Feature Study on a Programmable Network Traffic Classifier

Guerra Perez, K., Yang, X., Scott-Hayward, S., & Sezer, S. (2017). Feature Study on a Programmable Network Traffic Classifier. In 2016 29th IEEE International System-on-Chip Conference (SOCC) (pp. 108-113). (International SOC Conference (SOCC). Proceedings). DOI: 10.1109/SOCC.2016.7905446

Published in:
2016 29th IEEE International System-on-Chip Conference (SOCC)

Document Version:
Peer reviewed version

Queen's University Belfast - Research Portal:
Link to publication record in Queen's University Belfast Research Portal

Feature Study on a Programmable Network Traffic Classifier

Keissy Guerra Pérez, Xin Yang, Sandra Scott-Hayward, Sakir Sezer
ECIT Institute, Queen's University Belfast, Northern Ireland

***Abstract*—Monitoring and tracking of IP traffic flows are essential for network services (i.e. packet forwarding). Packet header lookup is the main part of flow identification by determining the predefined matching action for each incoming flow. In this paper, an improved header lookup and flow rule update solution is investigated. A detailed study of several well-known lookup algorithms reveals that searching individual packet header field and combining the results achieve high lookup speed and flexibility. The proposed hybrid lookup architecture is comprised of various lookup algorithms, which are selected based on the user applications and system requirements.**

Keywords—packet classification; multi-dimensional lookup; TCAM

I. INTRODUCTION

Network traffic classification plays an important role in network nodes to support packet- and flow- processing functions, such as IP forwarding, packet filtering, security policies, quality of services, etc. Classifying input packets into flows and searching against pre-defined rules is a fundamental classification technique. An action is associated to each flow entry based on the same header fields and is determined by the first matching rule or the Highest-Priority Matching Rule (HPMR).

There are several challenges in the classification process. Firstly, the entries in the routing tables change dynamically according to the state of the network and the information exchanged by routing protocols. It requires a solution with incremental update capability. Secondly, as the network link rate has rapidly increased, packet classification has become one of the fundamental challenges of high-speed network processing. Thirdly, working with IPv6 is becoming increasingly vital. IPv6 header varies in the number and length of the fields in comparison of its IPv4 counterpart. Consequently, for a fast adaptation between protocols, the adopted algorithms must be able to migrate to IPv6-based applications. Finally, the growth of the number of entries in the forwarding tables drives the need of scalability for large classifiers. Hence, efficient storage is a key for large classifier and the rule syntax should offer advantages to distribute rules into convenient groups.

In general, different classification methods can be characterised by a set of performance metrics. These include parameters such as speed of classification, update time, storage requirement, the ability to support dynamic rule definitions, flexibility, complexity of rulesets, scalability, etc. Routers and switches have to work with different packet header fields making classification more complicated. A method with multi-dimensional handling or multiple methods that work with a single field is essential.

This paper aims to include the studies and research development on the programmable multi-dimensional lookup architecture proposed for high-performance packet header lookup operations in network systems.

II. BACKGROUND STUDIES AND MOTIVATION

In packet classification, a certain number of fields from the packet headers are used to find the matching action from a predefined flow set. Main performance requirements regarding packet classification are listed and explained below:

- *Search speed*: the router and switch operations must work at the link rate of 40 GbE – 100 GbE and over [1]. The search time of a lookup algorithm should be independent of the number of rules.
- *Storage requirement*: a large number of rules must be stored in data structures whose storage methodology can be complex. The redundancy of rule tables is a challenge in terms of classification time.
- *Incremental update*: The data structure has to be changed when a new rule is inserted, updated or deleted. A data structure which supports incremental update does not need to be reconstructed for inserting or deleting entries.
- *Scalability*: It is expected that the system provides the ability to handle a large numbers of rules. The system should be scalable in terms of length and number of header fields used for packet classification. Furthermore, the two internet protocols (IPv4 and IPv6) with different field lengths must be considered.
- *Flexibility in algorithm specification*: A classification algorithm should support general rules, including prefixes, operators (range, less than, greater than, equal to, etc.) and wildcards.

The most common rules are composed of five fields from a header: source and destination IP addresses, source and destination ports and protocol. These fields are compared against input packet headers in order to apply an action

specified by the matching rule, where different matching patterns are applied on different fields. For example, the IP address fields are represented by prefixes and masks; the Port fields are characterised by intervals; the Protocol field is represented by exact values. Accordingly, methods of matching are divided into three categories, prefix matching, range matching, and exact matching, respectively. In particular, Longest-Prefix Matching (LPM) for IP address fields refers to the method that selects an entry with the most matching bits in a table of defined prefixes. The most commonly used approaches for LPM are Tree-based algorithms, such as binary tree and multi-bit trie [2]. Range matching is well suited to port field lookup. Interval tree, segment tree, and radix tree [3] are typical tree structures for range matching. Exact matching is deployed on those fields, such as the Protocol field, which requires exact values to be searched. According to the length of the data value, a variety of fast search methods can be applied from direct indexing and content-addressable memory (CAM) for simple data lookup, to hash-based search on data with big values.

To perform full packet classifications, multi-dimensional lookup should be used in order to inspect many header fields. Table I summarizes the state-of-the-art multi-dimensional lookup algorithms and comparisons of their performance in terms of lookup speed, storage complexity and support of fast incremental update.

TABLE I: PERFORMANCE EVALUATION OF TYPICAL MULTI-DIMENSIONAL LOOKUP ALGORITHMS

Lookup Algorithm	Lookup Speed	Storage Complexity	Incremental Update
HiCuts [10]	O(d*W)	$O(N^d)$	No
HyperCuts [13]	O(N)	$O(N^2)$	No
RFC [4]	O(d)	$O(N^d)$	No
AM-Trie [7]	O(h+d)	$O(N^2)$	Yes
Cross-producting [5]	O(W*d)	$O(N^d)$	No
DCFL [9]	O(d)	O(d*N*W)	Yes
ABV [6]	$O(d*W+N/M^2)$	$O(N^2)$	No
TSS [15]	O(M+N)	$O(W^d)$	Yes
Bitmap-Intersection	O(W*d+N/s)	$O(d*N^2)$	No
TCAM	O (1)	O(N)	Yes

d: number of dimensions
N: number of Rules
h: trie height
M: size of the bitmap vector
W: largest number of field bits
s: memory width

Despite the advantages of Ternary Content Addressable Memory (TCAM) of performing parallel search at one time, this technique has disadvantages of high power consumption and expensive storage. Additionally, the comparison in TCAM is only applicable to prefix fields and exact matching fields. Those fields given by range or other syntax necessitate a prefix conversion. TCAM suffers from memory blow-up if each range is converted to a set of prefixes.

Another option is to decompose the problem into smaller problems. RFC [4], Cross-Producting [5], ABV [6], AM-Trie [7], PCIU [8] and DCFL [9] are some popular decomposition methods, which are based on single-header-field search. Individual fields are handled independently, which presents advantages regarding lookup speed and/or update complexity but requires more storage space. The individual search results are combined in order to reach the highest-priority matching rule. As shown in Table I, RFC offers lookup advantages and DCFL improves the memory usage required by RFC.

Alternatively, a large number of approaches have been presented by splitting a multi-dimensional search space into equal-sized ranges and place them into a Trie structure. Consequently, the rule sets are altered and duplicated into the trie nodes. HiCuts [10], HSM [11], HiPAC [12], HyperCuts [13] and ExpCuts [14] are falling into this category. In these approaches, large memory space is not required because of hierarchical distribution. However, the problem is that incremental update is not supported, which results in a complicated update process. HiCuts and HyperCuts are highlighted in TABLE I as the featured methods in multi-dimensional packet classification.

Hash-based solutions can also be used for multi-dimensional lookup. However, as wildcard lookup is not supported in hash tables, rulesets must be converted into exact values. In this case, memory space becomes very large. Furthermore, hashing methods suffer from collision problems, which must be mitigated by sacrificing memory space or lookup time. Therefore hashing is normally used as part of a procedure together with other lookup algorithms. TSS [15], for example, combines the trie structure for update and collects the leaf nodes into a group of tables, which are used for lookup by addressing the tables with a hash function.

Based on the above analysis, it is concluded that the optimal lookup speed and update time are achieved by the decomposition approach, while trie structures require a trade-off between memory space and search time. The decomposition approach achieves high lookup performance by sharing individual fields in parallel. Managing individual fields with efficient dedicated algorithms also provides flexibility in packet classification, as will be discussed in Section III. In Perez's work [16], configurable lookup architecture based on the decomposition approach was presented with promising results.

The main motivation of this work comes from the results in [16] and [17]. The study of different lookup approaches presented in this paper concludes that there is no unique algorithm which can handle five or more fields efficiently with respect to the three main criteria: high lookup speed, memory storage efficient and fast incremental update. This highlights that existing lookup algorithms are optimally applied for one single application. Hybrid lookup architecture is therefore a key requirement for next-generation network devices and for the programmable platform, with greater flexibility for multiple applications.

III. PROGRAMMABLE LOOKUP ARCHITECTURE

In this section, a programmable lookup solution is described. The proposed architecture provides a configurable

lookup algorithm set for optimal performance. There are several challenges that need to be considered in the proposed hybrid system. The partition of header fields and the distribution of the algorithm structures are critical for lookup performance.

It is significant to have classification algorithms supporting incremental update since the rules, and accordingly the labels [9], must be deleted, inserted or changed at run time. Furthermore, when the individual results are obtained, the combination between these results must be performed to reach the desired HPMR. Therefore, the priority settings of the labels gains importance.

The proposed solution focuses on performing individual packet header field searches and combining the search results. The proposed system does not offer a fixed algorithm for each field, but presents with a certain number of algorithms for selections. The complete packet classification system, as shown in Fig. 1, is composed of two main elements: the *Decision Control Domain* for packet handling decisions and the *Lookup Domain* for packet classification with several sub-functions, including Packet Header Partition, Search Engine, Label Combination and Rule Filter.

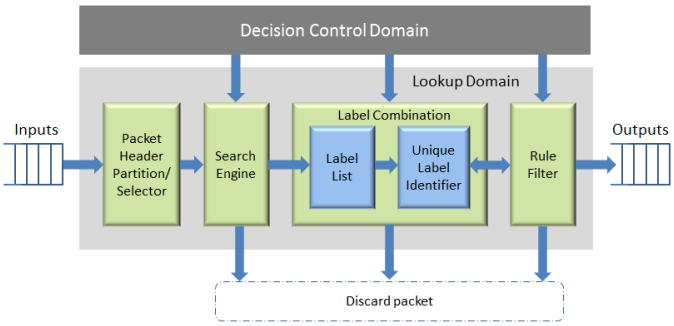

Fig. 1: Block diagram of the programmable lookup system

A. Decision Control Domain

In a pre-lookup phase, an individual algorithm for each field should be selected according to the application so as to provide an optimal lookup performance. For example, high speed is the critical parameter for a Multi-end videoconferencing application supporting real time connection. The selection and the characterization of the algorithm can be managed by the CPU instructions as part of the decision control operation. This information must be translated to set up the lookup domain on hardware.

B. Packet Header Partition/ Selector

The configurable lookup algorithm is activated when a packet header arrives in the system. The parallel search on each header field is a key to achieve higher search speed. Thus, the packet header is split into different fields. It is assumed that the packet header has a fixed (known) length and the header fields are organized in a certain order. Each field is sent to the corresponding selected algorithm (in accordance with the matching type of the header field) in the Search Engine module.

C. Search Engine

The Search Engine module of the presented architecture is based on three main engines according to the required match pattern, e.g. LPM, range matching and exact matching. Each process is performed individually using different algorithms in parallel. For an initial proof of concept, experimental setup is based on the common 5-tuple lookup and the available algorithms for test are collected and described below.

1. LPM Engine

The main advantage of the proposed system is that any lookup algorithm can be used provided that it supports the label method and it satisfies the user/application specific requirements in terms of lookup performance and/or memory consumption. For feature studies on a flexible LPM operation for IP address lookup, Multi-bit trie (MBT) and Binary search tree (BST) are selected as two candidates. MBT features high lookup speed but presenting inefficient storage. In comparison, BST offers a space efficient solution but it requires a large number of cycles for lookup.

2. Range-matching Engine

The main challenge in a Port lookup is to search intervals when a rule is being updated and perform a point search for the packet lookup. Trie structures based on interval lookup are used for range matching. Searching for all arbitrary ranges of port fields that overlap a given point presents a greater challenge than exact matching. Range matching can be made sufficiently fast for real filter sets using a set of parallel segment tree algorithms. These algorithms have inefficient memory usage, e.g. storing empty nodes for segment trie or duplication of rules in range search tree algorithm.

As a comparison, a small register bank is another option for Port field lookup, where the entries contain information about the boundary port values which define range and the corresponding labels.

3. Exact-matching Engine

There is a small set of values in a protocol field. In particular, three values are possible in any of the used filters, for example TCP, UDP or ICMP. For packet classification, it is straightforward to handle a small set of protocols using direct-indexing or using hash table for future expansions of the data set. Table II compares the single-field lookup algorithms that can be implemented in hardware. The comparison is based on the evaluation of LPM and range-matching lookup algorithms in terms of the hardware design requirements, such as label method supporting, lookup speed and memory space.

D. Label Combination

As mentioned earlier, it is important to support incremental update for the rules and the corresponding labels

to be modified. The labels play a key role in achieving the final HPMR. The update process should not produce a dynamic label value. For example, for inserting a new rule, the new labels created should not change the existing labels.

TABLE II: COMPARISONS OF LOOKUP ALGORITHMS FOR HARDWARE IMPLEMENTATION

Lookup algorithm	Label method support	Lookup speed	Memory space usage
LPM algorithms			
Multi-bit Trie	Yes	Fast	Moderate
AM-Trie	Yes	Moderate	Moderate
Binary Search Tree	Yes	Slow	Low
Binary tree with leaf pushing	No	Slow	Very low
Range-matching algorithms			
Range tree	No	Fast	High
Segment tree	Yes	Very slow	Moderate
Register bank	Yes	Very fast	Moderate

1. *Label List*

As proposed in [16], a label list instead of a rule list is maintained in the lookup domain. The selected algorithm set together with the corresponding labels is characterized by the *Decision Controller* in the update phase. The output from the Search Engine is a list of labels as the result from each individual lookup algorithm. These labels are combined to obtain the final index for HPMR. In order to speed up the lookup process towards HPMR, priority is also assigned to the labels and the resultant label lists are stored in a priority order. Likewise, the label combination is performed according to the label priority.

2. *Unique Label Identifier (ULI)*

As each algorithm produces a list of labels ordered by priority, the first label in the output list of each lookup algorithm refers to the highest priority matching rule. The ULI module receives the resulting label lists and the counter value, which indicates the number of valid labels in each list, and performs the combination to address the matching rule.

There are several solutions that can be applied for label combination, such as hash function or complex calculation. A simple manner to handle the label combination is that the highest priority labels of each field are combined and compared with a list of valid label combinations. If there is no match, the next highest priority labels are combined until the matching label combination is found. If there are no more possible label permutations and the valid label combination is not found, the input packet does not have a matching rule and should be discarded or sent to the control platform. Fig. 2 shows a simple Unique Label Identifier module design. *L_IPs, L_IPd, L_Ps, L_Pd* and *L_PRT* refer to the labels for source and destination IP fields, source and destination port fields and protocol field, respectively.

According to the research work in [4] and [6], the maximum number of labels in each field is limited to five labels. This is based on the observation that there is only a small set of matching rules that match with an input packet. If HPMR is not found, search on a new combination of labels is required. In this case, the combination process in this block is the bottleneck of the entire system because it consumes large label combination time (LCT). In the worst case, all the labels are combined and the LCT is calculated by the following equation:

$$\text{LCT} = O\,(n_1 * n_2 * n_3 * \ldots * n_d), \quad \forall\,(n \leq 5), \qquad \text{(Eq.1)}$$

where d is the number of fields and n_x is the total number of labels for the label vector of the field x.

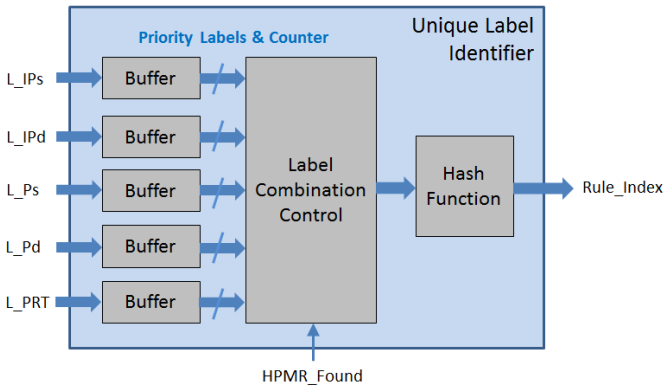

Fig. 2: Unique Label Identifier module

The above timing delay on looping search can be alleviated by shifting the problem from the lookup domain to the control domain.

A label-rule mapping module was added in the control domain in the host and managed during the update process. By operating this module, the actions of the original rule set are split into the labels and the rule set is optimized by reducing rule overlaps within each field. In this approach, the number of labels stored in the listed is dramatically reduced, resulting decreased label combination time.

E. *Rule Filter*

The Rule Filter module performs the lookup of the HPMR according to the label combination index resulting from the ULI module. If the resultant index addresses to a valid rule action, a rule acceptation signal is forwarded to the Unique Label Identifier module. However, if the index points to a non-valid rule (i.e. an empty address), the Rule Filter module returns the result to the ULI module and waits for a new index. If a matching rule is found, the proposed system output the original packet header and associated action. Practically, this output is sent to a function block to apply the matching action to the packet header. If there is no matching rule, the packet header is discarded.

In the case that the selected lookup algorithm is switched to satisfy new optimal parameters, the rest of the lookup

domain elements e.g. *Label Combination* and *Rule Filter*, remain the same.

IV. Performance Evaluation

A. System Integration

The proposed system consists of two domains, control domain and lookup domain. The control domain is running on a standard Intel CPU. The lookup domain was implemented on an Altera's Stratix® V FPGA (5SGXMB6R3F43C4). If a packet is processed or the rules are updated, the information from both domains is transferred via the same network interface**.** A Peripheral Component Interconnect (PCI), in particular the new protocol of PCI Express (PCIe), is a bus with high bandwidth for the control platform to connect the hardware device to the host. The communication between host PC and the FPGA board requires a connection between PCIe and other drivers, such as Jungo. In the proposed system, the tasks of the control domain, which focus on algorithm configuration and update process, are simply simulated using a file set with all the related information.

B. Rule update process

The update requirements differ among different applications: a very low update rate may be sufficient in firewalls where entries are added manually or infrequently, whereas a router with per-flow queues may require very frequent updates.

The update process is based on the information read from the *Decision Control platform*. This is represented in the simulation by files read and written to the hardware device to determine the number of clock cycles required to update the field label, rule and algorithm information. As mentioned in previous sections, the software platform supplies the information to be updated in a specified format. The packet header and action field information are read in and each field is stored independently in specific memory, as required. Non-repeated labels are stored using the label method.

The update process for rule information is more complex due to the hash function operation. All the necessary algorithm labels must be specified with each rule data and an extra clock cycle is required to calculate the final index. The labels are combined and hashed to obtain the final address. This process is performed in pipelining independent of the selected algorithm. The system are tested using different rule filters, such as Access Control List (ACL) 1K/5K/10K rule sets, Firewall (FW) 1K/5K/10K rule sets and IP Chain (IPC) 1K/5K/10K rule sets. For each type of rule filters, the field information related to the algorithm memories is pre-stored in different files. A test bench was created to stimulate the system and provide the header field information by reading the corresponding binary file for each selected algorithm. The update process cannot be performed for both MBT and BST modes at the same time because they share memory resources. The decision of which algorithm to choose is made by the control domain.

Fig. 3 shows the update time in terms of the number of CPU clock cycles for the MBT mode and BST mode. The average latency for the original rule filter update on hardware is two clock cycles per rule.

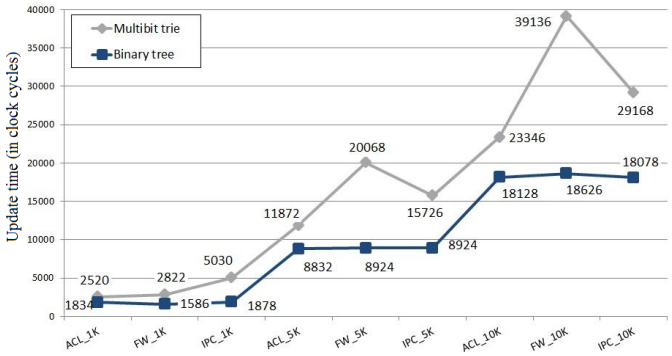

Fig. 3: Ruleset update time

It can be seen in Fig. 3 that the update latency using the BST algorithm is similar to the original rule filters. This is because the number of "lines of information" for binary tree update is proportional to the number of rules. However, a larger update time is expected using multi-bit trie because there are a larger number of trie nodes to store in different memory blocks.

C. Packet lookup process

The proposed designs are based on pipelined stages as described in Fig.1. Prior to the lookup, the *Decision Controller* has performed the rule updates and algorithm selections.

The protocol label search is executed in a single clock cycle. The range search engine produces the labels in two clock cycles. In the IP address search engine, the MBT data structure is executed with deep pipelining to support high throughput. As expected, the BST mode requires a large number of clock cycles per input packet. Fig. 4 shows the lookup process measured in number of clock cycles for different packet header sets (PHS) with different sizes. The impact of pipelining in MBT is clear as the lookup is completed 8 times faster with MBT than that with BST.

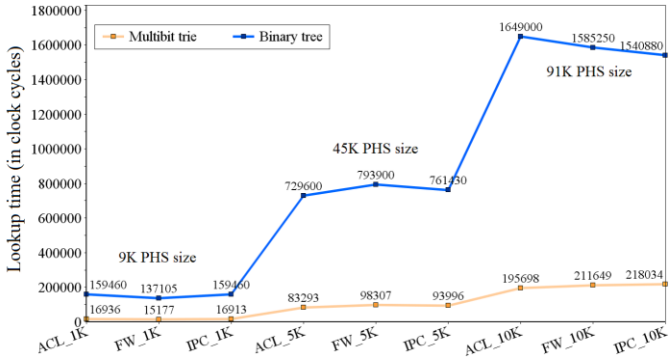

Fig. 4: Lookup time measured in number of clock cycles

As previously noted, the worst case of lookup occurs when the maximum number of labels are found for each field of a given packet header and all results have to be combined. This worst-case scenario of label combination is independent of the algorithm chosen and is very unlikely to occur due to the fact that the rulesets have been optimized in the decision controller.

D. Discussion

The range-matching and exact-matching engines produce results in the stages prior to the LPM engine. The LPM engine defines the critical path in terms of label search time since it requires extra time to find the labels. The lookup process for the HPMR is only performed when all the field searches match. Practically in this work, it is safe to operate the system at the clock of frequency of 200 MHz for timing closure, using FPGA embedded RAM blocks. This results in a lookup throughput of 95.23 million packets per second in MBT mode. Taking ACL 10K ruleset as an example, the proposed system is able to operate at 6.5 Gbps in BST mode with a small memory space and 54 Gbps throughput in MBT mode, given a minimum Ethernet frame size of 72 bytes.

V. CONCLUSION

The main objective of this research was to design a highly configurable parallel lookup system for packet classification. As is presented, the proposed and prototyped lookup architecture targets future network systems with fast update supports through software programmability. The proposed design offers optimal lookup performance by configuring the best performing set of algorithms for a given flow entry type and rulesets. Efficient memory utilization is also achieved by sharing memory resources among multiple lookup algorithms.

The presented results show a clear improvement in terms of memory space required and throughput in comparison with other lookup techniques. The proposed system is not restricted with a small number of algorithms as chosen for the experiment. More efficient search algorithms will be adopted into the search engine enabling high flexibility in future packet classifications.

REFERENCES

[1] IEEE 802.3 standards for Ethernet networks, IEEE 802.3 Ethernet working group, http://www.ieee802.org/3/index.html, accessed on 1st May 2016.

[2] M. A. Ruiz-Sanchez, E.W. Biersack, W. Dabbous, "Survey and Taxonomy of IP Address Lookup Algorithms", *IEEE The Magazine of Global Internetworking*, pp. 8-23, 2001.

[3] M. Berg, M. Kreveld, M. Overmars, O. Schwarzkopf, Computational Geometry: Algorithms and Applications. Springer, 1997.

[4] P. Gupta and N. Mckeown, "Packet classification on Multiple Fields", *SIGCOMM'99,* pp. 147-160, 1999.

[5] V. Srinivasan, S. Suri, G. Varghese, M. Waldvogel. "Fast and Scalable Layer four Switching," *ACM Sigcomm*, pp. 203-14, 1998.

[6] F. Baboescu, G. Varghese, "Scalable Packet Classification", *IEEE/ACM Transaction on Networking,* pp: 2-14, 2005.

[7] B. Zheng, C. Lin and X. Peng, "AM-Trie: An OC-192 Parallel Multidimensional Packet Classification Algorithm for Network Processor", *IMSCCS'06.* Vol.1 pp.:377-384, 2006.

[8] O. Ahmed, S. Areibi, K. Chattha, B. Kelly, "PCIU: Hardware Implementation of an Efficient Packet Classification Algorithm with an Incremental Update Capability*", International Journal of Reconfigurable Computing,* pp: 2011.

[9] D. E. Taylor and J.S. Turner, "Scalable Packet Classification using Distributed Crossproducting of Field labels", *IEEE INFOCOM 2005*, Vol. 1, pp. 269-280, 2005.

[10] P. Gupta and N. Mckeown, "Packet Classification using Hierarchical Intelligent Cuttings", *IEEE Symposium on High Performance Interconnects, HotI, 1999*.

[11] B. Xu, D. Jiang, J. Li, "HSM: a fast packet classification algorithm", *IEEE Advance Information Networking Application (AINA),* pp: 987-992, 2005.

[12] T. Heinz, PhD thesis "HiPAC High Performance Packet Classification for Netfilter", September 2003.

[13] S. Singh, F. Baboescu, G. Varghese, J. Wang "Packet Classification Using Multidimensional Cutting", *SIGCOMM,* pp. 213-224, 2003.

[14] Y. Qi, B. Xu, F. He, X. Zhou, J. Yu, J. Li "Towards Optimized Packet Classification Algorithms for Multi-Core Network Processors", *Parallel Processing (ICPP),* pp: 2, 2007.

[15] V. Srinivisan, S. Suri, G. Varghese, "Packet Classification using Tuple Space Search", *ACM SIGCOMM'99*, pp 135-146. 1999.

[16] K. Guerra Perez, X. Yang, S. Scott-Hayward, S. Sezer, "A Configurable Packet Classification Architecture for Software-Defined Networking". *IEEE SoCC'14,* pp. 353-358, 2014.

[17] K. Guerra Perez, X. Yang, S. Scott-Hayward, S. Sezer, "Optimized Packet Classification for Software-Defined Networking". *IEEE ICC'14.* pp. 859-864, 2014.