High Speed Packet Classification Using FPGA

M.Monika[1], M.Adiseshaiah[2]
[1, 2]Department of ECE
[1]DVR & Dr. HS MIC College of Technology, Kanchikacherla, A.P
[2]Assistant Professor, DVR & Dr. HS MIC College of Technology, Kanchikacherla, A.P

Abstract*-Performing Packet classification effectively in network infrastructure is becoming a major challenge because of demand in increasing throughput and speed of enhancement. Many previous techniques which are used for packet classification focused on throughput only and they cannot reduce the searching speed. So a dynamically updatable packet classification along with parity generator is proposed to enhance the speed of searching by reducing the number of comparison operations. A special hardware support is an attractive alternative to enhance the speed of operation. The time required for classifying a packet in network infrastructure is reduced greatly. This technique enhances the searching speed and also reduces the propagation delay.*

Keywords*-*Packet classification, pipelined architecture, FPGA, Dynamic updates

I. INTRODUCTION

SOFTWARE Defined Networking (SDN) has been proposed as a novel architecture for enterprise networks. SDN separates the software-based control plane from the hardware-based data plane; as a flexible protocol, Open Flow[10] can be used to manage network traffic between the control plane and the data plane. One of the kernel function Open-Flow performs is the flow table lookup[11]. The flow table lookup requires multiple fields of the incoming packet to be examined against entries in a prioritized flow table. This is similar to the classic multi-field packet classification mechanism[12]. hence we use interchangeably the flow table lookup and the OpenFlow packet classification in this paper. The major challenges of packet classification include: (1) supporting large rule sets (up to 100K rules), (2) sustaining high performance (over millions of packets per second[2]), and (3) facilitating dynamic updates. Many existing solutions for multifield packet classification employ Ternary Content Addressable Memories (TCAMs[13,14]). TCAMs cannot support efficient dynamic updates; for example, a rule to be inserted can move across the entire rule set. This is an expensive operation. TCAMs are not scalable with respect to the rule set size. Besides they are also very power-hungry. Field Programmable Gate Array (FPGA) technology has been widely used to implement algorithmic solutions for real-time applications. FPGA-based packet classification engine can achieve very high throughput for rule sets of moderate size. However, as the number of packet header fields or the rule set size increases (e.g., OpenFlow packet classifi- cation), FPGA-based approaches often suffer from clock rate degradation. Future Internet applications require the hardware to perform frequent incremental updates and adaptive processing. Because it is prohibitively expensive to reconstruct an optimal architecture repeatedly for timely updates, many sophisticated solutions have been proposed for packet classification supporting dynamic updates over the years. Due to the rapid growth of the network size and the bandwidth requirement of the Internet, it remains challenging to design a flexible and run-time reconfigurable hardware-based engine without compromising any performance.

II. RELATED WORK

Packet classification can be broadly categorized into Decision Tree based, Decomposition based.

Decision-tree-based approaches involve cutting the search space recursively into smaller subspaces based on the information from one or more fields in the rule. In a decision tree is mapped onto a pipelined architecture on FPGA; for a rule set containing 10K rules, a throughput of 80Gbps is achieved for packets of minimum size (40 bytes). However, the performance of decision-tree-based approaches is rule-set-dependent. A cut in one field can lead to duplicated rules in other fields (rule set expansion). As a result, a decision-tree can use up to O(Nd) memory. This approach can be impractical.

Decomposition-based approaches first search each packet header field individually. The partial results are then merged to produce the final result. To merge the partial results from all the fields, hash-based merging techniques can be explored. however, these approaches either require expensive external memory accesses, or rely on a second hard ware module to solve hash collisions.

BV-based approach:

Field-Split BV (FSBV) approach [9] and its variants split each field into multiple subfields of s-bits; a rule is mapped onto each subfield as a ternary string. Lookup operations can be performed in all the subfields in a pipelined fashion; the partial result in each PE1 is represented by a BV of N bits. Logical AND operations can be used to merge all the extracted BVs to generate the final match result on FPGA. We show the basic architecture [5],[9] of BV-based approaches; FSBV approach can be visualized as a special case of s = 1. To access an N-bit data, wires of length O (N) are often used for the memory; as N increases, the clock rate of the entire pipeline deteriorates.

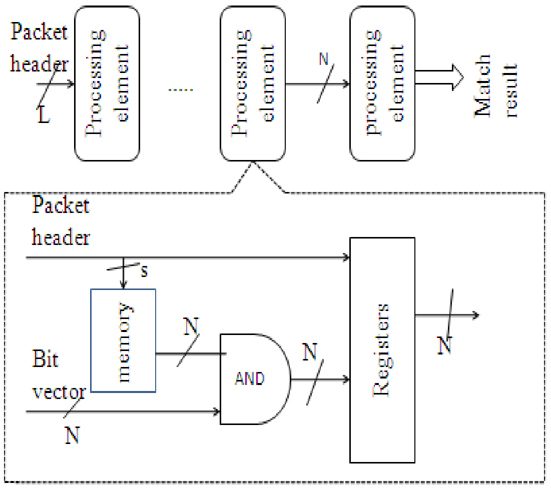


Fig 1: Basic architecture of BV-based approaches.

III. TWO-DIMENSIONAL PIPELINED ARCHITECTURE WITH SELF CONFIGURED MODULAR PE

Modular PE:

A modular PE is used to match a single packet header against one rule (N = 1) in a 1-bit subfield (s = 1). In order to minimize the number of I/O pins utilized, a modular PE is also responsible for propagating input packet headers to other PEs. A modular PE should be able to handle both prefix match and exact match.

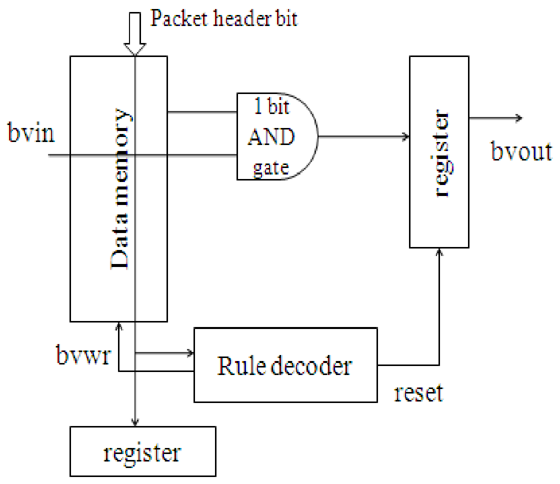


Fig 2: Modular Processing Element

Let us consider the internal organization of a modular PE as shown in above figure. The main difference between the modular PE in above, and the PEs used in the basic pipelined architecture, is that the modular PE in figure only produces the match result for exactly 1 rule at a time (2x1-bit data memory). The modular PE in this work has other components:

1) Rule decoder: the logic-based rule decoder is mainly responsible for dynamic updates.
2) s-bit register for input packet header: it is used in the construction of vertical pipelines.

We denote the register buffering the input packet header bits as the input register; we denote the register after the AND gate as the output register. Similar to FSBV, for a 1-bit subfield, the prefix rule can be handled efficiently. In above figure, the packet header bit is used to address the memory directly; the extracted BV is then ANDed with the BV output from the previous PE in the pipeline. A rule requiring exact match can be treated as a special case of a prefix rule. Hence we do not introduce any other new components for exact match.

2–Dimensional pipelined architecture:

In the basic pipelined architecture, the BVs in each PE for N rules are N- bit long. For distributed RAM (distRAM) or Block RAM (BRAM) modules of fixed size, the number of memory modules required for each PE grows linearly with respect to N. This means the length of the longest wire connecting different memory modules in a PE also increases at O (N) rate, which degrades the throughput of the pipeline for large N. To handle a larger number of rules and more input packet header bits, we use multiple modular PEs to construct a complete 2 dimensional pipelined architecture. We use PE [l,j] to denote the modular PE located in the l-th row and j-th column, where l = 0,1,2,3 and j = 0,1,2. We use distRAM for the data memory in each PE, so that the overall architecture can be easily fit on FPGA and the memory access in each PE is localized.

We are using incorporate range search method, in this method, two values are loaded at each pipeline stage, per rule the packet header value is checked against both lower and upper bound of rules. These values either can be stored locally in the pipeline stage or stored in stage memory. In the incorporate range search, stages do not require storage of entire bits. In the pipeline architecture each individual bit is independent on the other bits therefore partitioning is possible. Therefore there is no requirement to load all N bits at each pipeline stage, only N/P bits are required to load in pipeline stage, where P is number of partitions. Thus memory bandwidth requirement is decreased. XNOR gate is used for comparison of rules and header field. Rules and header field are the inputs of XNOR gate, due to use of XNOR gate we eliminate the generation of bit vectors by field splitting which improves the memory. Bits of header field are used as address to stage memory. The output bit vector of stage memory and the bit-vector generated from the previous stage are bitwise ANDed together and generate final bit vector of the current stage and the priority encoder find out the highest priority match from the resultant bit-vector.

We define horizontal direction as the forward (right) or backward (left) direction along which the BVs are propagated. We use output registers of modular PEs to construct horizontal pipelines (e.g., PE [0,0], PE [0,1], and PE [0,2]). The data propagated in the horizontal pipelines mainly consist of BVs.

The below diagram shows the 2 dimensional pipelined architecture for multi field packet classification. The processing elements are arranged in a 2 dimensional pipelined passion.

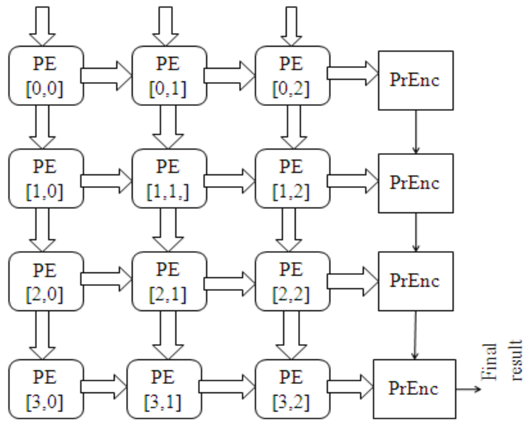


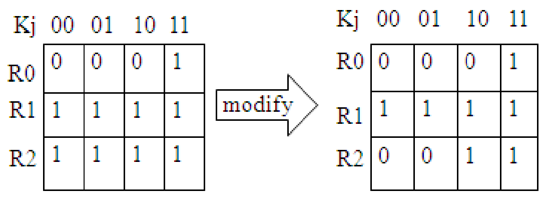
Fig 3: A 2 dimensional pipelined architecture (N=4, L=3) and priority encoders

Dynamic updates;

Open Flow packet classification requires the hardware to adapt to frequent incremental updates for the rule set during run-time. In this section, we propose a dynamic update scheme which supports fast incremental updates of the rule set without sacrificing the pipeline performance.

Modification

After the RID check, suppose RID R already exists in the rule set; Rule modification can be performed as: Given a rule with RID R, along with all of its field values and priority, compute the up-to-date BVs, and replace the outdated BVs in the BV arrays with the up-to-date BVs. In any subfield, a rule is represented by a ternary string {0,1,*}s. In reality, a rule is represented by two binary strings, the first string specifying the non-wildcard ternary digits while the second string specifying the wildcards.



Kj	00	01	10	11
R0	0	0	0	1
R1	1	1	1	1
R2	1	1	1	1

Kj	00	01	10	11
R0	0	0	0	1
R1	1	1	1	1
R2	0	0	1	1

Fig 4: modifying R2

Modifying Prefix Rule:

Algorithm 1 Up-to-date BVs in subfield j

Input n ternary strings each of s bits: $T_{i,j}$, where $T_{i,j} \in \{0,1,*\}^s$, $i = 0, 1, \ldots, n-1$.

Output 2^s BVs each of n bits: $B_j^{(k_j)} = \overline{b_{j,0}^{(k_j)} b_{j,1}^{(k_j)} \ldots b_{j,n-1}^{(k_j)}}$, where $b_{j,i}^{(k_j)} \in \{0,1\}$, $k_j = 0, 1, \ldots, 2^s - 1$, and $i = 0, 1, \ldots, n-1$.

```
1: for $i = 0, 1, \ldots, n-1$ do
2:   for $k_j = 0, 1, \ldots, 2^s - 1$ do
3:     if $k_j$ matches $T_{i,j}$ then
4:       $b_{j,i}^{(k_j)} \leftarrow 1$
5:     else
6:       $b_{j,i}^{(k_j)} \leftarrow 0$
7:     end if
8:   end for
9: end for
```

The BVs are arranged in an orthogonal direction to the rules in the data memory. To modify a single rule, 2s memory write accesses are required in the worst case. As can be seen later, even in the worst case, no more than 2s bits are modified in our approach. We show an example for rule modification in above figure. In this example, we modify the subfield j = 0 of the rule R2 in Figure 4. In this subfield, based on Algorithm 1 [9], R2 is to be updated from "0*" to"1*". A naive solution is to update the entire BV array. However, since we exploit distRAM for data memory, each bit of a BV is stored in one distRAM Entry; this means every bit corresponding to a rule can be modified independently. Hence in above figure, to update the subfield j = 0 of R2, only 4 bits have to be modified (in 4 memory accesses). To avoid data conflicts, memory write accesses are configured as single-ported. Hence in any subfield, we always allocate 2s clock cycles for 2s memory write accesses (worst case) for simplicity.

If the update process requires the priority of the old rule to be changed, i.e., the new rule and the old rule have different priorities; we update the priority encoders based on a dynamic tree structure. The time complexity to update the dynamic tree is O (log N). In general, if a prioritized rule set requires prefix match to be performed, the parallel time complexity for modifying a rule is O(max[2s,log N]).

Deletion:

After the RID check, suppose RID R already exists in the rule set; Rule deletion can be performed as: Given a RID R, delete the rule with RID Ri from the rule set. i.e., Ri should no longer produce any matching result. To handle rule deletion, let us first consider all the n rules handled by a particular horizontal pipeline consisting of [L/S] PEs. We propose to use n valid bits to keep track of all the n rules. A valid bit is a binary digit indicating the validity of a specific rule. A rule is valid only if its corresponding valid bit is set to "1".

For a rule to be deleted, we reset its corresponding valid bit to "0". An invalid rule is not available for producing any match result. We show an example for rule deletion in below figure. In this example, initially R0 and R1 are valid; R2 is invalid. R1 is to be deleted from the rule set. During the deletion, the valid bit corresponding to R1 is reset to "0". The n valid bits are directly ANDed with the bit vector of length n propagated through the horizontal pipeline. As a result, if a rule is invalid, the corresponding position for this rule in the final AND result can only be "0", indicating the input does no match this rule.

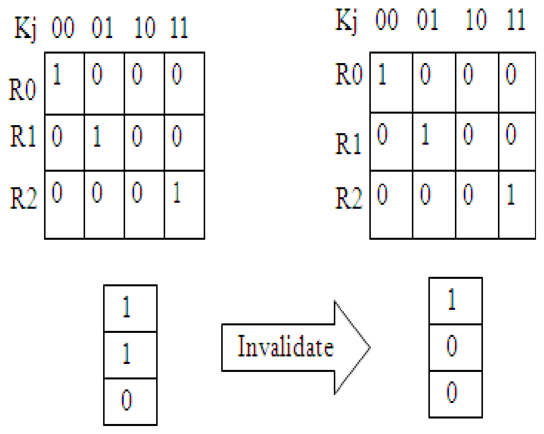


Fig 5: Deleting an old rule

Insertion:

After the RID check, suppose RID R does not exist in the rule set; Rule insertion can be performed as: Given a rule with RID R, along with all of its field values and priority, add the new rule with RID R into the rule set. i.e., checks the valid bits, and modifies one of the invalid rules and validates the new rule.

To insert a rule, (1) we first check whether there is any invalid rule in the rule set; we denote this process as validity check. (2) Then we reuse the location of any invalid rule to insert the new rule: we modify one of the invalid rules into the new rule by following the same algorithm presented. Finally, we validate this new rule by updating its corresponding valid bit.

The below figure shows how to insert a new rule. A new rule can be inserted by changing the rule value. Before that the RID check is performed whether the new rule inserted is present in the rules which are used before.

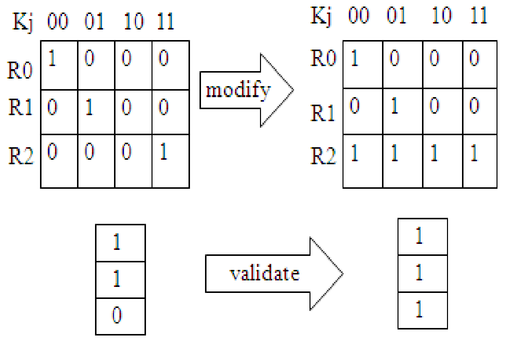


Fig 6: Inserting a new rule R as R2

Above figure shows an example of rule insertion in a subfield. In this figure, initially rule R2 is invalid as indicated by the valid bit. During insertion, the locality in the BV array corresponding to R2 is reused by the new rule R. We validate the new rule R by setting its valid bit to "1".

IV. TWO DIMENSIONAL PIPELINED ARCHITECTURE USING MODULAR PE WITH PARITY GENERATOR

Modular PE with parity generator

The below diagram explains the concept of classifying the packets while providing dynamic updates along with parity-generator.

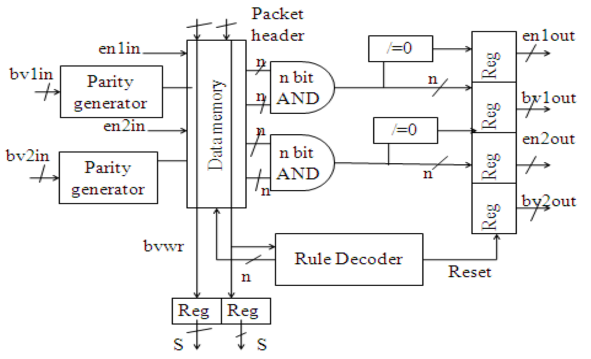


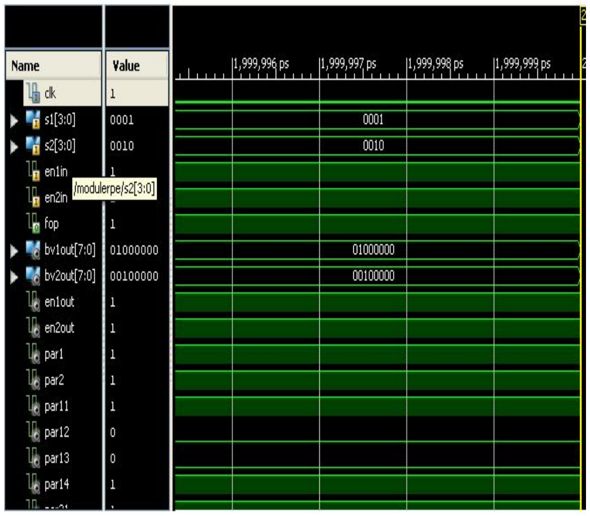
Fig 7: Modular PE with parity generator

The parity generator generates the parity value for the input bit vectors which are to be matched. For this purpose the XOR operation for the bit vector bit by bit individually. The corresponding parity value is set for the input bit vectors. The header fields are checked for packet matching with the given rules by performing XNOR operation. The result of XNOR operation gives individual bit vectors. All the bit vectors are merged to get the final output by performing and operation for the partial results. The final output is considered as matched output. The party values are also generated for the contents in the data memory.

If the parity of input bit vector matched with any of the contents in the data memory then that corresponding bit vector is the final bit vector of the output and the final output is said to be matched. The searching speed is also increased due to reduction in complexity and reduction in parameter comparison operations. By using parity bits, delay for each search operation is reduced. Hence, it boosts the search speed of parallel CAM. We employ dual-port (read) data memory on FPGA. Two concurrent packets can be processed in each modular PE. the input BVs for the two concurrently processed packets as bv1in and bv2in, respectively; we denote the output BVs for the two concurrent packets as bv1out and bv2out, respectively. The throughput is twice the maximum clock rate achieved on FPGA. Assuming the same clock rate can be sustained, this technique doubles the throughput achieved by the modular PE. For each of the two concurrent packets, the modular PE compares an s-bit subfield of the packet header against a set of n rules.

V. EXPERIMENTAL RESULTS

We conducted experiments using Xilinx ise design suite 14.5, targeting on FPGA.



Name	Value
clk	1
s1[3:0]	0001
s2[3:0]	0010
en1in	1
en2in	
fop	1
bv1out[7:0]	01000000
bv2out[7:0]	00100000
en1out	1
en2out	1
par1	1
par2	1
par11	1
par12	0
par13	0
par14	1

Fig8: Output waveforms for Multi field packet classification with parity

In our approach, the construction of BVs does not explore any rule set features, the performance of our architecture is rule set independent.

the shape or morphology in an image. According to the Comparision of Existed and Proposed Approach (S=2, L=4, N=8):

S:Stride, L:Packet header length, N:Size of the rule set

Table 1: Comparision of Existed and Proposed Approach (S=2, L=4, N=8)

Parameters	**Decomposition based self-reconfigurable PEs on FPGA**	**Decomposition based self-reconfigurable PEs with Parity generator on FPGA**
Latency(ns)	4.114	4.040
Time(ns)	15.339	12.338
Clockrate (MHz)	65.2	81.05

VI. CONCLUSION

Due to increase in demand of data, it is a great challenge to develop rule set independent solutions for next generation packet classification that supports larger rule set and more packet header fields. In this project dynamically updatable packet classification using parity generator is implemented which rule is set independent and reduces the look up operations and enhances the search speed. With the addition of extra parity block the delay is also reduced.

REFERENCES

[1] Yun R. Qu and Viktor K. Prasanna," High-performance and Dynamically Updatable Packet Classification Engine on FPGA" IEEE, 2015.

[2] Y. R. Qu, S. Zhou, and V. K. Prasanna, "High-performance architecture for dynamically updatable packet classification on FPGA," in ACM/IEEE Symp on Architectures for Networking and Communications Systems (ANCS), 2013, pp. 125–136.

[3] Z. P. Ang, A. Kumar, and Y. Ha, "High Speed Video Processing Using Fine-Grained Processing on FPGA Platform," in Proc. of IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2013, pp. 85–88.

[4] R. Salvador, A. Otero, J. Mora, E. de la Torre, T. Riesgo, and L. Sekanina, "Self-Reconfigurable Evolvable Hardware System for Adaptive Image Processing," IEEE Transactions on Computers, vol. 62, no. 8, pp. 1481–1493,2013

[5] T. Ganegedara and V. K. Prasanna, "StrideBV: Single Chip 400G+ Packet Classification," in Proc. of IEEE International Conference on High Performance Switching and Routing (HPSR), 2012, pp. 1–6.

[6] L. Frigerio, K. Marks, and A. Krikelis, "Timed Coloured Petri Nets for Performance Evaluation of DSP Applications: The 3GPP LTE Case Study," in VLSI-SoC: Design Methodologies for SoC and SiP. Springer Berlin Heidelberg, 2010, vol. 313, pp. 114–132.

[7] Y.-H. E. Yang and V. K. Prasanna, "High Throughput and Large CapacityPipelined Dynamic Search Tree on FPGA," in Proc. of ACM/SIGDA International Symp. on Field Programmable Gate Arrays (FPGA), 2010, pp. 83–92.

[8] A. Sudarsanam, R. Barnes, J. Carver, R. Kallam, and A. Dasu, "Dynamically Reconfigurable Systolic Array Accelerators: A Case Study with Extended Kalman Filter and Discrete Wavelet Transform Algorithms," Computers Digital Techniques, IET, vol. 4, no. 2, pp. 126–142, 2010.

[9] W. Jiang and V. K. Prasanna, "Field-split Parallel Architecture for High Performance Multi-match Packet Classification using FPGAs," in Proc. of Annual Symp. on Parallelism in Algs. And Arch. (SPAA), 2009, pp. 188–196.

[10] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar,L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow:Enabling Innovation in Campus Networks," SIGCOMM Comput. Commun. Rev., vol. 38, no. 2, pp. 69–74, 2008.

[11] "OpenFlow Switch Specification V1.3.1,"https://www.opennetworking.org/images/stories/downloads/sdnresources/onf-specifications/openflow/openflow-spec-1.3.1.pdf.

[12] P. Gupta and N. McKeown, "Algorithms for packet classification," IEEE Network, vol. 15, no. 2, pp. 24–32, 2001.

[13] F. Yu, R. H. Katz, and T. V. Lakshman, "Efficient Multimatch Packet Classification and Lookup with TCAM," IEEE Micro,vol. 25, no. 1, pp. 50–59, 2005.

[14] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary, "Algorithms for Advanced Packet Classification with Ternary CAMs," SIGCOMM Comput. Commun. Rev., vol. 35, no. 4, pp. 193–204, 2005.