A Scalable and Modular Architecture for High-Performance Packet Classification

Thilan Ganegedara, Weirong Jiang, and Viktor K. Prasanna, *Fellow, IEEE*

Abstract—*Packet classification* is widely used as a core function for various applications in network infrastructure. With increasing demands in throughput, performing *wire-speed* packet classification has become challenging. Also the performance of today's packet classification solutions depends on the characteristics of rulesets. In this work, we propose a novel modular Bit-Vector (BV) based architecture to perform high-speed packet classification on Field Programmable Gate Array (FPGA). We introduce an algorithm named StrideBV and modularize the BV architecture to achieve better scalability than traditional BV methods. Further, we incorporate *range search* in our architecture to eliminate ruleset expansion caused by range-to-prefix conversion. The post place-and-route results of our implementation on a state-of-the-art FPGA show that the proposed architecture is able to operate at 100+ Gbps for minimum size packets while supporting large rulesets up to 28 K rules using only the on-chip memory resources. Our solution is *ruleset-feature independent*, i.e. the above performance can be guaranteed for any ruleset regardless the composition of the ruleset.

Index Terms—Packet classification, firewall, router, network security, FPGA, ASIC, hardware architectures, networking

1 Introduction

Packet classification is a prominent technique used in networking equipment for various purposes. Its applications are diverse, including network security, access control lists, traffic accounting, etc [4], [5], [10], [11]. To perform packet classification, one or more header fields of an incoming packet is checked against a set of predefined rules, usually referred to as a *ruleset* or a *classifier*.

Performing packet classification is challenging since it involves inspection of multiple fields against a ruleset possibly containing thousands of rules. Performing such operations at *wire-speed* is even more challenging with the increasing throughput demands in modern networks. Various hardware platforms have been employed for packet classification in the past [2], [6], [10]. Field Programmable Gate Arrays (FPGAs) offer reconfigurability and deliver high performance due to the custom built nature of the architectures [13], [14]. Application Specific Integrated Circuits (ASICs) on the other hand offer little flexibility in the architecture once fabricated, but deliver superior performance compared with FPGA. Both platforms are widely used in networking applications to implement multi-gigabit packet forwarding engines. The throughput of hardware based routers routers can be dramatically increased by employing pipelining techniques [6], [10].

While numerous solutions exist for packet classification, the effectiveness of these solutions rely on the characteristics of the ruleset [12]. When the assumed features are not present in a ruleset, the memory consumption of such solutions may increase dramatically. For example, decision tree based approaches heavily rely on the distribution of the rules in the multi-dimensional space to reduce the tree height and rule duplication. Therefore the solution generated for a particular ruleset may not necessarily yield the same performance for a different ruleset with different features. Further, optimizations that target a specific ruleset feature, are not robust to be employed in environments where the ruleset changes dynamically and frequently.

In this work, we present a novel architecture for packet classification, whose performance is independent from ruleset features and is suitable for hardware platforms. We use a Bit-Vector (BV) based approach to represent the ruleset. Each rule is represented as a collection of sub-rules and we propose an algorithm named StrideBV to generate the sub-rules, which is an extension of the Field Split Bit Vector (FSBV) algorithm proposed in [6]. Our solution offers the user the flexibility of deciding the bit width of each sub-rule, which in turn decides the performance trade-off of the architecture. In order to handle the arbitrary ranges, we augment the architecture with explicit range search capability which does not require any range-to-prefix conversion. This yields higher memory efficiency which enhances the scalability of our approach. In addition, we propose a rule priority based partitioning scheme which allows us to modularize our solution to eliminate the inherent performance limitations in the traditional BV approaches.

We evaluate the proposed solution on a Xilinx Virtex-7 Field Programmable Gate Array (FPGA) device and we report the post place-and-route performance results in this paper. The modular packet classification engine is implemented as a pipelined architecture and we measure its performance with respect to throughput, memory, and power consumption. Our extensive experiments show that

- *T. Ganegedara and V. K. Prasanna are with the Department of Electrical Engineering, University of Southern California, Los Angeles, CA, USA. E-mail: {gangeda, prasanna}@usc.edu.*
- *W. Jiang is with the Xilinx Research Labs, San Jose, CA, USA. E-mail: weirongj@acm.org.*

Manuscript received 11 May 2013; revised 10 Sept. 2013; accepted 27 Sept. 2013. Date of publication 9 Oct. 2013; date of current version 21 Mar. 2014. Recommended for acceptance by S. Aluru.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2013.261

for varying module sizes, the throughput remains above 100 Gbps for minimum size packets, which makes our solution ideal for 100G line-card solutions. The dynamic power consumption of the proposed architecture is only 0.8 mW/rule on the average which is 5× lower than those of Ternary Content Addressable Memory (TCAM) solutions. To the best of our knowledge, this is the first solution available in literature that delivers deterministic performance for any given ruleset.

To summarize, we list our contributions in this work as follows:

- A memory-efficient bit vector-based packet classification algorithm suitable for hardware implementation;
- A modular architecture to perform large-scale high-speed (100+ Gbps) packet classification on a single chip;
- Inclusion of explicit range-search in the modular architecture to eliminate the expensive range-to-prefix conversion;
- Support for up to 28K rule classifiers using on-chip distributed and block Random Access Memory (RAM) on a Xilinx Virtex 7 FPGA;
- Deterministic performance for any given ruleset due to ruleset-feature independence.

2 Background and Related Work

Packet classification algorithms have been extensively studied in the past decade. A comprehensive survey can be found in [12]. Most of those algorithms fall into two categories: decomposition-based and decision-tree-based approaches. In this section, only decomposition based approaches will be discussed as the proposed solution belongs to this category.

Decomposition-based algorithms (e.g. Parallel Bit-Vector (BV) [7]), perform independent search on each field and finally combine the search results from all fields. Such algorithms are desirable for hardware implementation due to their parallel search on multiple fields. However, substantial storage is usually needed to merge the independent search results to obtain the final result. Taking the BV algorithm as an example, it performs the parallel lookups on each individual field first. The lookup on each field returns a bit-vector with each bit representing a rule. A bit is set if the corresponding rule is matched on this field; a bit is reset if the corresponding rule is not matched on this field. The result of the bitwise AND operation on these bit-vectors indicates the set of rules that matches a given packet. The BV algorithm can provide a high throughput at the cost of low memory efficiency.

By combining TCAMs and the BV algorithm, Song *et al.* [10] present an architecture called BV-TCAM for multi-match packet classification. A TCAM performs prefix or exact match, while a multi-bit trie implemented in Tree Bitmap [1] is used for source or destination port lookup. The authors never report the actual FPGA implementation results, though they claim that the whole circuit for 222 rules consumes less than 10 percent of the available logic and fewer than 20 percent of the available Block RAMs of a Xilinx XCV2000E FPGA. They also predict the design after pipelining can achieve 10 Gbps throughput when implemented on advanced FPGAs.

Taylor *et al.*[11] introduce Distributed Cross-producting of Field Labels (DCFL), which is also a decomposition-based algorithm leveraging several observations of the structure of real filter sets. They decompose the multi-field searching problem and use independent search engines, which can operate in parallel to find the matching conditions for each filter field. Instead of using bit-vectors, DCFL uses a network of efficient aggregation nodes, by employing Bloom Filters and encoding intermediate search results. As a result, the algorithm avoids the exponential increase in the time or space incurred when performing this operation in a single step.

3 Ruleset-Feature Independent Packet Classification

We develop our complete solution in two phases, which are 1) StrideBV algorithm, and 2) integration of range search and modularization of StrideBV architecture. These two phases are discussed in detail in this section. First we describe the usage of bit-vectors in the domain of packet classification.

3.1 StrideBV Algorithm

In [6] an algorithm named Field Split Bit-Vector (FSBV) was introduced to perform individual field search as a chain of sub-field searches, in the form of bit-vectors. The motivation was to improve memory efficiency of the packet classification engine by exploiting various ruleset features. Even though the FSBV algorithm itself does not rely on ruleset features, the architecture proposed in [6] relies on the features of the ruleset. However, as mentioned above, such ruleset features may not be present in all classifiers and the lack of thereof can potentially yield poor performance.

Considering the diverse nature of packet classification rulesets [11], [12], a robust solution that is able to guarantee the performance for any classifier is in demand. The FSBV algorithm proposed in [6] has the characteristics of being feature independent. However, the algorithm was applied only to a set of selected fields with memory optimization being the major goal. This caused the overall solution to be ruleset-feature reliant. In this work, we generalize the FSBV algorithm and extend the use of it to build a ruleset-feature independent packet classification solution, named *StrideBV*.

In the original FSBV algorithm, each W bit field was split into W 1 bit sub-fields. However, it is possible to generalize the sub-field bit length without affecting the operation of the packet classification process. In the proposed solution, a sub-field length of k bits is considered and we refer to such a sub-field as a *stride*. Due to this reason and the underlying data structure being bit-vector, the proposed solution is named StrideBV. We discuss the StrideBV algorithm in detail here.

Each k bits of the W bit rule can perform independent matching on the corresponding k bits of an input packet header. In other words, we can divide the W bit rule into W/k of k bit sub-fields. Each sub-field corresponds to 2^k N bit-vectors: a N bit-vector for each permutation of the k bits. A $wildcard(*)$ value in a ternary string is mapped to all

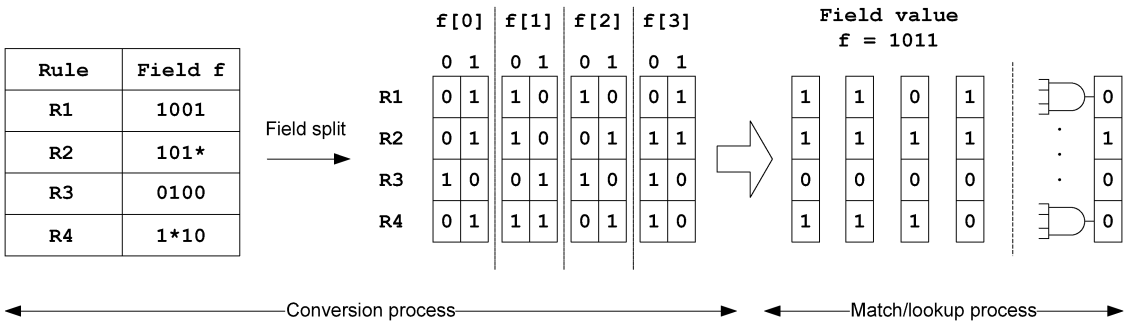


Fig. 1. FSBV bit-vector generation and header processing example.

bit-vectors. To match an input header with this W bit rule, each k bits of the header of the input packet will access the corresponding memory whose depth[1] is 2^k, and return one N bit-vector. Each such k bit header lookup will produce a N bit-vector and they are bitwise *AND*ed together in a pipelined fashion to arrive at the matching results for the entire classifier. The memory requirement of the StrideBV approach is fixed for a classifier that can be represented in a ternary string format (i.e. a string of 0, 1 and $*$) and can be represented as $\Theta(2^k \times N \times W/k)$. The search time is $O(W/k)$ since a bitwise *AND* operation can be done in $O(1)$ time in hardware and $O(W/k)$ such operations need to be performed to complete the classification process. Fig. 1 shows an example of applying the StrideBV algorithm for matching a packet header of 4 bits with a 4 bit ruleset. For simplicity and clarity, we show the BV generation and the lookup process for $k = 1$.

One caveat with StrideBV is that it cannot handle arbitrary ranges in an efficient manner. The two port fields (SP and DP) may be represented as arbitrary ranges. For example, when a certain rule needs to filter the incoming traffic belonging to one of the well-known port numbers, the rule will have in its DP field: 0-1023. In such cases, the arbitrary ranges need to be converted into prefix format where the value of each bit can be 0, 1, or $*$ (wildcard character). In the following analysis, we assume a port field of a rule can be represented as a ternary string and later, in Section 3.2.2, we demonstrate how to eliminate this requirement to represent arbitrary ranges in a memory efficient manner for improved scalability.

The formal algorithms for building the bit-vectors and for performing packet classification, are shown in Algorithms 1 and 2, respectively. The *Permutations* function accepts k bits of the rule and generates a 2-dimensional array of size $2^k(height) \times k(width)$, in which, each row value specifies the bit-vector values for the considered stride. For example, consider a stride size of $k = 2$ and the rule value for a particular stride is $0*$. In this case, the array output by the *Permutations* function will be {11, 11, 01, 01}, which is the match result of {00, 01, 10, 11} against rule value $0*$. The value at the jth position of the array indicates the bit-vector value for the considered stride, when the input packet header has a value of the binary representation of j.

Algorithm 1 Bit-vector Generation.

Require: N rules each of which is represented as a W-bit ternary string: $R_n = T_{n,W-1}T_{n,W-2}\cdots T_{n,0}$, $n = 0, 1, \ldots, N-1$
Ensure: $2^k \times W/k$, N-bit-vectors:
$V_{i,j} = B_{i,j,N-1}B_{i,j,N-2}\cdots B_{i,j,0}$,
$i = 0, 1, \ldots, W/k-1$, and $j = 0, 1, \ldots, 2^k-1$
1: Initialization: $V_{i,j} \leftarrow 00\cdots0\ \forall i, j$
2: **for** $n \leftarrow 0$ to $N-1$ **do** {Process R_n}
3: **for** $i \leftarrow 0$ to $W/k-1$ **do**
4: $\mathrm{S}[2^k][\mathrm{k}] = \mathrm{Permutations}(R[i*k : (i+1)*k])$
5: **for** $j \leftarrow 0$ to 2^k-1 **do**
6: $V_{i,j}[i*k : (i+1)*k] = S[j]$
7: **end for**
8: **end for**
9: **end for**

Algorithm 2 Packet Classification Process.

Require: A W-bit packet header: $P_{W-1}P_{W-2}\cdots P_0$.
Require: $2^k \times W/k$, N bit-vectors:
$V_{i,j} = B_{i,j,N-1}B_{i,j,N-2}\cdots B_{i,j,0}$,
$i = 0, 1, \ldots, W/k-1$, and $j = 0, 1, \ldots, 2^k-1$
Require: A N bit-vector V_m to indicate match result
Ensure: N bit-vector V_m indicates all match results
1: Initialize V_m: $V_m \leftarrow 11\cdots1$ All rules match initially
2: **for** $i \leftarrow 0$ to $W/k-1$ {bit-wise AND}
3: $j = [P_{i*k} : P_{(i+1)*k}]$
4: $V_m \leftarrow V_m \quad V_{i,j}$
5: **end for**

3.1.1 Multi-Match to Highest-Priority Match

In [6], [10], the output of the lookup engine is the bit-vector that indicates the matching rules for the input packet header. This is desirable in environments such as Intrusion Detection Systems (IDSs) where reporting all the matches is necessary for further processing. However, in packet

1. Throughout this paper, *depth* of a memory is defined as the number of entries in it. The *width* of a memory is defined as the number of bits in each entry.

classification, only the highest priority match is reported since routing is the main concern.

The rules of a classifier are sorted in the order of decreasing priority. Hence, extracting the highest priority match from the N bit-vector translates to identifying the first bit position which is set to 1, when traversing the bit-vector from index $0 \rightarrow N-1$. This task can be easily realized using a priority encoder. The straightforward priority encoder produces the result in a single cycle. However, when the length of the bit-vector increases, the time required to report the highest priority match increases proportionally. This causes the entire pipeline to run at a very slow clock rate for larger BV lengths, which affects the throughput.

As a remedy, we introduce a Pipelined Priority Encoder (PPE). A PPE for a N bit-vector consists of $\log_B N$ number of stages and since the work done per stage is trivial, the PPE is able to operate at very high frequencies. Here, the parameter B refers to the degree of the pipeline encoder, which essentially indicates how many comparisons are done in a particular stage. In other words, into how many partitions the bit-vector is split into in a given stage.

3.2 Modularization and Integration of Range Search

3.2.1 Modular BV Approach

Since each individual element of the BV is responsible for a single rule of the entire ruleset, each individual bit-level operation is completely independent of the rest of the bits in the BV. This allows us to partition the BV without affecting the operation of the BV approach. The benefit of partitioning is that we no longer need to load a N bit BV at each pipeline stage, but rather, a N/P bit BV, where P is the number of partitions, reducing the per pipeline stage memory bandwidth requirement by a factor of P. Partitioning is done based on the rule priority, i.e. first N/P rules in the first partition and so on. This eases the highest priority match extraction process.

To better explain the severeness of the unpartitioned BV approach, we provide quantitative insight using realistic example values. Consider a classifier (i.e. a ruleset) with 2000 rules and a pipelined architecture operating at 200 MHz. Considering a stride value of $k=3$, the pipeline length becomes $\lceil 104/3 \rceil = 35$. With no partitioning, each pipeline stage will be requesting a 2000 bit wide word at a 200 million requests per second rate. This translates to an on-chip memory bandwidth of 14 Tbps. On current FPGAs, having such high bandwidth is not possible. Therefore, the operating frequency drops with increasing ruleset size [3], affecting the performance of the packet classification engine.

Each partition will produce a portion of the longer BV, referred to as *sub-BV* hereafter, which contains the matching result(s). The aggregation of these sub-BVs is discussed in Section 3.2.3.

3.2.2 Scalable Range Matching on Hardware

As pointed out earlier, the effect of range-to-prefix conversion is significant, considering the fact that arbitrary ranges exist in real-life rulesets. The most prevalent range (well-known port numbers) 0-1023 can also cause serious effects on the memory footprint when converted to prefix format. For example, a single such range converts to 6 individual prefixes, causing a single rule to expand into 6 rules. With two such ranges appearing in a single rule for both source and destination port values, causes the rule to expand into 36 rules. However, the effect of range-to-prefix conversion may become severe for arbitrary ranges with the worst case $O(W^2)$ expansion factor.

In order to facilitate range search in hardware, we modify the pipeline stages corresponding to source/destination port fields accordingly. The search operation is a sequential comparison of unsigned integers. The result of a sub-field is carried on to the next stage and the comparison completes when the last sub-field of a range field is reached. Since the incoming packet header value needs to be checked against both lower and upper bounds of the rules, two values need to be loaded at each pipeline stage, per rule. These values either can be stored locally in the pipeline stage itself or stored in stage memory. In this work, we store them locally within the pipeline stage, by encoding the range values as logic functions. This allows us to achieve further improvements in memory efficiency as the range search stages do not require storage of BVs.

3.2.3 Multi-Level Priority Encoding

With the modular approach, multiple pipelines are required to support large-scale packet classification. Multiple rules from a classifier may match with an incoming packet, however, only the matching rule with highest priority is applied on the packet in the case of packet classification. The priority is simply the order in which the rules appear in the classifier. In order to extract this information from the BVs output from the pipelines, we implement a two stage, hierarchical priority encoder.

The first stage of the priority encoder deals with the sub-BVs produced by each sub-pipeline. It extracts the lowest bit index that is set to 1 in the sub-BV, which is the local highest priority match (Idx_{Local}). In order to compute the global match, the global index (Idx_{Global}) need to be computed. This can be easily done using $Idx_{Global} = Idx_{Local} + Idx_{Pipe} \times N/P$, where Idx_{Pipe} is the pipeline identifier, varying from 0 to $P-1$. By keeping the value of N/P (i.e. partition size) a power of 2, the multiplication can be converted to a simple bit-shift operation.

When these global indices are produced from all the sub-pipelines, they are stored in a BV in the second stage, and the same pipelined priority encoding is carried out on this BV to produce the global highest priority match. The latency introduced by the priority encoding process is $\log_{B1}(N/P)$ from the first stage and $\log_{B2} P$, where $B1$ and $B2$ are two user defined parameters with which the total latency can be controlled. $B1$ and $B2$ also defines the degree of the two priority encoders. In the case of binary mode, the values are set to $B1 = B2 = 2$. However, for larger ruleset sizes, the latency introduced by the binary priority encoder can be high. Depending on the requirement the parameters $B1$ and $B2$ can be chosen appropriately.

4 Hardware Architecture

4.1 StrideBV Architecture

In the StrideBV architecture, in each pipeline stage, k bits of header bits of the incoming packets is used as the memory address to the stage memory. The output bit-vector of stage

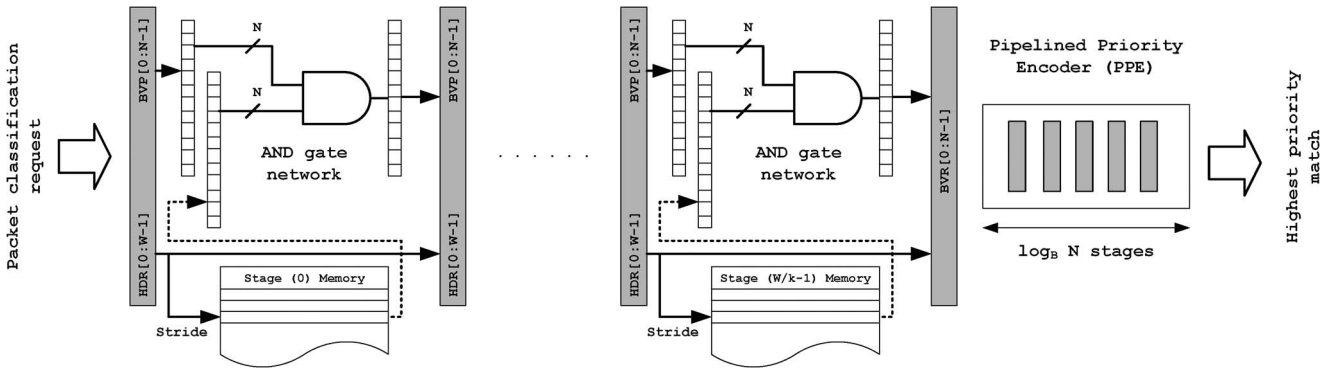


Fig. 2. StrideBV pipelined architecture (BVP/BVM/BVR-Previous/Memory/Resultant Bit-Vectors, HDR-5-field header, Stride-k bit header stride).

memory (BVM) and the bit-vector generated from the previous stage (BVP) are bitwise *AND*ed together to produce the resultant bit-vector (BVR) of the current stage. The same stage construction is adhered throughout the pipeline. The final stride lookup stage will output the multi-match result and the PPE extracts the highest-priority match from the resultant bit-vector. Note that BVP in stage 0 is set to all 1's to indicate that the entire ruleset is considered as potential matches. The architecture is presented in Fig. 2.

4.2 Range Search Integration

While there are numerous methods to handle ranges, in this work, we adopt explicit range search done in a serial fashion. The main reason for this choice is to not affect the uniformity of the pipeline stages significantly. As pointed out earlier, the StrideBV pipeline has a uniform structure, which can be easily mapped on to the FPGA fabric. In order to facilitate the stride access, at each range search stage, we consider only k bits of the header and perform a comparison with k of lower bounds (LBs) and upper bounds (UBs) of the range values to see whether the incoming value belongs in the stored ranges. The search is done from Most Significant Bits (MSBs) to Least Significant Bits (LSBs) to finally arrive at range match. This result is then **AND**ed with the bit-vector generated by the StrideBV pipeline to combine the range search results with prefix/exact match results. The architecture of the range search is shown in Fig. 3.

The operation of the sequential comparison is illustrated in Fig. 3. In each stage, the incoming packet's header stride is compared against the stride of lower and upper bounds of the rules by performing *greater than or equal* and *less than or equal* operations, respectively. The result of the first stride is forwarded to the second stride comparison and so on. When the sequential comparison is terminated these two BVs are *AND*ed together to form the complete match result for the range fields.

4.3 Modular Architecture

We present the overall architecture of the proposed solution in Fig. 4. The strides that process header fields with prefix and exact match are using StrideBV stages and the strides that perform range comparisons are using range-search stages. The search process is followed by the priority encoder network which extracts the highest-priority match.

Note that the order in which the headers are inspected does not affect the final search result. Therefore, the StrideBV stages and range-search stages can be permutated in any order the user desires. Fig. 4 takes a serial search approach, consuming consecutive strides of the header as the packet progresses through the pipeline stages. This causes the packet classification latency to increase linearly proportional to the header length. With longer header lengths, this can be an undesirable feature especially for applications that demand low latency operation, such as multimedia, gaming and data center environments.

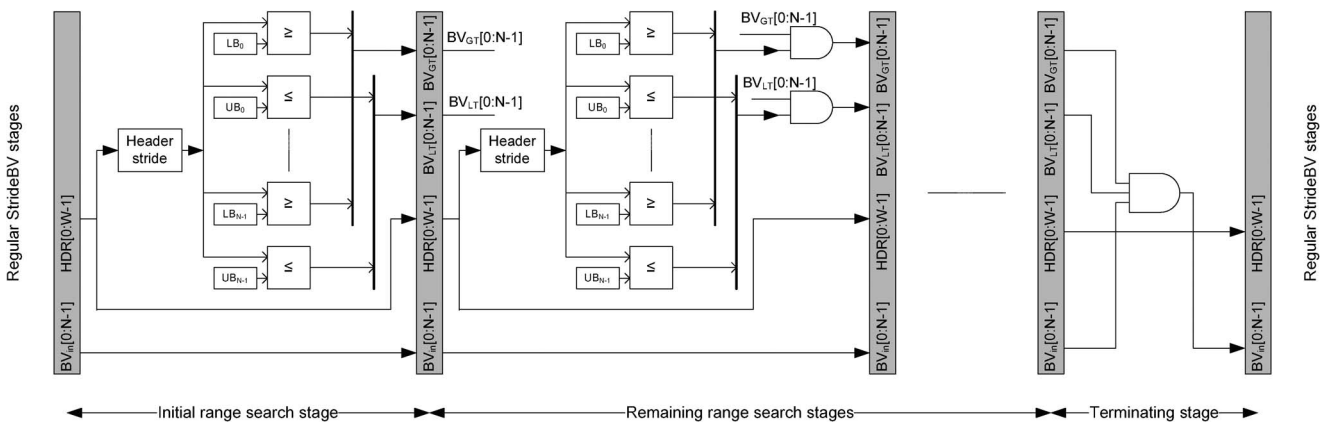


Fig. 3. Memory efficient range search implementation on FPGA via explicit range storage.

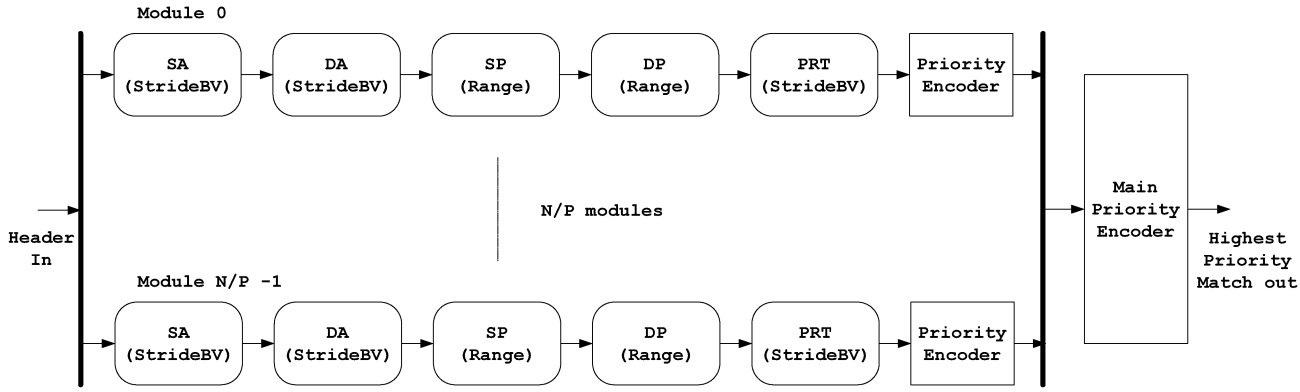


Fig. 4. Serial architecture for latency tolerant applications.

In order to facilitate such requirements, we propose a variant of the serial stride search architecture. Fig. 5 illustrates the low latency variant of the proposed architecture. As mentioned previously, since the order of search does not affect the final result, we perform the stride search in parallel. For example, a 5-tuple header is 104 bits in length. For stride size $k = 4$, if we perform serial stride search, the packet classification latency will be 26 clock cycles, excluding the priority encoder delay. However, if we perform the search in a parallel fashion, then the latency can be brought down significantly. One possible arrangement is to perform the following three searches in parallel: {SA}, {DA}, {SP, DP, PRT}. In order to aggregate the results from the parallel search operations, an additional stage needs to be added. In such a scenario, if we use a stride size of $k = 4$ for this arrangement, the delay will only be 11 clock cycles (with added delay stages for SA and DA lookups), which causes the packet classification latency to reduce $0.42\times$ compared with the serial search approach. This makes our solution suitable for networks that demand low-latency operation.

In both serial and parallel orientations, each module is implemented as a separate pipeline. For a classifier of size N (i.e. N rules), and a partition size of P, there will be N/P modules, hence N/P modular pipelines. Once the module-local highest priority match is extracted, this information is fed into the main priority encoder which determines the overall highest priority match. And the packet is treated based on the action specified in this rule.

5 Performance Evaluation

We evaluate the performance of both serial and parallel versions of StrideBV on a state-of-the-art Xilinx Virtex 7 2000T FPGA using post place-and-route results. We use memory consumption, throughput, power consumption and overall packet latency as performance metrics in this evaluation. Further, we highlight the benefits of using FPGA floor-planning techniques to map the proposed architecture on to the FPGA fabric. Finally, we compare our solution against several existing packet classification schemes to show the improvements rendered by the proposed scheme.

5.1 Memory Requirement

The memory requirement of StrideBV for a given packet classification ruleset is strictly proportional to the number of rules in the classifier. Since range-to-prefix conversion is not required in our approach, each rule storage takes only $O(1)$ space. Hence, for a classifier with N rules, the memory requirement is $\Theta(N)$. This unique property of our solution makes it a robust alternative for all ruleset-feature dependent schemes available in the literature.

The proposed architecture is flexible in that the user can decide the stride size and orientation (serial or parallel),

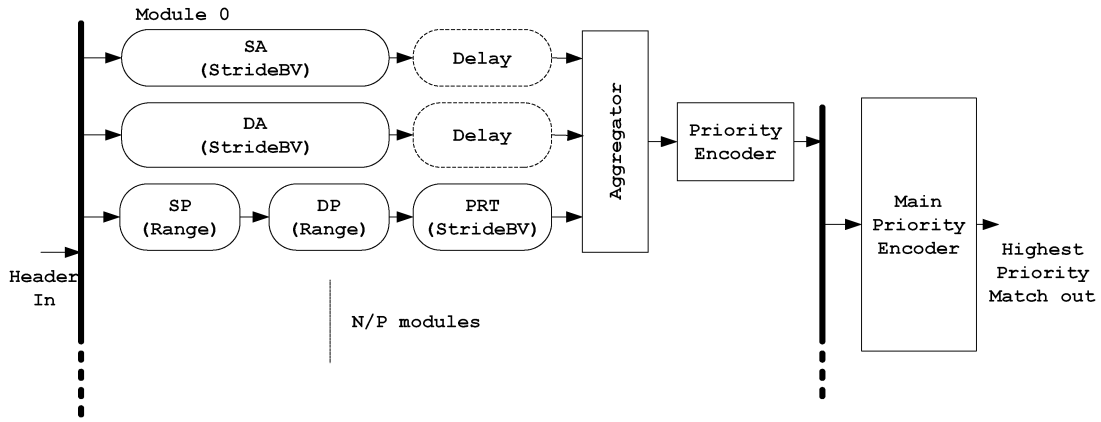


Fig. 5. Parallel architecture for low latency applications.

based on the latency and memory constraints of a given application. However, note that the orientation of the architecture has no effect on the memory requirement as no additional bit-vector storage is required to transform the architecture from serial to parallel or vice versa. Similarly, the modularization of the architecture has no effect on the memory requirement since the partitioning of the architecture does not introduce any additional rules. Further, the architecture proposed here, is able to exploit distributed RAM as well as BRAM available on the device. Hence, all on-chip memory resources can be made use of.

In Fig. 6 we illustrate the memory requirement of the proposed architecture for various stride sizes. The pipeline stage memory size increases by a factor of 2^k and the pipeline length decreases by a factor of $1/k$, for a stride of size k bits. Hence the overall effect of stride size on the memory consumption of the architecture is $2^k/k$. Consequently, this causes the latency of a packet to decrease by a factor of $1/k$ at the expense of a $2^k/k$ memory increase. Note that this latency is excluding the latency introduced by the priority encoder network. The overall latency will be discussed in Section 5.2.

Fig. 6 assumes a 100 percent utilization of on-chip memory of the considered FPGA. However, in reality, on-chip memory is available at a coarse granularity. For example, the minimum distributed RAM size is 64 bits and the minimum block RAM size is 18 Kbit. These memory blocks have predetermined height and width constraints that prevents one from using them at finer granularities. For example, the minimum depth configuration for block RAM on Virtex 7 devices is 512 (height) words of 72 bits (width). The effect of these constraints are discussed here.

Building stage memory of higher bit-widths (e.g. to store 1024 bit-vectors) is achieved by cascading multiple memory blocks together. However, when considering the height of stage memory, for a stride of size k bits, the height required is 2^k. The minimum height supported by block RAM is 512. In order to fully utilize such a block RAM, stride size has to be increased to $k = 9$. The side effect of using a higher stride size is that the memory required to store a rule increases by a factor of $2^k/k$ as shown earlier. Hence, for example, compared with $k = 2$, the memory required to store a rule increases by a factor of 28×. We show this tradeoff in Fig. 7. The multiplication factor is simply the ratio between the amount of memory required to store a rule using stride of k bits and the bit

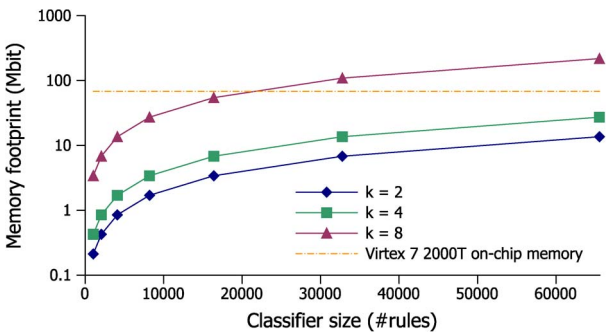


Fig. 6. Memory requirement of the proposed solution for stride sizes $k = \{2, 4, 8\}$ and increasing classifier size. Note the log scale in y-axis.

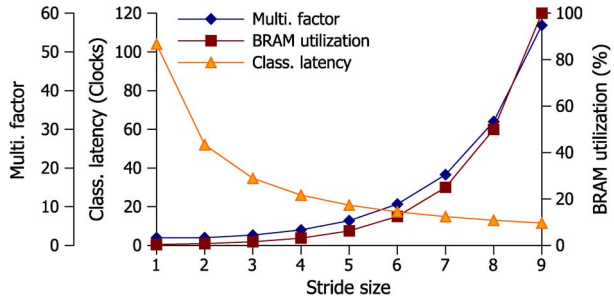


Fig. 7. Variation of 1) multiplication factor, 2) classification latency and 3) BRAM utilization with increasing stride size.

width of a single rule. This metric indicates how much space is required to store a single rule compared with its original size.

It can be seen that the amount of space required to store a single rule increases in an exponential fashion with increasing stride size. A Virtex 7 2000T device has 21 Mb of distributed RAM and 42 Mbit of BRAM, which will have 50 percent (a minimum height of 32) and 3 percent (a minimum height of 512) memory utilizations, respectively, that allows a maximum classifier size of 28K rules for a stride size of $k = 4$. The utilization of memory blocks is dictated by the minimum heights allowed by distributed and block RAMs available on FPGA. For a stride size of $k = 4$, only 16 rows is required and the excessive rows will not be utilized. Note that this constraint is imposed by the underlying architecture of FPGA and such limitations can be eliminated in custom built platforms such as ASICs, where the granularity of memory can be made finer than what is offered on modern FPGA platforms. Hence, the same architecture can be mapped onto ASIC, which can support higher classifier sizes and render higher throughput. However, the ASIC implementation of the proposed architecture and its performance evaluation is beyond the scope of this paper.

5.2 Throughput and Packet Latency

5.2.1 Throughput

In Section 4 we introduced the serial and parallel variants of the proposed architecture. We evaluate the throughput and latency of both architectures in this section in detail. The performance of networking hardware is measured in either the number of packets forwarded per second (PPS) or the number of data bits forwarded per second (bps). The conversion between the two metrics is straightforward and can be stated as: $bps = packet\ size\ in\ bits \times\ PPS$. In order to report the worst case throughput, we use minimum packet size which is 40 Bytes for IPv4 packets.

Table 1 shows the performance of the architecture. For these experiments, we fixed the stride size to $k = 4$. The reason for this choice is that $k = 3$ and $k = 4$ yields the best tradeoff of the metrics we consider as we will show later. In addition, considering the bit widths of the individual header fields, strides of size 4 can be perfectly aligned with header boundaries, which preserves the header field separation. This enables us to separate the range search fields from the prefix/exact search fields, which simplifies the architecture.

TABLE 1
Performance of the Modular StrideBV Architecture for Stride Size $k = 4$ and Priority Encoder Degree of $d = 4$

Partition size	Serial			Parallel		
	Clock Rate (MHz)	Throughput (Gbps)	Latency (Clocks/ns)	Clock Rate (MHz)	Throughput (Gbps)	Latency (Clocks/ns)
16	526	337	28/53	451	289	16
64	438	280	29/66	378	242	17
256	296	190	30/101	257	165	18
1024	211	135	31/146	174	111	19

In order to demonstrate the usefulness of the modular architecture, we report the performance for various partition sizes. Each module is implemented as a pipeline as discussed in Section 4. The advantage of modularization is that instead of storing and processing massively wide bit-vectors, the bit-vector length inside a module is restricted to a size such that the performance is not significantly deteriorated due to bit-vector length. With this approach, in order to support higher scalability, one simply can add more modules and the partition size can also be chosen based the latency and throughput demands of the network. The superior flexibility offered in the proposed architecture is appealing in modern networking environments where the demands vary from network to network.

We exploit the dual-ported feature of FPGA on-chip memory to increase the throughput of the architecture. Each modular pipeline is able to accept two packet headers per cycle. The reported clock frequency and throughput are for the dual-ported implementation of the on-chip memory. For these experiments, the stage memory was built using the distributed RAM available on FPGA. It is possible to implement the modular pipelines with BRAM and even a hybrid of distributed RAM and BRAM for higher scalability.

5.2.2 Chip Floor-Planning

The proposed architecture has a regular structure and the same structure is maintained in all pipeline stages. Therefore, it is relatively easy to map the pipelined architecture onto the FPGA fabric in such a way that the routing delays are minimized. While there are numerous ways of laying out the pipeline on the fabric, in our work we adopted the following methodology. The pipeline stages were placed in a contiguous manner, which required a snake-like layout for the serial case and a straight-line layout for the parallel case. This enabled us to constrain the occupancy of a single modular pipeline to a specified area of the chip (localized routing), which in-turn reduces the adverse effect on the performance of the other modular pipelines. Due to this reason, even if more modular pipelines are added the reported performance of a modular pipeline will not be significantly deteriorated.

5.2.3 Packet Latency

The cumulative packet latency comprise the packet classification latency and the latency introduced by the priority encoder. As mentioned previously, the latency introduced by the packet classifier can be calculated as W/k, where W is the header bit length and k is the stride size in bits. Once the packet classification process is complete, the highest priority match of the sub-BV produced by each modular pipeline is extracted. Then the main priority encoder resolves the highest priority match from all sub-BVs to find the matching rule for the input packet header.

The latency of the priority encoder can also be adjusted by appropriately choosing the degree. For example, a priority encoder with degree d and number of elements n will introduce a delay of $\log_d n$. With higher values of d, the latency can be minimized. However, there is a tradeoff. When d is increased, the number of serial comparisons that needs to be made at a stage also increases proportionally. Hence, the delay also increases. This delay is tolerable as long as the delay of a priority encoder pipeline stage is lower than the maximum stage delay of the packet classification pipeline. Increasing priority encoder delay beyond this limit will cause the throughput to degrade even though the delay is minimized. The latency values reported in Table 1 are for $d = 4$, and comprise packet classification latency and the total priority encoder latency for a modular pipeline.

The delay introduced by the main priority encoder depends on the size of the classifier under consideration. Considering the largest classifier that can be hosted on the given FPGA, a 28K rule classifier, and a partition size of 1024, the main priority encoder will be handling a 28 entry bit-vector. Here also, the delay can be adjusted by choosing the degree of the priority encoder accordingly, similar to the case of the priority encoder at the end of each modular pipeline. If we set $d = 4$, then the total delay introduced by the priority encoder network will be $\lceil \log_4 1024 \rceil + \lceil \log_4 28 \rceil$, which amounts to 8 clock cycles.

5.3 Power Efficiency

While the throughput of the networking has dramatically increased over the past few decades, from 100 Mbps connections to 100 Gbps connections, the power consumption

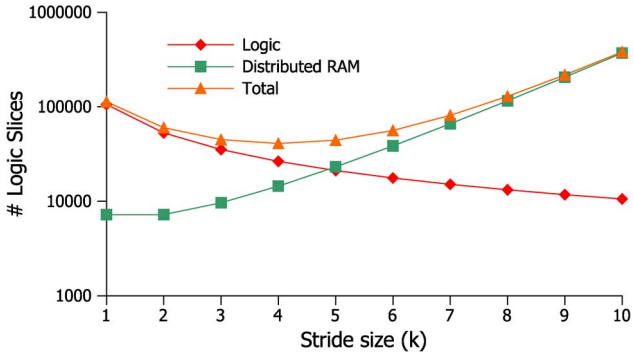


Fig. 8. Tradeoff between memory and logic slice utilization with increasing stride size.

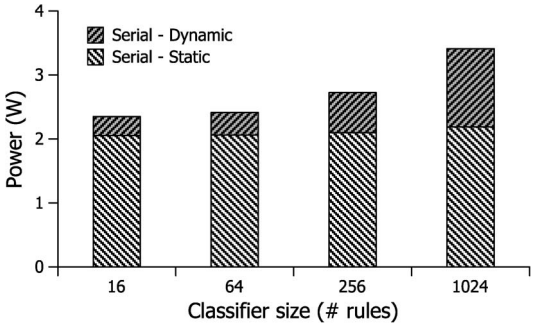


Fig. 9. Power consumption of serial architecture.

of networking hardware has also increased significantly. Power efficiency has gained much interest in networking community due to this reason. Operating under a restricted power budget is therefore imperative, which renders TCAMs a less attractive solution despite their simplicity. TCAMs have poor power efficiency due to their massively parallel exhaustive search conducted on a per packet header basis. Current TCAM devices are able to operate at 360 MHz speeds, support variable word sizes and are available in different capacities. A TCAM's power consumption ranges between 15-20 Watts/Mbit [8], which is relatively high compared with the power consumption of an architecture on FPGA that performs the same operation. However, despite the high power consumption, the packet latency of TCAM is minimal with only a single clock cycle latency.

Since the dynamic power is proportional to the amount of resources consumed, we evaluated the number of logic slices consumed by the logic and memory components of the architecture. The baseline was set for the architecture with stride $k = 1$ and we projected the results obtained from this experiment, to larger stride sizes by using $S_{L,k} = S_{L,1}/k$ and $S_{M,k} = S_{M,1} * 2^k/k$, where $S_{L,x}$ and $S_{M,x}$ denote the number of slices used as logic and memory portions of the architecture for stride size $k = x$, respectively. The projected results are depicted in Fig. 8. Even though this does not directly translate to power consumption of the different components due to the effect of stride size on frequency, if we assume a fixed frequency for all stride sizes, Fig. 8 can be used as a direct translation from resource to power consumption.

The conclusion we can draw from Fig. 8 is that increasing the stride size will have its tradeoffs both from memory and power consumption standpoints. In our experiments, the stride size that yielded best tradeoff between memory and power consumption was $k = 4$. Therefore, for all the experiments conducted on the proposed architecture, this stride size was used. Fig. 9 illustrates the results of the experiments conducted for $k = 4$ serial variant using different partition sizes. The results of the parallel architecture is not presented due to space limitations, but has similar behavior. It can be seen that the static power is almost constant while dynamic power increases with the increasing partition size. This is expected due to increased resource usage (i.e. more units being ''clocked''). Due to this reason, for smaller classifier sizes (ex: < 512 rules), the power consumption, of the proposed solution is worse compared with the power consumption for larger classifier sizes.

5.4 Comparison With Existing Literature

We compare several existing approaches with StrideBV and the details are presented in Table 2. For these experiments, we considered a classifier size of 512 rules. It must be noted that for the ruleset-feature dependent solutions, the performance can significantly vary depending on the ruleset characteristics, which consequently affects the performance whereas with the proposed solution, the reported performance is guaranteed for any 512 rule classifier. Here, we are assuming that each rule is translated into 2.32 rules, which is the average case reported in [12]. For both proposed schemes, stride of size $k = 4$ is assumed.

In order to compensate for the technology gap in cases where older devices were used (e.g. Xilinx Virtex II Pro), we assumed that all architectures operate at 300 MHz. We also report the worst case memory consumption reported for each approach to illustrate the caveats associated with ruleset-dependent solutions. In [11], the solution requires 5 sequential memory accesses to complete the lookup process, which yields lower throughput. While their solution is memory efficient for specific classifiers, it has been shown that the memory requirement can go up as high as 90 Bytes/rule. In BV-TCAM [10], the authors use a TCAM generate on FPGA. As shown in [9], the clock frequency that can be achieved with TCAMs built on FPGA is around 230 MHz for a size of 32×104. Hence the clock frequency of the complete architecture will be governed by the clock frequency of the TCAM despite our 300 MHz assumption.

For all approaches, the power efficiency reported here is the amount of dynamic power increment observed for each additional rule introduced. For this comparison we considered dynamic power only for all, as the static power remains almost the same despite the ruleset size. The values reported here are the gradient values of power

TABLE 2
Performance Comparison With Existing Literature. (*Has No Support for Arbitrary Ranges)

Approach	Throughput (Gpbs)	Memory (Bytes/rule)	Latency (Clocks)	Power Eff. (mW/rule)	Ruleset Dependence	Range-to prefix
Proposed - Serial ($k = 4$)	135	52	31	0.624	NO	NO
Proposed - Parallel ($k = 4$)	111	52	19	0.920	NO	NO
DCFL [11]	19	90	5	N/A	HIGH	NO
BV-TCAM [10]	75	154	11	0.846	HIGH	NO
Emulated TCAM [15]	64	24	1	N/A	LOW	YES*
TCAM [8]	115	30	1	4.901	LOW	YES

versus ruleset size lines for each approach (not specific for a ruleset size of 512 rules).

We also illustrate the ruleset dependence of each scheme to highlight the unique features of the proposed scheme. We have used *HIGH* to denote the algorithmic schemes that are heavily reliant on ruleset features and *LOW* to denote the schemes that require range-to-prefix conversion only. However, note that the effect of range-to-prefix conversion can be significant depending on the ranges present in the classifier. Unlike our scheme, [15] has no support for arbitrary ranges. Inclusion of arbitrary ranges could dramatically increase the memory required per rule in [15].

6 Conclusion

We presented a modular architecture for high-speed and large-scale packet classification on Field Programmable Gate Array (FPGA). We employed the Bit-Vector (BV) approach and overcame the inherent performance bottlenecks of the same via priority-based ruleset partitioning. Essentially, we reduced the per pipeline memory bandwidth requirement to improve the performance. Further, we incorporated *range search* in our architecture by performing an explicit range search without affecting the architecture, which is a significant improvement compared with range-to-prefix conversion which yields poor memory efficiency.

Acknowledgment

This work is supported by the United States National Science Foundation under grant No. CCF-1116781. Equipment grant from Xilinx Inc. is gratefully acknowledged.

References

[1] W. Eatherton, G. Varghese, and Z. Dittia, ''Tree Bitmap: Hardware/Software IP Lookups with Incremental Updates,'' *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 2, pp. 97-122, Apr. 2004.

[2] M. Faezipour and M. Nourani, ''Wire-Speed TCAM-Based Architectures for Multimatch Packet Classification,'' *IEEE Trans. Comput.*, vol. 58, no. 1, pp. 5-17, Jan. 2009.

[3] T. Ganegedara and V. Prasanna, ''StrideBV: 400G+ Single Chip Packet Classification,'' in *Proc. IEEE Conf. HPSR*, 2012, pp. 1-6.

[4] P. Gupta and N. McKeown, ''Classifying Packets with Hierarchical Intelligent Cuttings,'' *IEEE Micro*, vol. 20, no. 1, pp. 34-41, Jan./Feb. 2000.

[5] G. Jedhe, A. Ramamoorthy, and K. Varghese, ''A Scalable High Throughput Firewall in FPGA,'' in *Proc. 16th Int'l Symp. FCCM*, Apr. 2008, pp. 43-52.

[6] W. Jiang and V.K. Prasanna, ''Field-Split Parallel Architecture for High Performance Multi-Match Packet Classification Using FPGAs,'' in *Proc. 21st Annu. SPAA*, 2009, pp. 188-196.

[7] T.V. Lakshman and D. Stiliadis, ''High-Speed Policy-Based Packet Forwarding Using Efficient Multi-Dimensional Range Matching,'' *SIGCOMM Comput. Commun. Rev.*, vol. 28, no. 4, pp. 203-214, Oct. 1998.

[8] C.R. Meiners, A.X. Liu, and E. Torng, *Hardware Based Packet Classification for High Speed Internet Routers*. Berlin, Germany: Springer-Verlag, 2010.

[9] A. Sanny, T. Ganegedara, and V. Prasanna, ''A Comparison of Ruleset Feature Independent Packet Classification Engines on FPGA,'' in *Proc. IEEE IPDPS RAW*, 2013, pp. 124-133.

[10] H. Song and J.W. Lockwood, ''Efficient Packet Classification for Network Intrusion Detection Using FPGA,'' in *Proc. ACM/SIGDA 13th Int'l Symp. FPGA*, 2005, pp. 238-245.

[11] D. Taylor and J. Turner, ''Scalable Packet Classification Using Distributed Crossproducing of Field Labels,'' in *Proc. 24th Annu. Joint IEEE INFOCOM*, Mar. 2005, vol. 1, pp. 269-280.

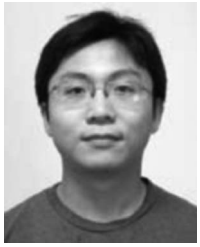
[12] D.E. Taylor, ''Survey and Taxonomy of Packet Classification Techniques,'' *ACM Comput. Surv.*, vol. 37, no. 3, pp. 238-275, Sept. 2005.

[13] XilinxVirtex 7. [Online]. Available: http://www.xilinx.com/products/silicon-devices/fpga/virtex-7/index.htm

[14] XilinxXilinx xcell Journal. [Online]. Available: http://www.xilinx.com/publications/xcellonline/

[15] C.A. Zerbini and J.M. Finochietto, ''Performance Evaluation of Packet Classification on FPGA-Based TCAM Emulation Architectures,'' in *Proc. IEEE GLOBECOM*, 2012, pp. 2766-2771.

Thilan Ganegedara received the BS degree in electrical engineering from University of Peradeniya, Sri Lanka, in 2008 and PhD degree in electrical engineering from University of Southern California, Los Angeles, in 2013. He is currently a Software Engineer at Cisco Systems, USA. His research interest include high-performance networking, reconfigurable computing and network virtualization.



Weirong Jiang received the BS degree in automation and the MS degree in control science and engineering, both from Tsinghua University, Beijing, China, in 2004 and 2006, respectively, and the PhD degree in computer engineering from the University of Southern California, Los Angeles, in 2010. He is currently a Staff Research Engineer at Xilinx Labs, USA. His research interests include high-performance low-power networking, reconfigurable computing, network virtualization, machine learning, and wireless communication. He is a member of the ACM.

Viktor K. Prasanna is Charles Lee Powell chair in engineering in the Ming Hsieh Department of Electrical Engineering and professor of computer science at the University of Southern California (USC). He is the executive director of the USC-Infosys Center for Advanced Software Technologies and is an associate director of the USC-Chevron Center of Excellence for Research and Academic Training on Interactive Smart Oilfield Technologies. He serves as the director of the Center for Energy Informatics at USC. He served as the editor-in-chief of the IEEE TRANSACTIONS ON COMPUTERS from 2003 to 2006. Currently, he is the editor-in-chief of the Journal of Parallel and Distributed Computing. His research interests include high-performance computing, parallel and distributed systems, reconfigurable computing, and embedded systems. He is a recipient of the 2009 Outstanding Engineering Alumnus Award from PSU. He was the founding chair of the IEEE Computer Society Technical Committee on Parallel Processing. He is the Steering co-chair of the IEEE International Parallel and Distributed Processing Symposium and is the Steering chair of the IEEE International Conference on High-Performance Computing. He is a Fellow of the IEEE, the ACM, and the American Association for Advancement of Science.