Many-Field Packet Classification for Software-Defined Networking Switches

Cheng-Liang Hsieh
Southern Illinois University
Carbondale, Illinois, USA
hsieh@siu.edu

Ning Weng
Southern Illinois University
Carbondale, Illinois, USA
nweng@siu.edu

ABSTRACT

Packet classification is a core problem for OpenFlow-based software-defined networking switches, which required 38 packet header fields per flow to be examined against thousands of rules in a ruleset. With the trend of continue growing number of fields in a rule and the number of rules in rule set, it will be a great challenge to design a high performance packet classification solution with the capability to easy update new rule and fields. In this paper, we present a scalable many-field packet classification algorithm with varying rulesets and its prototype implementation on a graphics processing unit. The proposed algorithm constructs multiple lookup tables and merges partial lookup results for a small ruleset to accelerate the overall packet classification process by using effective bit positions in a ruleset with three selecting metrics: wildcard ratio, independence index, and diversity index. Those lookup tables made with effective bit positions are flat with a low rule replication ratio. Besides, they are adjustable to meet different implementation environments for a good performance scalability between different ruleset sizes. Our prototype on a single NVIDIA K20C GPU achieves 198 MPPS, 186 MPPS, 163 MPPS throughput for 1K, 32K, and 100K 15-field ruleset.

Keywords

Packet Classification; GPU; SDN

1. INTRODUCTION

Software-defined networking (SDN) abstracts network infrastructures as programmable resources for network applications. OpenFlow-based software-defined networking switches [15] require 38 packet header match fields and 6 pipeline match fields to be examined for a flow. The requirements on latency, throughput, update cost, and storage for a system like OpenFlow Switch become stricter with the increasing complexity of a ruleset. Hence, packet classification remains an open and challenging problem for next generation network device development.

ANCS '16, March 17-18, 2016, Santa Clara, CA, USA
© 2016 ACM. ISBN 978-1-4503-4183-7/16/03...$15.00
DOI: http://dx.doi.org/10.1145/2881025.2881036

Current solutions generate good system performance for traditional 5-tuple packet classification problems [21][9][3][10][1] but they do not scale well to many-field packet classification problems. Longer processing latency and bigger space requirement caused by additional fields in a ruleset are expected. Many-field packet classifications with current 5-tuple TCAM solutions are with high implementation cost due to the bigger size of each rule and arbitrary field types in a many-field ruleset. Many-field implementations with current 5-tuple decision-tree-based solutions will generate a deeper and wider tree due to those additional fields and result in impractical memory usage and inefficient tree traversal to reach leaf nodes. Moreover, many-field implementations with current 5-tuple decomposition-based solutions will increase both process latency and storage requirement due to the merge process for those those additional fields.

Several research groups observed these new challenges in many-field packet classification and tried to come up with different solutions. On the one hand, some solutions avoid the repeated packet classifications on the same packet at different network nodes and free extra processing capacity. For example, tagging approach [2] classifies packets at the edge switch and inserts tags into packets for a fast packet forwarding in a network. Moreover, hashing approach [16] classifies the first packet in a flow and uses hash table lookups for the following packets to improve the performance of packet forwarding. On the other hand, some solutions improve the packet classification performance directly by designing new methods. For example, decomposition-based approach [19] uses bit vector with hash tables, decision-tree-based approach [6] uses 2-d pipeline architecture with tree-to-pipeline mapping scheme, and hybrid-based approach [4] uses effective bits with look-up tables to lower the computation cost and improve the system performance.

In this paper, we propose a scalable many-field packet classification algorithm using multidimensional-cutting via selective bit-concatenation (MC-SBC) for OpenFlow-based SDN switches. MC-SBC uses simple lookup tables to avoid computation divergence problems and leverages the high efficient instructions in a massive computation platform for a higher system throughput. Due to the sparsity and biased rule distribution in a ruleset, MC-SBC is designed to quickly find out few candidate rules using effective bits for the full match as shown in Figure 1. In summary, there are three main contributions of this work:

- MC-SBC demonstrates scalability on both the system performance and rule replication ratio with varying ruleset types and sizes.

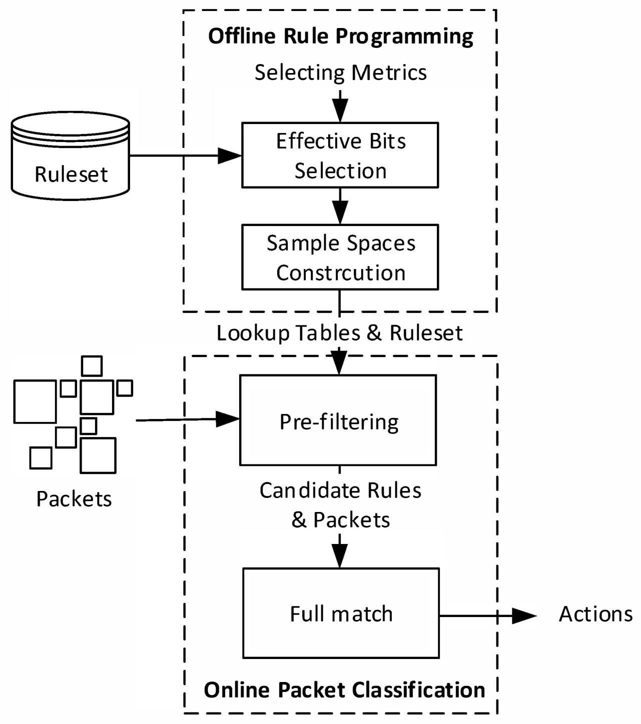


Figure 1: The proposed packet classification architecture. The system consists of off-line rule programming and on-line packet classification. Rule programming is conducted for the first time when a system is initiated or its ruleset is updated.

- MC-SBC designs a two-stage architecture with feasibility and flexibility with easy implementations by leveraging effective bit positions in a ruleset.
- A GPU prototype is implemented with packet classification throughput of 198 MPPS and 163 MPPS for 1K and 100K 15-field rules.

The remainder of this work is organized as follows. Section 2 gives background for related researches on packet classification problems and discusses the motivation of this paper. Section 3 presents the proposed architecture. Section 4 discusses the selecting metrics of effective bits. Section 5 gives an example of algorithm operations. Section 6 discusses the complexity analysis and shows the experiment results. Finally, the paper is concluded in Section 7.

2. BACKGROUND

In this section, we review major types of the packet classification algorithms to show the challenges and the trend of packet classification research area. Then we present the motivations for new many-field packet classification solutions.

2.1 Ternary Content-Addressable Memory Solutions

Ternary content-addressable memory (TCAM) is a specialized memory that can store and query three different types of data: 0, 1, and * (wildcard). TCAM solutions (TCAMs) employ multiple TCAM devices for each rule in a ruleset, and construct a parallel matching architecture for the target packet header fields with those pre-defined rules to fulfill the line rate requirement. However, for a ruleset with more packet header fields and bigger rule size, TCAMs need more TCAM devices and a complicated hardware implementation. Though TCAMs [9][13] leverage hardware parallel computation power to generate high packet classification throughput, they suffer problems such as high power consumption, high manufacturing cost, and low scalability [21][27].

2.2 Tuple Space Solutions

Besides of the hardware-based implementation like TCAMs, many researchers [16][25] also work on the software-based algorithmic solutions for a better system scalability with a lower implementation cost. Tuple space solutions are software-based and they leverage the fact that the number of distinct tuples is much less than that of rules in a ruleset. A tuple defines the number of significant bits in a prefix match field, the nesting level and range ID of a range field, and the existence of a value for an exact match field in a ruleset. Tuple-space-based solutions efficiently compress a ruleset by storing those valid bits of each field only. Besides, tuple-space-based solutions perform the search of each tuple independently and take advantage of parallelism. With the growth field number in a ruleset, both tuple number and tuple size increase. A longer processing latency could be expected.

2.3 Decomposition-based Solutions

Decomposition-based solutions [7][22][28] work on each field in a ruleset independently and merge the intermediate results from different fields as the final match result. Since each field is processed individually, more intermediate results will be generated with the growth of field number in a ruleset. The increasing merge stages of those additional fields in a ruleset result in a bigger memory requirement and longer processing latency. Recently, many-field decomposition-based solutions [18][17] leverage range-tree and hash functions to process the 15-field packet classification.

2.4 Decision-tree-based Solutions

Decision-tree-based approaches [8][11][24][28] analyze all fields in a ruleset to construct decision trees for efficient packet header lookup. For the matching process, decision-tree-based solutions traverse the tree by using individual field values to make branching decisions at each node until a leaf is reached. Tree depth and rule duplication in a decision tree affect the searching efficiency and memory requirement of one implementation. Both of them increase with the growth of field numbers which results in an exponential increase of memory requirement and increasing processing latency. Recently, many-field decision-tree-based solution [6] divides a ruleset into several subsets which have their own individual optimized decision trees and implement the design as a pipeline architecture on FPGA for good performance on a 12-field ruleset. An improved algorithm [20] leverages similar techniques and designs fine grained processing elements with a 2-dimensional pipelined architecture on FPGA with better performance for a 15-field ruleset.

2.5 Motivation of Many-field Packet Classification Solutions

New applications like OpenFlow Switch examine more than 15 fields in multiple lookup tables to categorize incoming packets into different flows [16]. For the development of

Rules	Field 1	Field 2	Field 3
r_1	0010	1101	1001
r_2	1001	000*	100*
r_3	1010	0110	1110
r_4	1110	1010	10**
r_5	0000	****	0000
r_6	****	110*	101*
r_7	1001	1010	0000
r_8	1111	110*	0000
r_9	0101	1010	101*
r_{10}	****	0110	101*
r_{11}	0001	0110	0000
r_{12}	0010	110*	101*
r_{13}	0100	000*	1110
r_{14}	1100	0010	000
r_{15}	1110	000*	0000
r_{16}	0011	110*	101*

Table 1: An example ruleset of sixteen rules with three fields

advanced network services such as SDN and NFV, the number of packet header fields in a rule are expected to grow in the future. However, the growing field number in a ruleset poses new challenges to packet classification problems in term of system throughput and storage requirement. To address the scalability problem and improve the system performance of many-field packet classification problem, it is necessary to lower the computation complexity with the number of rules and fields in a ruleset. In this paper, MS-SBC leverages the statistical characteristics between different bits in a ruleset to pre-compute lookup tables to lower computation complexity with the growth of field number and ruleset size.

The performance of a network application is affected by not only the implemented algorithm but also by the host platform. Heterogeneous system architecture is proposed to generate high system throughput with low implementation cost and high flexibility. However, it is a challenge to align different computation resources in a heterogeneous system. GPU is a typical commercial off-the-shelf device. Compared to other platforms such as Mulitcore General Processor (GPP) or FPGA, GPU supports fewer instruction but has more computation units to run in parallel. Besides, using GPU to implement an algorithm is a purely software-based solution and the implementation does not change GPU's hardware characteristics. Hence, GPU platforms provide a high flexibility on how to implement an algorithm and a high portability to move the designed algorithm to another GPU with similar architecture. With a proper alignment between the algorithm and GPU, a GPU can generate high throughput. All these benefits make GPU attractive for packet classification problems.

3. MC-SBC ARCHITECTURE

Packet classification multidimensional-cutting via selective bit-concatenation (MC-SBC) is implemented as a two-stage classification system with an off-line rule programming process to construct the lookup tables for the following on-line packet classification process as shown in Figure 1. This two-stage system shifts the computation cost to off-line stage to generate lookup tables to be used later to increase the system performance at on-line stage by doing simple lookups instead of data extracting and composition.

Algorithm 1: The pseudo code of off-line rule programming process.

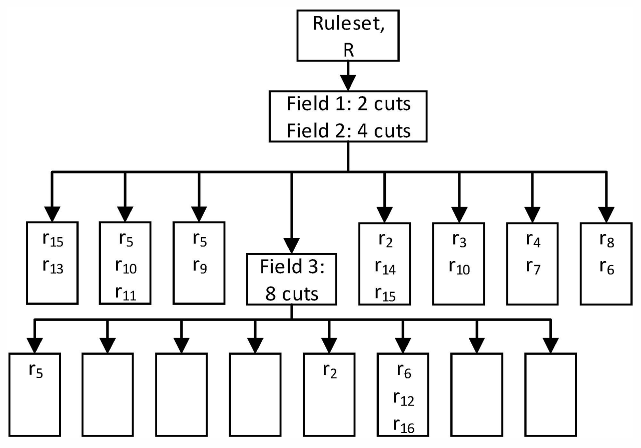
```
   input  : A many-field ruleset, $R$
   output : Effective Bit Sets, $EBS$
1  for All bits in a ruleset do
       // Get wildcard ratio
2      Get $P_{**}^{R^i,R^j}$;
       // Get diversity index
3      Get $Div^{R^i}$;
       // Get indepedence index
4      Get $Ind^{R^i,R^j}$;
       // Generate selection factor
5      $SF^{R^i,R^j} = \alpha \cdot P_{**}^{R^i,R^j} + \beta \cdot Div^{R^i} + \gamma \cdot Ind^{R^i,R^j}$;
6  end
7  Mark all bits as unused;
8  while Unused bits are available do
9      for All unused bits in a ruleset do
10         if $SF^{R^i,R^j}$ is maximum then
11             if $P_{**}^{R^i,R^j} > P_{th}$ and $Div^{R^i} > Div_{th}$ and
               $Ind^{R^i,R^j} > Ind_{th}$ then
12                 Append $R^i, R^j$ to current EBS;
13                 Mark $R^i, R^j$ as used;
14             end
15         end
16     end
17     Move to next EBS;
18 end
```

3.1 Off-line Rule Programming

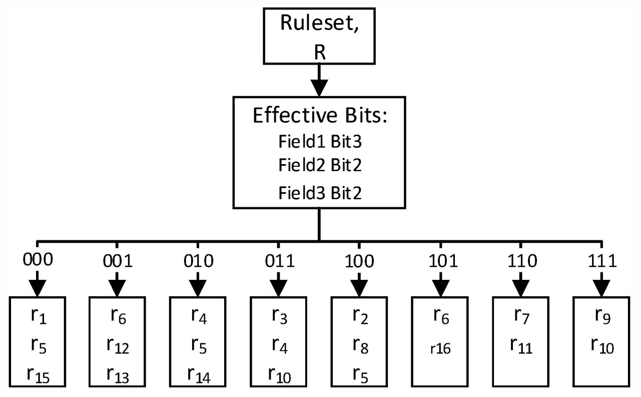
Off-line rule programming stage shifts the computation cost of field traversals in many-field packet classification using the pre-computed lookup tables. It stores the partition results of a ruleset into these lookup tables with rules' ID and uses effective bits as indices to access them. These effective bits are selected to provide the best discrimination between rules in a ruleset. Since only few effective bits are used, those bits could be quickly collected from a packet header. A lookup generates preliminary packet classification results without full comparison between rules and packets.

MC-SBC leverages wildcard ratios, independence indices, and diversity indices to identify effective bit positions in a ruleset and uses these bits to generate lookup tables. MC-SBC makes one lookup table by using one set of the effective bit positions of each rule. To save the required memory space and improve the processing latency, MC-SBC constructs multiple lookup tables with high independence to each other. The intermediate lookup results from each table are merged to become a much smaller candidate rule subset for the full match process in on-line packet classification stage.

To update lookup tables with low cost, each lookup table is stored discretely in system memory with two reference tables: the quantity table and the address table. The address table tells the memory address of each location in a lookup table. The quantity table tells the number of rule IDs in that location. Once there is an update in a ruleset, only those affected locations are updated. For example, if we want to add a new rule into a ruleset. By using the pre-defined effective bit position set, we can find the affected locations for



(a) HyperCuts



(b) MC-SBC

Figure 2: The examples of (a) HyperCuts and (b) MC-SBC data structures for a ruleset as in Table 1. Both algorithms are set up to have at most 3 rules in a leaf node with minimum rule duplication ratio.

this new rule. The address table tells where to find the data of rule IDs in memory and the quantity table tells how many rule IDs are stored there. MC-SBC modifies rule ID data at this location without interference to others then updates the quantity table.

The pseudo-code of rule programming process is shown in Algorithm 1 where R is the target ruleset, R^i is the bit value of i-th bit, $SF^{i,j}$ is the selection factor for i-th and j-th bits in a ruleset. $P_{**}^{R^i,R^j}$ is the wildcard ratio of i-th and j-th bit in a ruleset and the threshold is P_{th}. Div^{R^i} is the diversity index of i-th bit in a ruleset and the threshold is Div_{th}. Ind^{R^i,R^j} is the independence index of i-th and j-th bits in a ruleset and the threshold is Ind_{th}.

An example of the data structures between HyperCuts [23] and MC-SBC with a ruleset in Table 1 is shown in Figure 2. HyperCuts finds the best way to cut a ruleset heuristically with a bigger processing latency before constructing its data structure. MC-SBC detects those effective bits by using the statistical characteristics in a ruleset deterministically and creates its data structure directly. Besides, more leaf nodes could be found in this example for HyperCuts' data structure resulting in insufficient memory usage. Moreover, when verifying an incoming packet, HyperCuts checks each header field's value to make branch decisions at each node until a leaf node is reached. MC-SBC saves the time to access each field 's value with the concatenation value of those effective bits to make branch decision. Instead of conducting at most two separate lookups on the data structure used in HyperCuts to reach a leaf node, MC-SBC only needs one lookup on its data structure.

3.2 On-line Packet Classification

The on-line packet classification contains two processes: the pre-filtering process and the full match process. The pseudo code of on-line packet classification is shown in Algorithm 2 where r is the target rule and pkt is an incoming packet. The pre-filtering process is designed to find the related rule IDs for an incoming packet. It takes the values of effective bit position in a packet as lookup table indices and checks with pre-computed lookup tables for preliminary results. To accelerate the merging process, rule IDs in lookup tables are stored in order. Lookup results from different tables are merged to come out candidate rule IDs. All candidate rule IDs are then forwarded to the full match stage.

The full match process is designed to report matches between packets and candidate rules. It checks candidate rule IDs and retrieves both packet data and rule data in full. Then full match process compares them based on each field's characteristics, such as pre-fix match, range match, and exact match. During the matching process, if the match within a field fails, the full match process will move to next candidate rule immediately. Once there is a match with the highest priority rule, it terminates the matching process and reports this match. If there is no match, the full match process conducts the default action.

Algorithm 2: The pseudo code of on-line packet classification process

```
   input : A many-field ruleset, Effective bit position
           sets, Sample space lookup tables
   output: Match results
   // Get packet's sample value
 1 for All packet in queue do
 2     Get packet sample value;
       // Find sample space lookups
 3     Get the location of each space space based on
       sample value;
       // Find common rule ID
 4     for All Rule ID in each lookup do
 5         Get common rule from different lookups;
 6     end
       // Conduct full match
 7     Retrive packet data;
 8     for All candidate rules do
 9         Retrice rule data;
10         if $pkt = r^i$ then
11             Report match;
12         end
13     end
14 end
```

4. SELECTING METRICS FOR EFFECTIVE BIT POSITION

Effective bits are those bits in a ruleset which partition the ruleset evenly at best effort. At the off-line rule programming stage, MC-SBC uses effective bits to cut a ruleset effectively into subsets and generate lookup tables as the cutting results. At the on-line packet classification stage, MC-SBC uses the values of effective bits of a packet to find candidate rules for the full match process. Hence, the aim of MC-SBC is to make the subsets' sizes as small as possible and to distinguish different rules in an efficient way. Therefore, for a set of effective bit positions, any two of them are designed to be with low wildcard ratio ($P_{**}^{(i,j)}$) and high diversity (Div) to lower the total number of duplicated rules. Besides, the independence index (Ind) between any two bits is designed to avoid possible bias of data structure. To find out effective bits, MC-SBC leverages several key statistic characteristics in a many-field ruleset as discussed in the following sections.

4.1 Wildcard Ratio

Wildcard ratio measures how many wildcard symbols (*) appear on two chosen bit positions in a ruleset. A wildcard (*) symbol in a bit position means it could be either 0 or 1. When an effective bit position encounters a wildcard symbol in a specific rule due to the prefix and range fields, this wildcard symbol has to be converted into 0 and 1 to cover all possible matches and causes rule duplication. The duplication of a rule increases sharply when some rules have many wildcard symbols on those effective bits. For example, a rule has its sample value as 01**, and the rule ID of this rule will be duplicated to subsets 0100, 0101, 0110, and 0111 in a sample space. A higher wildcard ratio for a bit position means there are many wildcard symbols on this bit position and more duplications are produced. Therefore, more rules will be duplicated and distributed to different subsets in a lookup table, and it results in a higher replication ratio, bigger averaged number of rules in a subset, and longer processing latency.

MC-SBC chooses those bit positions with a lower wildcard ratio when other parameters are the same, and examines wildcard symbols for any two chosen bit positions in a ruleset to avoid the impacts caused by wildcard ratio. MC-SBC uses Equation 1 to estimate the *combined wildcard ratio* for any two chosen bit positions where N is the number of rules in a ruleset, i and j are for bit positions in a ruleset:

$$P_{**}^{(i,j)} = \frac{N_{**}^{(i,j)}}{N}. \qquad (1)$$

4.2 Independence Index

Bit positions with low correlations to each other can draw good distinctions between different rules when checking sample values, and *independence index* ensures that different bit positions are with high independence to each other. MC-SBC chooses a set of effective bit positions to guarantee the high independence among them. MC-SBC determines the independence index by the calculation for any two bit positions by the following Equation 2:

$$Ind^{(i,j)} = \sum_{x=0,1,*} \sum_{y=0,1,*} P_{xy}^{ij} - P_x^i \cdot P_y^j. \qquad (2)$$

4.3 Diversity Index

Diversity index ensures the distribution of rule number in the subsets is even for better performance and lower processing latency. The diversity index of a bit position is calculated by calculating the entropy of the distribution on 0 and 1 as Equation 3:

$$Div^i = -\frac{P_0^i}{1-P_*^i} log \frac{P_0^i}{1-P_*^i} - \frac{P_1^i}{1-P_*^i} log \frac{P_1^i}{1-P_*^i}. \qquad (3)$$

With a high diversity index, the subset sizes are distributed more evenly, and if the subset sizes are with uniform distribution, the processing latency under the worst case can be greatly reduced for a given duplication.

The above indices can be generalized to the cases of many bit positions (even two sets of bit positions), and the corresponding definition can be derived in a similar way. The proposed algorithm becomes more accurate when more bit positions are evolved. However, more computational cost will arise. Thus, we use these indices of one or two bit positions to reduce the computational complexity and the general cases for them are neglected here. Examples of how to implement MC-SBC are discussed in the next section.

5. EXAMPLES OF MC-SBC IMPLEMENTATION

Examples of MS-SBC algorithm are given in this section to show how MC-SBC works as a two-stage system for many-field packet classification. In the off-line rule programming stage, an example is given to illustrate how MC-SBC detects effective bits from a ruleset, uses those bits to create lookup tables, and updates those tables when a ruleset changes. In the on-line packet classification stage, an example is given to illustrate how the prefiltering process uses a packet header data with those pre-computed lookup tables and extracts candidate rules by merging intermediate results from table lookups.

5.1 Effective Bits Selection

MC-SBC uses only few effective bits in a ruleset to construct lookup tables to accelerate the overall processing speed. Since only those effective bits in the ruleset are used, MC-SBC converts range match fields to prefix match fields with an expanded value to avoid rule explosion problem but still keeps the characteristics of each field for the packet classification. For an effective bit position set, MC-SBC finds any two bit positions where Div is big, Ind is small, and P_{**} is small. Here, $R^{(i,j)}$ is denoted as the bit position j in field i of a ruleset R. For the ruleset as in Table 1, $R^{(1,1)}$, $R^{(1,2)}$, $R^{(1,3)}$, $R^{(1,4)}$, $R^{(2,1)}$, $R^{(2,2)}$, $R^{(2,3)}$, $R^{(3,1)}$, and $R^{(3,3)}$ are with higher Div indices compared to other bits for an effective bit position set. However, if $R^{(1,1)}$, $R^{(1,2)}$, $R^{(1,3)}$, $R^{(1,4)}$ or $R^{(2,1)}$, $R^{(2,2)}$, $R^{(2,3)}$ are in the same set, a higher P_{**} index of that set is found and it results in a higher rule duplication. Hence, there are at most two sets with up to 3 elements in each set: $\{R^{(1,1)} \vee R^{(1,2)} \vee R^{(1,3)} \vee R^{(1,4)}, R^{(2,1)} \vee R^{(2,2)} \vee R^{(2,3)}, R^{(3,3)} \vee R^{(3,4)}\}$ or at most 24 sets with up to 2 elements in each set. To decide the of the elements in a set, Ind indices are used and $\{R^{(2,3)}, R^{(3,3)}\}$, $\{R^{(1,1)}, R^{(2,1)}\}$, and $\{R^{(1,4)}, R^{(3,1)}\}$ are found as effective bit position sets which have smaller Ind values. Thus those effective bit position sets are good to divide a ruleset into smaller groups and construct lookup tables for each group by using other effective bit position sets.

The effective bit position sets decide the size of a lookup table, the number of groups for a ruleset, and how many lookup tables can be generated for a ruleset. We take the same matric as mentioned in Section 4 for differnt types of ClassBench rulesets. Table 2 shows the number of effective bits we can find after the rule programming on 5-tuple ClassBench rulesets. The available number of effective bits varies from one ruleset to another but there are sufficient bits from most fields. With this observation, a many-field synthetic ruleset with sufficient bit position sets is assumed to generate lookup tables with low dependence to each other.

Ruleset	SA	DA	Ptrl	SP	DP
ACL	4.36	6.64	2.93	1.14	8.64
FW	1.64	1.51	2.43	5.14	6.93
IPC	2.05	2.89	2.67	5.01	7.71

Table 2: The experimental average number of effective bits from the traditional 5 tuples for different rulesets

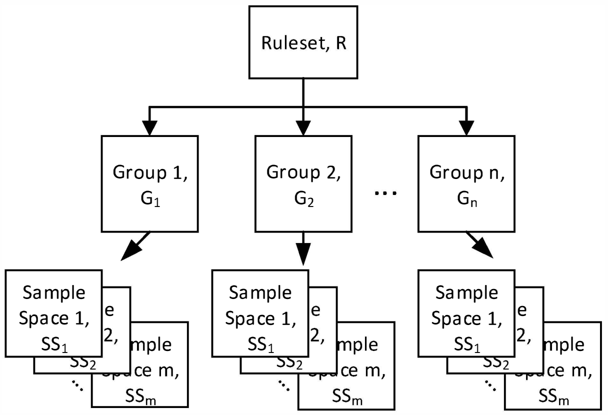


Figure 3: An example of the relationship between ruleset, groups, and sample spaces. A ruleset is divided into several small group. Each subset creates its own sample spaces.

5.2 Lookup tables construction

To improve system scalability and performance, MC-SBC divides a ruleset into smaller groups with minimum overlaps to each other and constructs lookup tables for each group separately. MC-SBC collects the target rules by putting rules with same attributes into a smaller subset and gives each subset a group index. Later, when doing a lookup, MC-SBC can only check those rules with same attributes by using group indices without wasting resources on those unrelated rules. For a big ruleset size, MC-SBC can generate more groups to maintain system scalability and performance. When the groups of a ruleset are ready, MC-SBC creates lookup tables for each group to maintain the system performance and memory requirements. The relationship among a ruleset, groups, and lookup tables is shown in Figure 3.

An example of the lookup table construction process is shown in Figure 5 for a ruleset as in Table 1. From Section 5.1, multiple effective bit position sets could be found from the example ruleset and there are three sets with the

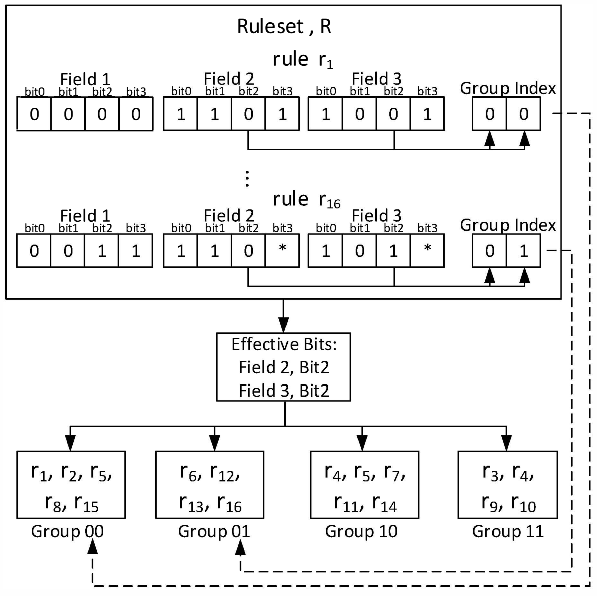


Figure 4: An example of the rule programming process on subset construction. In this example, we use the effective set: $\{R^{(2,3)}, R^{(3,3)}\}$ to generate the group indices. The ruleset is divided into several groups based the group indices.

best ruleset discrimination: $\{R^{(2,3)}, R^{(3,3)}\}$, $\{R^{(1,1)}, R^{(2,1)}\}$, and $\{R^{(1,4)}, R^{(3,1)}\}$. The first set is used to generate the group index and the following two sets are used to construct lookup tables. For example, the concatenation value of $\{R^{(2,3)}$ and $R^{(3,3)}\}$ in rule r_{16} is 01 and it acts as a group index as shown in Figure 4. Set $\{R^{(1,1)}, R^{(2,1)}\}$ and $\{R^{(1,4)}, R^{(3,1)}\}$ are used for lookup tables. In the group 00, there are 5 rules in it: r_1, r_2, r_5, r_8, and r_{15} and the lookup tables are shown in Figure 5.

When a ruleset is updated, only those related groups and lookup tables are updated and the following part remains unchanged. To update these lookup tables, MC-SBC finds which groups the target rule belongs to originally and removes the target rule from all related locations in the sample spaces. For example, to update r_{15}:[1110, 000*, 0000] as r'_{15}:[1110,000*,1111], r_{15} belongs to group 00 but now r'_{15} belongs to group 01. Hence, r_{15} has to be removed from lookup tables of group 00 and r'_{15} has to be inserted into group 01. Instead of checking all group 00 sample spaces, r_{15} is stored in location 10 of the first lookup table and in location 11 of the second table of group 00 by its original sample values. Hence, MC-SBC checks the address table to find where to modify in the memory space, and updates the quantity table with the updated number of rules in both locations. Then MC-SBC inserts r'_{15} into in group 01 lookup tables and updates the associated reference tables accordingly.

5.3 Pre-filtering

The pre-filtering process merges the lookup results from each sample space to extract candidate rules for the full match process. It takes the group index and the sample values of an incoming packet to find out which group's lookup tables should be used to retrieve rule IDs in a target location. The pre-filtering process then extracts candidate rules from the

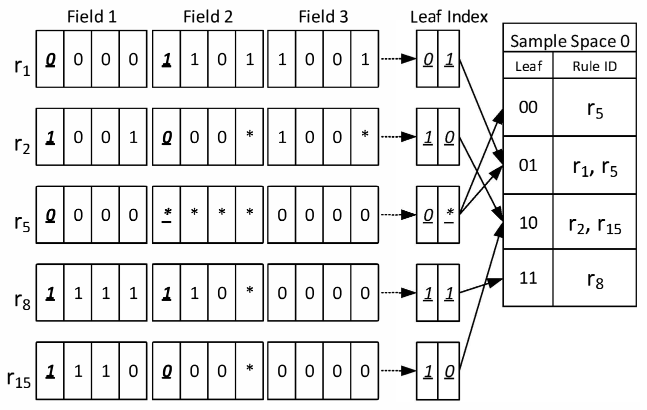


Figure 5: An example of rule programming process on lookup table construction for group 00 with r_1,r_2,r_5, r_8 and r_{15} in it. The concatenation of bit values is used to lookup the different locations in a sample space for the rule IDs. Rule IDs with same sample values are kept in the same location.

lookup results of different sample spaces for the following full match process.

For example, to verify an incoming packet p_y as shown in Figure 6, MC-SBC takes the group index and table indices from p_y, and uses them to check the lookup tables for rule IDs. Hence, p_y has 00 as its group index, 10 and 00 as the table indices for lookup tables. Thus, MC-SBC gets r_2, r_{15} from location 10 of first lookup table and r_5,r_{15} from location 00 of second lookup table in group 00. By merging the lookup results, the pre-filtering process generates r_{15} as the candidate rule ID and passes it to the following full match process.

5.4 Full match

To accelerate the matching process, the pre-filtering process uses only the rule ID to find out the candidate rules for the full match process for each incoming packet. However, a lookup only shows the match result on those effective bit positions between a rule and the incoming packet. To guarantee a match between packets and rules, a full comparison between each header field of a packet and the candidate rules is necessary. The full match process retrieves those candidate rules' data and packet data in full for the matching process and reports the matching results. With only few rules, we are able to implement those efficient match algorithms and derive the match results quickly at the full match stage. Other solutions can be implemented at this stage for a faster matching process without the restrictions caused by the complexity on a ruleset. To demonstrate comparable results with other algorithmic solutions, we only conduct the linear search at the full match process.

6. RESULTS

In this section, the theoretical results of time and space complexity are given to demonstrate how MC-SBC reduces the complexity by using multiple lookup tables. Then the experimental results are given to show the feasibility and effectiveness of the proposed MS-SBC algorithm with the comparison along with other existing many-field packet classification solutions.

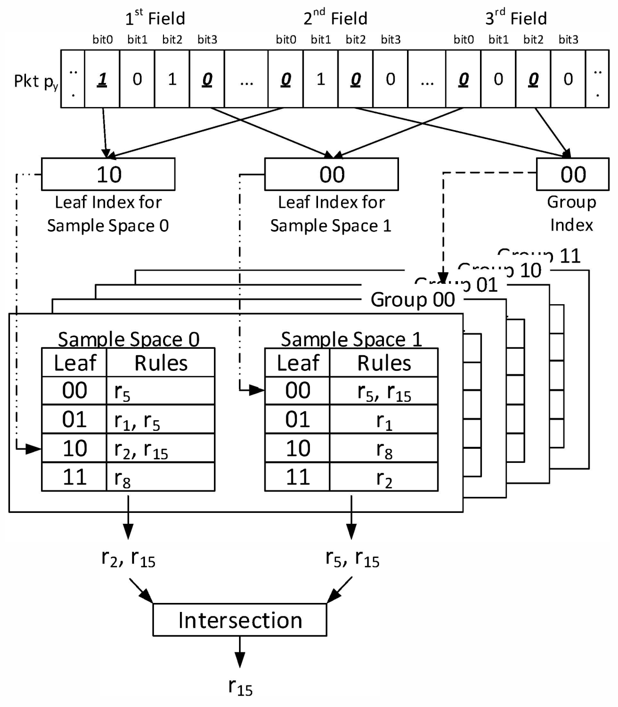


Figure 6: An example of the pre-filtering process for packet p_y. The subset index and sample values of packet p_y are generated with effective bit position sets used for sample space construction. The group index is used to decide which group's sample spaces should be used for the look-ups. The packet's sample values are used for lookups in different sample spaces. The intersection process in pre-filtering derives the candidate rules and pass them to the following full match process.

6.1 Theoretical Results

In this section, the time and space complexity are given for both D^2BS [28] and MC-SBC at first to show how MC-SBC can lower the complexity by using multiple lookup tables. Then a example with practical setting is given to show the performance of MC-SBC.

6.1.1 Time Complexity

Assume there are m rules in a ruleset, R, and each rule has h fields in it. For this ruleset, there are up to q effective bit position sets and each set has up to k bits in it. When using one set of effective bit position to partition R, R could be divided into at most 2^k subset. Due to wildcard symbol in the ruleset, one rule could appear in multiple subsets. Hence, the replication ratio caused by each effective bit is ρ and each subset could have up to $\rho^k \cdot m$ rules. When the partition result is stored as a lookup table and uses the concatenation values of those effective bits as indexes for the table, there are up to q lookup tables for this ruleset and only rule's ID is stored in the lookup table. Based on the selecting metric of effective bits, the rule distribution between different lookup tables is designed to be different. Hence, there are up to α rules will be the same from any subset in two different lookup tables.

D^2BS and MC-SBC use similar effective bit concept to cut a ruleset. By following the definition in D^2BS, the time

Algorithm	Time Complexity	Space Complexity
D^{2}BS	$O(\rho^k \cdot m \cdot c(h))$	$O((2 \cdot \rho)^k \cdot m)$
MC-SBC	$O(\rho^k \cdot m(q + \alpha^{q-1} \cdot c(h)))$	$O(q \cdot (2 \cdot \rho)^k \cdot m)$

Table 3: Complexity comparison between D^{2}BS [28] and MC-SBC where ρ is the average replication ratio, m is the number of rules in a ruleset, h is the number of fields in a rule, q is the number of effective bit sets, k is the number of effective bits, and α is the duplicated ratio between subsets in two lookup tables.

complexity to find a rule ID is $O(\rho^k \cdot m)$. When more effective bits are chosen, less rule IDs are stored in a subset. However, the number of fields in a rule also impacts the time complexity to generate the results of packet classification. Hence, the time complexity of D^{2}BS with full match process is expanded as $O(\rho^k \cdot m \cdot c(h))$, where $c(h)$ depends on the implemented matching algorithm and the number of field in a rule.

The time complexity of MC-SBC is the sum of the cost to find candidate rules from different tables and the cost to compare candidate rules with a packet. However, MC-SBC allows multiple lookup tables to lower the complexity when doing the full match. Hence, the time complexity of MC-SBC is $O(q \cdot \rho^k \cdot m + \rho^k \cdot m \cdot \alpha^{q-1} \cdot c(h)) = O(\rho^k \cdot m(q + \alpha^{q-1} \cdot c(h)))$ where $q \cdot \rho^k \cdot m$ is the cost to find candidate rules from q different tables with ordered sorting and $q \cdot \rho^k \cdot \alpha^{q-1} \cdot c(h)$ is the cost to match a packet with candidate rules.

6.1.2 Space Complexity

Both D^{2}BS and MC-SBC store rule IDs in their data structures as pointers to the full rule data. When the classification process is launched, the full rule data will be retrieved and compared with incoming packets. Hence, data structures in D^{2}BS and MC-SBC are not increasing proportionally with the number of field in a ruleset. By following the definition in D^{2}BS, the space complexity of D^{2}BS is $O(2^k \cdot \rho^k \cdot m) = O((2 \cdot \rho)^k \cdot m)$.

MC-SBC provides the flexibility to trade space complexity for time complexity. With the multiple lookup tables, MC-SBC can checks only rule IDs to quickly filter out unrelated rules to lower the number of rules for full match. When there are more lookup tables used in MC-SBC, the space complexity of MS-SBC is increasing proportionally. Hence, the space complexity of MC-SBC is $O(q \cdot (2 \cdot \rho)^k \cdot m)$.

6.1.3 Complexity Comparison

The comparison between D^{2}BS and MC-SBC is shown as Table 3. MC-SBC provides the level of time and space complexity as D^{2}BS under the same setting. Moreover, MS-SBC provides the flexibility to lower the time complexity at the cost of the increasing space complexity. The cost to examine each packet for a match result is proportional to the number of fields in a ruleset. For applications like OpenFlow switch that needs more fields in a packet to be examined, MC-SBC is able to support this application development trend.

For example, the probability of 0, 1, and *(wildcard) of for all fields in a random generated ruleset with 100K rules are $(P_0, P_1, P_*) = (0.45, 0.45, 0.1)$. Assume two sets of effective bits are found and each set has 15 effective bits. Hence, there are at most 0.00013 similar rules in average from any subset in two different lookups. Hence, ρ is 0.55 and $/alpha$ is 0.00013 in this case. Assume only linear search is conducted and the cost to match each field is equal for both D^{2}BS and MC-SBC. With D^{2}BS, the cost to find a match result is: $0.55^{15} \cdot 100000 \cdot 15 = 191$ unit cost and the storage requirement is: $0.55^{15} \cdot 100000 \cdot 2^{15} = 417725$ unit space. With MC-SBC, the cost to find a match result is: $0.55^{15} \cdot 100000(2 + 0.00013 \cdot 15) = 26$ unit cost and the storage requirement is: $2 \cdot 0.55^{15} \cdot 100000 = 835450$ unit space. By doubling the space requirement, MC-SBC could decrease the time complexity to about one seventh of the original value.

6.2 Experimental Results

In this section, MC-SBC is compared with other solutions in terms of data structure and system performance. The off-line rule programming stage in MC-SBC generates lookup tables for the on-line packet classification stage. The lookup tables are similar concepts with trees in decision-based solutions. The subset of each lookup table is actually the lead node in a decision tree. To show the efficiency of those lookup tables, MC-SBC is compared to those existing decision-based solutions with three key parameters: tree depth (D_t), replication ratio (f), and the maximum number of rules stored on a leaf node ($binth$). The on-line packet classification stage in MC-SBC affects the overall system performance. To keep the flexibility of implementation with high system throughput, MC-SBC is implemented on a GPU with different optimization strategies. Moreover, MC-SBC is compared with other many-field packet classification solutions on different platforms using the synthetic rulesets with same setting.

6.2.1 Experiment Setup

All experiments are conducted on an Intel Xeon E5410 CPU machine with 4GB DDR2 RAM as the main memory. This machine is equipped a NVIDIA K20C GPU [14] with 13 streaming multiprocessors (SMXs) and 5Gb GDDR5 memory for general computation. Each SMX has 192 single-precision CUDA cores, 64 double-precision units, 32 special function units, and 32 load/store units. Besides, MC-SBC is implemented on the Debian 7.3 64-bit operating system with Cuda 7.0 as the software development environment. All necessary packets and tables are assumed to be ready on GPU before the packet classification process starts. This is a feasible and practical assumption because in the network system there are an abundant number of packets buffered to be processed in a network system [29]. In this paper, the synthetic rulesets are generated by taking ClassBench ruleset for the traditional 5-tuple first [26]. We attach 10 additional fields with wildcard ratio as 0.1 and the unique values setting as in [18] to those ClassBench rulesets to create the synthetic rulesets for the experiments.

6.2.2 Off-line Rule Programming Results

The off-line rule programming stage determines the size of each subset in a lookup table and impacts the overall system performance. The characteristics of a ruleset and the selecting criteria both play key roles at this stage. Both ideal and synthetic rulesets are verified to show the mixed interaction between system setting and varying ruleset characteristics.

6.2.2.1 Results with ideal rulesets.

An ideal ruleset provides more effective bits compared to practical rulesets and the distribution of rule IDs are evenly.

# of Subsets	Wildcard Ratio (%)	Binth			Candidate Rules		
		Sample Space = 1			Sample Space = 2		
		Space size (bits)			Space size (bits)		
		5	10	15	5	10	15
2	0	156K	4K	108	4K	5	11
	0.1	157K	4K	156	5K	6	1
	1	164K	5K	178	6K	11	1
	10	251K	12K	640	23K	280	4
8	0	39K	1K	27	1K	2	1
	0.1	39K	1K	39	1K	2	1
	1	41K	1K	45	1K	3	1
	10	62K	3K	160	5K	70	1
32	0	9K	306	10	306	1	1
	0.1	9K	310	10	315	1	1
	1	10K	355	12	405	1	1
	10	15K	793	40	1K	18	1

Table 4: Binth and candidate rules for a ruleset with 10 M rules by giving different subset and space size settings. All fields in the ruleset varying with the field wildcard ratio for each setting. Binth shows how many rule are stored in a leaf node of a sample space. Candidate rules are the intersection results of any two sample spaces.

The analysis of ideal rulesets can show a trend to design a MC-SBC system. The *binth* of a lookup table is affected by bit position wildcard ratio, subset size, sample space size, and the number of lookup tables. We evaluate field wildcard ratio ranging from 0.01 % to 10%. The evaluation of ideal ruleset is summarized as in Table 4. More subsets and bigger sample space will decrease the number of rules stored in each leaf node (binth) for higher system performance. However, the wildcard ratio for each field in a ruleset increases binth and results in longer processing latency and a higher memory requirement. By changing these design factors, an improved system could found with a trade-off between system performance and memory requirement.

6.2.2.2 *Results with synthetic rulesets.*

The evaluation of MC-SBC with synthetic rulesets is conducted to compare the data structure of MC-SBC with other well-known algorithms like BC, SBC, Hi-Cuts, and Hyper-Cuts to show the effectiveness of the proposed system. MC-SBC is implemented with 3 major types of ClassBench rulesets. The trade-off between *binth* and f helps a designer to choose a setting which is suitable for the implementation environment.

Figure 7a shows the maximum number of rules per leaf node and Figure 7b shows the maximum replication ratio in a sampling space with subsets. The number of rules per leaf node (binth) decreases when the number of subsets increases. Binth also increases when the number of rules in a ruleset increases. The replication ratio increases when the number of subsets increases. However, it does not increase when the number of rules in the same ruleset increases. The proposed method controls the replication ratio to achieve better scalability in terms of storage, and provides the flexibility to trade the number of subsets for the number of rules stored in a leaf node, at the cost of memory storage. Besides, the binth of synthetic ruleset is bigger and does not decrease as fast as the ideal ruleset. When more effective bits are chosen, the dependence between effective bits becomes stronger and it weakens the effectiveness of MC-SBC.

MC-SBC is compared with other decision-tree-based solutions such as BC, SBC, HiCuts, and HyperCuts. Table 5 shows the comparison result with other algorithms. All algorithms are set to have the same number of rules stored in a leaf node (*binth*) to count accesses of tree traversal (D_t) and the rule replication ratio (f). The proposed method (MC-SBC) is able to maintain the same level of binth and replication ratio with lower and deterministic tree depth and fast searching time.

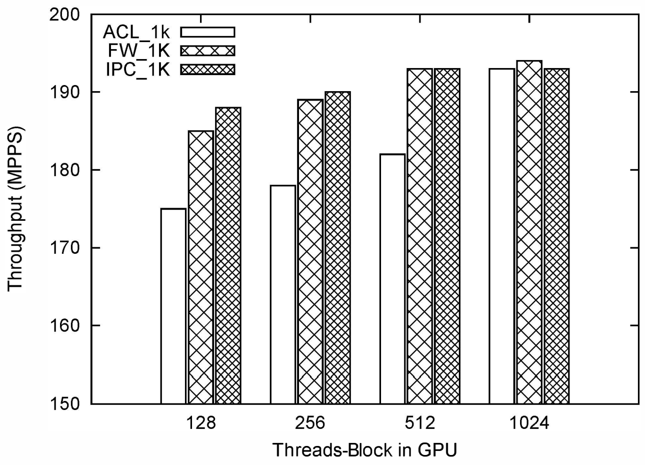
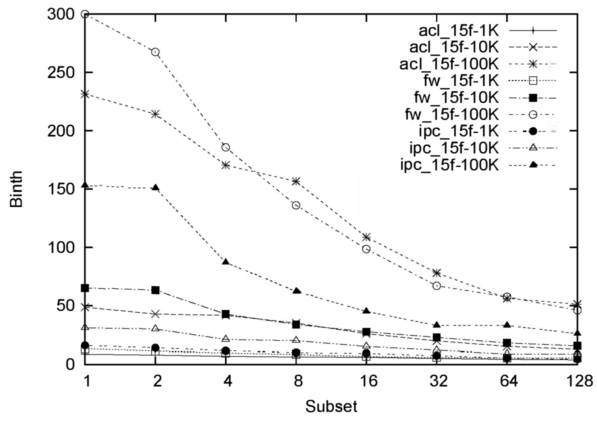


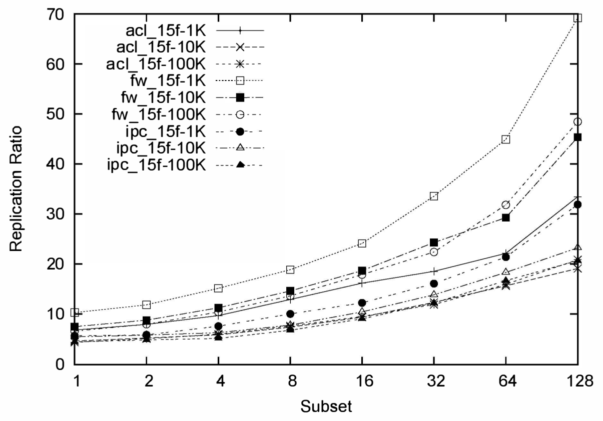
Figure 8: System throughput of GPU prototype implementation with varying GPU thread-block setting.

6.2.3 *On-line Packet Classification Results*

A GPU prototype is implemented to show the performance of the proposed system with comparisons to the other many-field packet classification solutions on different platforms. Each packet is designed to have 15 fields in it and the size of a packet header is 354 bits as defined in OpenFlow. The system throughput is calculated based on the processing time of GPU kernel by CUDA Visual Profiler. The time duration of the GPU kernel is used as process latency for packet classification process. We make an assumption that packet data could be pre-loaded into GPU memory. This is a feasible and practical assumption because in a network system there are abundant packets in a network device waiting to be processed. Therefore, we can pre-load packet data to hide the transfer latency. However, the up and coming memory architectures may remove this bottleneck and allow for the full performance potential of MC-SBC. To avoid the performance penalty of kernel synchronization, we implement only one kernel to run both pre-filtering and full match with buffers to store packet subset indices, packet sample values, and suspected packets. Once the prefiltering process is done, the same kernel is going to run full match process and report the match results for each thread. Figure 8 shows the impact on system performance with different GPU thread-block setting. MC-SBC is able to mitigate the variance between different types of rulesets and generate consistent performance. Besides, MC-SBC is also able to utilize the massive computation platform by generating a higher system performance with more threads and higher instructions level parallelism.



(a) The number of rules in a leaf node (binth)



(b) The replication ratio (f)

Figure 7: The experimental maximum number of rules per leaf node (binth) and the replication ratio (f) by dividing a ruleset into 1 to 128 subsets with 15-bit sample space for each subset

Ruleset	binth	BC		SBC		HiCuts		HyperCuts		MC-SBC	
		5-field ruleset								15-field ruleset	
		Dt	f	Dt	f	Dt	f	Dt	f	Dt	f
ACL_1K	6	5	9	5	6	66	20	17	5	2	12
ACL_10K	10	5	41	5	25	63	34	14	3	2	16
ACL_100K	46	5	93	5	56	55	106	14	7	2	28
FW_1K	11	5	42	5	27	64	98	22	3	2	12
FW_10K	17	5	157	5	107	63	454	18	122	2	29
FW_100K	54	6	7683	6	6237	85	151514	18	825	2	48
IPC_1K	5	5	12	4	10	70	11	22	3	2	6
IPC_10K	16	6	66	6	33	78	270	19	279	2	10
IPC_100K	39	6	277	6	101	76	738	18	359	2	16

Table 5: The comparison between different algorithms: Boundary Cutting(BC), Selective Boundary Cutting(SBC), HiCuts, HyperCuts [12], and Multidimensional-Cutting Via Selective Bit-Concatenation(MC-SBC)

Based on the observation of ClassBench rulesets, we can find that a field normally has less than 10% as wildcards for a ruleset. However, the wildcard ratio of a field does impact the effectiveness to sample a ruleset to construct operational lookup tables. Hence, the system performance and memory requirement for varying number of rules with different wildcard setting is shown in Figure 9 to show the feasibility of the proposed system. Normally, only few fields in a ruleset are with high wildcard ratio compared to other fields. MC-SBC can mitigate this problem by finding only effective bits. In Figure 9, all field are set up with a high wildcard ratio as the worst scenario and MC-CBS still maintains a good scalability in term of wildcard ratio.

Figure 10 shows the comparison of system throughput with other many-field packet classification algorithms such as GPP-BV [18] and GPU-BV [19]. The processing latency is about 4 ms in GPP-BV when processing 64 packets with 32 K rules and is about 22 ms in GPU-BV when processing 81920 packets with 32 K rules. However, the processing latency of this work is about 80 ms when processing 26624 packets with 32 K rules. The proposed system is able to maintain scalability with the increasing ruleset size and the growth field number in a ruleset. Although FPGA solutions [5][19] are with a higher throughput (upto 650 MPPS), the FPGA platform is constrained by its limited memory space and

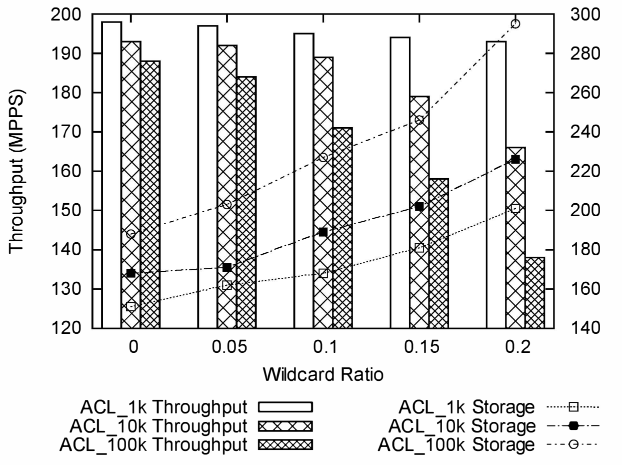


Figure 9: System throughput and memory requirement of GPU prototype implementation for different 15-field rulesets with varying wildcard setting.

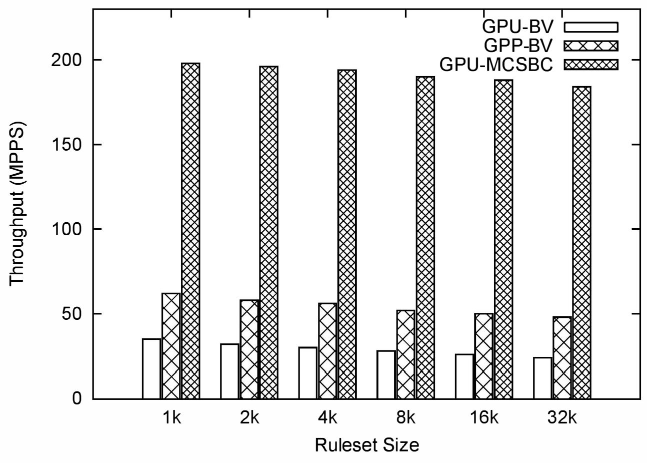


Figure 10: The comparison between GPP-BV [18], GPU-BV [19], and GPU-MCSBC(this paper).

can only support a ruleset with around 1K 15-field rules in it. Hence, the comparison with FPGA platforms is not conducted here.

7. CONCLUSION AND DISCUSSION

In this paper, we present a many-field packet classification algorithm by extracting only few candidate rules for full match process to improve the system performance. The proposed method converts a huge and biased rule space into several small subsets by using key selecting metrics to construct flat data structures for fast processing and easy updating. Both data structures and matching processes in the proposed system are designed for massive computation platforms to hide short latency tasks behind a long access latency task for a better system performance. The proposed method is examined with ideal and synthetic rulesets with all key factors related to system performance. Besides, the cutting results are compared with well known cutting algorithms to show the effectiveness. The throughput of GPU prototype achieves around 198 MPPS for 1K 15-field rules and around 163 MPPS for 100K 15-field ruleset. The prototype result is also compared with other many-field implementations in terms of scalability and throughput. This paper demonstrates the feasibility of Openflow-based SDN switches using software-based packet classification instead of TCAM solutions.

8. ACKNOWLEDGMENTS

We want to thank Dr. Wei Wei from Beihang University for his help on the mathematical problem formulation when he was a visiting professor at Southern Illinois University.

9. REFERENCES

[1] F. Baboescu and G. Varghese. Scalable packet classification. volume 31, pages 199–210, 2001.

[2] H. Farhadi and A. Nakao. Rethinking flow classification in sdn. In *2014 IEEE International Conference on Cloud Engineering (IC2E)*, pages 598–603, 2014.

[3] P. Gupta and N. Mckeown. Classifying packets with hierarchical intelligent cuttings. *IEEE Micro*, 20(1):34–41, 2000.

[4] C. Hsieh and N. Weng. Scalable many-field packet classification using multidimensional-cutting via selective bit-concatenation. In *Proceedings of the Eleventh ACM/IEEE Symposium on Architectures for networking and communications systems*, pages 187–188, 2015.

[5] W. Jiang and V. K. Prasanna. Field-split parallel architecture for high performance multi-match packet classification using fpgas. In *Proceedings of the Twenty-first Annual Symposium on Parallelism in Algorithms and Architectures*, pages 188–196, 2009.

[6] W. Jiang and V. K. Prasanna. Scalable packet classification on fpga. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 20(9):1668–1680, 2012.

[7] W. Jiang, V. K. Prasanna, and T. Ganegedara. A scalable and modular architecture for high-performance packet classification. *IEEE Transactions on Parallel and Distributed Systems*, 25(5):1135–1144, 2014.

[8] A. Kennedy and X. Wang. Ultra-high throughput low-power packet classification. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(2):286–299, 2014.

[9] K. Kogan, S. Nikolenko, O. Rottenstreich, W. Culhane, and P. Eugster. Sax-pac (scalable and expressive packet classification). In *Proceedings of the 2014 ACM Conference on SIGCOMM*, pages 15–26, 2014.

[10] T. V. Lakshman and D. Stiliadis. High-speed policy-based packet forwarding using efficient multi-dimensional range matching. In *Proceedings of the ACM SIGCOMM 1998 Conference*, pages 203–214, 1998.

[11] H. Lim, N. Lee, G. Jin, J. Lee, Y. Choi, and C. Yim. Boundary cutting for packet classification. *IEEE/ACM Transactions on Networking*, 22(2):443–456, 2014.

[12] H. Lim, N. Lee, G. Jin, J. Lee, Y. Choi, and C. Yim. Boundary cutting for packet classification. *IEEE/ACM Transactions on Networking*, 22(2):443–456, 2014.

[13] Y. Ma and S. Banerjee. A smart pre-classifier to reduce power consumption of tcams for multi-dimensional packet classification. 42(4):335–346, 2012.

[14] Nvidia Corp. *Nvidia K20 GPU Accelerator*, 2012. http://www.nvidia.com/content/PDF/kepler/Tesla-K20ActiveBD06499001v02.pdf.

[15] Open Networking Foundation. *OpenFlow Switch Specification, Version 1.5.1*, 2015. https://www.opennetworking.org/images/stories /downloads/sdn-resources/onf-specifications /openflow/openflow-switch-v1.5.1.pdf.

[16] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado. The design and implementation of open vswitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 117–130, 2015.

[17] Y. Qu, S. Zhou, and V. K. Prasanna. Scalable many-field packet classification on multi-core processors. In *2013 25th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 33–40, 2013.

[18] Y. Qu, S. Zhou, and V. K. Prasanna. A decomposition-based approach for scalable many-field

packet classification on multi-core processors. *International Journal of Parallel Programming*, 438(6):965–987, 2014.

[19] Y. R. Qu, H. H. Zhang, S. Zhou, and V. K. Prasanna. Optimizing many-field packet classification on fpga, multi-core general purpose processor, and gpu. In *Proceedings of the Eleventh ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, pages 87–98, 2015.

[20] Y. R. Qu, S. Zhou, and V. K. Prasanna. High-performance architecture for dynamically updatable packet classification on fpga. In *2013 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pages 125–136, 2013.

[21] O. Rottenstreich, I. Keslassy, A. Hassidim, H. Kaplan, and E. Porat. On finding an optimal tcam encoding scheme for packet classification. In *2013 Proceedings IEEE INFOCOM*, pages 2049–2057, 2013.

[22] A. Sanny, T. Ganegedara, and V. K. Prasanna. A comparison of ruleset feature independent packet classification engines on fpga. In *IPDPS Workshops*, pages 124–133, 2013.

[23] S. Singh, F. Baboescu, G. Varghese, and J. Wang. Packet classification using multidimensional cutting. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 213–224, 2003.

[24] H. Song and J. S. Turner. Abc: Adaptive binary cuttings for multidimensional packet classification. *IEEE/ACM Transactions on Networking*, 21(1):98–109, 2013.

[25] V. Srinivasan, S. Suri, and G. Varghese. Packet classification using tuple space search. *ACM SIGCOMM Computer Communication Review*, 29(4):135–146, 1999.

[26] D. E. Taylor and J. S. Turner. Classbench: A packet classification benchmark. *IEEE/ACM Transactions on Networking*, 15(3).

[27] B. Vamanan and T. N. Vijaykumar. Treecam: Decoupling updates and lookups in packet classification. In *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*, pages 1–12, 2011.

[28] B. Yang, J. Fong, W. Jiang, Y. Xue, and J. Li. Practical multituple packet classification using dynamic discrete bit selection. *IEEE Transactions on Computers*, 63(2):424–434, 2014.

[29] S. Zhou, S. Singapura, and V. Prasanna. High-performance packet classification on gpu. In *2014 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–6, 2014.