Memory Cost Analysis for OpenFlow Multiple Table Lookup

Guerra Perez, K., Scott-Hayward, S., Yang, X., & Sezer, S. (2015). Memory Cost Analysis for OpenFlow Multiple Table Lookup. In Proceedings of the 2015 28th IEEE International System-on-Chip Conference (SOCC). (pp. 322-327). (System-on-Chip Conference (SOCC) Proceedings). Institute of Electrical and Electronics Engineers (IEEE). DOI: 10.1109/SOCC.2015.7406975

Published in:
Proceedings of the 2015 28th 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

Memory Cost Analysis for OpenFlow Multiple Table Lookup

Abstract—**Multiple Table Lookup architectures in Software Defined Networking (SDN) open the door for exciting new network applications. The development of the OpenFlow protocol supported the SDN paradigm. However, the first version of the OpenFlow protocol specified a single table lookup model with the associated constraints in flow entry numbers and search capabilities. With the introduction of multiple table lookup in OpenFlow v1.1, flexible and efficient search to support SDN application innovation became possible. However, implementation of multiple table lookup in hardware to meet high performance requirements is non-trivial. One possible approach involves the use of multi-dimensional lookup algorithms. A high lookup performance can be achieved by using embedded memory for flow entry storage. A detailed study of OpenFlow flow filters for multi-dimensional lookup is presented in this paper. Based on a proposed multiple table lookup architecture, the memory consumption and update performance using parallel single field searches are evaluated. The results demonstrate an efficient multi-table lookup implementation with minimum memory usage.**

Keywords— Packet Classification; Lookup Algorithms; Software-Defined Networking; OpenFlow; Multi Table Lookup.

I. Introduction

Software-defined Networking has evolved as a platform to increase flexibility and innovation in network traffic management. It achieves this by separation of the control and data plane with the control plane responsible for defining network-wide traffic behaviour and the data plane responsible for implementing packet processing decisions. The control plane consists of a set of Controllers and the data plane consists of the network elements e.g. switches, routers. The controllers connect to the data plane via an Application Programming Interface (API).

OpenFlow [1] is the most popular API for the control-data interface. The OpenFlow protocol describes an extensive list of packet header fields for packet and flow classification. The classification of packets into flows uses multiple fields of the packet header. This supports fine-grained flow classification based on which new network applications can be developed. However, with increasing granularity of the flow definition, a greater volume of flow match entries are held in the flow tables of network devices [2].

The first version of the OpenFlow protocol specified a single table lookup model with the associated constraints in flow entry numbers and search capabilities. The complexity for a single lookup table to support a large number of flow entries and arbitrary flow features of different applications is high [3]. In order to optimize the lookup process, a multiple table pipeline model was introduced in OpenFlow v1.1. Using this method, packets can be matched against a defined set of flow tables as specified by the application. This reduces the number of flow entries per table and increases the classification performance accordingly.

There are several challenges when implementing the multiple table lookup model in hardware. For example, different application characteristics e.g. no. of packet header fields, field lengths, no. of actions etc. must be mapped into appropriate lookup tables [3]. In addition, for next-generation networks, packet classification must support high network throughput, e.g. 40-100 Gbps. The lookup efficiency in terms of scalability, flexibility, capacity, incremental update ability, memory usage and speed must all be considered.

A possible method to implement the multiple table lookup model in hardware involves the use of multi-dimensional lookup algorithms. For high lookup performance, the flow entries can be stored in embedded memory. The memory requirements for such a solution are analyzed in this paper. A detailed study of OpenFlow flow filters for multi-dimensional lookup is presented and a multiple table lookup architecture is proposed based on parallel single field searches. The architecture is implemented and synthesized on a Stratix V FPGA [4] and memory consumption and update performance results are presented for the OpenFlow flow filters analyzed (Routing and MAC Filters). The results demonstrate an efficient implementation of multiple table lookup in hardware with minimum memory usage.

The rest of the paper is organized as follows: In section II, related work is discussed. A survey of different flow filters is presented in section III and the multiple table lookup architecture is described in section IV. In section V, performance evaluation results are presented and discussed. Finally, in section VI, the paper is concluded.

II. Related Work

There is a small body of research on packet/flow classification for SDN and OpenFlow-based devices for single and multi-table lookup. Flow table mapping is considered in [5], [6]. In [[6] a simple table distribution with parallel tables for non-dependent fields is proposed. In [7], the limitations of an OpenFlow v1.3 software switch with Intel DPDK are highlighted. For example, the packet processing performance drops rapidly with an increasing number of flow rules. The authors suggest improvements such as flow caching, better lookup algorithms, and lookup algorithm selection per table with tables containing rules of similar types.

Multi-dimensional lookup techniques can be categorized into four groups; Trie-based (HyperCuts [8], HyperSplit [9]), Decomposition (RFC [10], DCFL [11]), Hashing (TSS [12], TTSS [13]) and Hardware-based (TCAM, Bitmap-insertion

[14]). The advantages and disadvantages of these techniques are outlined in Table I.

TABLE I
EVALUATION OF MULTI-DIMENSIONAL LOOKUP ALGORITHMS

Category	Advantages	Disadvantages
Trie-Geometric	Efficient Memory	Moderate lookup Very Complex update
Decomposition	Fast Lookup	Memory explosion Complex update
Hashing-based	Fast Lookup	Collision issue Memory explosion
Hardware-based	Very Fast Lookup	Memory Limitation Poor Flexibility

The most common hardware-based structure proposed for OpenFlow multiple table devices [15]-[17] is a combination of Ternary Content Addressable Memory (TCAM) for wildcard matching using linear search and SRAM for exact matching using a hash function. In [16], the authors use an Extended RFC algorithm. The TCAM and SRAMs are divided into blocks to represent each table. The lookup process involves an exact match search followed by a TCAM lookup if no match was found in the first stage. Although TCAM is a popular method for classification due to its high lookup speed, it has disadvantages in terms of high power consumption, storage limitation and the difficulty of rule ternary conversion.

An alternative to TCAM is to use multi-dimensional lookup algorithms. Two methods have been proposed [18]-[19] based on the HyperCut algorithm. However, they are limited to single table lookup and the number of rules stored reduces with increasing number of lookup fields. The high lookup speed of the decomposition approach was demonstrated for a single table lookup in [20]. The decomposition technique uses parallel search of packet header fields (using a set of algorithms) and combines the results for the final matching rule lookup.

In comparison to the existing research, this work presents a solution to replace the TCAM with a multi-field, multiple table lookup model. The focus of the paper is the analysis of memory requirements for this solution, which is critical to achieve high lookup performance.

III. FILTER ANALYSIS

In order to determine the appropriate search algorithms for optimal memory consumption across the multiple table lookup, a deep study of network flow filters is performed. The terms filter and rule are used here interchangeably.

A. *OpenFlow Match Fields*

The number of matching header fields that can be used for packet classification based on OpenFlow v1.3 is 39 (excluding metadata) [1]. The metadata field is an additional field composed of 64 bits. The system uses the metadata internally to pass information between lookup tables during packet processing. There are 15 common matching fields supporting applications, such as Access Control List, MAC learning or Routing**.** They are listed in Table I along with the field length in bits and the matching method required. In addition to the field length, the definition of the field in terms of syntax such as range or prefix is relevant for the lookup process. Based on this syntax, each field requires a different matching method, for example Exact Matching (EM), Range Matching (RM) or Longest Prefix Matching (LPM). Different matching types present a challenge to existing packet classification algorithms. The EM approach compares all bits of the packet header field against the flow entry field. For the RM approach, the narrowest range is selected from all the ranges of the filter that match against the packet header field. The LPM selects the entry with the prefix with the largest number of matching bits from all the matching prefixes of the filter.

TABLE II
OPENFLOW MATCH FIELD, FIELD LENGTH AND MATCHING METHOD

Matching Field	Number of Bits	Matching Method Required
Ingress Port	32	Exact Matching (EM)
Source Ethernet	48	Wildcard matching (LPM)
Destination Ethernet	48	Wildcard matching (LPM)
Ethernet Type	16	Exact Matching (EM)
VLAN ID	13	Exact Matching (EM)
VLAN Priority	3	Exact Matching (EM)
MPLS Label	20	Exact Matching (EM)
Source IPv4	32	Wildcard matching (LPM)
Destination IPv4	32	Wildcard matching (LPM)
Source IPv6	128	Wildcard matching (LPM)
Destination IPv6	128	Wildcard matching (LPM)
IPv4 Protocol	8	Exact Matching (EM)
IPv4 ToS	6	Exact Matching (EM)
Source Port	16	Wildcard matching (RM)
Destination Port	16	Wildcard matching (RM)

B. *Lookup Methods*

Three lookup methods have been identified: EM, RM, and LPM. For the fields requiring exact matching, this lookup can be handled by a hash function. However, the lookup process for fields that require wildcard matching is more complex. For this reason, this search method is considered in detail in this work in order to achieve high lookup performance. The Ethernet and IP address fields are a focus of this work as they present the largest field length.

Rule replication is an issue for multi-dimensional lookup algorithms, which implies the storage of the copied rules through the algorithm structure. For example, HyperCuts requires that the same rule be stored in several trie nodes, which leads to inefficient memory use. However, the individual field management provides flexibility to handle each rule field. An analysis based on the repetition of each rule field is therefore performed.

C. *MAC and Routing Filter Analysis*

For the analysis presented in this section, the filter set presented in [21] is studied. This filter set contains a range of flow sets based on different applications e.g. ACL entries (*_rtr_config*), Routing/Packet Forwarding (*_rtr_route*), MAC learning (*_rtr_mac_table*) and ARP (*_rtr_arp*). Each of these flow sets is comprised of 16 different flow filters of different sizes. These are named according to the Router ID e.g. *bbra*.

The analysis begins with a survey of the number of unique values identified in the filter set for a given field.

In Table III, the MAC learning application ruleset is analyzed. The two fields in this set are VLAN ID and Destination Ethernet address. For each filter, the total number of rules and the number of unique VLAN ID and Ethernet addresses within that ruleset are identified. Based on a previous study of field partition [22], the evaluation of the Ethernet address is presented for 16-bit field partitions.

As illustrated in Table III, there are relatively few unique values of VLAN ID field. For example, there are no more than 209 different VLAN ID values (*gozb* filter) out of any of the analyzed filter sets. The VLAN ID field uses exact matching (Table II) and this observation contributes to the multiple table lookup design presented in Section IV. The Ethernet address field is also in exact value format. The Ethernet address is assigned to a network interface for communication. The first 3 bytes are the Organizationally Unique Identifier (OUI) while the second 3 bytes are Network Interface Controller (NIC) specific. The results are as expected i.e. there is a smaller number of unique values of the higher 16-bit partition compared with the other partition values.

TABLE III
NUMBER OF UNIQUE FIELD VALUES OF FLOW-BASED MAC FILTER

Flow Filter	**Number of Rules**	**Number of unique values for MAC filter**			
		VLAN ID	**Higher 16-bit Ethernet**	**Middle 16-bit Ethernet**	**Lower 16-bit Ethernet**
bbra	507	48	46	133	261
bbrb	151	16	26	38	55
boza	3664	139	136	3276	2664
bozb	4454	139	137	1338	3440
coza	3295	32	225	1578	2824
cozb	2129	32	194	1101	1861
goza	6687	208	172	2579	5480
gozb	7370	**209**	159	1946	6177
poza	4533	153	195	2165	3786
pozb	4999	155	169	1759	4170
roza	3851	114	136	2389	3264
rozb	3711	113	140	1920	3175
soza	3153	41	187	1115	2682
sozb	2399	39	161	821	2132
yoza	3944	112	178	1655	3180
yozb	2944	101	162	1298	2351

For further analysis, the Routing filter is also evaluated. This filter contains the IPv4 address and interface fields (e.g. ingress port). The ACL filter set also contains the IP address field. However, the Routing filters contain a larger number of wildcard flow entries and require larger prefix lookups (e.g. 0.0.0.0/0). Therefore, to better illustrate the LPM, the Routing filter analysis results are presented as an example of the worst case. As for the MAC filter, the Routing filter survey is based on the unique field values for IP address with 16-bit partitions. Table IV summarizes the analysis of the routing filter.

TABLE IV
NUMBER OF UNIQUE FIELD VALUES OF FLOW-BASED ROUTING FILTER

Flow Filter	**Number of Rules**	**Number of unique values for Routing filter**		
		Ingress Port	**Higher 16-bit IP Address**	**Lower 16-bit IP Address**
bbra	1835	40	82	1190
bbrb	1678	20	82	1015
boza	1614	26	53	1084
bozb	1455	26	53	952
coza	184909	43	**20214**	**7062**
cozb	183376	39	**20212**	**5575**
goza	1767	21	57	1216
gozb	1669	22	57	1138
poza	1489	18	54	976
pozb	1434	20	54	932
roza	1567	17	52	1053
rozb	1483	16	52	988
soza	184682	48	**20212**	**6723**
sozb	180944	36	**20212**	**3168**
yoza	4746	**77**	58	3610
yozb	2592	48	55	1955

Similar to the MAC filter results, it can be seen in Table IV that the number of unique ingress port fields achieves a maximum of 77 different values (*yoza* filter). Even the largest flow filter for routing (*coza* with 184909 entries), only has 43 unique ingress port values. The ingress port field uses exact matching, similar to the VLAN ID field of Table II.

In contrast, the IP address field can be composed of wildcard or exact values. The routing filters are comprised of IP address field with wildcards. Since the address field identifies the network device (similar to the Ethernet address for network interface), the higher bits of the address identify the network and the lower bits identify a particular network device i.e. the host. Therefore, it is expected that the higher 16-bit partitions contain a smaller number of unique values compared with the lower 16-bit partitions. This is illustrated in Table IV. The exceptions to this are *coza, cozb, soza* and *sozb* filters, as highlighted. For these four filters, the higher 16 bits of the IP address field present a greater number of unique values, indicating a wider range of network addresses in these filter sets. The *coza* filter of routing application (Table IV) reaches a maximum of 20214 unique address values corresponding to 11% of the total flow entries.

Based on the filter analysis observations, the proposed multiple table lookup design is described in the next section.

IV. MULTIPLE TABLE LOOKUP DESIGN

As previously noted, the proposed multiple table lookup architecture is based on the decomposition technique. The architecture is illustrated in Fig. 1 with the individual elements described in the next sections.

A. Packet header and Partition/Selector

For the lookup process, the packet header is split into the selected fields used for the first table lookup. Each field partition is sent to the corresponding single-field algorithm.

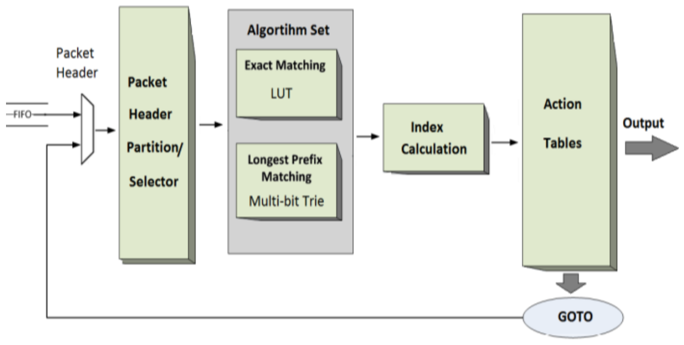

Fig. 1 Multiple Table Lookup Architecture

B. Algorithm Set and Label Method

From the analysis of the unique values of each flow field presented in Section III, it was identified that the filters are comprised of a set of entries based on field repetition. Labelling the unique rule fields is a key method for efficient storage and to avoid rule replication. A label method was presented in [11] for this purpose. The label method is an efficient technique for algorithms with fixed structures such as Multi-bit Trie (MBT), and is not applicable to dynamic structures, which require a re-built algorithm structure.

For the proposed architecture, the exact matching fields for the analysed flow filters are handled using a simple hash-based Lookup table (LUT). However, the prefix-based fields require a more complex lookup to support LPM. Several one-dimensional lookup algorithms support LPM, of which the multi-bit trie algorithm is the most popular. MBT searches several bits at one tree level simultaneously.

The label method is applied to each of these lookup algorithms so that each unique field value is labelled and stored in the corresponding algorithm.

C. Index Calculation and Action Tables

The result from each algorithm search is a label, which is used to obtain the final index to address the action tables.

In the applications presented in Section III (Routing and MAC learning), there are two fields that can be distributed into two tables. The flow entries have been adapted to support the required OpenFlow-based instructions [1] for multiple table lookup. Hence, when the packet header matches with a flow entry, there are two required instructions:

- *Goto-Table*: Forward the packet header being processed to the next table.
- *Write-action*: Apply the actions to the packet header e.g. forward the packet to a specific output port.

In the case that no matching rule has been found for the packet header, the instruction is "*Send to controller*".

The results of the performance evaluation for this multiple table lookup architecture are presented in the next section.

V. Performance Evaluation

The proposed architecture is implemented and synthesized on 5SGXMB6R3F43C4 of the Stratix® V FPGA family [4]. The evaluation of the proposed architecture is measured in terms of memory consumption and update performance.

A. Figures and Tables

Implementation of the proposed architecture based on the MAC learning and Routing filters consumes 5 Mb of total memory. In this case, 4 OpenFlow Lookup Tables are implemented along with two independent multibit trie structures and two exact matching LUTs. The MBT implementation consumes the majority of the total storage (2Mb for both MBT structures).

For these two use cases, 209 values must be addressed on implemented LUTs based on the worst case of unique fields (VLAN ID from Table III). However, it can be seen that IP and Ethernet address fields are more complex and the memory space of MBT algorithms depends on the number of stored trie nodes, which depends on the prefixes.

In order to evaluate the memory consumption for MBT, Fig. 2 shows the results regarding the number of nodes stored for Ethernet address fields (Fig. 2 (a)) and IPv4 address fields (Fig. 2 (b)). The Ethernet address field is 48 bits and requires three 16- bit MBT structures. The highest 16 bits of the fields are searched in the higher trie, the following 16 bits are sent to the middle trie and, finally, the lower trie handles the lowest remaining 16 bits of the field. All tries are distributed with three levels. The IPv4 address field is split into two 16-bit partitions and sent to two 3-level trie structures (Higher trie and Lower trie). Every trie structure works in parallel to find the corresponding label.

In Fig. 2(b), it can be seen that for all filters except *coza/b* and *soza/b*, the lower tries consume larger memory space to allocate the required stored nodes. This corresponds to the number of unique values identified in Table IV. Analysing the maximum number of nodes, the largest memory space must store 54010 nodes (MAC learning *gozb* filter).

To support pipeline lookup in the multiple table approach, each lookup algorithm is implemented in a separate memory block, and each node level of the multi-bit trie is searched in a different pipeline stage. To optimize the distribution of the pipeline trie structure, each level has been analyzed. A study presented in [22] concluded that the distribution of 3-level trie is optimal for a tradeoff between fast lookup and efficient memory space. The memory space required for each memory block (L1, L2 and L3) and the total space are evaluated. As shown in Fig. 2, the lower trie structures present the worst case. Thus Fig. 3 and Fig. 4(a) present the number of Kbits required to store the lower trie structures for Ethernet address and IP address, respectively. The trie node data is composed of the child pointer, the label and a flag bit. However, each level node requires different child pointer sizes. This size is determined by the worst case (lower trie). The very low number of nodes of L1 is notable across the graphs of Fig. 3 and Fig. 4(a). The maximum stored nodes in L1 are 32 and the memory consumption is less than 1 Kbit (832 bits).

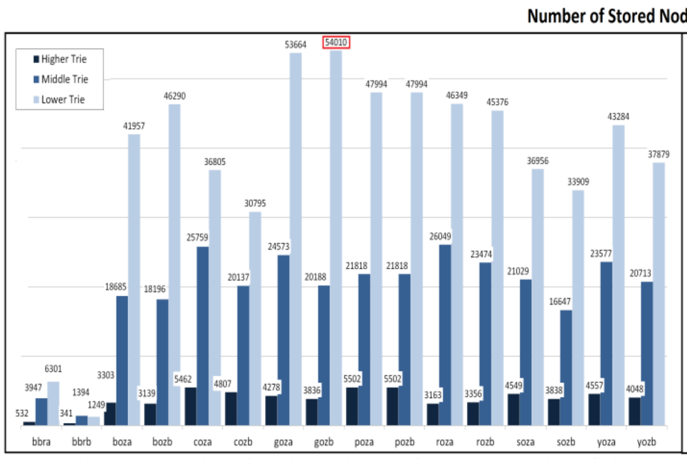

Fig. 2 (a) Number of total stored nodes for Ethernet address fields using different flow filters

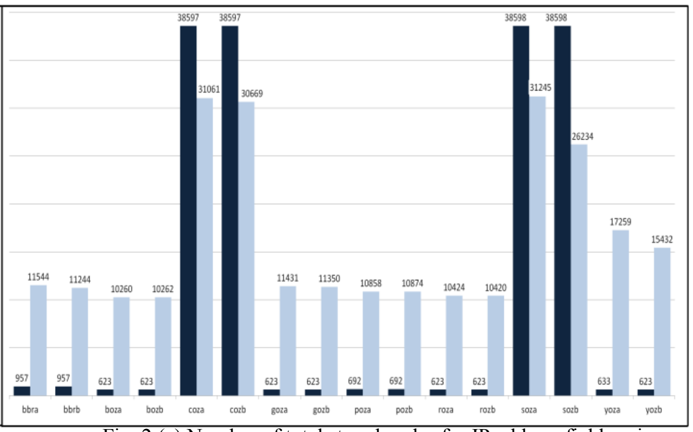

Fig. 2 (a) Number of total stored nodes for IP address fields using different flow filters

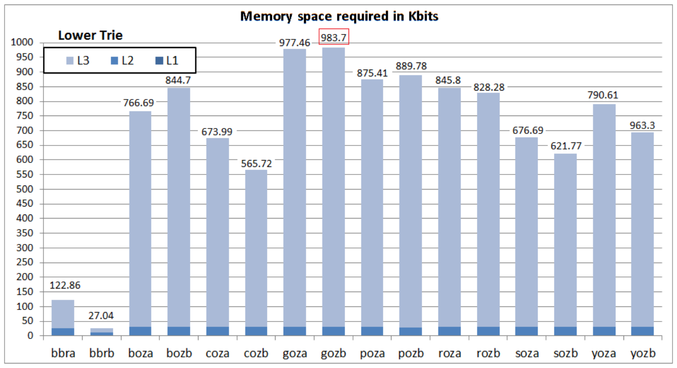

Fig. 3 Memory space required for each level of Ethernet Lower trie.

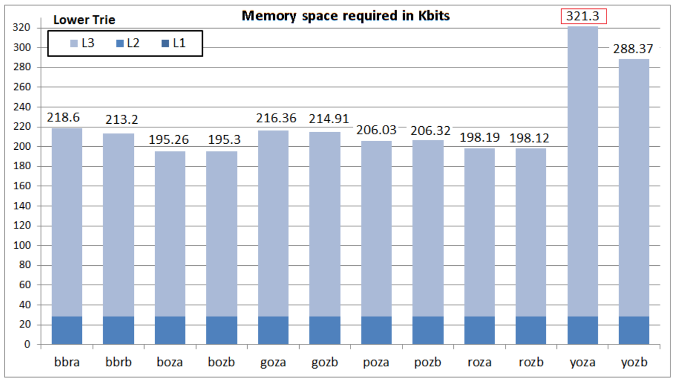

Fig. 4 (a) Memory space required for each level of IP address Lower trie.

The coza/b, soza/b filters are shown separately in Fig. 4 (b) due to their higher number of stored nodes compared with the other filters. Due to the exception described in Section II, the evaluation is performed for both higher and lower tries.

For Routing filters, the max. memory space required is 572.57 Kbits for the lower trie implementation using the coza/b or soza/b filters. For these filters, the higher trie structure requires higher memory space for L2 and L3 with a total of 706.06 Kbits. Otherwise, 321.3 Kbits are needed for the lower trie implementation.

Comparing the two MBT evaluations, the Ethernet address fields require a higher number of stored nodes for all cases with a maximum requirement of 54010. Consequently, the max. no. of bits required for the three levels of trie structure using this worst case is 983.7 Kbits (*gozb* filter).

For the IP address of the Routing filter, the number of stored nodes is less than 40000 even for the worst case filters with more than 180K rules (coza, cozb, soza and sozb). This difference is due to the fact that the MAC learning filters are composed of exact values for Ethernet address fields.

The multi-bit trie can suffer from memory explosion in cases of coz/b, soza/b filters from Routing filters due to all the individual values that must be stored. The label method selects the unique values to store, avoiding value repetition.

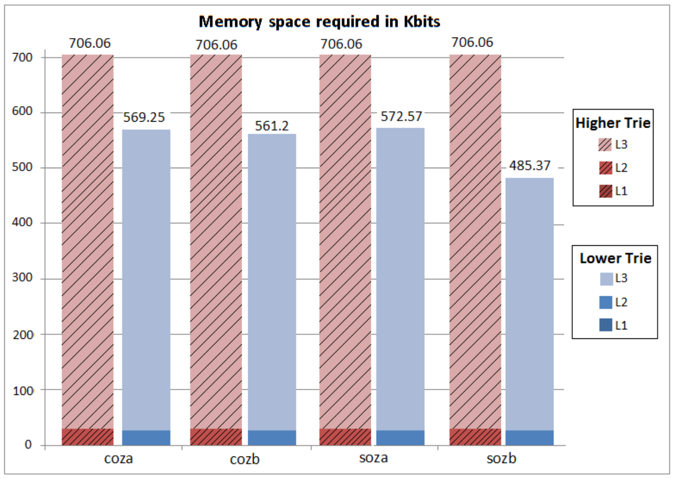

Fig. 4 (b) Memory space required for each level of IP address Higher trie and Lower trie for coza/b and soza/b filters.

B. *Evaluation of Update Process*

In order to simulate the Software Controller platform, two files are generated with the information to characterize each algorithm and table block. For each entry, the required information is extracted and interpreted to update the algorithm structures and the action tables. For example, the trie node information is determined according to the address fields, and the exact match LUTs are characterized from

VLAN ID or Ingress port fields. The processed information is stored in an update file. The timing evaluation is based on an update process using optimized algorithm files and action files. On average, two clock cycles are required for each update. The update data is composed of the label and the information for each lookup algorithm structure or table. The index used to address the algorithm data is calculated in the first clock cycle and stored in the second clock cycle. The same process is performed for both algorithm and lookup table update. Fig. 5 shows the number of CPU clock cycles required to update the lookup algorithms using the optimized algorithm files, in which the label method is applied, and initial algorithm files (without the label method).

The advantage of the update time reduction using the label method is considerable, achieving a 56.92% fewer CPU clock cycles on average. Consequently, the update process is faster and the memory space is reduced using the label method.

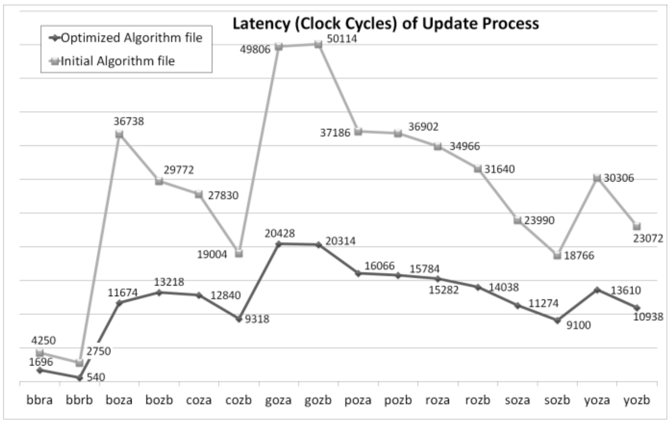

Fig. 5 Number of CPU clock cycles required for algorithm update using the original algorithm and using the label method.

VI. Conclusion

The OpenFlow protocol describes an extensive list of packet header fields that can be used for packet/flow classification. The leads to a potentially large rule set for fine-grained flow classification. The multiple table pipelined approach introduced in OpenFlow v1.1 is designed to optimize the lookup process. An analysis of OpenFlow filters has been presented in this paper, highlighting the potential to select a lookup method based on the packet header field type. Based on this analysis, a multiple table lookup architecture is proposed in which parallel one-dimensional field searches are performed with an individual field set managed for each table. The performance study, focused on memory consumption for the MBT algorithm, has identified an optimal approach to minimize memory usage. For the investigated prototype, only 5 Mbits of total memory are required. A significant improvement of the update process is also demonstrated.

References

[1] Open Networking Foundation, OpenFlow https:// www.opennetworking.org/Openflow . Accessed February 2015.

[2] S. Sezer, S. Scott-Hayward, P. Kaur Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, N. Rao. "Are we ready for SDN?- Implementation Challenges for Software- Defined Networks". *IEEE Communications Magazine,* Vol 51, pp. 36-43, 2013.

[3] Open Networking Foundation "The Benefits of Multiple Flow Tables and TTPs ", Version Number 1.0, February 2, 2015.

[4] Altera, Stratix V. https:// www.altera.com/products/fpga/stratix-series/stratix-v/support.html. Accessed December 2014.

[5] H. Song, "Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking,* 2013, pp. 127-132.

[6] P. Bosshart, D. Daly, M. Izzard, N. McKeown, J. Rexford, D. Talayco, A. Vahdat, G. Varghese and D. Walker, "Programming protocol-independent packet processors," *ArXiv Preprint arXiv:1312.1719,* 2013.

[7] G. Pongrácz, L. Molnar and Z. L. Kis, "Removing roadblocks from SDN: OpenFlow software switch performance on intel DPDK," in *Software Defined Networks (EWSDN), 2013 Second European Workshop on,* 2013, pp. 62-67.

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

[9] Y. Qi, J. Fong, W. Jiang, B. Xu, J.Li, V. Prasanna, "Multi-dimensional Classificaiton on FPGA: 100 Gbps and Beyond", *Field-Programmable Technology,* pp: 241-248, 2010.

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

[11] 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.

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

[13] R. Avudaiammal, R. SivaSubramanian, R. Pandian, P. Seethalakshmi, "TTSS Packet Classification Algorithm to enhance Multimedia Applications in Network Processor based Router", *International Journal of Computer Science and Information Security*, vol. 2, 2009.

[14] T. V. Lakshman and D. Stiliadis, "High-Speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching", *ACM Sigcomm*, 1998.

[15] NetFPGA 10G OpenFlow Switch Architecture [Online] Available: https://github.com/NetFPGA/NetFPGA-public/wiki/NetFPGA-10G-OpenFlow-Switch

[16] O. Ferkouss, I. Snaiki, O. Mounaouar, H. Dahmouni, R. Ben Ali, Y. Lemieux and C. Omar, "A 100gig network processor platform for openflow," in *Network and Service Management (CNSM), 2011 7th International Conference on,* 2011, pp. 1-4.

[17] G. Lu, C. Guo, Y. Li, Z. Zhou, T. Yuan, H. Wu, Y. Xiong, R. Gao and Y. Zhang, "ServerSwitch: A programmable and high performance platform for data center networks." in *Nsdi,* 2011, pp. 2-2

[18] Y. Qi, J. Fong, W. Jiang, B. Xu, J. Li and V. Prasanna, "Multi-dimensional packet classification on FPGA: 100 gbps and beyond," in *Field-Programmable Technology (FPT), 2010 International Conference on,* 2010, pp. 241-248.

[19] W. Jiang, V. Prasanna "Scalable Packet Classification on FPGA". *IEEE Transactions on Very Large Scale Integration (VLSI) System,* pp. 1668-1680, 2012.

[20] 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

[21] GitHub Yang (Jack) Wu, https:// github.com/wuyangjack/stanford-backbone. Accessed February 2015.

[22] K. Guerra Perez, X. Yang, S. Scott-Hayward, S. Sezer "An Improvement of IP address Lookup based on Rule Filter Analysis*". IEEE ICC'14,* pp. 688-693, 2014.