-tuple rule-set. A packet is considered matching a rule only if it matches all the fields within that rule. Each field of a rule can be specified as either a prefix or an exact number. The goal of a packet classification algorithm is to find the matching rule and output of the algorithm is the number of the matched rule. Packet classification is one of the most challenging functions in next-generation routers since it involves a multi-dimensional search that should be performed at wire-speed.

2.2. Packet Classification Algorithms

Packet classification can be viewed as extension from 5-tuple packet classification whose solutions have been extensively studied. Comprehensive surveys can be found in [15] and [16]. Most of those algorithms fall into two categories: decomposition-based and decision-tree-based approaches. Decomposition-based algorithms desirable for hardware implementation, due to their parallel search on multiple fields. However, these algorithms have poor scalability and work well only for small-scale rule sets. In these methods, the multiple field searches is decomposed into instances of single field searches, then independent searches are performed on each packet field and the results are combined. Examples are bit-vector scheme or Parallel Bit-Vectors (BV) [21], Aggregated Bit-Vector (ABV) algorithm [22], Recursive Flow Classification (RFC) [23], Parallel Packet Classification (P^2C) [24] and Distributed Crossproducting of Field Labels (DCFL) [16]. Decision-tree-based algorithms construct a decision tree from the rules in the rule-set and use the packet fields to traverse the decision tree [16]. Decision-tree-based algorithms have scale better than decomposition-based algorithms, and allow incremental rule updates. Some of these algorithms are Grid-of-Tries and Crossproducting algorithms [14], Extended Grid-of-Tries (EGT) that supports multiple fields searches without the need for many instances of the data structure [13], Hierarchical Intelligent Cuttings (HiCuts) [17], Modular Packet Classification [18], HyperCuts algorithm [19] and Fat Inverted Segment (FIS) Trees [20]. The outstanding representative of decision-tree-based packet classification algorithms are pipeline-based architectures. Pipeline-based decision tree is one of the best choices for increasing throughput on packet classification. Pipeline architecture is suitable for time level parallelism when all instruction independence from and does not exist hazard. IP packets entering the network are inherently serial and execute instruction the same. Hence pipeline is perfect solution for speed processing packet header and can be to achieve a high throughput of one packet per clock cycle. Dependent on the number of the stages and processing time any stage of the pipeline can provide high throughput. Most of the pipeline schemes are based on ASIC or FPGA.

3. Related Work

3.1. FPGA Designs

The since the little work has been done on FPGA. Most implementation of packet classification engine on FPGA is based decomposition-based packet classification algorithm. Song et al. [30] presented an architecture called BV-TCAM

for multi-match packet classification. This architecture combined TCAMs and the BV algorithm. It predicted the design after pipelining can achieve 10 Gbps throughput when implemented on advanced FPGAs also claims that the whole circuit for 222 rules consumes less than 10% of the available logic and fewer than 20% of the available Block RAMs of a Xilinx XCV2000E FPGA. Jedhe et al. [5] presented the DCFL architecture in firewall implementation on a Xilinx Virtex 2 Pro FPGA. It used a memory intensive approach so that on-the-fly update is feasible. They achieved a throughput of 50 million packets per second, for a rule-set of 128 entries. They also predicted the throughput can be 24 Gbps when the design is implemented on Virtex-5 FPGAs. Papaefstathiou et al. [8] proposed a memory-efficient decomposition-based packet classification algorithm. It used multi-level Bloom filters to combine the search results from all fields. Their FPGA implementation shows that the design can takes 26 clock cycles on average to classify a packet. In this architecture throughput is 1.875 Gbps on average. Implementation [8] on FPGA called 2sBFCE [9]. Kennedy et al. [29] implemented a simplified HyperCuts algorithm on an Altera Cyclone 3 FPGA. It discussed several issues in implementing decision-tree-based packet classification algorithms on FPGA. Implementation shows that the design can takes 23 clock cycles on average to classify a packet. In this architecture throughput is 0.47 Gbps on average.

3.2. Pipeline Architecture

A simple way to pipeline a decision tree is to assign each decision tree level to a distinct stage so that a rule can be issued every cycle. However, this simple pipeline scheme results in unbalanced memory distribution, leading to low throughput, inefficient source allocation and non-liner pipeline.

Implementation of unbalanced decision tree leads to different distribution nodes on the several stages of pipeline, as the result various access time. It also memory overflow occurs for some stages of the pipeline when rules are updated. Hence it becomes a bottleneck pipeline stages and the overall performance is degraded. Solutions for the problem caused by unbalanced pipeline have been proposed to achieve a balanced distribution of the memory. However, these methods are non-linear structure of the pipeline and reduced pipeline throughput and various delays of different stages. Baboescu et al. [25] proposed a new circular pipeline architecture to minimize memory cost and maximize throughput. The pipeline stages are configured in a circular, multi-point access pipeline so that lookup requests can be initiated at any stage. In other words, every stage can be used as the starting stages of any packets. The approach is starting by split trie data structure into several small subtries of equal size. These subtries are then mapped to different stages to create a balanced pipeline. Each stage accommodates two data paths. The first data path is active during the odd clock cycles and it is used for the IP lookup request's first traversal of the pipeline. The second data path is active during the even clock cycles, it will allow the request to continue its execution in the pipeline until it is finished. Therefore, the throughput of the design is 0.5 lookups per clock. In [26], Kumar proposed another pipeline with a new architecture called Circular, Adaptive and Monotonic Pipeline (CAMP). The scheme aims on develop a balanced pipeline. Unlike the Ring pipeline, CAMP provides multiple entry stages and multiple exiting stages. To partition the trie it uses several initial bits as the hashing index to get the hash value to decide which particular stages as its entering stage. Different packets as input stream may have different entry and exit stages, since to manage the access conflict several request queues are employed. CAMP is circular, thus all neighboring stages are connected in the same direction. The throughput of the design is 0.8 lookups per clock in the worst case.

4. Motivation

Due to the rapid growth of the Internet traffic, as well as the rule-set size and the number of overlapping rules, the complexity of the rule-set increases exponentially. FPGA can handle flexible classifiers and rule specification but due to limited available resource can not support larger rule-sets. Hence, we propose an optimized rule-set partitioning algorithm to reduce the complexity of the rule-set and the resource requirement.

4.1. Rule-Set Partitioning

To speed up the search and reduce the complexity of the architecture implemented, the rule-set can be partitioned into the separately subsets. An appropriate partitioned for set of rules can be reduced the time to update, eliminated duplicated rules and make the search process parallel on different subsets. Both power efficiency and throughput improvement can be obtained by such partitioning and parallelization. The main idea is to have a regular structure and flexible architecture so this way we found a high performance hardware implementation.

Table 1. Example 5-tuple rule set

Rules	SA	DA	AP	DP	Protocol
R1	64.91.106.0/24	168.104.0.0/16	0 : 65535	0 : 65535	0x06/0xFF
R2	64.91.107.58/32	64.91.108.0/22	0 : 65535	1734 : 1521	0x06/0xFF
R3	64.91.107.60/32	251.226.233.0/24	0 : 65535	0 : 65535	0x00/0x00
R4	192.151.11.0/24	0.0.0.0/5	0 : 65535	5001 : 65535	0x00/0x00
R5	192.151.11.127/32	208.184.188.39/32	0 : 65535	1221 : 1221	0x00/0x00
R6	95.105.143.38/32	15.0.0.0/8	0 : 65535	0 : 65535	0x00/0x00
R7	95.105.143.56/32	175.85.68.245/32	0 : 65535	1521 : 1521	0x06/0xFF
R8	64.91.106.0/24	136.0.0.0/6	0 : 65535	0 : 65535	0x00/0x00
R9	192.151.11.54/32	0.0.0.0/5	0 : 65535	0 : 1649	0x06/0xFF
R10	192.151.11.41/32	15.0.120.4/32	0 : 65535	1521 : 1521	0x06/0xFF

Rules	SA	DA	AP	DP	Protocol	Subset
R1	64.91.106.0/24	168.104.0.0/16	0 : 65535	0 : 65535	0x06/0xFF	[1]
R2	64.91.107.58/32	64.91.108.0/22	0 : 65535	1734 : 1521	0x06/0xFF	[2]
R3	64.91.107.60/32	251.226.233.0/24	0 : 65535	0 : 65535	0x00/0x00	[2]
R4	192.151.11.0/24	0.0.0.0/5	0 : 65535	5001 : 65535	0x00/0x00	[3]
R5	192.151.11.127/32	208.184.188.39/32	0 : 65535	1221 : 1221	0x00/0x00	[3]
R6	95.105.143.38/32	15.0.0.0/8	0 : 65535	0 : 65535	0x00/0x00	[4]
R7	95.105.143.56/32	175.85.68.245/32	0 : 65535	1521 : 1521	0x06/0xFF	[4]
R8	64.91.106.0/24	136.0.0.0/6	0 : 65535	0 : 65535	0x00/0x00	[1]
R9	192.151.11.54/32	0.0.0.0/5	0 : 65535	0 : 1649	0x06/0xFF	[3]
R10	192.151.11.41/32	15.0.120.4/32	0 : 65535	1521 : 1521	0x06/0xFF	[3]

Prefix SA	Subset	Rules
64.91.106.*	[1]	R1, R8
64.91.107.*	[2]	R2, R3
192.151.11.*	[3]	R4, R5, R9, R10
95.105.143.*	[4]	R6, R7

Figure1. An example of the rule-set partitioning of table 1

Motivated by the observation that the number of unique fields in the rule-set is much smaller than all rules, the division can be made based on one or more fields of rules. Those rules which have greater overlap with each other in a subset are identical. Then for each of these subsets, a decision tree is implemented separately. For example, in a rule-set including hundred rules may be forty-seven unique destination prefix address and four source prefix address. Partitioning is based on the one of these states (destination/source address field). Selection of a state for partitioning is depended on the lowest number of unique states of the rule-set. In this technique, all rules of the rule-set that have the same source prefix address are a subset. Hence, the rules of rule-set are divided into k subsets and then created a separate subtree for each subset. An example of rule-set partitioning of table 1 is shown in figure 1. Given these definitions of rule-set partitioning, we list the states of each rule in table 1 in the last column. The rule-set is divided into four subsets. This partitioning is based on the longest prefix matching on the source address. For example, the source address prefix of all rules that are 192.151.11. *, must be placed in subset 3. Hence the rules {R4, R5, R9, R10} are the subsets 3. Each subset is defined by several rules that they are implemented with a pipeline-based decision tree. Each field of a rule is processed in each stage of the pipeline. Because the processing is performed separately for each field in each stage and is independent of the overall architecture, therefore, the best architecture for the processing of each field can be implemented. Note that to have linear pipeline architecture each stage must have equal delay. Partitioning rule set based on this technique causes the overall architecture has regular structure. In last proposed technique, when a rule will be added or removed to architecture, the overall structure of architecture must be reconfigured. In this technique, rules are partitioned to small subsets based on specific criteria until both easier to find location of the rules (for reduced the time to update) and with change of the architectural structure is not required to reconfigure the entire structure. In our technique for adding or removing a rule, only a subset can be searched. In this method, without disturbing the operation of the rest of the architecture, rules can be updated. One of the main challenges of the packet classification engine is complexity of rules that with increasing number of the rules and overlapping rules grows exponentially. Overlapping rules in many classification algorithms such HyperCuts or HiCuts increase the consumable resources. To resolve this problem, some techniques have been proposed, but they will have a negative effect on performance. On the other words, complexity of rules is increasing with growth of the rule-set size. We presented a method to address these problems. In this method, overall structure is based on a special order which increasing the number of rules does not effect on the complexity of the architecture and it has flexibility for greater development. Moreover, whatever rules have a greater overlap so fewer resources are used to implement the architecture. The use of prefix rules will be effective in reducing the consumption of resources because does not require that each existing rules or each rules that are in a certain range to be implemented. Another feature of this technique is to remove duplicate rules and overhead of the search algorithm in the final stage. In many classification algorithms based on decision tree, the end nodes of a decision tree are encountered with several rules. This is because to best matching the rules used of search algorithms such as linear search, CAM or binary search. In this technique, the end nodes of a decision tree are consistent with the single rule and no need to additional algorithms for selecting the best rule among of several rules.

4.2. Minimize the Hardware Implementation

In packet classification each rule includes several fields that based on these fields will be determined what action should be done. Thus, if incoming packet is the match with a rule in the rule-set, this rule determines what action should be done on the packet. When we design an architecture for packet classification, in fact fields of the rules are implemented.

Decision tree-based algorithms have good scalability. They are suitable for rule-sets where the rules have little overlap with each other but the duplicate rules will cause an inefficient utilization of hardware resources. Rules overlapping can be cause rule duplication in decision tree algorithms. For example, consider field DP and SP in table 1

As can be observed source and destination port fields of the rules R1, R3, R6 and R8 are defined for all states. In the packet classification tables, these cases are called “don’t care” and often marked with * indicate.

Don’t care states are due to rules that have been saved in prefix form and their values are not specified. The presence or absence of the don’t care fields have not effect in the how decision of the rules. These fields are considered as an overhead implemented in the hardware architectures based on decision tree. Therefore, there is no need to implement. Hence, this technique can dramatically reduce the amount of hardware overhead. For resource saving in the hardware implementation, idea is that eliminating duplicate rules and ignoring don’t care fields. We used rule-set partitioning technique, described in the previous section, for eliminating duplicated rule.

In this technique, exist duplicated rules in the implementation are completely eliminated. For example, figure 2 indicates an example of eliminating don’t care fields of the rules to reduce consumption of resources. This decision tree is state 1 of figure 1 to implement rules R1 and R8. Tree is composed of two sub-branches that each is to matching intended rule. Note that eliminating of the fields of the third and fourth levels of the tree do not effect on the overall function because they are full range. Hence, during the implementation of the architecture, they are only being consuming resources which are considered overhead. By eliminating these fields, height of the tree is less and reduces the resources consumption. The less height of the tree is lead to increased speed of the rules matching.

5. Our Proposed Architecture

5.1. Overview

Due to the serial transmission of the packets in the network links, implementation of the pipeline architecture is one of the best choices for packet classification engines. Pipeline is one of the best architecture for serial processing that provides parallelism at the level of the time. Therefore, to achieve high throughput, we map N decision tree onto a parallel multi-pipeline architecture with N linear pipelines, as depicted in figure 3, where N=2. This architecture is based on the decision sub-trees of the rule-sets that the number of linear pipelines is equal to the number of sub-trees.

Each pipeline is used for traversing a decision tree as well as matching the rule attached to the leaf nodes of that subtree. Architecture is five stages that each stage includes one or several specific Processing Units (PUs). One field of the header packet have been processing in the one of the stages of the pipeline so that to be decided about it according to value of each field.

After processing of fields in each stage of the pipeline information is sent to the next stage that this information is called the Code-ID. Code-ID for each stage is used as an activator of the PUs. At the end of traversal tree to reach leaf nodes along the pipeline, one of the rules for incoming packet will be mached. Each incoming packet goes through all the N pipelines in parallel and each output of pipeline is a rule-ID or its corresponding action. In different pipelines used a different subset of header fields of the packet. According to the pipeline structure after 5 clock (number of pipeline stages) will match one rule per clock.

5.2. Processing Units

In the proposed architecture, as a decision-tree-based architecture, the child nodes of the tree can be traversed in order during the pipeline. Finally, the best rule is matching in the end stages. Depending on the architecture implementing the number of stages of pipeline can be different for used. . In the decision tree algorithms depth of a decision tree can be as large as O(W),where W denotes the total number of bits per packet for lookup and W>104 in 5-Tuple. Although the throughput can be boosted by using a deeper pipeline but the large delay passing the packet classification engine requires the router to use a large buffer to store the payload of all packets being classified.

The proposed architecture is a parallel architecture with five stages for routing of local routes. This architecture can be seen in figure 3. The internal structure of each stage includes one or more PU and a set of control signals that are responsible to matching one of the fields in the packet header. One of the problems of SRAM-based multi pipeline architectures is memory overflow problem in different stages. In this architecture, the rule-set is implemented in memory blocks that should be preconfigured.

But if at the time of rules update, the memory blocks do not have enough space to store the new rules then has occurred memory overflow for this architecture. Moreover, it is possible that memory blocks in different stages have various sizes and result in each stage will have a different delay. In this case, pipeline is out of the linear state.

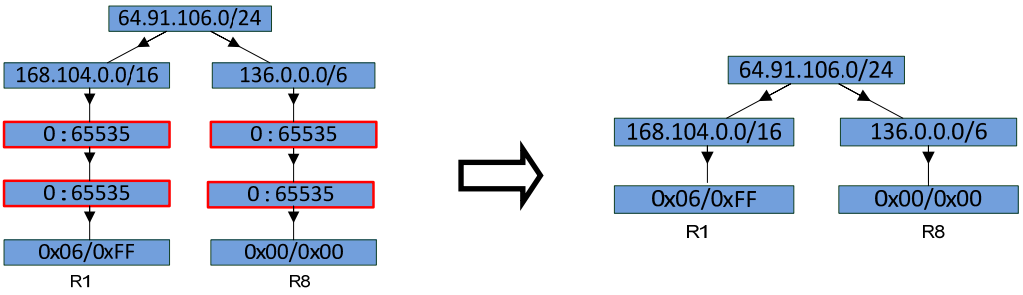

Figure 2. Eliminate the indeterminate field and reduce duplication rules

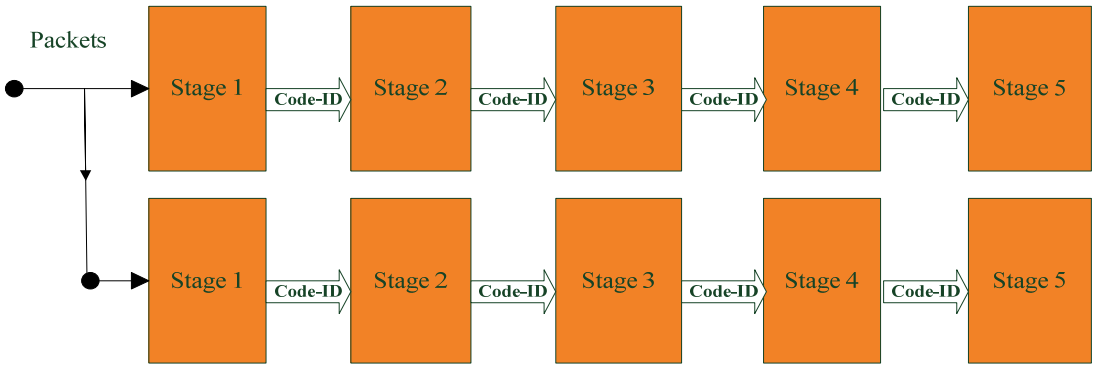

Figure 3. Parallel multi-pipeline architecture with N=2 linear pipelines

We used a different structure to solve the problem of memory overflow. The proposed architecture, memory blocks have been removed and replaced Look-Up tables (LUTs). A LUT is a small one bit wide memory array, where the address lines for the memory are inputs of the logic block and the one bit output from the memory is the LUT output. A LUT with K inputs would then correspond to a $2^K \times 1$ bit memory, and can realize any logic function of its K inputs by programming the logic function's truth table directly into the memory. In our architecture, the rules of the rule set are located in processing units (PU). Each stage of pipeline includes several PU which in the anytime only one PU is active and will process one the fields of packet header as depicted in figure 4.

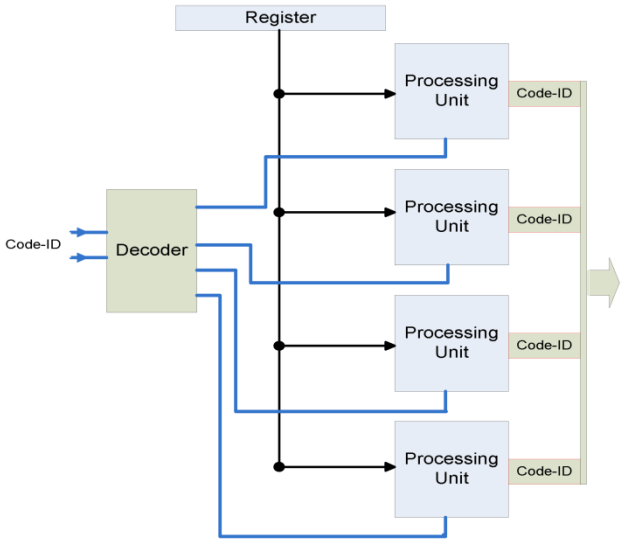

Figure 4. Example of a stage of the pipeline, including several processing units

This figure shows the internal structure of a stage of pipeline that consists of the four PU. Each stage after decision for packet will send a Code-ID for the next stage of pipeline. This Code-ID is inputs of a decoder that depend on this code one of its output are activated.

The PUs includes an activator input, input field of the header packet and an output Code-ID. Activator input is one of the outputs of decoder. With activation of this input line, PU becomes active until comparison and decisions be done. The internal structure of PU consists of multiple-bit comparator. Depend on input field of the header packet, of through the register a field is entering in the stage and then it will be processed by multiple-bit comparator of the PUs. After the process a Code-ID is produced and then sends to the decoder of next stage of pipeline by Code-ID output line of PU. This process is continued until the reaches the latest stage of pipeline. Finally the best rule is matching in the last stage.

In this architecture, each stage of the pipeline is conformable from the generated code of the previous stage. figure 5 is an example of table 1 that shows the implementation of the proposed architecture. As can be observed, there are several PUs on each stages that to processing its domain of incoming packets.

In the table 1 four states are defined that here as the four lines of pipeline have been implemented. The line 4 of the pipeline, is shown in red color, no processing units has been implemented in the third stage because of all state are don't care and directed leading to latest stage of pipeline. Furthermore, because the one of the fields of the all rules is don't care state, pipeline architecture is four stages. In the general case, straight parts of the architecture are represents of don't care state. For these states not considered PU for implementing. For example, suppose a packet including the header (64.91.107.29, 251.226.233.94, 1365, 0x00) to be entered. First, this is entering the all lines of pipeline but only line 4 of pipeline can be provides the processing and generating Code-ID for next stage. After passing through the first stage, the Code-ID will through decoder to active second PU of the second stage. In the third stage because existed don't care state, this field with any value will not be processed. In addition Code-ID is so that does not cause any interference with other PUs and they will not active. Finally, R3 is match to the last stage. PUs according to the rules partitioning technique are simple to implement.

Pseudo VHDL code related to a PU is shown below. These units are based on the Code-ID that will be issued of each line of the previous stage of pipeline.

Input: Field-x of the packet
Input: Code-ID_X (Previous Stage)
Output: Code-ID_Y (Next Stage)

```
CASECode-ID_X IS
        WHEN "000" => IF (Field-x = "*..*") THEN
                        Code-ID_Y <= "..";
                      Else
                        Code-ID_Y <= "ZZ";
                      END IF;
        WHEN "001" => IF (Field-x = "*...*") THEN
```

```
                    Code-ID_Y  <= "..";
                    ELSE
                    Code-ID_Y  <= "ZZ";
                    END IF;
                    .
                    .
          WHEN OTHERS  => Code-ID_Y  <= "ZZ";
End CASE;
```

The motivation for using this method, the code activated processing units, reduce delays in processing time and speed up the update. Actually when the numbers of rules that must be implemented in a stage are high, the processing time will increase compared to the other stages. But if the rules become distributed in several separate units and at any given time only a particular unit is active, delays in processing time will be limited to those units. Moreover, if the rules be added or removed only a unit is retrieving that is included relevant rules. In addition, there will be no overflow. The our proposed architecture is compared with other hardware approaches such as TCAM greater development capability, low power consumption, less space and consume resources and is consistent with prefix rules. In compare with hashing techniques such as Bloom filter have determined performance and throughput higher. In compare with geometric methods (than with rules that are greater overlapping) such as HiCuts and HyperCuts have better functionality, lower resource consumption and higher processing speed (due to less complexity).

5.3. Rule Update

One of the most important criteria in the proposed architecture is capable to the updated. Due to the dynamic structure of the network applications and rules, it is very important to consider the update time because may be architecture needs to be modified frequently. Architecture must be capable to adding or removing one or more rules without disturbing of the overall architecture and the network. Hence, we consider this feature in our architecture. As seen in figure 3, different internal nodes in a decision tree are implemented in the pipeline with the same structures. These nodes are connected with each other through a simple wired. The PUs that makes up the main part of each node are updateable and can be easily added or removed. Unlike other type of pipeline which have problems of memory overflow and update, in the proposed architecture have not these problems and can simply to adding new nodes to the decision tree with adding PUs. Systems implemented with FPGAs can make use of their reprogram ability in one of two ways: compile-time reconfiguration or run-time reconfiguration. Compile-time reconfiguration systems do not change the FPGA's configuration for the life-time of the application. Run-time reconfiguration systems change the FPGA configuration during the course of operation, either by full reconfiguration or partial reconfiguration. Tools and a design methodology have been developed to support partial run-time reconfiguration of FPGA logic on the Field Programmable Port Extender. The present architecture uses partial run-time reconfiguration in an FPGA to provide dynamically developers of hardware. Partial reconfiguration allows an FPGA to implement multiple functions and to change those functions while the system is running. With the implementation of the architecture on the FPGA can use this feature so that new rules can be updated to simply while is to running.

6. Experimental Results

To evaluate the performance of our scheme for 5-tuple packet classification problem, conducted experiments result. We used the rule sets from [27], which generated using Class Bench [33]. These rule-sets extracted from real-life rules and the size varies from hundreds of to tens of thousands of rules. To evaluate the scheme, used following performance metrics:

Average look up table (LUT) requirement per rule-set:

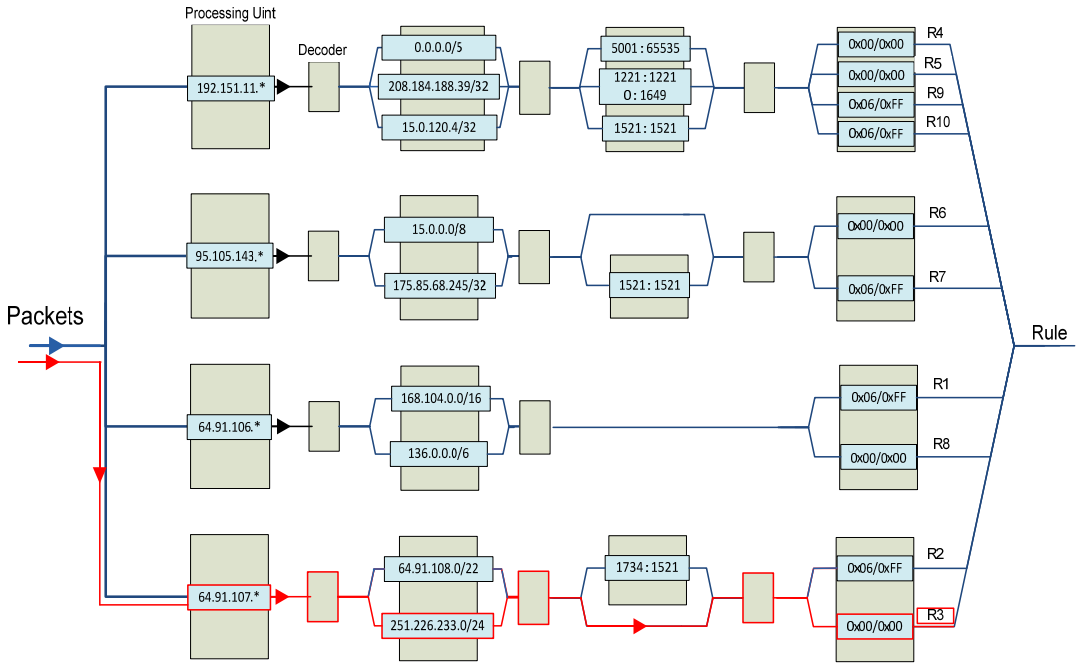

Figure 5. An example of table 1 that shows the implementation of the proposed architecture

Set: area integration is the quantity of consumed FPGA logic resources, which is measured by the consumed lookup tables (LUT). It is the scalability of our scheme and computed as the total LUT requirement of rules for building the tree. Evaluation criteria of the pipeline architectures which so far proposed have been based on memory blocks consumption. In our architecture, to address the challenges of the recent architecture, we unused memory blocks. To evaluate the hardware overhead imposed by applying this method, we applied proposed method to ISE 14.2 environment and synthesized it. Table 2 shows the average LUT requirement for different the number of rule sets.

Table 2. The average LUT requirement whit increasing size of rule-sets

Rule set	# of rules	LUT	Available	Utilization %
ACL_100	98	464	354,240	0/1
ACL_1K	916	4786	354,240	1/35
ACL_5K	4415	21830	354,240	6/16
ACL_10K	9603	47242	354,240	13/33

The fourth column of the table is the available total number of LUT and in the last column of the table shows the percentage of these resources. The results show that over ten thousand rules can be implemented in the FPGA.

The depth of the decision tree: The depth of the decision tree is depending on both the size of the rule-set and the characteristics of the rule set. In the many decision tree-based architectures, depth of tree is indeterminate. For example, in Hicuts and HyperCuts algorithms with the increasing number rules of the rule-set, the complexity and processing time at each stages increases. Hence, in the rule-set with different sizes, depth of the tree is varies.

Table 3. Pipeline depth for rule sets of various sizes

		Tree depth		
Rule set	# of rules	Our algorithm	HyperCuts[25]	HiCuts[28]
ACL_100	98	5	23	8
ACL_1K	916	5	20	11
ACL_5K	4415	5	29	12
ACL_10K	9603	5	29	9

In our architecture, pipeline depth is fixed and with change of the rule-set size does not change. According to table 3, our pipeline needed at 5 stages to map the decision tree for all rule-sets.

Implementation results: We implemented our design on FPGA using Xilinx ISE 14.2 development tools. VHDL language is used to describe our design. The target device was Virtex-6 XC6VHX565T with -2 speed grade. Post place and route results showed that our design achieved a clock frequency of 384 MHz. The resulting throughput was 123.07 Gbps for minimum size (40 bytes) packets in the worst case. The FPGA design supports Over 10K 5-tuple rules. Table 4 compares our design with the state-of-the-art FPGA-based packet classification engines. Based on this table, it can be observed that the proposed approach can achieve 123.07 Gbps throughput for a single engine. It is clear that with increasing the number of PUs, higher throughput is achieved.

Table 4. Performance comparison

Packet classification engine	Platform	# of rules	Throughput (Gbps)
Our design	FPGA	9603	123.07
Hicuts [28]	FPGA	9603	80.23
HyperCuts [29]	FPGA	10000	10.84
BV-TCAM [30]	FPGA	222	10
2sBFCF [9]	FPGA	4000	2.06
Memory-based DCFL [5]	FPGA	128	24

Average number of pipeline lines: It is very important witch consider tradeoff between complexity, area integration and rule match delay. The easiest method is that all rules to be implemented on a separate line of the pipeline. Also, if all rules can be implemented in just one line pipeline then area integration will be reduced. Moreover, complexity and matching delay will be increases. To optimize the implementation and tradeoff between the parameters, we used of the rule-set partitioning technique. We compute unique state of SA and DA fields in the different rule-set. We have considered the number of lines of pipeline based on unique state of SA field because the number of states is less and there is greater overlap between rules. As a result, amount of consumed hardware and the number of line of pipelines is reduced. The table 5 shows unique states of the SA and DA fields with different sizes of rule-set. The last column shows the number lines of the pipeline.

Table 5. Average number of pipeline lines

Rule set	# of rules	# of unique SA	# of unique DA	# of linear pipelines
ACL_100	98	3	53	3
ACL_1K	916	5	171	5
ACL_5K	4415	25	489	25
ACL_10K	9603	230	1700	230

7. Conclusion

In this paper leads us to design the high-throughput network processing tasks e.g packet classification, longest prefix matching using FPGA that can be inserted as a co-processor in the new routers. This means that each router including some reconfigurable modules for the network processing tasks which need performance, flexibility and the possibility of replacement to new task. This paper presented a new decision-tree-based linear multi-pipeline architecture on FPGAs. We considered the packet classification problems that 5-tuple packet header fields would be classified. Optimization techniques were proposed to improve throughput degradation, delay variation and memory overflow of the state-of- the-art decision-tree based packet classification algorithm. In this architecture processing units

used instead of memory blocks. Partial reconfiguration in FPGA used to reduce the time that is needed to change the behavior of architecture.

References

[1] W. Jiang, and V. K. Prasanna, "Large-scale Wire-speed Packet Classification on FPGAs," *Proc. IEEE Intl Conf. Field-Programmable Gate Array*, pp. 219-228, 2009.

[2] F. Yu, R. H. Katz, and T. V. Lakshman, "Efficient Multimatch Packet Classification and Lookup with TCAM," *IEEE Trans. Microarchitecture*, vol. 25, no. 1, pp. 50-59, 2005.

[3] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary, "Algorithms for Advanced Packet Classification with Ternary CAMs," *Proc. IEEE Intl Conf. ACM Special Interest Group on Data Communication*, pp. 193-204, 2005.

[4] W. Jiang, and V. K. Prasanna, "Sequence-preserving Parallel IP Lookup Using Multiple SRAM-based Pipelines," *Journal of Parallel and Distributed Computing*, vol. 69, no. 9, pp. 778-789, 2009.

[5] G. S. Jedhe, A. Ramamoorthy, and K. Varghese, "A Scalable High Throughput Firewall in FPGA," *Proc. IEEE Intl Conf. Field-Programmable Custom Computing Machines*, pp. 43-52, 2008.

[6] Y. Qi, J. Fong, W. Jiang, X. Bo, J. Li, and V. Prasanna, "Multi-dimensional Packet Classification on FPGA: 100Gbps and Beyond," *Proc. IEEE Intl Conf. Field-Programmable Technology*, pp. 56-64, 2010.

[7] S. Dharmapurikar, H. Song, J. S. Turner, and J. W. Lockwood, "Fast Packet Classification Using Bloom Filters," *Proc. IEEE Intl Conf. Architecture for Networking and Communications Systems*, pp. 61-70, 2006.

[8] A. Nikitakis, and I. Papaefstathiou, "A Memory-efficient FPGA-based Classification Engine," *Proc. IEEE Intl Conf. Field-Programmable Custom Computing Machines*, pp. 53-62, 2008.

[9] A. G. AlaguPriya, and H. Lim, "Hierarchical Packet Classification Using a Bloom Filter and Rule-priority Tries, " *IEEE Journal on Computer and Communications*, vol. 33, no. 10, pp. 1215-1226, 2010.

[10] H. Yu, and R. Mahapatra, "A Power and Throughput-Efficient Packet Classifier with n Bloom Filters," *IEEE Trans. Comput.*, vol. 60, no.8, pp. 1182-1193, 2011.

[11] Xilinx Virtex-6 FPGA Family, www.xilinx.com/products/virtex6/, May 2009.

[12] Altera Stratix IV FPGA, http://www.altera.com/products/devices/stratix_fpgas/stratix/, May 2009.

[13] F. Baboescu, S. Singh, and G. Varghese, "Packet Classification for Core Routers: Is There an Alternative to CAMs?," *Proc. IEEE Intl Conf. Computer and Communication*, pp. 22-32, 2003.

[14] V. Srinivasan, S. Suri, G. Varghese, and M. Waldvogel, "Fast and Scalable Layer Four Switching," *Proc. IEEE Intl Conf. ACM Special Interest Group on Data Communication*, pp. 203-214, 1998.

[15] P. Gupta, and N. McKeown, "Algorithms for Packet Classification," *IEEE Trans. Network*, vol. 15, no. 2, pp. 24-32, 2001.

[16] D. E. Taylor, "Survey and Taxonomy of Packet Classification Techniques," *Journal of ACM Computing Surveys*, vol. 37, no. 3, pp. 238-275, 2005.

[17] P. Gupta, and N. McKeown, "Packet Classification Using Hierarchical Intelligent Cuttings," *Proc. IEEE Intl Conf. Hot Interconnects*, pp. 34-41, 1999.

[18] T. Y. C. Woo, "A Modular Approach to Packet Classification: Algorithms and Results," *Proc. IEEE Intl Conf. Computer and Communication*, pp. 32-44, 2000.

[19] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet Classification Using Multidimensional Cutting," *Proc. IEEE Intl Conf. ACM Special Interest Group on Data Communication*, pp. 63-71, 2003.

[20] A. Feldmann, and S. Muthukrishnan, "Tradeoffs for Packet Classification," *Proc. IEEE Intl Conf. Computer and Communication*, pp. 43-51, 2000.

[21] T. V. Lakshman, and D. Stiliadis, "High-Speed Policy-based Packet Forwarding Using Efficient Multi dimensional Range Matching," *Proc. IEEE Intl Conf. ACM Special Interest Group on Data Communication*, pp. 190-202, 1998.

[22] F. Baboescu, and G. Varghese, "Scalable Packet Classification," *Proc. IEEE Intl Conf. ACM Special Interest Group on Data Communication*, pp. 193-204, 2001.

[23] P. Gupta, and N. McKeown, "Packet Classification on Multiple Fields," *Proc. IEEE Intl Conf. ACM Special Interest Group on Data Communication*, pp. 500-512, 1999.

[24] J. Van Lunteren, and T. Engbersen, "Fast and Scalable Packet Classification," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 2, pp. 560-571, 2003.

[25] F. Baboescu, D. M. Tullsen, G. Rosu, and S. Singh, "A Tree-based Router Search Engine Architecture with Single Port Memories," *Proc. IEEE Intl Symp. on Computer Architecture*, pp. 123-133, 2005.

[26] S. Kumar, M. Becchi, P. Crowley, and J. Turner, "CAMP: Fast and Efficient IP Lookup Architecture," *Proc. IEEE Intl Symp. Architecture for Networking and Communications Systems*, pp. 51-60, 2006.

[27] Packet Classification Filter Sets, http://www.arl.wustl.edu/~hs1/pclasseval.html, May 2009.

[28] W. Jiang, and V. K. Prasanna, "Scalable Packet Classification on FPGA," *IEEE Trans. Very Large Scale Integration Systems*, vol. 20, no. 9, pp. 1668-1680, 2012.

[29] A. Kennedy, X. Wang, Z. Liu, and B. Liu, "Low Power Architecture for High Speed Packet Classification," *Proc. IEEE Intl Conf. Architecture for Networking and Communications Systems*, pp. 131-140, 2008.

[30] H. Song, and J. W. Lockwood, "Efficient Packet Classification for Network Intrusion Detection Using FPGA," *Proc. IEEE Intl Conf. Field-Programmable Gate Array*, pp. 238-245, 2005.

Rashid Hatami received the B.S. degree in Computer engineering From Islamic Azad University of Dezful, Dezful, Khuzestan, Iran in 2010. He received the M.Sc. degrees in Computer architecture and engineering from Razi University, Kermanshah, Iran in 2013. His research interests include Computer architecture, high-performance architectures in network router and Digital Systems on FPGA. His primary focuses are on custom hardware, memory-efficient data structures, and high-speed architectures.
E-mail: r.hatami@pgs.razi.ac.ir

Mahmood Ahmadi received the B.S. degree in Computer Engineering from Isfahan University, Isfahan, Iran in 1995. He received the M.Sc. degrees in Computer architecture and engineering from Tehran Poly technique University, Tehran, Iran in 1998.From 1999 to 2005, he was a faculty member at Razi university in Kermanshah in Iran. In October 2005, he joined the Faculty of Electrical Engineering, Mathematics, and Computer Science (EEMCS), Delft University of Technology, Delft, The Netherlands, as a full-time Ph.D. student. He got his PhD in May 2010. Currently, he is working as assistant professor at Computer Engineering Department of Razi University of Kermanshah in Iran. He is member of IEEE and HIPEAC.
His research interests include Computer architecture, network processing, Bloom filters, performance modeling and reconfigurable computing.
E-mail: m.ahmadi@razi.ac.ir

Paper Handling Data:

Submitted: 29.10.2013
Received in revised form: 06.04.2014
Accepted: 10.05.2014
Corresponding author: Dr. Mahmood Ahmadi,
Department of Computer Engineering, Razi University,
Kermanshah, Iran.