FAST DYNAMICALLY UPDATABLE PACKET CLASSIFIER ON FPGA *

Yun R. Qu  
Ming Hsieh Dept. of Electrical Engr.  
University of Southern California  
Los Angeles, CA 90089  
email: yunqu@usc.edu

Viktor K. Prasanna  
Ming Hsieh Dept. of Electrical Engr.  
University of Southern California  
Los Angeles, CA 90089  
email: prasanna@usc.edu

ABSTRACT
Packet classification requires multiple fields of the packet header to be matched against entries in a prioritized table; it is still challenging to support dynamic updates for packet classification without sacrificing throughput performance. In this paper, we present a high-throughput pipelined architecture for packet classification on FPGA supporting dynamic updates of the rule set. This architecture is based on Dynamic Bit Vector (Dynamic-BV) approach and supports modify, delete and insert operations during run-time with very little impact on sustained throughput. Experimental results show that, for a 1K rule set on a state-of-the-art FPGA, a throughput of 120 Gbps with 1 million updates/second can be sustained using a single pipeline.

1. INTRODUCTION
The emerging class of network applications requires the hardware to perform packet classification [1] with incremental updates. It would be very expensive, if at all feasible, to reconstruct an optimal packet classifier repeatedly to support timely updates. While many sophisticated solutions have been proposed for packet classification supporting dynamic updates [2], [3], due to the rapid growth of the Internet, supporting dynamic updates of the rule sets without loss of throughput performance remains a challenging problem.

Many existing hardware-based solutions for multi-field packet classification employ ternary content addressable memories (TCAMs) [4], [5]. TCAMs are expensive [6]. Recent research has proposed to use Field-Programmable Gate Array (FPGA) technology as a power-efficient alternative for implementing real-time network processing engines [7]. FPGA-based packet classification engines can achieve very high throughput for rule sets of moderate size. However, little work has been done to support efficient dynamic updates of rule sets for packet classification on FPGA.

In this paper, we propose Dynamic-BV, a classification approach which supports incremental updates of the rule set during run-time. We present a parallel architecture for Dynamic-BV on FPGA; this architecture simultaneously sustains high throughput (120 Gbps) and supports high update rate (1 million updates/second) for 1K rules on a state-of-the-art FPGA. The rest of the paper is organized as follows: Section 2 introduces the packet classification problem. We present the algorithms and the architecture for Dynamic-BV in Section 3. Section 4 provides the performance evaluation. Section 5 concludes the paper.

2. BACKGROUND

2.1. Packet Classification with Dynamic Updates
The classic five-field packet classification involves classifying packets based on the five fields in the packet header [6]. The individual predefined entries for classifying a packet are called rules, which are stored in a rule set. Each rule has a rule ID (RID), 5 fields and their associated values, a priority, and an action to be taken if matched. A packet is considered matching a rule only if it matches all the fields in that rule.

Incremental updates (modification, deletion and insertion) of the rule set are often required for packet classification. In [2], two algorithms based on tree/trie structures are proposed to support dynamic updates. For a d-dimensional rule set consisting of N rules, they require $O(\log^d N)$ and $O(\log^d N)$ update time, respectively. For OpenFlow, $d = 12$; thus, these algorithms are expensive. A dynamically updatable packet classification scheme using tree structures on TCAM is presented in [8]. In this approach, rules are partitioned among TCAM subarrays; in the worst case, an inserted rule moves across different subarrays, which is expensive.

2.2. Prior Work on FPGA
Decision-tree based approaches [9] involves cutting the search space into smaller subspaces based on the information from one or more fields in the rule. The resulting tree can be mapped onto a pipelined architecture. However, if the rule...
set is updated, the decision-tree must be recomputed and mapped onto FPGA; this is an expensive operation.

Decomposition based approaches [11] first search each packet header field individually. The partial results are merged to produce the final result. Hash-based techniques can be used to merge the partial results [12]. However, reconfiguring the hash functions is an expensive operation. Bit Vector (BV) [10] is another decomposition based approach; in BV, the lookup in each field returns a vector, each bit corresponding to a rule. A bit is set to “1” only if the input matches the corresponding rule in that field. A bit-wise logical AND operation gathers the matches from all the fields in parallel.

2.3. BV-based approaches
A Field-Split BV (FSBV) approach has been proposed on FPGA [13]. Suppose the rules are represented using field values (strings of ternary digits) as shown in Figure 1. FSBV splits a \( W \)-bit field into \( W \) subfields. In a subfield, for each rule, 2 bits are generated for “0” and “1” bit vectors, respectively. The rules are translated into a set of bit vectors and kept in a bit vector matrix (denoted as \( B \)-matrix). A pipelined architecture is constructed to perform packet header match: each stage extracts a bit vector in a 1-bit subfield. The bit vectors for all subfields are ANDed to report the final match. Although FSBV does not require knowledge of any rule set features except the rule set size, it does not support dynamic updates of the rule set.

3. DYNAMIC-BV
We base the algorithms for Dynamic-BV on FSBV approach [13] and support rule modify, delete and insert operations: (1) For rule modification, we update the corresponding “0” and “1” bit vectors in the \( B \)-matrix. (2) For rule deletion, we keep a “valid” matrix (\( V \)-matrix), each bit of which associated with a particular rule; we set the bit to “0” to invalidate a rule. (3) For rule insertion, we reuse the locations of the B-matrix whose corresponding V-matrix bits are invalid.

3.1. Modify
Let \( N \) denote the total number of the rules. In the FSBV approach as shown in Figure 1, each of the \( W \) subfields corresponds to a pair of \( N \)-bit vectors. Let us use \( k \in \{0, 1\} \) to index those two bit vectors.

To modify a rule in a 1-bit subfield, we need to update two bit vectors in the worst case: one update to edit the “0” bit vector and another update for the “1” bit vector. Since each bit vector corresponds to \( N \) rules, multiple rules can be updated concurrently during a single update of the bit vector. We construct the bit vectors that need to be placed into the \( B \)-matrix; then we replace the bit vectors that are to be retired.

3.2. Delete
To handle rule deletion efficiently, we use a \( V \)-matrix to keep a valid bit for each rule. A rule is valid for producing matching results if and only if the corresponding valid bit in the \( V \)-matrix is “1”. If a rule is to be deleted from the rule set, we change the bit to “0” to invalidate that rule. If a rule is invalid, the rule is not available for producing any matching output. To validate or invalidate a rule, the \( V \)-matrix needs to be updated. We show an example of rule deletion in Figure 2, where rule \( R_1 \) is deleted from the rule set.

3.3. Insert
We insert a rule by first checking whether there is any invalid bit in the \( V \)-matrix. We denote this process as vacancy check. If there is any bit in the \( V \)-matrix having a “0” value, we set that bit to “1” and modify the bit vectors in the \( B \)-matrix in the corresponding position.
Stage Register
0 …
1 …
Bit vector
Input packet header
0 …
1 …
AND
Stage 0
Stage 1
…
Stage ...
Highest-priority match
Stage Register
Bit vector
ID Valid bit
V-matrix
valid bits
AND
Index \( k \)

An example of rule insertion is shown in Figure 3. In this example, initially two rules (\( R_0, R_1 \)) with 4-bit field values 000* and 110* are in the rule set; \( R_1 \) is invalid. The new rule \( R_2 \) having 4-bit field value 11** is placed in the corresponding position of \( R_1 \) in the B-matrix. Notice only a single bit is flipped during the deletion. Also, the vacancy check is independent of the packet header match process, therefore it has little effect on the performance of the pipeline.

3.4. Architecture

We show the pipeline architecture in Figure 4. All the control signals are omitted for simplicity.

For a \( d \)-field rule set, each field having at most \( W \) bits, we partition the B-matrix into \( d \cdot W \) pairs of bit vectors, each pair corresponding to a 1-bit subfield. As shown in Figure 4, we store each pair of bit vectors in the on-chip memory of a pipeline stage.

In each pipeline stage, to extract a bit vector from the memory, the packet header bit in a specific subfield is compared with the index \( k \) of the bit vectors stored in the corresponding pipeline stage. Then the bit vectors extracted from the memory and the bit vectors output by the previous stage are ANDed; the resulting bit vector is output to the next stage. However, in the first pipeline stage, the bit vectors extracted from memory is directly ANDed with the \( N \) valid bits from the V-matrix. Also, since the update scheme does not require any sequencing for the bits in the bit vectors, to report the match with the highest priority, we use a priority encoder to sort all matching rules.

We implement the V-matrix using small-sized distributed RAM (distRAM) since the size of the V-matrix is only \( N \) bits. We implement the V-matrix in the first stage only, allowing the V-matrix update to be completed before updates of bit vectors. For bit vectors stored in each stage, we use either dual-port distRAM or dual-port Block RAM (BRAM).

4. PERFORMANCE EVALUATION

4.1. Experimental Setup

We conducted experiments using Xilinx ISE Design Suite 14.4, targeting the Virtex 6 XC6VLX760 FFG1760-2 FPGA [15]. It has 1200 I/O pins, 26Mb BRAM (720 RAMB36 blocks), and can be configured to realize large amount of distRAM (up to 8Mb). Clock rate and resource consumption are reported using post-place-and-route results. Since the algorithms and the proposed architecture for Dynamic-BV do not depend on any rule set features, we use randomly-generated bit vectors and packet headers for 5-field classification in order to evaluate our design. In our experiments, the number of rules is varied from 128 to 1K. Considering 40-byte packets, we study the performance trade-offs with respect to: (1) Peak throughput when no update is performed; (2) Sustained throughput with update operations; (3) Update rate of the rule set; (4) Resource consumption on FPGA.

4.2. Evaluation Results

We denote the designs where the bit vectors are stored in distRAM as distRAM-based designs; similarly, BRAM-based designs can be defined. For both types of designs, we configure the memory modules to be dual-ported for read access. We summarize the achievable post-place-and-route clock rates in Table 1. (1) As the number of rules \( N \) increases, the length of the bit vectors also increases. More memory module of fixed size\(^1\) have to be used. Therefore the clock rate drops due to longer wires connecting larger amounts of memory modules. (2) The distRAM-based designs outperform the BRAM-based designs since distRAM

\(^1\)RAMB36 or 64-bit Lookup Up Table (LUT).
In this paper, we proposed Dynamic-BV and its corresponding high-throughput parallel architecture on FPGA for rule modify, delete and insert operations during run-time. We plan to design a 2-dimensional distRAM-based pipelined architecture on FPGA in the future to further enhance the overall performance.

## 6. REFERENCES


