sustainability

MDPI

Article

Enabling Packet Classification with Low Update Latency for SDN Switch on FPGA

Chenglong Li *, Tao Li, Junnan Li, Zilin Shi and Baosheng Wang

Computer College, National University of Defense Technology, Changsha 410073, China
* Correspondence: lichenglong17@nudt.edu.cn

Received: 2 March 2020; Accepted: 4 April 2020; Published: 11 April 2020

Abstract: Field Programmable Gate Array (FPGA) is widely used in real-time network processing such as Software-Defined Networking (SDN) switch due to high performance and programmability. Bit-Vector (BV)-based approaches can implement high-performance multi-field packet classification, on FPGA, which is the core function of the SDN switch. However, the SDN switch requires not only high performance but also low update latency to avoid controller failure. Unfortunately, the update latency of BV-based approaches is inversely proportional to the number of rules, which means can hardly support the SDN switch effectively. It is reasonable to split the ruleset into sub-rulesets that can be performed in parallel, thereby reducing update latency. We thus present SplitBV for the efficient update by using several distinguishable exact-bits to split the ruleset. SplitBV consists of a constrained recursive algorithm for selecting the bit positions that can minimize the latency and a hybrid lookup pipeline. It can achieve a significant reduction in update latency with negligible memory growth and comparable high performance. We implement SplitBV and evaluate its performance by simulation and FPGA prototype. Experimental results show that our approach can reduce 73% and 36% update latency on average for synthetic 5-tuple rules and OpenFlow rules respectively.

Keywords: SDN switch; Packet Classification; update latency; Bit-Vector; FPGA

1. Introduction

Software-Defined Networking (SDN) [1] satisfies both flexibility and programmability for routers. SDN separates the control plane (named controller) and data plane (named switch) of routers. OpenFlow [2] is deployed in SDN as a de facto standard, which defines the communication mechanism and message format between the controllers and the switches [3]. SDN controllers have a necessary demand on SDN switches that the control commands must be transmitted timely and executed efficiently, namely update performance guarantees.

The SDN switches use lookup tables with match-action rules to enforce the forwarding strategy, which is essentially a multi-field packet classification problem [2]. Commands issued by the SDN controllers usually become regular rule update operations (insert, delete, and modify) for the lookup tables (called rulesets). Recently, the research topic puts more emphasis on achieving both fast packet classification and fast rule updates. Ternary Content Addressable Memory (TCAM) has been widely adopted in the industrial field [4,5] for enabling line-speed classification. Despite that, TCAM has the significant disadvantages of capacity-limited, power-hungry, and invalid for range-match [5–7]. Field Programmable Gate Array (FPGA) has been recognized as a promising alternative in real-time network processing applications [8–10]. Due to its high performance with flexible reconfigurability, FPGA is increasingly being utilized for accelerating SDN software switches [11,12]. Decision-tree-based algorithms [13–16] and Tuple-Space-Search (TSS) algorithms [17–20] are two major approaches implemented on the CPU/GPU platform. However, the characteristics of these algorithms conflict with FPGA devices and cannot be effectively migrated to FPGA. Bit-Vector-based (BV-based)

algorithms [21–25] exploit hardware parallelism of FPGA and easily achieve line-speed classification performance. In particular, the state-of-the-art research TPBV [24] and further optimized WeeBV [25,26] have implemented microsecond-level data structure updates on the hardware pipeline by using self-reconfiguration processing elements.

Unfortunately, the worst-case latency of completing one update operation by the above researches [24–26] is unbearable, which is crucial for SDN controllers. It has been confirmed that most flows are small in SDN [27], which generate plenty of new rules that needed to be updated. Excessive latency of the update operation raises two serious problems: (1) process pressure of the control plane caused by accumulative new packets [28]; (2) losing packets on the forwarding plane, which has a destructive impact on latency-sensitive applications [29].

In this paper, we present SplitBV to ensure as small latency as possible for high-performance SDN switches on FPGA. SplitBV selects several distinguishable exact-bits to split the ruleset into independent sub-rulesets without rule replication. These diminutive sub-rulesets can be implemented parallelly in BV-based pipelines, thus reducing update latency tremendously. Moreover, the update and lookup processes between different pipelines do not conflict with each other, which further reduces the damage of rule updates to classification performance. Our contributions in this work include:

- *A novel architecture for packet classification, SplitBV*. SplitBV divides the ruleset into independent sub-rulesets and uses BV-based pipelines to match the sub-rulesets in parallel.
- *A recursive splitting algorithm based on greedy sorting, SplitAL*. SplitAL first determines the searching steps of matching fields based on greedy strategy and then employs a constrained recursive algorithm to resolve the problem of exponential computational complexity.
- *A hybrid matching pipeline on FPGA, SplitHP*. SplitHP contains a hash element for mapping different sub-rulesets into different two-dimensional pipeline consisted of processing elements (PEs, proposed in [24])).

We generate synthetic 10K 5-tuple rules and OpenFlow1.0 rules from well-known ClassBench [30] and the lastest ClassBench-ng [31] to evaluate our scheme. It is reasonable to choose rules around 10K since recent researches [32,33] have validated against dozens of thousands of rules. Experimental results show that SplitBV reduces the update latency by average 37% and 41% compared with TPBV on the two typical rulesets respectively.

The rest of the paper is organized as follows. Section 2 reviews the background and discusses problem of this paper. Section 3 surveys and analyses prior works. In Section 4, we detail our scheme. The hardware architecture is proposed in Section 5. We present experimental results in Section 6. Finally, the paper is concluded in Section 7.

2. Background

2.1. Multi-Field Packet Classification

Packet classification can be defined as: Given a ruleset consisted of N rules and an input packet header consisted of F fields, find all the rules matching the packet header and export the highest priority rule. Each rule is an individual predefined entry used for classifying a packet stream, which is associated with a unique rule ID (RID), a priority and an action. A packet header matches a rule, which means that matches all matching fields of the rule.

Classical packet classification contains five matching fields: Source IP Address, Destination IP Addres, Source Port, Destination Port, and Protocol. Each matching field has a different match type. For OpenFlow, classical packet classification is transformed into multi-field packet classification. OpenFlow1.0 has 12 matching fields, while version 1.5.1 extends to 45 matching fields. Besides, the multi-field packet classification in OpenFlow has the following characteristics: (1) more rules, to support fine-grained management of SDN controllers; (2) faster rule update, caused by plenty of short streams in SDN. Packet classification for SDN switches should achieve both high-performance and low update latency.

2.2. Rule Updates on SDN Switch

SDN divides the function of the router into two parts: controller and switch. Figure 1 shows a regular router model that uses FPGA acceleration. A controller is typically implemented on the CPU platform to generate policies for new packets. The switch on FPGA usually consists of three modules: Parser, Packet Classification (Classific.), and Action. The Parser module takes head messages from the input packet as a key. Packet Classification is the core module, which uses the key to match the rule with the highest priority from the ruleset, and then transport results to the Action module for execution. A new packet generates a miss that will be uploaded to the controller, and then the controller dispatch the corresponding rule update into the Packet Classification module.

From the beginning of the first miss generation, continuously arriving packets will generate misses successively until the new rule is configured on the data plane. It is worth noting that the new rule entering the data plane will not take effect immediately. The controllers calculate a new rule at the microsecond level, while the data plane completes the update configuration at the millisecond level [34]. Excessive latency will accumulate plenty of misses for controller, resulting in a sharp decline in controller performance. Excessive latency will accumulate plenty of misses for the controller, resulting in a sharp decline of controller performance [28]. Meanwhile, massive miss-route packets on the data plane can seriously affect latency-sensitive applications [29].

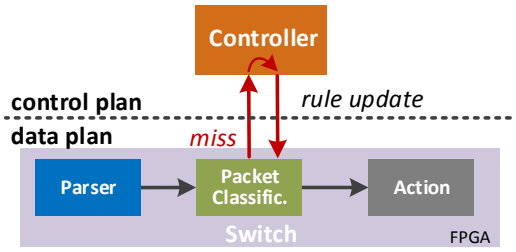


Figure 1. An example of SDN controller and switch with FPGA.

3. Prior Works

3.1. Packet Classification Algorithms

Packet classifications recently can be divided into two main categories: RAM (Random Access Memory)-based algorithmic solutions and TCAM (Ternary Content Addressable Memory)-based solutions [3,35]. Each storage unit in TCAM can have three different types of states: {0, 1, and *}, and TCAM can search all rules in parallel in a single lookup cycle. TCAM suffers high cost and high power consumption, and hard to perform range matching [3]. Anat Bremler-Barr et al. proposed gray code [36] and layered interval code [37] to enable TCAM support range matching at the cost of entry explosion. Thus in practice, the number of range fields is severely limited.

Decision-tree-based approaches [13–15] analyze all fields in a ruleset to construct decision trees for packet classification. Tree depth and rule duplication in a decision tree affect the searching efficiency and memory requirement of one implementation. Also, the state-of-the-art CutSplit [16] can hardly make an excellent trade-off among storage, performance, and updating,

Tuple space solutions (TSS) [17,18] leverage the fact that the number of distinct tuples is much less than that of rules in a ruleset. TSS can insert or delete rules from distinct tuples in amortized one memory access, resulting in high update performance. However, with the growing field number in a ruleset, both tuple number and tuple size increase. A longer processing latency could be expected. PartitionSort [19] reduces average lookup times by presenting a pre-computed priority for each tuple space, but there is no guarantee of its worst-case performance. The further TupleMerge [20] relaxes the rule's restrictions on entering the same tuple to reduce the number of tuples. However, too many tuples merging can lead to hash conflicts, which can seriously affect performance.

3.2. BV-Based Solutions Analyses

The BV-based approaches split matching fields (e.g., source IP address) into $\frac{L}{s}$ sub-fields (s $(1 \ll s \ll L)$ denotes the length of a sub-field, and L denotes the length of total matching fields). A bit-vector (BV) of length N is used to represent the matching result of N rules on a sub-field; usually, N is the number of the entire ruleset. Each bit of BV corresponds to a rule, with 0 for success and 1 for failure. When performing AND operations on $\frac{L}{s}$ BVs of all sub-fields, the matching result of the ruleset for the input packet is obtained. FSBV [22] is a particular case where s is 1, which creates bit-level sub-field partitioning. StrideBV [23] increases s for enhancing performance. The state-of-the-art TPBV [24] divides a ruleset into $\frac{N}{n}$ sub-rulesets for improving the scalability on large rulesets. WeeBV [26] further reduces the memory consumption of TPBV but damages rule updates on the hardware.

To support large-scale ruleset and ensure performance, TPBV implements a two-dimensional pipeline consisted of processing elements (PEs) and priority encoders (PrEncs) on FPGAs [24]. Each PE is responsible for matching a sub-field and is self-reconfigurable. However, the increasing number of rulesets forces the pipeline array to expand, which severely affects update latency. Figure 2 shows the worst-case update latency of TPBV, where parameters s and n are the best scheme in [24]. The update latency of TPBV consists of two parts: (1) *propagation time*, the time from entering the pipeline to finding the updated location; (2) *processing time*, the time from the execution of the updated instruction at the updated location to the new rule effective. It can be seen from Figure 2 that propagation time has a significant impact on update latency.

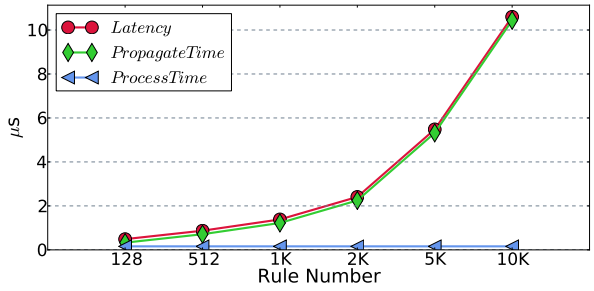


Figure 2. The worst-case latency of TPBV in [24] with $s = 4$ and $n = 8$.

An matching example of TPBV is illustrated in Figure 3a. A bit-vector ${BV_i}^{K_j}$ is used to represent the matching result of K_j for the corresponding matching sub-field$_j$ of the sub-ruleset$_i$. The values of ${BV_i}^{K_j}$ can be calculated by algorithms in [23]. In this example, parameter s determines that each horizontal pipeline has two PEs ($\frac{L}{s}$=$\frac{4}{2}$=2), and parameter n determines that there are two horizontal pipelines ($\frac{N}{n}$=$\frac{6}{3}$=2). The propagation path of packet header is: $PE[0,0] \rightarrow \{PE[0,1], PE[1,0]\} \rightarrow \{PrEnc[0], PE[1,1]\} \rightarrow \{PrEnc[1]\}$. If the update location is in $PE[1,1]$, the update command have to wait 2 clock cycles for propagating.

The parameters s and n are generally related to the characteristics of the FPGA devices used. In general, each FPGA has its fixed s and n to achieve maximum performance. This means that the "number of layers" of a two-dimensional pipeline is related to the size of the ruleset, in other words, a larger ruleset causes a larger propagation time. We note that the exact-bits in the rules can be used to split the ruleset. As shown in Figure 3b, selecting the first bit in the rule as the split-bit, ruleset can be divided into two sub-rulesets $\{R_0, R_1, R_4\}$, and $\{R_2, R_3, R_5\}$. With the same s and n, the two sub-rulesets correspond to two independent pipelines, respectively. In particular, the input packet head will enter only one pipeline according to 1st bit. If the 1st bit of input packet is 1, the rules in the previous sub-ruleset $\{R_0, R_1, R_4\}$ must not be matched. Therefore, these two pipelines can handle updates and matches concurrently. In this case, the propagation time of the entire pipeline is reduced to half of its original value. Based on the above observation, we are motioned to reduce propagation time to reduce rule update latency.

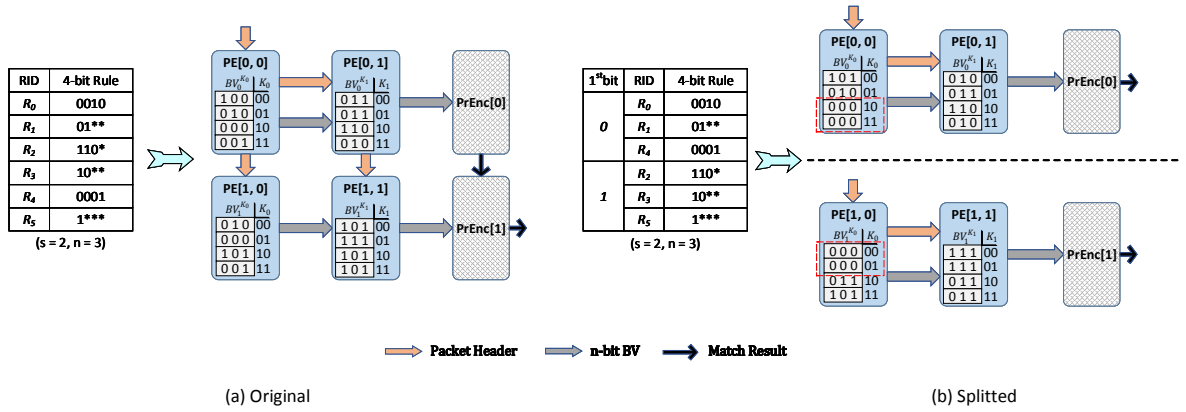


Figure 3. A matching example of TPBV for a 4-bit ruleset with $s = 2$ and $n = 3$.

4. Proposed Scheme

4.1. Ideas and Architecture

As mentioned above, our proposed SplitBV divides the whole ruleset into different sub-ruleset by selecting several specific bit locations. Notice that there is no rule replication between these sub-rulesets. Rule replication brings two problems: more memory consumption and the challenge of rule updates. If a rule exists in more than one sub-ruleset, it will result in multiple rule updates. Rule replication occurs when there is a wildcard in the selected location. Given the selected bit positions, the candidate rules that are exact bits at these positions constitute a sub-rule set.

Figure 4 shows the architecture of SplitBV. The splitting of ruleset can be divided into F steps (F is the number of matching fields). Each step corresponds to a field partition process because the rules have different characteristics in different matching fields. Ideally, the original ruleset can find p bit positions on one field, Splitting all rules into a HASH table of 2^p table items. In this case, only one step, $sub - ruleset_1$ needs to be executed, which is the original ruleset. Each item in a HASH table contains rules that are looked up using TPBV-pipeline [24]. However, it is often difficult to find such a bit position in a single field. Because the rules are not usually accurate in all fields, that is, in some fields is a full wildcard match. To avoid rule replication, SplitBV tries to partition the ruleset in different fields. It is a noteworthy fact that no rule can be a full wildcard match on all fields. Therefore, up to F sub-rulesets can be obtained from the original ruleset without rule replication. At the same time, for each sub-ruleset, at least one-bit position can be found to generate a HASH table. Each item corresponds to the TPBV pipeline responsible for matching $N_{(i,j)}$ rules.Moreover, if the number of remaining rules is less than the minimum size of the HASH table ($2 \times n$), then we can directly use TPBV for matching without splitting.

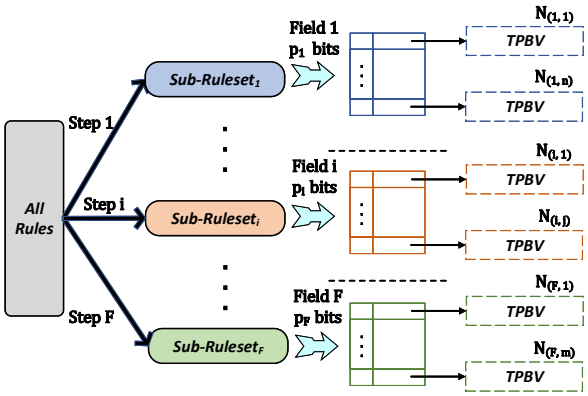


Figure 4. The architecture of SplitBV.

The worst-case propagation time of SplitBV = $\frac{L}{s} + \lceil \frac{Max(N_{(i,j)})}{n} \rceil - 1$. This is because the TPBV pipelines corresponding to the different HASH tables can be matched in parallel. For a HASH table, the input packet will enter only one TPBV pipeline corresponding to a table item. Therefore, in order to minimize the latency, of SplitBV, especially propagation time, it is necessary to find a field selection order and a bit location scheme on each field to minimize $Max(N_{(i,j)})$.

4.2. Brute Force Strategy

We first define the following variables: the total number of rules N, the total length of all matching fields L, and the number of matching fields F. An alternative combination consists of two parts: the matching field selection order and the bit position selection on each field. Given an alternative combination, the original ruleset can be quickly split, and the propagation time can be easily calculated. In order to get the combination of minimal propagation time, the brute force strategy is to find out all possible alternative combinations and then calculate each alternative combination. The selection order of matching fields is a sequential arrangement problem, and there are a total of $F!$ matching arrangement schemes. For matching fields with matching length of f_i, p ($1 \ll p \ll f_i$) bits can be randomly selected. Formula 1 gives the total number of selected location combinations.

$$Comb_p(field_i) = C_{f_i}^{p=1} + C_{f_i}^{p=2} + \cdots + C_{f_i}^{p=f_i} = 2^{f_i} \tag{1}$$

Given a matching field order permutation $StepList$, formula 2 gives the total number of selected position combinations on F matching fields. Therefore, the brute force strategy must calculate the values of $F! * 2^L$ schemes, and then select the optimal solution through the sorting algorithm. This kind of computational complexity is obviously unbearable.

$$\prod_{i=0}^{F} Comb_p(field_i) = 2^{f_1} \times 2^{f_2} \times \cdots \times 2^{f_F} = 2^L \tag{2}$$

4.3. Split Algorithm

Notice that the two parts of an alternative combination are in sequence. Therefore, we first try to determine the near-optimal matching field selection order quickly and then propose a constrained recursive algorithm to find all the bit position combination schemes.

4.3.1. Distinguish by field

We use Dis to denote the distinguishability of a field, and formula 3 gives the calculation function of Dis. $PrefixKind$ represents the number of different prefixes of the original rule set in the current field, and the prefix consists of three characters. The greater the distinguishability of a field, the more precise specific locations can be found in this field. Meanwhile, it also means that these exact bit positions can be used to build a HASH table, where each table item contains as small a set of rules as possible. We adopt the idea of greedy choice: calculate the distinguishable for each field, and then give $StepList$ in descending order.

$$Dis(field_i) = \frac{PrefixKind}{N} \tag{3}$$

4.3.2. Select-Bits Combination

There is a noteworthy fact that wildcards in prefix matching appear continuously and must appear after exact bits. We adopt a simple strategy that we select consecutive p_i ($1 \ll p_i \ll f_i$) bits starting from the first bit of $field_i$ for the sub-rulesets. Even with the above strategy, the total number

of bit location selection combination is still exponential, and formula 4 gives the upper limit (derived by inequality of arithmetic and geometric means).

$$\prod_{i=0}^{F} Comb_{continuous\_p}(field_i) = f_1 \times f_2 \times \cdots \times f_F \ll (\frac{L}{F})^F \tag{4}$$

In order to further reduce the number of alternatives and approach the optimal solution as much as possible, we propose a recursive algorithm *SelectBits* with constraints. Algorithm 1 shows the pseudo-code of the SelectBit algorithm. *SelectBits* recurses the matching fields in *StepList* sorted by *Dis* in descending order from beginning to end. In particular, line 8 of *SelectBits* gives constraints to reduce the number of recursions. $CurRuleSet.num$ indicates the number of rules currently waiting to be partitioned. Each table item in, HASH table is expected to have at least n rules. For $field_i$, the p_i needs to satisfy: $2^{p_i} \times n \ll CurRuleSet.num \Leftrightarrow p_i \ll log_2 \frac{CurRuleSet.num}{n}$. In our tests, we found that $log_2 \frac{CurRuleSet.num}{n}$ is usually much less than f_i, which greatly reduces the number of recursions. For the 10K rules, the average computing percentage ($\frac{Search\ times}{Rule\ number}$) of the 5-tuple rule is 1.18% (0.85% of ACL, 2.13% of FW, and 0.55% of IPC), and the average computing percentage of the OpenFlow1.0 (12-tuple) rules is 2.37%. Line 2 gives the conditions for recursive termination: (1) all fields have been searched stepped by *StepList*; (2) the number of remaining rules is less than the minimum size of the HASH table ($2 \times n$, when p = 1), and then set $p = 0$ and return. In addition, considering that the greedy strategy of *StepList* may lead to a locally but bot globally optimal solution, line 18 allows skipping of $field_i$ that currently cannot be effectively partitioned. Skipping all fields is obviously the wrong solution, and line 3 excludes this situation.

Algorithm 1 SelectBits

Input: The list of field information, *FieldList* The list of index of *FieldList* sorted in descending order of Dis, *StepList*; The current remaining rules for searching, *CurRuleSet*; The current value of step, *CurStep*; The list of p_i on each field, *PList* The number of cluster, n;

Output: The global set of combination of p_i on each field, *G_ResultSet*;

```
1:  function SELECTBITS(CurRuleSet,CurStep,CurMax,PList)
2:      if CurStep > F or CurRuleSet.num < 2 × n then
3:          if PList is all −1 then                                   ▷ return
4:          end if
5:          Add PList+[0] to G_ResultSet                              ▷ return
6:      end if
7:      f_i = FieldList[CurStep].len
8:      for p = 1 to MIN(f_i, log2 CurRuleSet.num / n) do
9:          for rule in CurRuleSet do
10:             if no wildcard in rule[1 : p] then
11:                 Add rule to HASH_p[rule[1 : p]]
12:             end if
13:         end for
14:         if HASH_p has empty item then
15:             if CurRuleSet.num ≪ CurMax then
16:                 Add PList+[0] to G_ResultSet                      ▷ return
17:             else
18:                 SelectBits(CurRuleSet, CurStep + 1, CurMax, PList + [−1])   ▷ return
19:             end if
20:         else
21:             Remove HASH_p.allrules in CurRuleSet
22:             CurMax = MAX(CurMax, HASH_p.max)
```

23: SelectBits($CurRuleSet, CurStep + 1, CurMax, PList + [p]$)
24: **end if**
25: **end for**
26: **end function**

4.4. A Example

Table 1 gives an example of a ruleset that contains 12 rules for two matching fields, with the cluster parameter n. First, we calculate $Dis(Field_1) = \frac{3}{4}$ and $Dis(Field_2) = \frac{1}{2}$. The selection order of matching field is $[Field_1, Field_2]$. First, the ruleset is divided on the $Field_1$. According to line 8 of the *SelectBits* algorithm, the optional range of p_1 is 1 and 2. When $p_1 = 1$, the ruleset is divided into $sub - ruleset_1 = \{0 : \{R_0, R_3, R_6, R_8\}, 1 : \{R_1, R_2, R_5, R_{10}, R_{11}\}\}$ and $sub - ruleset_2 = \{R_4, R_7, R_9\}$. Then the remaining $sub - ruleset_2$ is divided on $Field_2$, and the search will be stopped since the number of remaining rules is less than $2 \times n = 6$. Therefore, the bit position selection scheme $Cb_1 = [1, 0]$ for p_1=1 (0 meaning that the remaining rules are not split in $Field_2$ after the $Field_1$ selection), and $\lceil \frac{Max(N_{(i,j)})}{n} \rceil = \lceil \frac{5}{3} \rceil = 2$. In the same way, the bit location selection scheme $Cb_2 = [2, 0]$ for pendant 2 is divided into $sub - ruleset_1$=$\{00 : \{R_0, R_3\}, 01 : \{R_6, R_8\}, 10 : \{R_1, R_5, R_{11}\}, 11 : \{R_2, R_{10}\}$, $sub - ruleset_2 = \{R_4, R_7, R_9\}$, and $\lceil \frac{Max(N_{(i,j)})}{n} \rceil$=$\lceil \frac{3}{3} \rceil = 1$. It can be seen that Cb_2 has less propagation time than Cb_1. However, 4 horizontal pipelines will be used for Cb_1 with the given parameter n, while 5 horizontal pipelines will be used for Cb_2. It is necessary to make a trade-off between propagation time and increasing memory consumption caused by pipeline expansion.

Table 1. An example of ruleset with two fields for $n = 3$.

RID	$Field_1$	$Field_2$
R_0	0010	11
R_1	1001	0 *
R_2	1110	10
R_3	0000	**
R_4	****	1 *
R_5	1001	10
R_6	0101	1 *
R_7	****	00
R_8	0100	0 *
R_9	****	00
R_{10}	1101	0 *
R_{11}	1011	1 *

* denotes the wildcard bit.

5. Hardware Design

5.1. Hybrid SplitHP

We propose a hybrid two-dimensional pipeline to implement SplitBV, named SplitHP on FPGA. The SplitHP consists of two stages, the first stage is composed of the splitting elements (SEs), and the second stage is composed of several horizontal two-dimensional pipelines composed of PEs and PrEncs.

Figure 5 shows an example of the hardware framework of SplitHP for Figure 3b. In the first stage, each $SE\text{-}p_i$ corresponds to a sub-ruleset, obtained by splitting on $field_i$ with p_i bits. $SE\text{-}p_i$ associates different rules corresponding to 2^{p_i} items of the HASH table to different horizontal pipelines subsequently. In the second stage, the rules in each table item are matched using a two-dimensional pipeline implemented by TPBV. Note that, partitioning the ruleset changes the original order of

rules, but does not affect the correctness of the final match results. PrEnc [23] reports a local highest priority match, which does not limit the order in which rules are arranged. The final match result is collected by the vertical pipeline of the priority encoders. The input packet header will be sent to all $SE\text{-}p_i$s in the first stage in parallel, while the entire ruleset is matched to obtain the correct result. The two-dimensional pipelines corresponding to a $SE\text{-}p_i$ are isolated and independent, that is, the packet header will only enter one of the two-dimensional pipelines and only one PrEnc will output a valid matching result. We use a multiplexer (MUX) to integrate the matching results of the $SE\text{-}p_i$ and send it to PrEnc in the bottom horizontal pipeline. As mentioned in Section 3.1, the worst-case propagation time of SplitHP depends on $Max(N_{(i,j)})$. In Figure 5, $PrEnc[4]$ executes one clock cycle ahead of $PrEnc[1]$. We adopt a simple strategy that the PrEnc in the bottom horizontal pipeline waits until all the matching results are received.

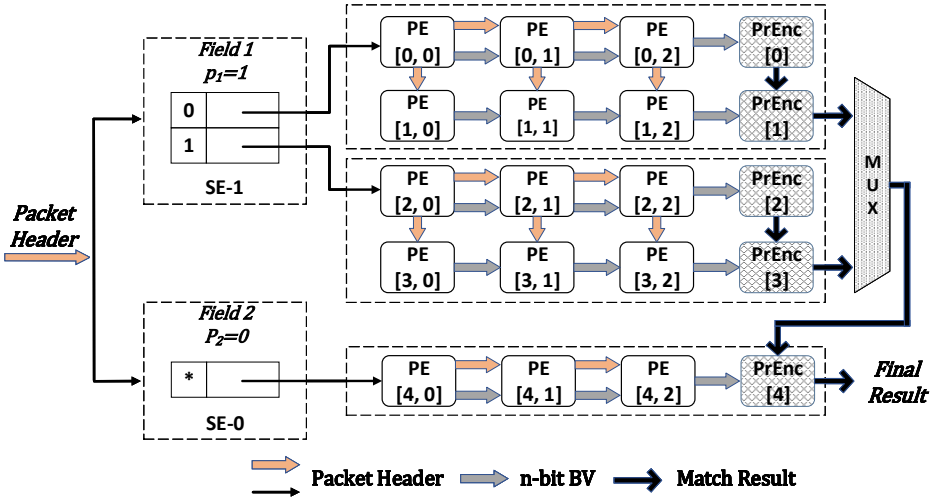


Figure 5. Hardware framework of SplitHP for Figure 3b.

The detailed design of the main elements in SplitHP is described below and shown in Figure 6.

5.1.1. Splitting Element (SE)

There are two types of SE: SE-0 and $SE\text{-}p_i$. SE-0 is usually used only in the bottom horizontal pipeline and is responsible for matching the remaining sub-ruleset (the number of rules is less than $2 \times n$). $SE\text{-}p_i$ is used to find the corresponding item of the HASH table for the packet header. The $SE\text{-}p_i$ contains the following components: (1) Controller, is responsible for extracting split-bits from packet header; (2) DMUX, one or multi-level demultiplexer for outputting packet header to the corresponding pipeline. Figure 6b,c show the SE with p_i equals 1 and 2, respectively. When p_i is greater than 1, multi-level DMUX is used to satisfy 2_i^p table items. Research has shown that multi-level small DMUX can provide higher performance than a large DMUX [38].

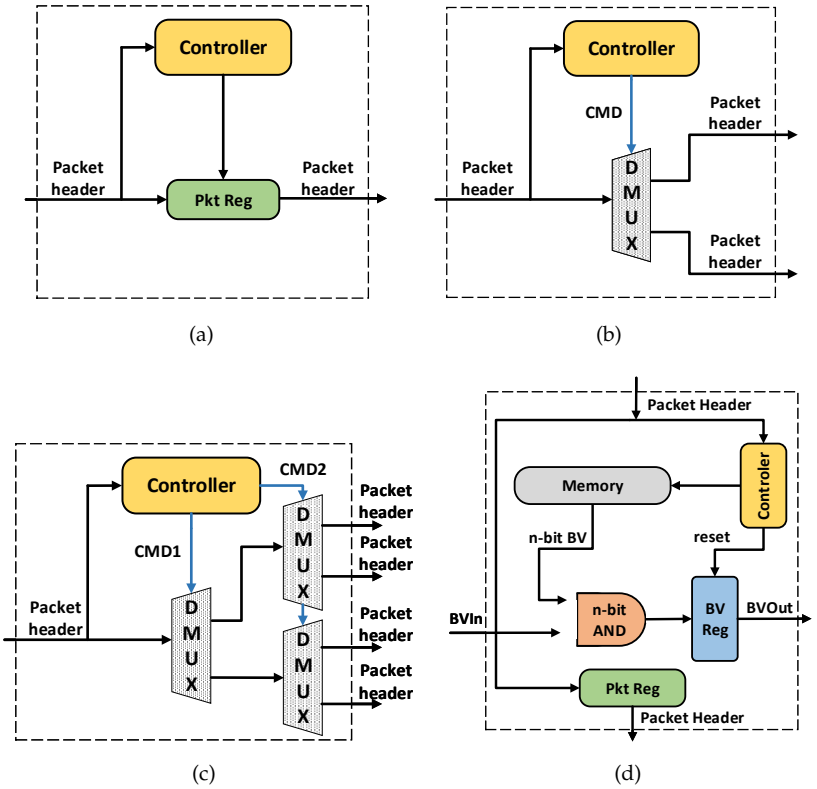


Figure 6. Various elements of SplitHP. (**a**) SE-0; (**b**) SE-1; (**c**) SE-2; (**d**) PE in [26].

5.1.2. Process Element (PE)

Considering the simplicity of implementation, we reused the PE in [26]. In particular, for the two-dimensional pipeline corresponding to each item of the SE, the packet header is transmitted in the horizontal direction in sequence only in the uppermost horizontal pipeline. Then, the packet header is propagated to the next horizontal pipeline at the same time every other clock cycle.

5.2. Update Strategy

Rule update can be seen as three operations: modification, insertion, and deletion. The controller component in PE can automatically configure the writable memory according to update commands. In addition, valid-bit is used to support fast deletion [24]. We follow the strategy of reuse, where inserts are processed at the location of invalid rules.

Delete and Modify. To delete a rule is to set the valid-bit of the rule to 0. The BV that PE fetches from memory will be ANDed with valid-bits, and invalid rules will not be matched. A modification can be divided into a deletion and an insertion at the same location.

Insert. For inserts, the controller component of PE can convert the new rules into the entire BV table and rewrite the memory component. The main difficulty is whether an invalid rule can be found in the corresponding table item. Plenty of short streams in SDN also means frequent invalidation of rules, which generates increasing deletion operations. Therefore, finding an invalid rule is easy. In the worst case, SE may need to be refactored, which can be quickly resolved by using FPGA technology DPR(Dynamic Partial Reconfiguration) [39,40].

6. Evaluation

6.1. Experimental Setup

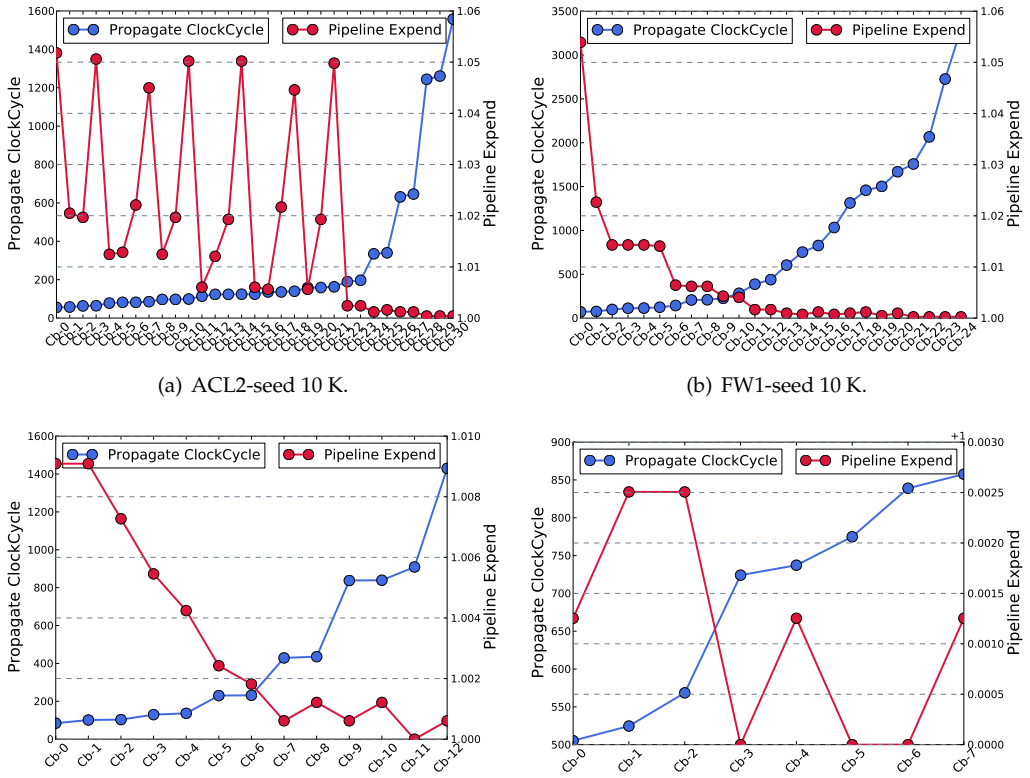
Synthetic classifiers: To test the performance of our scheme and prior art, we generate the 5-tuple rules by the well-known ClassBench [30] and the OpenFlow1.0(OF) rules (with 12 fields) by the inherited ClassBench-ng [31]. In our experiments, the 5-tuple rules we have used contain Accesses Control List (ACL), Firewall (FW) and IP Chain (IPC). The seed files from real-life 5-tuple rules and a data-center can make the performance as close to practice as possible.

Implementation platform: We implement the SplitHP with an Intel Arria II GX EP2AGX65DF25C4 FPGA, which consists of 4.35Mb Block RAM, 50,600 Combinational ALUTs, and 51336 logic registers. The splitting algorithm is implemented by software and runs on a machine with Intel Xeon E5-3650 CPU and Ubuntu 16.04 LTS operation system.

Parameter selection: There are two key parameters s and n in TPBV, which will affect the update delay of the hardware pipeline and the memory footprint of the BV vector table. In order to make a fair comparison with TPBV, we choose the best parameter given in [24], that is, swatches 4 and 8.

6.2. Split-Bits Trade-Off

As mentioned in Section 4.3, SplitBV will lead to the expansion of the hardware pipeline when choosing the location of dividing bits. This is because the selection bit length p is increased so that the propagation time reduces the number of items in the HASH table and increases at the same time. Because each table item corresponds to at least one horizontal pipeline, the total number of pipelines implemented by SplitBV hardware increases. We used 12 different seed files (5 kinds of ACL,5, 5 kinds of FW, 2 kinds of IPC, 2 kinds of OF) to generate 10 K rules for evaluation.



(a) ACL2-seed 10 K. (b) FW1-seed 10 K.

(c) IPC1-seed 10 K. (d) OF2-seed 10 K.

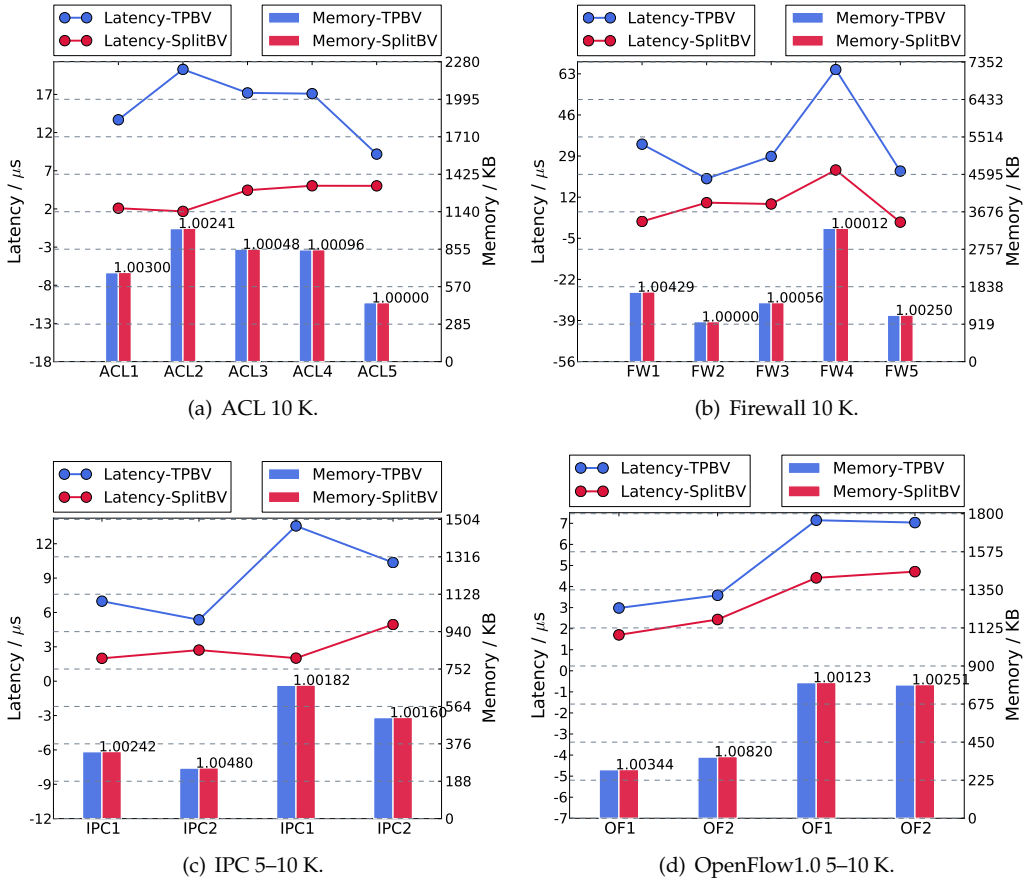
Figure 7. Tradeoff of *SelectBits* for the different rulesets.

Due to limited space, Figure 7 shows the tradeoff of running split-algorithm only on rule sets generated by four different types of representative seed files. In terms of a clock cycle, we give the expansion multiple of the number of horizontal pipelines of the combined scheme represented by propagation time, pipeline expend compared with the number of horizontal pipelines of the original TPBV. We arrange all the alternatives in the ascending order of the propagation clock cycle.

As can be seen from Figure 7, both metrics are generally in line with an exponential rise (or decline). Because the *SelectBits* algorithm allows you to skip a field, where two propagation clock cycles are close to each other, there may also be a big difference in pipeline expend. The best solution is near the intersection of the trend curves of the two metrics.

6.3. Update Latency

We measured the worst-case update latency for TPBV and SplitBV, respectively. The update delay in the worst case is an important index to evaluate the performance and reliability of the SDN switch. We assume that the clock rate of the FPGA implementation is the usual 100 MHz, then the clock cycle is 10ns. Figure 8 shows the update delay comparison between SplitBV and TPBV. SplitBV reduces update latency by an average of 73%, 74%, 65% and 36% for ACL, FW, IPC, and OpenFlow1.0, respectively. In particular, it is an accepted fact that SDN can achieve 42,000 rules updates per second [34]. This means that the update delay must be controlled at a subtle level of 20. The results in Figure 8 show that SplitBV meets the update latency requirements under SDN for various rulesets of the order of magnitude of 10 K.



(a) ACL 10 K. (b) Firewall 10 K.

(c) IPC 5–10 K. (d) OpenFlow1.0 5–10 K.

Figure 8. Comparison of latency and memory for the different rulesets.

6.4. Memory Consumption

We also measured the memory footprint of SplitBV and TPBV. The limited Block Memory on FPGA severely limits the scalability of BV-based approaches. A large number of BV tables must be stored to support fast parallel matching on the hardware pipeline. Considering that our implementation of PE is not consistent with the details in the original text, we only compare the memory footprint of the BV table. Figure 8 also shows the comparison of SplitBV's best scheme with the original TPBV. The data on the red column is a multiple of memory expansion. SplitBV increases memory consumption by an average of 1.00137, 1.00149, 1.00266 and 1.00385 times for ACL, FW, IPC, and OpenFlow1.0, respectively. That is because SplitBV requires more pipelines than the original TPBV. In other words, the cost of reducing update latency is the expansion of memory consumption. Fortunately, the increase of memory is acceptable.

6.5. Resource and Throughput

Limited by hardware resources, we tested an additional simple prototype system on FPGA for 36 rules. The size of the Block Memory of the FPGA we use is m9K, which is equal to $9 \times 1024 = 9 \times 2^9$ bits. Therefore, we set parameter $s = 9$ and $n = 36$ so that each PE has two memory blocks. In this way, TPBV only needs one horizontal pipeline, and SplitBV has a maximum of 2 SEs. Table 2 shows the resource consumption and performance of TPBV and SplitBV, and there are three schemes for SplitBV to be tested. Scheme $Cb1$, $Cb2$ and $Cb3$ represent the first SE with 1, 2 and 3 levels of DMUXs, respectively. For level i, $SE - i$ connects 2^i horizontal pipelines. Considering that a small number of rules can lead to wasted memory, we are more concerned with logical resource overhead. We use Clock Rate to reflect throughput. SplitBV increases the number of horizontal pipelines, so logical resources increase accordingly. Table 2 shows that the increase in the number of DMUX series in SE results in linear growth of logical resources without significantly compromising performance.

Table 2. Resource consumption and performance for $s = 9$ and $n = 36$.

	ALUTs Number	Registers Number	Clock Rate (MHz)
TPBV [24]	848	2501	158.63
SplitBV $Cb = 1$	1512	4421	152.0
SplitBV $Cb = 2$	1979	5873	145.45
SplitBV $Cb = 3$	3200	9379	145.33

7. Conclusions

In this paper, we present a scheme named SplitBV to support diminutive update latency of SDN switch while ensuring high throughput processing performance. SplitBV splits the ruleset into independent sub-rulesets without rule replication by several distinguishable exact-bits. SplitBV then utilizes BV-based pipelines to parallelly match these sub-rulesets. Besides, SplitBV utilizes a constrained recursive algorithm, which is used to determine the near-optimal bit-position combination schemes. A two-stage hybrid pipeline called SplitHP is proposed to implement SplitBV on FPGA. SplitHP uses the DMUX to send packet header to different pipelines and uses MUX to integrate the final matching results. Experimental results show that SplitBV has an excellent update latency reduction on both 5-tuple and OpenFlow rulesets. Moreover, SplitBV brings negligible memory growth but maintain comparable high performance. As our future work, we will improve SplitBV from the following aspects: (1) test MUX and DMUX model for large-scale ruleset; (2) improve splitting element with DPR technology.

Author Contributions: Conceptualization, C.L. and T.L.; Methodology, C.L., T.L. and J.L.; Software and hardware, C.L. and J.L.; Validation, C.L., J.L. and T.L.; Formal analysis, C.L. and B.W.; Investigation, Z.S.; Writing—original draft preparation, C.L.; Writing—review and editing, J.L. and T.L.; Funding acquisition, Z.S. and B.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China under grant NO. 61702538 and partially by the Scientific Research Program of National University of Defense Technology under grant NO. ZK17-03-53.

Acknowledgments: The authors thank Yuhao Han, Jinli Yan, Wenwen Fu, and Yue Jiang for great help.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. O. N. F. (ONF). Software Defined Networking. Available online: https://www.opennetworking.org/software-defined-standards/overview/ (accessed on 9 April 2020).
2. McKeown, N.; Anderson, T.; Balakrishnan, H.; Parulkar, G.M.; Peterson, L.L.; Rexford, J.; Shenker, S.; Turner, J.S. Openflow: Enabling innovation in campus networks. *ACM SIGCOMM Comput. Commun. Rev.* **2008**, *38*, 69–74. [CrossRef]
3. Yang, T.; Liu, A.X.; Shen, Y.; Fu, Q.; Li, D.; Li, X. Fast openflow table lookup with fast update. In Proceedings of the 2018 IEEE Conference on Computer Communications, INFOCOM 2018, Honolulu, HI, USA, 16–19 April 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 2636–2644. [CrossRef]
4. Lakshminarayanan, K.; Rangarajan, A.; Venkatachary, S. Algorithms for advanced packet classification with ternary cams. In Proceedings of the ACM SIGCOMM 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Philadelphia, PA, USA, 22–26 August 2005; Guérin, R., Govindan, R., Minshall, G., Eds.; ACM: New York, NY, USA, 2005; pp. 193–204. [CrossRef]
5. Ma, Y.; Banerjee, S. A smart pre-classifier to reduce power consumption of tcams for multi-dimensional packet classification. In Proceedings of the ACM SIGCOMM 2012 Conference, SIGCOMM '12, Helsinki, Finland, 13–17 August 2012; Eggert, L., Ott, J., Padmanabhan, V.N., Varghese, G., Eds.; ACM: New York, NY, USA, 2012; pp. 335–346. [CrossRef]
6. Li, X.; Lin, Y.; Li, W. Greentcam: A memory- and energy-efficient tcam-based packet classification. In Proceedings of the 2016 International Conference on Computing, Networking and Communications, ICNC 2016, Kauai, HI, USA, 15–18 February 2016; IEEE Computer Society: Washington, DC, USA, 2016; pp. 1–6. [CrossRef]
7. Rottenstreich, O.; Keslassy, I.; Hassidim, A.; Kaplan, H.; Porat, E. Optimal in/out TCAM encodings of ranges. *IEEE/ACM Trans. Netw.* **2016**, *24*, 555–568. [CrossRef]
8. Qu, Y.R.; Prasanna, V.K. Enabling high throughput and virtualization for traffic classification on FPGA. In Proceedings of the 23rd IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM 2015, Vancouver, BC, Canada, 2–6 May 2015; IEEE Computer Society: Washington, DC, USA, 2015; pp. 44–51. [CrossRef]
9. Li, B.; Tan, K.; Luo, L.L.; Peng, Y.; Luo, R.; Xu, N.; Xiong, Y.; Cheng, P. Clicknp: Highly flexible and high-performance network processing with reconfigurable hardware. In Proceedings of the ACM SIGCOMM 2016 Conference, Florianopolis, Brazil, 22–26 August2016; Barcellos, M.P. , Crowcroft, J., Vahdat, A., Katti, S., Eds.; ACM: New York, NY, USA, 2016; pp. 1–14. [CrossRef]
10. Yang, X.; Sun, Z.; Li, J.; Yan, J.; Li, T.; Quan, W.; Xu, D.; Antichi, G. FAST: Enabling fast software/hardware prototype for network experimentation. In Proceedings of the International Symposium on Quality of Service, IWQoS 2019, Phoenix, AZ, USA, 24–25 June 2019; ACM: New York, NY, USA, 2019; pp. 32:1–32:10. [CrossRef]
11. Zhao, T.; Li, T.; Han, B.; Sun, Z.; Huang, J. Design and implementation of software defined hardware counters for SDN. *Comput. Netw.* **2016**, *102*, 129–144. [CrossRef]
12. Fu, W.; Li, T.; Sun, Z. FAS: Using FPGA to accelerate and secure SDN software switches. *Secur. Commun. Netw.* **2018**, *2018*, 5650205:1–5650205:13. [CrossRef]
13. Singh, S.; Baboescu, F.; Varghese, G.; Wang, J. Packet classification using multidimensional cutting. In Proceedings of the ACM SIGCOMM 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, Karlsruhe, Germany, 25–29 August 2003; Feldmann, A., Zitterbart, M., Crowcroft, J., Wetherall, D., Eds.; ACM: New York, NY, USA, 2003; pp. 213–224. [CrossRef]

14. Vamanan, B.; Voskuilen, G.; Vijaykumar, T.N. Efficuts: Optimizing packet classification for memory and throughput. In Proceedings of the ACM SIGCOMM 2010 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, New Delhi, India, 30 August–3 September 2010; Kalyanaraman, S., Padmanabhan, V.N., Ramakrishnan, K.K., Shorey, R., Voelker, G.M., Eds.; ACM: New York, NY, USA, 2010; pp. 207–218. [CrossRef]
15. He, P.; Xie, G.; Salamatian, K.; Mathy, L. Meta-algorithms for software-based packet classification. In Proceedings of the 22nd IEEE International Conference on Network Protocols, ICNP 2014, Raleigh, NC, USA, 21–24 October 2014; IEEE Computer Society: Washington, DC, USA, 2014; pp. 308–319. [CrossRef]
16. Li, W.; Li, X.; Li, H.; Xie, G. Cutsplit: A decision-tree combining cutting and splitting for scalable packet classification. In Proceedings of the 2018 IEEE Conference on Computer Communications, INFOCOM 2018, Honolulu, HI, USA, 16–19 April 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 2645–2653. [CrossRef]
17. Srinivasan, V.; Suri, S.; Varghese, G. Packet classification using tuple space search. In Proceedings of the SIGCOMM'99, Cambridge, MA, USA, 31 August–3 September 1999; pp. 135–146. [CrossRef]
18. Pfaff, B.; Pettit, J.; Koponen, T.; Jackson, E.J.; Zhou, A.; Rajahalme, J.; Gross, J.; Wang, A.; Stringer, J.; Shelar, P.; et al. The design and implementation of open vswitch. In Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation, NSDI 15, Oakland, CA, USA, 4–6 May 2015; USENIX Association: Berkeley, CA, USA, 2015; pp. 117–130. Available online: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff (accessed on 9 April 2020).
19. Yingchareonthawornchai, S.; Daly, J.; Liu, A.X.; Torng, E. A sorted-partitioning approach to fast and scalable dynamic packet classification. *IEEE/ACM Trans. Netw.* **2018**, *26*, 1907–1920. [CrossRef]
20. Daly, J.; Bruschi, V.; Linguaglossa, L.; Pontarelli, S.; Rossi, D.; Tollet, J.; Torng, E.; Yourtchenko, A. Tuplemerge: Fast software packet processing for online packet classification. *IEEE/ACM Trans. Netw.* **2019**, *27*, 1417–1431. [CrossRef]
21. Lakshman, T.V.; Stiliadis, D. High-speed policy-based packet forwarding using efficient multi-dimensional range matching. In Proceedings of the SIGCOMM'98, Vancouver, BC, Canada, 31 August–4 September 1998; pp. 203–214. [CrossRef]
22. Jiang, W.; Prasanna, V.K. Field-split parallel architecture for high performance multi-match packet classification using fpgas. In Proceedings of the SPAA 2009, 21st Annual ACM Symposium on Parallelism in Algorithms and Architectures, Calgary, AB, Canada, 11–13 August 2009; auf der Heide, F.M., Bender, M.A., Eds.; ACM: New York, NY, USA, 2009; pp. 188–196. [CrossRef]
23. Ganegedara, T.; Jiang, W.; Prasanna, V.K. A scalable and modular architecture for high-performance packet classification. *IEEE Trans. Parallel Distrib. Syst.* **2014**, *25*, 1135–1144. [CrossRef]
24. Qu, Y.R.; Prasanna, V.K. High-performance and dynamically updatable packet classification engine on FPGA. *IEEE Trans. Parallel Distrib. Syst.* **2016**, *27*, 197–209. [CrossRef]
25. Li, C.; Li, T.; Li, J.; Yang, H.; Wang, B. A memory optimized architecture for multi-field packet classification (brief announcement). In Proceedings of the The 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, 22–24 June 2019; Scheideler, C., Berenbrink, P., Eds.; ACM: New York, NY, USA, 2019; pp. 395–397. [CrossRef]
26. Li, C.; Li, T.; Li, J.; Li, D.; Yang, H.; Wang, B. Memory optimization for bit-vector-based packet classification on fpga. *Electronics* **2019**, *8*, 1159. [CrossRef]
27. Benson, T.; Akella, A.; Maltz, D.A. Network traffic characteristics of data centers in the wild. In Proceedings of the 10th ACM SIGCOMM Internet Measurement Conference, IMC 2010, Melbourne, Australia, 1–3 November 2010; Allman, M., Ed.; ACM: New York, NY, USA, 2010; pp. 267–280. [CrossRef]
28. Chen, H.; Benson, T. Hermes: Providing tight control over high-performance SDN switches. In Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies, CoNEXT 2017, Incheon, Korea, 12–15 December 2017; ACM: New York, NY, USA, 2017; pp. 283–295. [CrossRef]
29. Chowdhury, M.; Zhong, Y.; Stoica, I. Efficient coflow scheduling with varys. In Proceedings of the ACM SIGCOMM 2014 Conference, SIGCOMM'14, Chicago, IL, USA, 17–22 August 2014; Bustamante, F.E., Hu, Y.C., Krishnamurthy, A., Ratnasamy, S., Eds.; ACM: New York, NY, USA, 2014; pp. 443–454. [CrossRef]
30. Taylor, D.E.; Turner, J.S. Classbench: A packet classification benchmark. *IEEE/ACM Trans. Netw.* **2007**, *15*, 499–511. [CrossRef]

31. Matouvsek, J.; Antichi, G.; Lucansky, A.; Moore, A.W.; Korenek, J. Classbench-ng: Recasting classbench after a decade of network evolution. In Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ANCS 2017, Beijing, China, 18–19 May 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 204–216. [CrossRef]
32. Kogan, K.; Nikolenko, S.I.; Rottenstreich, O.; Culhane, W.; Eugster, P. SAX-PAC (scalable and expressive packet classification). In Proceedings of the ACM SIGCOMM 2014 Conference, SIGCOMM'14, Chicago, IL, USA, 17–22 August 2014; Bustamante, F.E., Hu, Y.C., Krishnamurthy, A., Ratnasamy, S., Eds.; ACM: New York, NY, USA, 2014; pp. 15–26. [CrossRef]
33. Hsieh, C.; Weng, N. Many-field packet classification for software-defined networking switches. In Proceedings of the 2016 Symposium on Architectures for Networking and Communications Systems, ANCS 2016, Santa Clara, CA, USA, 17–18 March 2016; Crowley, P., Rizzo, L., Mathy, L., Eds.; ACM: New York, NY, USA, 2016; pp. 13–24. [CrossRef]
34. Kuzniar, M.; Peresíni, P.; Kostic, D. What you need to know about SDN flow tables. In *Passive and Active Measurement, Proceedings of the 16th International Conference, PAM 2015, New York, NY, USA, 19–20 March 2015*; ser. Lecture Notes in Computer Science; Mirkovic, J., Liu, Y., Eds.; Springer: Berlin, Germany, 2015; Volume 8995, pp. 347–359._26. [CrossRef]
35. Taylor, D.E. Survey and taxonomy of packet classification techniques. *ACM Comput. Surv.* **2005**, *37*, 238–275. [CrossRef]
36. Bremler-Barr, A.; Hendler, D. Space-efficient tcam-based classification using gray coding. *IEEE Trans. Comput.* **2012**, *61*, 18–30. [CrossRef]
37. Bremler-Barr, A.; Hay, D.; Hendler, D. Layered interval codes for tcam-based classification. *Comput. Netw.* **2012**, *56*, 3023–3039. [CrossRef]
38. Kuekes, P.J.; Williams, R.S. Demultiplexer for a Molecular Wire Crossbar Network (MWCN DEMUX). U.S. Patent 6,256,767, 3 July 2001.
39. Abel, N. Design and implementation of an object-oriented framework for dynamic partial reconfiguration. In Proceedings of the International Conference on Field Programmable Logic and Applications, FPL 2010, Milano, Italy, 31 August–2 September 2010; IEEE Computer Society: Washington, DC, USA, 2010; pp. 240–243. [CrossRef]
40. Kalb, T.; Göhringer, D. Enabling dynamic and partial reconfiguration in xilinx sdsoc. In Proceedings of the International Conference on ReConFigurable Computing and FPGAs, ReConFig 2016, Cancun, Mexico, 30 November–2 December 2016; Athanas, P.M., Cumplido, R., Feregrino, C., Sass, R., Eds.; IEEE: Piscataway, NJ, USA, 2016; pp. 1–7. [CrossRef]