A Comparison of Ruleset Feature Independent Packet Classification Engines on FPGA

Andrea Sanny, Thilan Ganegedara, Viktor K. Prasanna
Ming Hsieh Dept. of Electrical Engineering
University of Southern California
Los Angeles, CA 90089
Email: {sanny, ganegeda, prasanna}@usc.edu

***Abstract*—Packet classification is used in network firewalls to identify and filter threats or unauthorized network access at the application level. This is realized by comparing incoming packet headers against a predefined ruleset. Many solutions to packet classification are available, but most of these solutions exploit some features of the ruleset in order to minimize the memory footprint of ruleset storage. However, when the expected ruleset features are not present, feature-reliant solutions may yield poor memory efficiency. In this paper, we focus on two ruleset independent packet classification schemes, Ternary Content Addressable Memory (TCAM), a brute force search method, and StrideBV, a bit-vector-based algorithmic solution, to determine which solution is more suited for high performance packet classification. Using ruleset sizes ranging from 32 to 2048 (targeted for firewall rulesets), we implement both schemes on a Field-Programmable Gate Array (FPGA) to evaluate their performance. We measure the performance using memory efficiency, resource consumption, throughput and power efficiency metrics for both solutions. The post place-and-route results on a state-of-the-art FPGA reveal that StrideBV has $4.5\times$ and $3.5\times$ higher power efficiency in comparison with TCAM, along with $6\times$ and $4\times$ higher throughput when using distributed RAM and block RAM as memory respectively. TCAM has better memory efficiency, though its improvement over StrideBV varies.**

***Index Terms*—Packet classification, FPGA, ASIC, Internet, Firewalls, Network Security, TCAM**

I. Introduction

With growing security threats, network security has become a major concern in any type of a network. Network firewalls act as the preliminary checkpoint for such threats which provides security at the application (Layer-4) level [10], [7]. Packet classification is the scheme by which such firewalls are realized. Packet classification inspects a predetermined number of header fields from the packet and checks against a set of rules to determine the forwarding decision. If a threat or malicious traffic is encountered, they are dropped and will not be further processed. Packet classification is also used to distinguish between flows of traffic for packet reassembly when performing Deep Packet Inspection (DPI) type operations [4], [15], [16].

The challenge with implementing firewalls is supporting wire-speed classification [3], [5]. With the high throughput and Quality of Service (QoS) demands in networking, high-speed networking has become critical. Compared with best-effort IP lookup, packet classification is more challenging due to multiple header field inspection [13], [19]. Due to various search requirements in each header field, the lookup operation can be complex. The amount of storage required to store the increasing number of rules is also a constraint especially on hardware platforms such as Field Programmable Gate Arrays (FPGAs) due to the limited amount of on-chip memory. Hence, supporting scalability and high throughput is challenging.

To reduce memory consumption, most, if not all, solutions exploit some properties (or features) of the ruleset, however these solutions might not work as expected for all rulesets [18]. Different rulesets have different features, and may not contain the exploited features used by a particular solution. Instead, this ruleset dependency may yield poor memory efficiency for solutions that are heavily reliant on the features. Ternary Content Addressable Memory (TCAM) is a solution technique, a brute force search, which is not dependent on ruleset features [1], [2]. TCAM performs a parallel search against all stored rules to find the matching rule within its memory in a single clock cycle. They are commonly used for packet classification and IP lookup because of its high-speed and simplicity. Although, they are known to be expensive, as well as power hungry compared with alternative solutions such as Static Random Access Memory (SRAM)-based architectures. Unlike SRAM that has only one row of memory array active at a time, during a TCAM lookup, all entries are active, causing relatively high power consumption. Though it is used in practice, TCAM has many drawbacks, and there are many algorithmic solutions available as alternatives to TCAM. One such solution that also does not rely on the ruleset features is StrideBV, a bit-vector-based approach [5]. StrideBV splits the packet header fields and through the use of parallel pipelines, which can achieve high throughput even for large ruleset sizes. These two schemes can avoid the pitfalls of feature-reliance, and be unaffected by the absence of expected features, making them strong possibilities for a very reliable packet classification solution despite the lack of knowledge of the ruleset features, which could cripple solutions that rely heavily on feature-exploitation.

We demonstrate in this paper a detailed comparison between the ruleset feature independent packet classification engines.

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

Specifically, we compare the exhaustive search solution (i.e. TCAM) versus the algorithmic solution (i.e. StrideBV) on FPGA, in order to determine which approach is more suited for high performance packet classification. Although TCAMs are still used in networking equipment due to their simplicity, their high power and resource consumption make them a less desirable solution. An algorithmic solution such as StrideBV is a potential alternative to TCAM, which can deliver high throughput while having moderate resource consumption. We employ the same implementation platform and evaluate the performance of these two solutions head-to-head.

Using post place-and-route results on a state-of-the-art Xilinx Virtex 7 XC7VX1140T FPGA [22], we compare TCAM and StrideBV for ruleset sizes starting at 32 rules to 2048 rules. To ensure a strong and a fair comparison, looking at both ends of the spectrum in terms of possible firewall ruleset sizes. The contributions of this work are summarized below:

- A quantitative and an exhaustive comparison of TCAMs and StrideBV approaches for packet classification on state-of-the-art FPGA
- A thorough evaluation of TCAM generated on FPGA for various ruleset sizes
- Performance comparison of the two approaches with respect to throughput, power, resource consumption and scalability of each approach
- Use of FPGA chip floor-planning for StrideBV for further improved performance

II. Background and Related Work

A. Background

Packet classification is a widely deployed scheme as a mechanism to filter malicious traffic and to enforce security in a network by inspecting network traffic at application level. It is also used as a mechanism to differentiate network flows from each other where per flow operation is required. Compared with the simplistic IP lookup, packet classification is a more complex problem due to the inspection of multiple fields. A match indicates that the incoming packet's header matches all fields of the rule. Typically, $5-8$ fields are used in packet classification which are collectively called as a *tuple*. The most prominent scheme is 5 field packet classification in which a tuple has the following header fields: $\langle Source\ IP,\ Destination\ IP,\ Source\ Port,\ Destination\ Port,\ Protocol\rangle$. Other multi-field packet classification schemes such as OpenFlow also exist which consider $12+$ number of fields, but are used for different purposes such as Software Defined Networking (SDN).

To determine the forwarding information for an incoming packet, the packet header needs to be matched against a predefined set of rules which is referred to as *ruleset* or *classifier*. The rules are prioritized and the priority is determined by the order in which the rules are stored in the classifier. The topmost rule has highest priority while the last rule has the lowest priority. Priority is necessary to determine the forwarding information for packets that match with more than one rule. In such situations, the highest priority rule out of the matched rules is used as the rule to be applied on the packet. However, it should be noted that some applications such as Intrusion Detection Systems (IDSs) require all matching rules to be reported.

Table I shows a sample packet classification ruleset. As it can be seen, each field has its own matching requirement. SIP and DIP fields require prefix matching, SP and DP fields require range matching and the port field requires exact match. This hinders one from using a single lookup scheme for all the fields. Also, the SP and DP fields can be arbitrary ranges in the sense that they might not be represented as a single prefix. For most implementations (e.g. TCAM), a prefix representation (binary or ternary) is desirable due to the data structure requirements. While an arbitrary range can be converted to prefixes, the number of rules generated can go up to $2(w-1)$ in the worst case, where w is the bit width of the header field. Since two port fields exist, a single range rule may potentially expand to $4(w-1)^2$ rules, which is highly undesirable from the memory requirement standpoint.

In the following section, we discuss some prominent implementation methods of packet classification.

B. Related Work

A myriad of solutions can be found in the literature that perform packet classification [8], [18]. These techniques vary from brute force solutions such as TCAM (and variants of it) to more efficient algorithmic solutions. Even though algorithmic solutions are plentiful in literature, TCAMs are still being used in networking equipment mainly due to their simplicity. However, TCAM is known to be power hungry and expensive, hence is not considered as an efficient solution for network related tasks. Efforts have been put on reducing the power consumption of TCAM based solutions via partitioning so as to disable the TCAMs that are not relevant for a given search operation. Although this helps improving power efficiency, the cost and power requirements are still not justifiable compared with algorithmic solutions. The TCAMs being referred to here are mainly Complementary Metal-Oxide Semiconductor (CMOS) based. However, it can be seen in literature that TCAMs are generated on FPGA fabric using the logic resources available, to perform lookup operations. As we show in Section V, such implementations are highly resource hungry and may potentially use entire chip resources to build a small size TCAM.

Algorithmic solutions on the other hand, are more efficient and offer high flexibility. The existing algorithmic solutions can be categorized into two main groups, namely, 1) Decision tree-based and 2) Decomposition-based. In decision tree based approaches, the ruleset is treated as a d-dimensional space (for d number of fields), in which an axis represents one field. [14], [7], [9] The decision tree is a mechanism to partition the d-dimensional space into smaller size partitions such that, the number of rules inside is small enough to perform a linear search. Many variants of the decision tree based approach exist in which, the tree height, the number of cuts per node,

TABLE I: Example packet classification ruleset

Source IP (SIP)	Destination IP (DIP)	Source Port (SP)	Destination Port (DP)	Protocol (PRT)	Priority	Action
128.125.2.0/24	128.143.200.0/24	$0-1023$	$*$	UDP	0	PORT 4
128.125.0.0/17	$*$	$20-21$	$20-21$	TCP	1	PORT 3
128.143.63.0/25	128.96.182.64/27	80	$0-1023$	$*$	2	DROP
128.96.12.45/32	128.69.43.0/28	≥ 1023	$*$	$*$	3	PORT 0
128.143.0.0/16	128.125.44.0/24	$*$	$*$	$ICMP$	4	PORT 3

the number of dimensions per node, the number of rules per small partition, etc., are selected based on some optimization criterion. The advantage of this method is that, since it is a tree traversal, the tree structure can be mapped to a linear pipelined architecture to achieve high throughput.

The second group of solutions, decomposition-based, takes each field separately, perform search in each field and aggregates the partial results to arrive at the highest priority match [16], [11]. For example, the field searches can be realized as tree/trie traversal. The output of each such field is the identities of the rules that matched with the considered header field. Once all the partial matches are available, the final step is to identify the highest priority match, which is done by finding the set intersection of the partial matches. The main challenge in decomposition based techniques is aggregating the partial results. Hashing, bit-vector and set intersection techniques have been employed to perform this step and each technique has its own limitations. For example, hashing may potentially result in a large hash table, which might not fit on the available on-chip memory of a device and set intersection may not scale to larger problem sizes when each header matches a larger number of rules than the set intersection network can handle.

Both above mentioned algorithmic solutions are ruleset feature reliant. Since the main limitation of high-performance hardware platforms is memory, the main goal is to reduce the memory footprint of the ruleset storage. This is not desirable due to two reasons:

- The assumed features may not be present in a given ruleset. In such cases the expected memory savings will not be achieved and may potentially increase the memory requirement instead of decreasing.
- Most solutions use some form of tree to perform search. Typically, with increasing depth, the number of nodes in a given level increases exponentially. When mapping such solutions to pipelined hardware engines, the performance will be dictated by the slowest stage and the slowest stage is generally the one with the highest memory usage. Due to the non-uniform memory distribution, the performance is adversely affected.

III. Ruleset Feature Independent Packet Classification Engines

As mentioned in Section II, the only two approaches that can be found in literature that performs ruleset feature independent packet classification are StrideBV and TCAM. In this section, we explain each approach in a concise manner.

A. StrideBV: Algorithmic Solution

1) Bit-Vector based Packet Classification: The first bit-vector based approach for packet classification appears in [17]. Further developments of the same architecture are available in [16], [11]. The usage of bit-vector is as an indexing mechanism to identify the rules that matched with a header. The operation of the bit-vector based packet classifier is explained here. Each field lookup is performed using a chosen search technique and the output of each lookup operation is a bit-vector. The representation is as follows: For a classifier with N number of rules, a bit-vector of N bits is used. If an incoming header matched with the rule r_i where i is the index (or the priority) of the rule, the i^{th} bit of the bit-vector is set to `1`. For the rule indices that did not match, the bit values are set to `0`. Therefore, at the end of each field search, a bit-vector in which bit values corresponding *all* the matched rule indices set to `1` is output.

Once this step is complete, the bit-vector's output from each field search is aggregated. This is done by performing a bit-wise logical *AND* of the partial results (i.e. bit-vectors). This generates a bit-vector of length N, in which a bit at index i is set to `1` if and only if all the partial bit-vectors had `1` at i^{th} index. Essentially, the bit-vector indicates the rules that matched with an incoming header, in all fields. In order to identify the highest priority match, the index corresponding to the lowest bit position set to `1` is extracted. At this point, the packet classification is complete for the incoming header.

2) Field Split Bit-Vector (FSBV) Algorithm: FSBV was first introduced in [11] as a solution to minimize the memory footprint of packet classification engines. The key idea in FSBV is to consider each bit of a header as a field and perform search as explained in Section III-A1. The bit-vector generation and packet classification operations are depicted in Figure 1.

Each bit of a rule is viewed as a sub-field and two bit-vectors are generated for each sub-field. One corresponding to header value `1` and the other for `0`. For example, for sub-field i, two bit vectors are stored and based on the incoming packet header's i^{th} bit value, one of the two bit-vector is loaded. Similarly, bit-vectors are loaded for other bit positions and then they are logically *AND*ed together to find the matching rules in all fields. In [11], FSBV was applied to only the SP and DP fields since FSBV results in memory efficiency only when the number of unique rules per field is less than twice the bit length of a field. Note that here a field means one of the five fields in the 5-tuple header. For the fields that did not satisfy the aforementioned condition, TCAMs generated

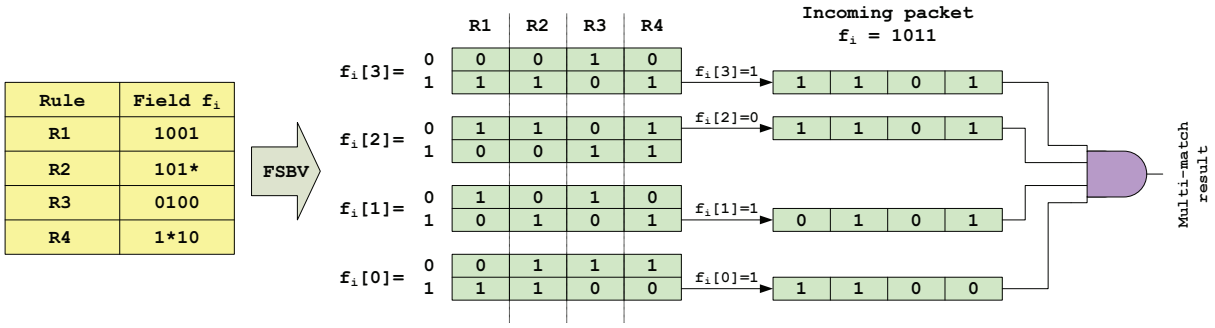

Fig. 1: Example of FSBV bit-vector generation and packet classification processes [5]

on FPGA were used.

3) StrideBV: In StrideBV the main motive is improving the throughput of packet classification engines. It employs the FSBV algorithm for the entire rule rather than restricting it to SP and DP fields. In addition, instead of considering each bit of the header as a sub-field, a stride of k bits is considered a sub-field. It should be noted that by considering a k bit stride, the number of bit-vectors per each sub-field increases to 2^k, as the bit-vectors corresponding to each k bit combination needs to be stored. However, the number of such sub-fields is reduced by a factor of $1/k$ compared with the uni-bit stride (or FSBV). Hence, the total memory requirement increases only by a factor of $2^k/k$.

The main advantage of StrideBV is that, it can be mapped onto FPGA/ASIC type hardware architectures to achieve high performance. Most solutions available in the literature require either tree traversal or hash lookup. In the case of tree search, the tree size grows exponentially with increasing depth and the memory consumption of the last level of the tree can be considerably high compared with the remaining stages. This, especially on FPGA platforms, translates to higher access time as larger memory blocks are generated by cascading multiple smaller memory blocks and routing complexity increases when the number of blocks used to generated a larger memory increases. This has a negative impact on the clock rate since in pipelined architectures, the clock rate is determined by the slowest stage. Hence, even though the initial few stages are able to operate at high clock rates, the overall pipeline speed is set to the minimum clock rate across all stages. Similarly in hash based search operations, the large hash tables cause the same effect, hence achieving higher throughput is challenging.

However, with StrideBV, the memory consumption across the pipeline is uniform and the stage structure is regular. Therefore, the clock rate of the pipeline is not governed by a single stage. Since the amount of computation in a single stage is merely loading a bit-vector from stage memory and performing bit-wise logical *AND* operation, the time spent for the computation is minimal. Further, considering the resource distribution of a FPGA chip, the StrideBV architecture can be mapped onto the chip fabric in such a way that the routing complexity can be minimized to achieve high-throughput.

B. TCAM

TCAM is a type of memory which operates differently than SRAM/DRAM. For SRAM/DRAM, an address is given and then the data residing in that address is output. With TCAM, the content is given and the address at which the content is available is output. In the case of packet classification, the TCAM contains the ruleset. Similarly, for IP lookup, the content will be the routing table. The distinction between TCAM and Binary CAM (BCAM) is that a TCAM is able to handle wildcards while BCAMs can only handle binary strings. Therefore, TCAMs are widely used for IP lookup and packet classification type applications.

The operation of a TCAM can be described as follows. Upon receiving an input, the input is compared against all the stored entries in a parallel fashion, in a single clock cycle. For a given input, there can be more than one match since wildcard search allows one to perform prefix search and a given input may match multiple prefixes. In order to resolve the issues of multiple matches, a priority encoder is integrated into TCAMs to extract the highest priority match. The priority can simply be the order in which the content is stored. In the case of IP lookup, the prefixes can be stored by their prefix length and this yields longest prefix match [20].

Although the operation of the TCAM is fairly simple, due to the massively parallel search, the power consumption of TCAM is relatively high compared with SRAM/DRAM based solutions. A single TCAM cell requires 16 transistors while a single SRAM cell requires only 6 transistors. In the case of DRAM, the transistor count is 1 and it also includes a capacitor. Even though the TCAMs and RAMs are not directly comparable due to the nature of their operation, in the case of networking applications, power efficient architectures can be developed that outperform TCAMs with respect to power and throughput.

IV. Architecture

A. StrideBV

The basic StrideBV architecture utilizes uniform stages, implemented with Static Random Access Memory (SRAM) for stage memory storage of the 2^k bit vectors corresponding to each possible combination of the k bit stride. At each stage s of the pipeline, the stride $[sk : (s+1)k - 1]$ is used as the address to stage memory. The resulting output of stage

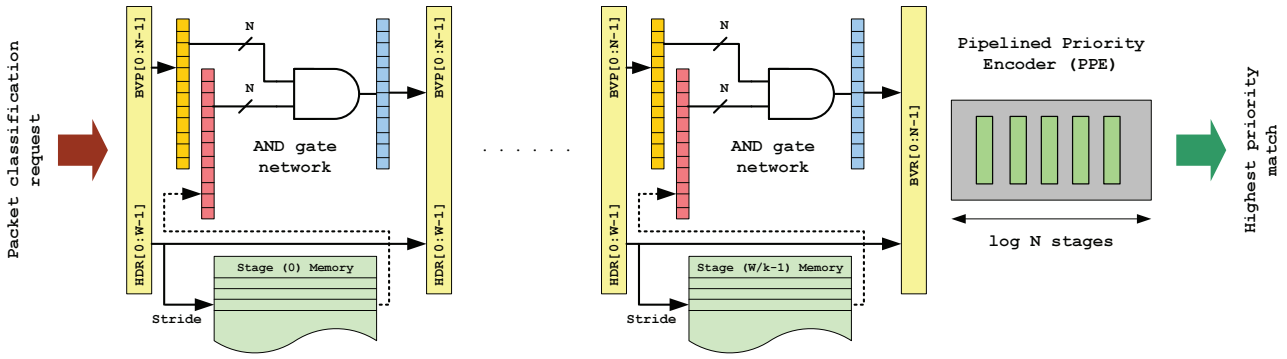

Fig. 2: StrideBV pipelined architecture with pipelined priority encoder [5]

memory is an N bit vector, which is ANDed with the bit vector from the preceding stage to produce an intermediate result to be sent to the next stage. The final output of the initial pipeline is a multi-match result, however, in packet classification, only the highest priority match is reported since secure packet routing is the major concern than reporting which rules matched for a given packet header. A priority encoder can easily extract the desired match in a single cycle. Although with longer bit vectors, the time it takes to find the highest priority match increases proportional to the bit vector length, which leads to a loss of throughput. To avoid this potential problem, a Pipelined Priority Encoder (PPE) was used at the end of the StrideBV pipeline. For an N bit vector, a PPE consists of $\log N$ number of stages. In each stage, only a small amount of work needs to be done, which means that the PPE can operate at very high frequencies, avoiding the performance bottleneck incurred by a single stage priority encoder. StrideBV architecture is illustrated in Figure 2.

StrideBV can be implemented using either distributed RAM or block RAM as stage memory. Originally, StrideBVs main goal was to achieve a high throughput, with the concept that using distributed RAM over block RAM would lower routing delays. We consider both types of memory in this paper, measuring performance in terms of power efficiency, throughput, and resource consumption to determine which memory is better suited for StrideBV, and to verify the scalability o StrideBV using both types of memory while sustaining higher throughput.

1) Distributed-Memory Based: The benefit of distributed RAM lies in its location. Distributed RAM is implemented using logic slices, allowing the logic and stage memory to reside closely together, and with proper logic placement, will result in significant reduction in wire length. This reduction leads to a lower clock period, though the tradeoff for increased throughput is the increase in loss of memory availability. With larger architectures that require greater amounts of logic, there may not be enough slices to map all of the required memory. However, during the experiments with StrideBV, the memory requirements never overreached the available distributed RAM and, with careful placing, the clock rate could be further improved, making the use of distributed RAM instead of block RAM as the stage memory a strong contender.

2) Block-Memory Based: Block RAM is generally used when memory consumption is a concern, due to its high availability as a dedicated memory resource. Unlike distributed RAM, there is no competition between ensuring enough memory with logic slices while also using the logic for the rest of the architecture. The separation from the logic though is both the strength and weakness in block RAM. Since block RAM is already firmly laid out on the FPGA, there are greater limitations on placing the block RAMs close to the logic, leading to longer wire lengths, which leads to increased routing delay and poorer throughput. When running experiments with block RAM, we attempted to improve the routing delay with careful placement of logic in conjunction to block RAM, to dampen the weakness of block RAM.

Further improvement can be done by employing a combination of distributed and block RAM for multiple pipelines, achieving high throughput and utilizing all available resources. The combination is not explored here in this paper, but can be done to achieve 400G+ throughput.

B. TCAM

We use an FPGA implementation of TCAM [21] for our experiments, instead of a standard ASIC implementation, for a focused comparison of the two methods (the brute-force solution versus the bit-vector based solution) on FPGA. The architecture of the FPGA implementation is built differently than the ASIC implementation, using the resources available on the FPGA.

The basic functionality of the TCAM is shown as Figure 3. A control block determines what needs to be done and manages all functions. Before reading or writing, the TCAM needs to process the input data and its address in order to map to the appropriate memory block as well as encode the ternary bits. For the memory block, at the output, the TCAM interprets which address(es) contain the desired data and generate the MATCH flags as well as register the outputs.

1) FPGA-based Architecture: Ternary CAM constructed on the FPGA uses SRL16E-based memory, considering one SRL16E to be thought of as a 16-bit deep by 1-bit wide RAM. For a regular CAM, this translates into a 4-bit wide by 1-bit deep CAM, however in the case of TCAM, one SRL16E translates into a 2-bit wide by 1-bit deep, when taking into account that a two-bit data input has a two-bit mask value for

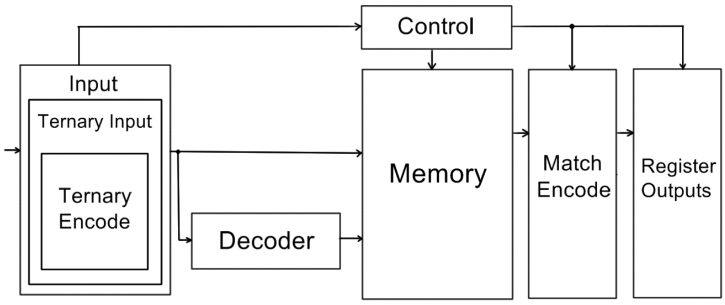

Fig. 3: TCAM Implementation on FPGA [21]

the "don't care" or $*$ condition (a 4-bit word). A 1 as the data mask bit indicates $*$ and a 0 indicates that the value of the data input matters for a match. A ternary encoder is used to convert the data input and mask into a 4-bit output, each bit indicating whether a 2-bit encoded value can match the incoming 2-bit ternary value. Consider the four bits of the encoder as the A, B, C, and D bit for the following example.

Using a single SRL16E, assume that the A bit is high only if the ternary value can match 00 (the possible matches would be 00, 0$*$, $*$0, and $**$). ABCD are used as the address into the SRL16E, and the output of the SRL16E is only high if the ternary value matches the data stored at the address, signifying a match. This logic is repeated for each address, and in the case of more than one match, the TCAM will return the address with the highest priority.

C. ASIC-based Implementations

In this work, we compare the two architectures implemented on FPGA only. However, it is possible to implement both architectures more efficiently on ASIC for better performance in terms of throughput and power efficiency. Such customized architectures can be optimized in terms of routing and circuit layout by exploiting the fine-grained control one has over ASIC designs. This yields more efficient architectures although due to the custom nature of these architectures, they offer little or no flexibility for modifications, while an architecture implemented on FPGA can be easily reconfigured either statically or dynamically. When considering the Non-Recurring Engineering (NRE) cost and overall time taken to develop an architecture, FPGAs are superior compared with ASIC.

An ASIC-based TCAM chip typically supports $200+$ MHz with a capacity of 18 Mbit and a power consumption of 15 W [2]. The amount of power dissipated is proportional to the number of entries active in the chip as it is possible to control the enabling of entries on a per entry basis. Hence, the dynamic power can be controlled at a per active entry granularity. In [2] it has been shown that the total static power of a TCAM chip is around 0.8 W for 70 nm feature size. Therefore, the per bit power consumption of a TCAM can be calculated as $0.8 + \frac{(15-0.8)}{104} * N$ W, where N is the number of packet classification rules (each 104 bits long). The direct comparison of ASIC-based TCAM, mentioned previously, with FPGA implementations of StrideBV reveal that ASIC-based TCAMs have superior power performance, however, the same power efficiencies can be achieved if StrideBV is implemented on ASIC platforms. However, a comparison of ASIC-based implementations is beyond the scope of this paper.

V. Performance Evaluation

In this section, we discuss our detailed performance analysis, evaluating the results of experiments conducted on FPGA Virtex 7 XC7VX1140T [22] with -2 speed grade. For a thorough comparison of the two packet classification schemes, we used ruleset sizes ranging from 32 to 2048 rules, and used a stride size of 3 and 4 for StrideBV. The smaller the stride size, the more resources are consumed, and the larger the stride size, the more memory is needed per stage; the two demands require a delicate balance to achieve the best results. We employed stride size 3 and 4 to maintain that balance between memory consumption and resource usage. Due to the exponential increase in memory along with stride size [5], going beyond the selected strides of 3 and 4 will result in additional undesirable memory consumption. The performance of the two architectures is measured in throughput, memory efficiency, power and resource usage, using post place-and-route performance from the Xilinx ISE 14.1 development tools. The Virtex-7 device used has 178k logic slices which supports a maximum of 18 Mbit distributed RAM, 68 Mbit of block RAM, and 1100 Input/Output (I/O) pins.

A. Clock Frequency

Dual-port RAMs were used for stage memory, allowing the StrideBV architecture to process two packets every clock cycle, resulting in higher throughput than single-port RAM. Furthermore, multiple pipelines could be employed through the use of a combination of distributed and block RAM as well to continue improving throughput, however we chose not include those extra pipelines in our experiments, instead focusing on the comparison the two approaches in a fair and a detailed manner. Figure 4 shows the comparison of throughput against the number of rules. The results are shown for the TCAM implementation on FPGA, StrideBV with distributed

RAM as stage memory and StrideBV with block RAM as stage memory.

With regards to throughput, the TCAM implementation on FPGA has noticeably lower throughput than any of the cases of the StrideBV architecture. Both approaches show the similar trend of degrading throughput with regards to increase in ruleset size. There was an average $6\times$ throughput improvement over TCAM for both a stride of 3 or 4 using distributed RAM, and an average $4\times$ improvement over TCAM for either stride length using block RAM for memory. The advantage of TCAM is that it has a lookup time of $O(1)$, achieving a steady access time when attempting to match an incoming packet with the ruleset independent of any change in the size of the ruleset. However, due to its large resource footprint and increased routing on FPGA, the clock rate is lower, resulting in lower throughput despite the $O(1)$ lookup time. Using block RAM results in a lower throughput than distributed RAM because of the increased wire length between logic and memory, and the distributed RAM version of StrideBV has an average of $1.3\times$ improvement in throughput over the block RAM implementation. Block RAM has lower decline in throughput overall as the ruleset size increases, unlike distributed RAM, which must be carefully placed in order to effectively shorten routing delays, and has equivalent results with its counterpart for larger rulesets. However, as we show next, using chip floor-planning the clock rate of the StrideBV implementation can be further improved.

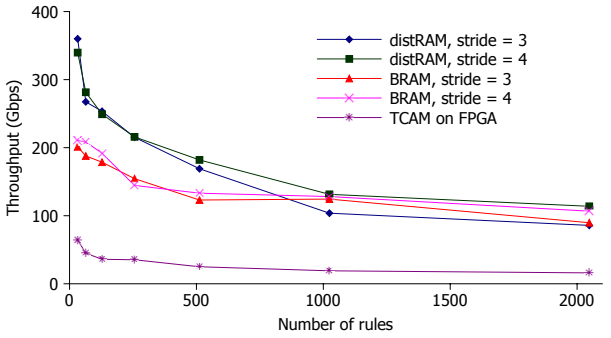

Fig. 4: Throughput vs number of rules

The advantage of the StrideBV approach is the possible exploitation of the resource layout of the FPGA architecture to map the pipeline in a regular fashion onto the chip. Using Xilinx PlanAhead, we thoroughly aligned the StrideBV pipeline with the chip layout to ensure short wire lengths over a minimized area, gaining considerable performance improvements over simple place-and-route results. Figure 5 and 6 show two examples of the gains we achieved when mapping the pipeline using PlanAhead versus allowing the place-and-route tool to do the placement of the pipeline without attempting to optimize, one example with distributed RAM as memory and a stride of 4, and the other example with block RAM as memory and a stride of 3. Whether using distributed RAM or block RAM, there is notable improvement with regards to throughput when extensively mapping the pipeline. Careful mapping can be the difference between 100 Gbps and 150 Gbps, as shown for stride 4 using distributed RAM for the ruleset size of 1024.

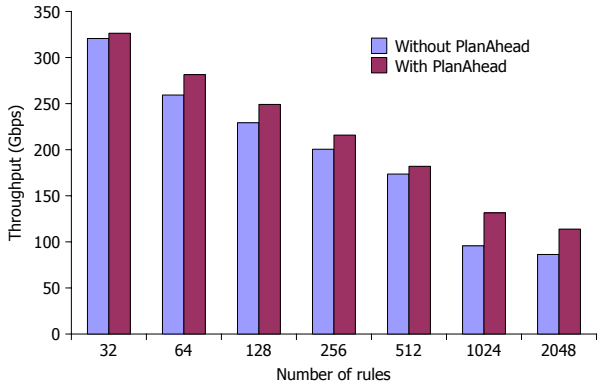

Fig. 5: Throughput comparison: Distributed RAM, stride 4

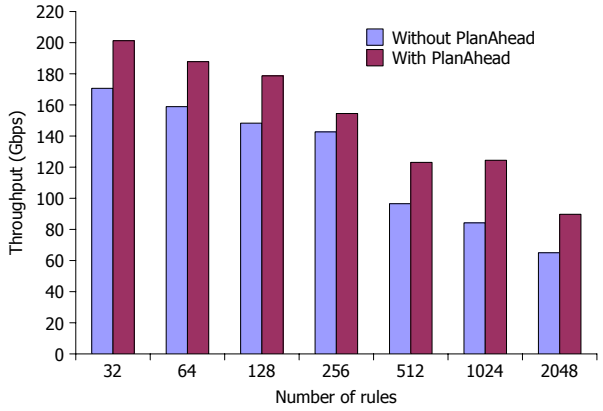

Fig. 6: Throughput comparison: Block RAM, stride 3

B. Memory Consumption

Memory efficiency has always been a concern with packet classification solutions, especially when working with platforms like FPGA, which has limited on-chip memory. TCAM has the lowest memory requirement in comparison with StrideBV, which uses the stride lengths of 3 and 4 for this comparison. Unlike StrideBV, TCAM's architecture requires only one table that contains the ruleset in its entirety, resulting in a lookup time of $O(1)$ when searching for a matching rule to the incoming data. The memory requirement for TCAM is double that of a regular CAM because of the added mask bit included to cover the "don't care" bit possibility. StrideBV's higher memory demand is a result of the pipelined architecture and $N \times 2^k$-sized stage memory for stride length k, though the memory requirement can be lowered by using a smaller stride, if increased pipeline length (hence, slightly increased packet latency) is acceptable.

Even with the worst case of 2048 rules, either architecture can be implemented using only on-chip memory of distributed RAM or block RAM, with the maximum of $<$900 Kbit for

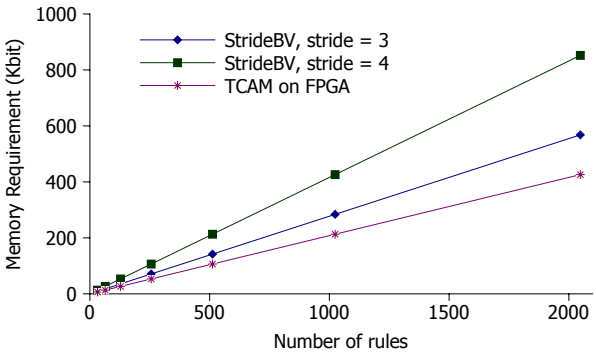

Fig. 7: Memory vs number of rules

StrideBV with a stride of 4. The memory requirement is dependent on the number of rules and the stride length, and so does not change when using block versus distributed RAM for the stage memory for StrideBV; those effects resulting from using different RAM as memory can be seen in the comparison of throughput, power efficiency, and resource consumption. Figure 7 shows the memory requirements for both architectures, which have linear increasing trends for all looked-at packet classification solutions.

Though extra parallel pipelines were not included in these experiments, if additional parallel pipelines were desired either with a single type of memory or by a combination of distributed RAM and block RAM, the total memory consumption would need to include multiplication factors depending on the number of pipelines. For example, using dual-ported stage memory and implementing 6 parallel pipelines, a multiplication factor of $3\times$ would have to be included in the total memory consumption calculation.

C. Resource Usage

We define resource consumption in this paper as the percentage of slices (a *slice* is a unit of logic resource) and bonded IOBs utilized. The utilization results can be seen in Figure 8 and are shown for the TCAM and StrideBV architectures used on FPGA. Unlike some of the other results which have higher variance between TCAM and StrideBV (such as throughput), the resource consumption between the different architectures is a similar percentage despite the fact that the expectation may be that less slices would be used for StrideBV using block RAM instead of distributed RAM. However, work needs to be done to go between the isolated BRAM and the logic used, resulting in an increase in slices despite no conversion of logic to memory. Only until larger ruleset sizes are used, after $N = 1024$, does the consumption begin to delineate, with block RAM consuming more than any of the other options.

TCAM and StrideBV for stride length 3 require more slices and IOBs than stride length 4 of StrideBV. This phenomena is directly linked to the tradeoff off between memory requirement per stage and the number of stages. The larger the stride length, the less stages are employed within the StrideBV architecture, leading to less overall resource requirements even if the stage memory increases. In the case of $k = 4$, there is an improvement of $1.3\times$ less slice usage than any of the other options. When comparing the use of block RAM against distributed RAM, more slices are required for the separate memory, and significantly more is needed when larger ruleset sizes are used. While the use of block RAM results in more resource consumption for $N = 2048$ or greater, the other consequence is similar throughput to StrideBV using distributed RAM, which otherwise had noticeably higher throughput at lower ruleset sizes until $N = 1024$, after which throughput results become similar.

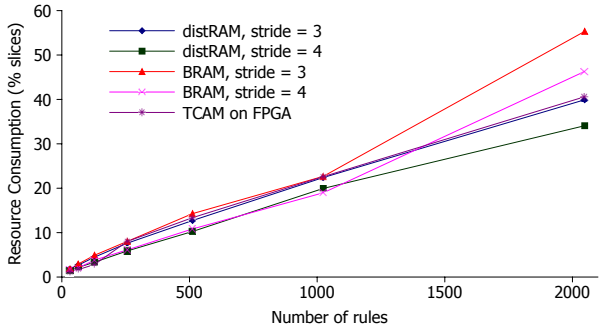

Fig. 8: Resource consumption vs number of rules

We also observe the percentage of block RAM utilization for StrideBV, shown in Figure 9. The worst-case scenario occurs at $k = 3$ with $N = 2048$, utilizing all the available block RAM fully. Real-life firewall classifiers rarely have more than 1000 rules [9], but if larger ruleset sizes were desired beyond block RAM capabilities, distributed RAM can be utilized as stage memory as well to continue to implement StrideBV solely with on-chip resources available and increase allowable ruleset size further than the current limit of 2048 if block RAM is the desired memory. Distributed RAM has a higher limit, since even at $N = 2048$, only about 40% of the resources are consumed as shown in Figure 8. With the availability of block RAM and distributed RAM, StrideBV can be very flexible in its performance, and any mixture of distributed RAM and block RAM can be used to increase throughput, or ensure maximum resource consumption. Figure 7, Figure 8 and Figure 9 all have similar trends of increasing requirements with the number of rules, due to the nature of the StrideBV architecture, requiring more resources and memory for more stages as N increases.

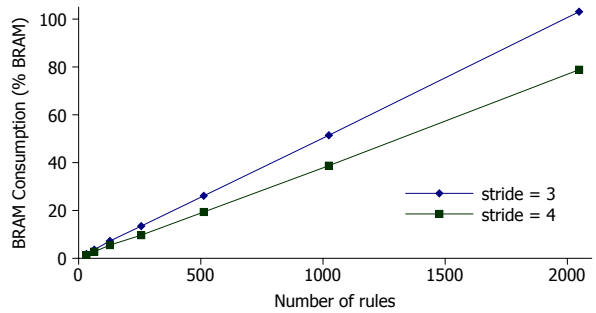

Fig. 9: BRAMs consumed by StrideBV vs number of rules

TABLE II: Performance Comparison

Approach	Memory Req. (Bytes/rule)	Throughput (Gbps)	Power Eff. (μW/Gbps)
StrideBV (k = 3) distRAM	35	169	4934
StrideBV (k = 4) distRAM	52	182	6186
StrideBV (k = 3) BRAM	35	123	42404
StrideBV (k = 4) BRAM	52	133	30133
TCAM-FPGA	26	25	36837
TCAM-SSA [23]	13	20	5150
Pattern-Matching [6]	15	10	-
B2PC [12]	164	15	-

D. Power Efficiency

We used the XPower Analyzer tool available in the Xilinx ISE 14.1 suite to measure the power consumption of our device for the various cases shown. The results are presented in Figure 10 and the metric used is Watts per Gbps (W/Gbps). The power consumed by logic, distributed RAM, and block RAM are all reported in this figure.

There is a key improvement to power efficiency when distributed RAM is used instead of block RAM. For both stride lengths 3 and 4, distributed RAM has improved power efficiency over block RAM by a noticeable margin. When block RAM is utilized for a stride length of 3, its power efficiency is $4.5\times$ worse than either case of distributed RAM employment, and $3.5\times$ worse for a stride length of 4.

There is a significant gap between the power efficiency of stride length 3 versus 4 when employing block RAM, which increases with ruleset size. The average improvement for $k = 4$ is $1.3\times$ power efficiency in comparison to $k = 3$. The lower power efficiency results from the higher resource demands as well as the power demands of block RAM. Since distributed RAM utilizes logic slices, there will be no power consumption from additional block RAMs, leading to distributed RAM having lower power and resource demands, along with higher throughput, making it the stronger candidate for memory in the StrideBV architecture. Block RAM may be better suited as additional stage memory instead of the main memory in the case of more pipelines being required beyond the resource capabilities of distributed RAM alone to improve throughput further.

The notable increase in power when using block RAM for memory can be partially attributed to the necessity for longer wires to connect the block RAM to the registers holding the values between stages. When memory is held in distributed RAM, the LUTs are physically closer to the memory and are flexible in placement, while the block RAM is available only in specific areas on the chip, separated from the registers. Another reason that may have impacted the power results is that the amount of block RAM required for memory is smaller than the actual amount active. Block RAM has a minimum size requirement, and at the strides of 3 and 4, some of the block RAM is unused but still dissipates power. By using a larger stride size, block RAM could be better utilized however, the increase in memory consumption would be far beyond the memory requirements of other approaches, making the approaches incomparable.

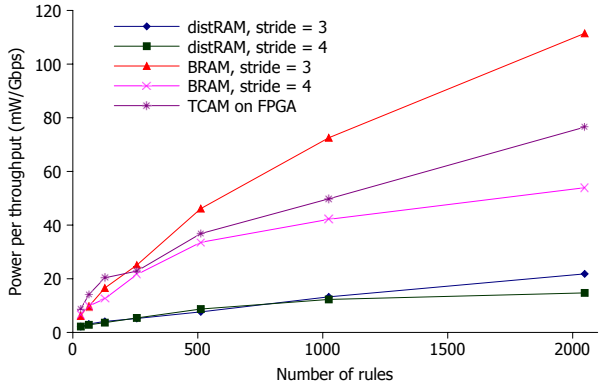

Fig. 10: Power per unit throughput vs number of rules

E. Comparison with Other Approaches

We also compare our results against a set of other existing solutions for multi-match packet classification, continuing with the assumption of 5-field classification and using a ruleset size of 512 rules for this performance comparison. Table II shows each approach's results for memory requirement, throughput, and power efficiency.

Using worst-case performance in this comparison, [23] and [6] have better memory efficiency than either the TCAM implementation on FPGA or StrideBV for either case of using distributed RAM or BRAM for memory. The TCAM implementation has more demanding memory requirements due to the need for a data bit and a mask bit, which is required to sustain the "don't care" bit for every bit of the stored ruleset, but it still has better memory requirement than StrideBV. StrideBV has one of the highest demands due to the parallel pipelined implementation, only lower than [12]. StrideBV, however, can make up for this deficit by being able to improve its memory efficiency through the use of a smaller stride length to achieve desired results, while other methods cannot be as flexible with their memory efficiency, making StrideBV a unique solution in this sense.

With regards to throughput, StrideBV has the highest throughput by at least $6\times$ for distributed RAM and $4\times$ for block RAM memory when compared against any other approach, and also has the best power efficiency when using distributed RAM for stage memory and k = 3, though close to the power efficiency of [23]. Due to the separation of block RAM and logic, StrideBV's power efficiency is poor when using block RAM and we can conclude from these results that using distributed RAM can achieve higher performance results with regards to both throughput and power efficiency. Though the power efficiency of StrideBV using distributed RAM is similar to the power efficiency of TCAM-SSA, a TCAM-only approach which splits the filters for separate TCAM lookup, StrideBV can be improved by adding more pipelines and increasing the throughput further without immense gains in power consumption, resulting in a stronger performance against other solutions' power efficiency at the cost of added resources. The FPGA implementation of TCAM has low power efficiency in comparison to other solutions except for

block RAM, and along with its small throughput, shows that StrideBV is the more efficient method when using distributed RAM. The memory efficiency of StrideBV can be altered with stride size, but the high performance against other approach cannot be so easily breached by other inflexible methods.

Overall, the lower performance of StrideBV using block RAM as the main memory instead of distributed RAM, both against the distributed RAM-based StrideBV and other packet classification solutions, further cements the conclusion that the decision in [5] to use distributed RAM was the correct choice. Though the throughput is not immensely disproportionate in the comparison, the power efficiency for block RAM shows clearly that even though block RAM can hold its own with only a slightly lower throughput, its low power efficiency in comparison to distributed RAM makes distributed RAM a much stronger choice. From the above comparison it is obvious that the quality of being ruleset feature independent and delivering high performance makes StrideBV a unique packet classification solution.

VI. Conclusion

In this paper, we extensively evaluated two ruleset independent packet classification solutions on a state-of-the-art Field Programmable Gate Array (FPGA) platform. The two schemes are Ternary Content Addressable Memory (TCAM) (a brute-force solution) and StrideBV (an algorithmic approach). We compared the performance of these two approaches with respect to throughput, power efficiency, resource consumption and memory efficiency. The post place-and-route results on a large Xilinx Virtex 7 device showed that the algorithmic solution, StrideBV, outperforms the brute-force solution TCAM by a considerable margin with respect to all the metrics except for memory efficiency. The higher memory consumption of StrideBV is due to the increased per stage memory consumption.

We showed that the throughput of the StrideBV architecture is nearly $6\times$ and $4\times$ higher than that of when using TCAM when using distributed RAM and block RAM, respectively. The power efficiency of StrideBV is $4.5\times$ and $3.5\times$ compared with TCAM for distributed and block RAM, respectively. Further, we showed that by exploiting the regular architecture of StrideBV, it can be mapped on to the FPGA chip in an efficient manner in order to reduce routing delays, which yields higher clock frequency, hence higher throughput. Overall, we conclude that the StrideBV solution is a more efficient, algorithmic implementation of TCAM which yields higher performance and hence become a potential alternative for brute-force TCAM.

References

[1] B. Agrawal and T. Sherwood. Modeling tcam power for next generation network devices. In *Performance Analysis of Systems and Software, 2006 IEEE International Symposium on*, pages 120 – 129, march 2006.

[2] B. Agrawal and T. Sherwood. Ternary cam power and delay model: Extensions and uses. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 16(5):554 –564, may 2008.

[3] Mike Attig and Gordon Brebner. 400 gb/s programmable packet parsing on a single fpga. In *Proc. 7th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, pages 12–22, oct. 2011.

[4] Bro. The bro network security monitor. http://bro-ids.org/.

[5] Thilan Ganegedara and Viktor Prasanna. Stridebv: 400g+ single chip packet classification. In *Proceedings of the IEEE Conference on High Performance Switching and Routing*, HPSR'12, 2012.

[6] N. Guinde, S. Ziavras, and R. Rojas-Cessa. Efficient packet classification on fgpas also targeting at manageable memory consumption. In *Signal Prcoessing and Communication Systems (ICSPCS)*, pages 1 –10, december 2010.

[7] P. Gupta and N. McKeown. Classifying packets with hierarchical intelligent cuttings. *Micro, IEEE*, 20(1):34 –41, jan/feb 2000.

[8] P. Gupta and N. McKeown. Algorithms for packet classification. *Network, IEEE*, 15(2):24 –32, mar/apr 2001.

[9] Pankaj Gupta and Nick McKeown. Packet classification on multiple fields. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '99, pages 147–160, New York, NY, USA, 1999. ACM.

[10] G.S. Jedhe, A. Ramamoorthy, and K. Varghese. A scalable high throughput firewall in fpga. In *Field-Programmable Custom Computing Machines, 2008. FCCM '08. 16th International Symposium on*, pages 43 –52, april 2008.

[11] Weirong Jiang and Viktor K. Prasanna. Field-split parallel architecture for high performance multi-match packet classification using fpgas. In *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*, SPAA '09, pages 188–196, New York, NY, USA, 2009. ACM.

[12] I. Papaefstathiou and V. Papaefstathiou. Memory-efficient 5d packet classification at 40 gbps. In *Proc. INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 1370 –1378, may 2007.

[13] T. Sasao. On the complexity of classification functions. In *Multiple Valued Logic, 2008. ISMVL 2008. 38th International Symposium on*, pages 57 –63, may 2008.

[14] Sumeet Singh, Florin Baboescu, George Varghese, and Jia Wang. Packet classification using multidimensional cutting. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '03, pages 213–224, New York, NY, USA, 2003. ACM.

[15] Snort. Snort: Network intrusion prevention and detection system (ips/ids). http://www.snort.org/.

[16] Haoyu Song and John W. Lockwood. Efficient packet classification for network intrusion detection using fpga. In *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, FPGA '05, pages 238–245, New York, NY, USA, 2005. ACM.

[17] T. Srinivasan, N. Dhanasekar, M. Nivedita, R. Dhivyakrishnan, and A.A. Azeezunnisa. Scalable and parallel aggregated bit vector packet classification using prefix computation model. In *Parallel Computing in Electrical Engineering, 2006. PAR ELEC 2006. International Symposium on*, pages 139 –144, sept. 2006.

[18] David E. Taylor. Survey and taxonomy of packet classification techniques. *ACM Comput. Surv.*, 37(3):238–275, September 2005.

[19] D.E. Taylor and J.S. Turner. Scalable packet classification using distributed crossproducing of field labels. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 1, pages 269 – 280 vol. 1, march 2005.

[20] Wikipedia. Longest prefix match. http://en.wikipedia.org/wiki/Longest_prefix_match.

[21] Xilinx. Cam application notes. http://www.xilinx.com/support/documentation/anmeminterfacestorelement_cam.htm,.

[22] Xilinx. Virtex-7 fpga family. http://www.xilinx.com/products/silicon-devices/fpga/virtex-7/index.htm.

[23] F. Yu, T. Lakshman, M. Motoyama, and R. Katz. Ssa: A power and memory efficient scheme to multi-match packet classification. In *Proceedings of the 2005 ACM symposium on Architecture for networking and communications systems*, pages 105 –113, 2005.