StrideBV: Single Chip 400G+ Packet Classification

Thilan Ganegedara, Viktor K. Prasanna
Ming Hsieh Dept. of Electrical Engineering
University of Southern California
Los Angeles, CA90089
Email: {ganegeda, prasanna}@usc.edu

***Abstract*—Hardware firewalls act as the *first line of defense* in protecting networks against attacks. Packets are organized into *flows* based on a set of packet header fields and a predefined rule is applied on the packets in each flow to filter malicious network traffic. This is realized using packet classification, which is implemented in secure networking environments where mere best-effort delivery of packets is not adequate. Existing packet classification solutions are highly dependent on the properties (or features) of the ruleset. We present a bit vector based lookup scheme and a parallel hardware architecture that does not rely on ruleset features. A detailed performance analysis of the proposed scheme is given under different configurations. Post place-and-route results of our parallel pipelined architecture on a state-of-the-art Field Programmable Gate Array (FPGA) device shows that for real-life firewall rulesets, the proposed solution achieves 400G+ throughput. To the best of our knowledge, this is the first packet classification engine that achieves 400G+ rate on a single FPGA. Further, on the average we achieve $2.5\times$ power efficiency compared with the state-of-the-art solutions.**

I. Introduction

Secure networking is becoming crucial with the various attacks networks are being exposed to [13]. To protect networks from such attacks, various hardware equipment as well as software tools are being extensively used [11]. Packet classification and deep packet inspection (DPI) are implemented in these systems to detect and neutralize potential threats by discarding malicious traffic. Packet classification stands as the initial filter to the network in which network traffic is classified into *flows* based on a pre-defined set of rules. It requires the inspection of multiple fields of the packet header compared with best-effort IP forwarding where only the destination IP address is inspected. Hence, it is more challenging especially in environments where *wire-speed* processing of packets is mandatory.

From a hardware perspective, the main bottleneck in implementing packet classification engines has been the amount of memory required to store the ruleset. Especially on platforms such as Field Programmable Gate Arrays (FPGAs), on-chip memory is limited and exploiting external memory is challenging in applications of this nature. To circumvent this, various solutions have been proposed in the literature to reduce the memory footprint of ruleset storage. Most, if not all, of these solutions exploit some properties (or features) of the ruleset to achieve memory efficiency [8], [5], [12], [14], [6]. While these features may be present in some rulesets, we cannot always assume that to be true. In such cases, the heavily feature-dependent solutions may yield poor memory efficiency.

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

With the advancements in memory technology, coping with the memory bottleneck has become a secondary concern. However, high-speed networking is becoming critically important with the stringent throughput demands and Quality of Service (QoS) requirements. To achieve this, networking hardware must be capable of delivering the required throughput and QoS. While 100 Gbps networking is becoming standard, both the research community and the industry are targeting 400 Gbps and even 1 Tbps networks to support future networking demands [1], [2].

Combining memory efficiency and high throughput in the same solution is difficult and challenging due to many reasons. For example, in the trie/tree based approaches, on-chip resources are easily exhausted due to pipelined tree traversal and multiple field lookup. Therefore, implementing parallel pipelines to improve the throughput becomes infeasible. In this work, we consider improving throughput as the primary goal while memory efficiency is kept secondary.

We present *StrideBV* a simple, yet high-throughput packet classification scheme based on field-split algorithm [8]. We avoid depending on features of the ruleset to optimize for memory efficiency, which makes our packet classification engine a robust solution. Using FPGA as a hardware platform, we demonstrate that via tight integration of logic and memory resources available on-chip, our parallel pipelined architecture can achieve a throughput of 400 Gbps and beyond for real-life rulesets. Further, the classification engine can be customized as a single 400 Gbps engine or four 100 Gbps engines for added flexibility. While achieving high-throughput, on the average, the proposed scheme achieves $2.5\times$ power efficiency compared with the state-of-the-art solutions available in the literature. We summarize our contributions in this work as follows:

- The *first* 400G+ packet classification engine solution for real-life rulesets on a single chip
- Performance independent of ruleset features
- Detailed performance analysis wrt throughput, power and resource usage under various configurations
- $2.5\times$ average power efficiency compared with state-of-the-art packet classification engines

II. Background and Related Work

A. FPGA for High-speed Networking

FPGAs are widely used in various networking applications such as packet forwarding, packet classification, DPI, etc [16], [14], [2]. The most salient features of FPGA for networking are reconfigurability, abundant parallelism, vast amount of on-chip logic and memory resources and the enormous on-chip memory bandwidth. These features make it an ideal platform to implement algorithmic solutions for networking applications. Even though the operating frequency of FPGA is low (typically 100-400 MHz) fine-grained pipelining can be used to dramatically improve the performance. Further, parallel architectures can also be effectively implemented on FPGA fabric.

B. Related Work on Packet Classification

Packet classification rulesets, or classifiers, are not made publicly available due to security reasons. However, in [5], the authors acquired access to real-life firewall rulesets from a set of Internet Service Providers (ISPs) and they revealed some statistics and common characteristics of the rulesets. One important fact revealed by the studies on real rulesets is the ruleset size. These rulesets are fairly small in size with a mean of 50 rules per ruleset. Further, only 0.7% of the rulesets contained more than 1000 rules and nearly 99% of the classifiers contained less than 500 rules. Even though the largest ruleset is targeted in most of the solutions proposed in the literature, according to the studies on real-life classifiers, the average number of rules is far less compared with the upper bound.

Solutions for packet classification can be categorized into two main groups: 1) Decomposition based and 2) Decision Tree based. Decomposition based approaches are two phased. In the first phase, individual field search is performed separately to produce partial search results and in the second phase the partial results are combined using an aggregation network. Various solution techniques based on Ternary Content Addressable Memory (TCAM), bit vector, tree/trie traversal and hashing are present in the literature [7], [17], [14], [4].

Decision tree based approaches are radically different in that, the ruleset is mapped to a multi-dimensional space where each dimension represents a header field used in the classifier [6], [12]. Each rule forms a hypercube in the multi-dimensional space which represents the volume covered by that rule. Due to rule overlap, the hypercubes formed by the rules intersect with each other. A packet header becomes a point in this space and the challenge is to identify the hypercube with the highest priority, this point belongs to. A decision tree partitions the multi-dimensional space so that the search can be performed effectively guided by the packet header.

As mentioned in Section I, all the above solutions rely on the features of the classifier and the main goal is improving memory efficiency. The memory efficiency ranges from $10 \sim 80$ bytes/rule and the best throughput observed in these solutions is 100 Gbps [8]. Our goal is to avoid relying on the features of the ruleset and provide a high-throughput solution for packet classification.

III. StrideBV: Algorithm and Classification Process

A. Problem Definition

Packet classification is a generic scheme in which an arbitrary number of header fields of a packet can be inspected for the classification purpose. The most widely used scheme is 5-field packet classification in which the following tuple of headers of each incoming packet is inspected: ⟨*Source IP (SA), Destination IP (DA), Source Port (SP), Destination Port (DP), Protocol (PR)* ⟩. The type of lookup required for each field can be different. In 5-field classification, the two IP address fields require prefix match, the two port fields and protocol field require either range or exact match. With this understanding, we formally define our problem as follows:

Given a packet classification ruleset that has N number of rules that considers d number of packet header fields, $f_0, f_1, ..., f_{d-1}$, devise:

- A lookup scheme whose performance is independent of the features or properties of ruleset
- A hardware architecture to perform *wire-speed* packet classification for 400 Gbps and beyond

B. Field-split Algorithm

The field-split algorithm, in a networking context, was originally studied in Field-Split Bit Vector (FSBV) [8]. In FSBV, the primary goal is reducing memory consumption. Their studies on the 5-field packet classification rules appearing in the Snort [13] ruleset show that the SA, DA and PR fields of the rules contain a very small number of unique values compared with the ruleset size. For example, the SA field has 11 unique values in a ruleset of size 323. This provided them the opportunity to use TCAM/CAM for those fields that satisfy the condition $2 \times w_i > u_i$, where w_i is the bit-width of field f_i and u_i is the number of unique values in field f_i found within the classifier. The intuition behind this condition is the comparison against the bit vector based approach [14], whose memory requirement may increase in the order of $O(u_i \times N)$ for field f_i. Since reducing the memory requirement was the main goal in FSBV, the authors applied the field-splitting algorithm only to the SP and DP fields, which satisfied the aforementioned condition.

The field-split algorithm is as follows: In a rule R_k, for a given field f_i that has a bit width of w_i, it can be partitioned into a set of w_i sub-fields ($f_i[w_i]$), where each sub-field corresponds to a single bit position in field f_i. This limits the number of values each sub-field can take to 2, which are $[0, 1]$. Extending this to all the rules in the classifier yields two N bit vector that corresponds to the two possibilities of the sub-field $f_i[w_i]$, which are $f_i[w_i] = 0$ and $f_i[w_i] = 1$. To better understand this process, we use an example ruleset as shown in Figure 1. Here we consider a hypothetical header field of bit width 4.

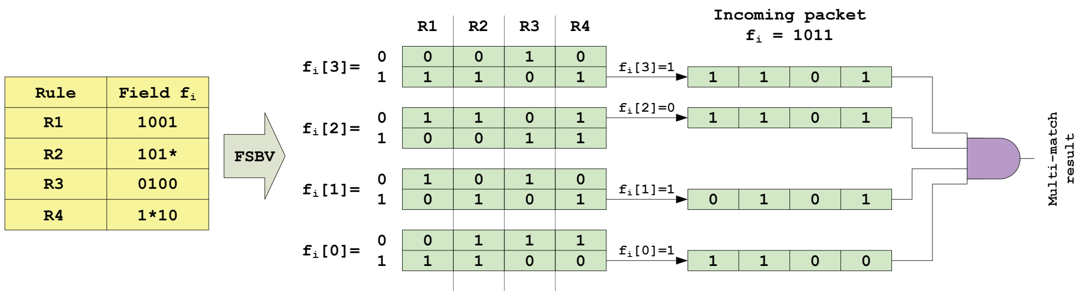

Fig. 1. Field-split bit vector generation and classification operation

In Figure 1, the bit vector generation process as well as packet lookup process is illustrated. The individual sub-fields form two vectors of size N. When a packet header arrives requesting classification, the corresponding bit vectors are loaded and a bit-wise logical AND operation is performed to identify the matching result. A bit vector in the context of packet classification is an N bit vector of which, each bit is representing a rule of the classifier. A bit position set to `1` indicates a match, while a `0` indicates otherwise. The correctness of this scheme is proved in [8]. The result of the AND operation is another bit-vector of size N, in which, each bit position k indicates whether or not there was a match with rule R_k for the incoming packet header.

Note that field-split has full support for wildcard ($\star$) matching by setting the corresponding bit vector positions to `1`. In rules R_2 and R_4, wildcards appear at bit positions 0 and 2, respectively. A wildcard indicates that the corresponding bit in the header could be either `0` or `1`. The importance of the wildcard character is prefixes can be easily expressed and implemented. Hence, prefix and exact matching is directly supported in the field-split algorithm.

However, expressing ranges in binary format requires range to prefix conversion [10]. This conversion results in partitioning a single rule into multiple rules, until the rule with the range can be expressed as a prefix. An alternative to this is to compare the field value with the explicit bounds. For example, for a range $[x, y]$, the comparison $x < f_i < y$ can be performed to test whether the packet header belongs to the given range or not. This requires additional hardware (comparators) and slight modification of the search algorithm. However, as shown in [5], range specifiers are seldom used (at most 10%, only in SP and DP fields). Hence, for our experiments, we assume that a rule can be expressed as a ternary string given by the following regular expression: `[0 1 ⋆]`$\{w_i\}$, for a field with w_i bits.

C. StrideBV: Algorithm

As mentioned earlier, in [8], the field split algorithm is applied only for the SP and DP fields due to the concern on memory consumption. For the other fields TCAM/CAM is used since the number of unique values is small. Our goal is to devise a solution that avoids relying on such ruleset features and achieves high throughput. Implementing TCAM on FPGA can become inefficient due to high circuit complexity and poor performance compared with pipelined architectures [14].

TABLE I
COMPARISON OF VARIATIONS TO STRIDEBV

Method	Memory size	Total memory bandwidth	# stages
1	$(2^k/k) \times N \times W$	$N \times W/k$	W/k
2	$N \times W$	$N \times W$	W/k

This limits the scalability as well as performance of FSBV as ruleset features change. We consider memory consumption as a secondary concern mainly because of the ability of the proposed solution to exploit various memory resources available on FPGA. Hence, in this work, we apply field-split algorithm to all the 5 fields.

The challenge in doing the above is the pipeline length. The resulting pipeline length becomes $W = \sum_{i=0}^{d} w_i$. In the case of 5-field packet classification, this amounts to 104 pipeline stages. From a network perspective, this causes packet latency to increase by a significant factor. From a hardware perspective, particularly on FPGA, this necessitates significant amount of routing which causing the performance to degrade. Reducing pipeline length in this approach can be done using multiple bits (k bit stride) than a single-bit inspection. This can be performed in two different methods by storing:

1) Bit vectors corresponding to the 2^k combinations of the k bit stride and load a single bit vector per stage
2) $2 \times k$ bit vectors corresponding to the individual bits of the k bit stride and load multiple bit vectors per stage

Table I compares the characteristics of the two approaches. The first method consumes more memory while reducing memory bandwidth and second method saves memory at the cost of memory bandwidth. However, it should be noted that in the second case, in a single stage, k number of N bit AND operations need to be performed. This increases the amount of work to be done per stage which causes the clock period to increase. Since our goal is to implement a high-throughput packet classification engine, we opt for the first method at the cost of increasing the memory consumption. Considering the stride memory access and use of bit-vector to perform classificaiton, we call the proposed scheme *StrideBV*.

D. Multi-match to Single-match

In [8], [14], 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 classification, only

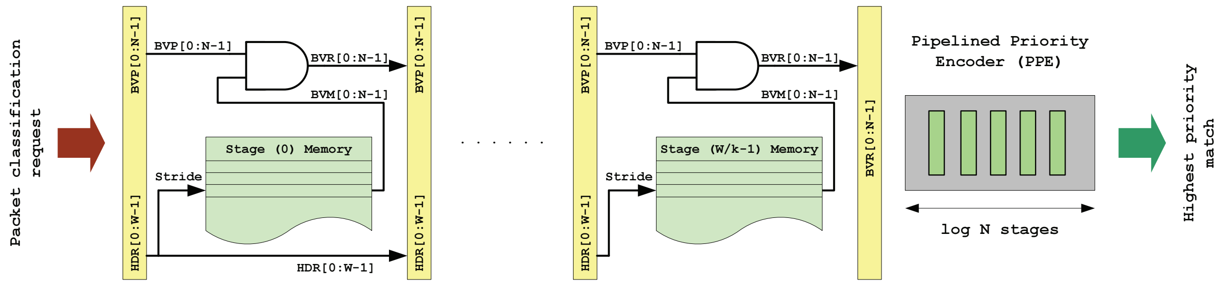

Fig. 2. StrideBV pipelined architecture (BVP/BVM/BVR - Previous/Memory/Resultant Bit-Vectors, HDR - 5-field header, Stride - k bit header stride)

the highest priority match is reported since routing is the main concern.

The rules of a classifier is 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 significantly. This causes the whole pipeline to run at a very slow clock rate which degrades the throughput.

As a remedy, we introduce a Pipelined Priority Encoder (PPE). A PPE for a N bit-vector consists of $\log N$ number of stages and since the work done per stage is trivial, the PPE is able to operate at very high frequencies. Hence, the performance bottleneck introduced by the single stage priority encoder can be effectively eliminated using a PPE.

E. StrideBV: Lookup Process and Hardware Architecture

StrideBV lookup process is fairly simple. At each stage s of the pipeline, the stride $[sk : (s+1)k-1]$ is used as the address to the stage memory. The output of the stage memory is the N bit-vector corresponding to the input stride. This bit-vector is ANDed with the bit-vector from the preceding stage to produce the intermediate result. This process is implemented as a linear Static Random Access Memory (SRAM) based pipeline. The output of the initial pipeline is the multi-match result. In order to extract the highest-priority match, the StrideBV pipeline is followed by a PPE. The overall architecture of the proposed solution is depicted in Figure 2.

IV. Performance Analysis on FPGA

In this section, we provide a detailed analysis of the StrideBV architecture under different configurations. The performance of the proposed architecture is measured in throughput, memory efficiency, power and resource usage. A state-of-the-art Xilinx Virtex-6 device (XC6VLX760) [15] was used for the experiments and the results presented here are based on the post place-and-route performance. The considered device has 118K logic slices which can be used either as pure logic or distributed RAM (8 Mbit max), 26 Mbit of block RAM and a total of 1200 Input/Output (I/O) pins. Since StrideBV does not rely on ruleset features, to evaluate the performance, we used ruleset sizes ranging from 32 to 512 rules, considering real-life firewall classifiers [5].

A. Throughput

The main goal of this work is to implement a high-throughput packet classification engine. For this, the architecture depicted in Figure 2 was mapped on to the FPGA fabric. In doing so, there were two options: Use 1) distributed RAM or 2) block RAM as stage memory. Here the trade-off is memory size vs. clock period. Since distributed RAM is implemented using logic slices, the wire length between logic and stage memory can be dramatically reduced which in turn reduces the clock period. In the case of block RAM, the increased wire length between memory and logic increases the routing delay which affects the throughput in a negative manner. Hence, we opted to use distributed RAM since the memory requirement for real-life classifiers in our application is less than the maximum distributed RAM available on the considered FPGA.

A single pipeline was not adequate to support 400 Gbps (this requires an operating frequency of 1.25 GHz, which current FPGA device do not support [15]). For this, we employed 4 pipelines for ruleset sizes less than 512 and 6 pipelines for ruleset size 512. With increasing ruleset size, the routing complexity of the design becomes significantly high, causing the clock frequency to decrease. Hence, multiple lookup engines were required to sustain 400 Gbps throughput. The dual-port feature of distributed RAM was used to efficiently share stage memory between two pipelines. This allows us to parallelize the architecture by sharing resources. Figure 3 shows the throughput variation with the size of the classifier for various stride sizes for minimum packet size (40 Bytes). It can be observed that the proposed scheme sustains 400G+ throughput for all considered classifier sizes. For lower stride size and large classifiers, mainly due to the increase of complexity in routing, a decrease in performance can be observed.

It can be seen that on the average, using a larger stride size is desirable from a throughput point of view. Further, this reduces pipeline length which in turn reduces packet delay. However, this comes at the cost of using extra memory. In the following section, we show the memory efficiency of this approach for varying stride size.

B. Memory Efficiency

Memory efficiency is not considered a major concern in this work. However, we show that the proposed architecture can be easily fitted on a single-chip while achieving 400 Gbps or beyond. Further, we employed only the distributed RAM

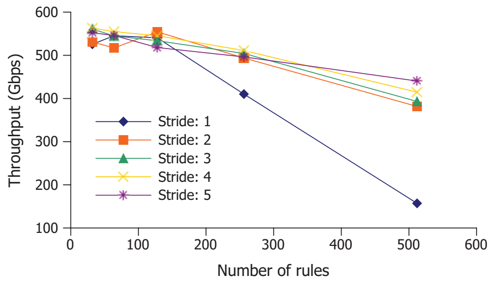

Fig. 3. Throughput vs. number of rules

(built using logic) in this work. Additional parallel pipelines can be implemented by utilizing the $3\times$ more block RAM available on-chip. Figure 4 illustrates the per pipeline memory efficiency. Since we use dual-ported stage memory, to implement 4 and 6 parallel pipelines, to calculate the total memory consumption, multiplication factors of $2\times$ and $3\times$ has to be introduced, respectively.

The worst case in memory efficiency is when stride size is 5 and $N = 512$. Even in this scenario, the memory consumption per pipeline is < 700 Kbit. Hence, even if we require $3\times$ more memory, the architecture can still be accommodated on-chip using only the distributed RAM. This allows instantiation of multiple parallel pipelines to achieve high-throughput.

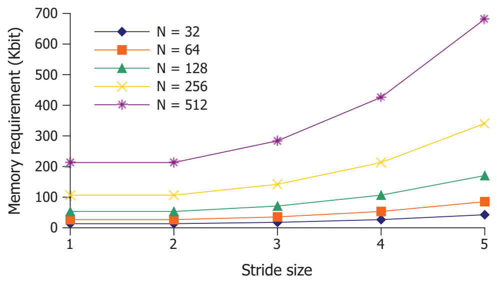

Fig. 4. Memory vs. stride size

C. Power Per Unit Throughput

The proposed StrideBV architecture is entirely logic based, however, it can be extended to a hybrid of memory (BRAM) and logic. To measure power consumption of our device, we used the XPower Analyzer tool available in the Xilinx ISE 12.4 suite. The metric used to measure power efficiency of networking devices is Watts per Gbps (W/Gbps) [3]. We evaluate our architecture using this metric and the results are shown in Figure 5. Here, we report the power consumed by logic and distributed RAM.

The key observation is that, using a small stride size yields lower power efficiency. This is mainly because of the extensive resource usage. As shown in Table I, using a stride of size k results in a pipeline of length W/k. In our application, using a stride size of 1 renders a pipeline of length 104. This increases the resource consumption which dictates the power consumption. From a power efficiency point of view, using a larger stride size is desirable.

Further, we observed that increasing k does not continually improve the power efficiency. When larger stride sizes are used, the per stage memory requirement increases by a factor of $2^k/k$ while the amount of logic decreases by a factor of $1/k$. Initially, the impact of memory power is less and logic power dominates. However, as k increases, memory power dominates the total power consumption, hence causes the total power to increase than decrease. For example, for $N = 512$, we observed $k = 5$ to be the minimum (i.e. the best) k value for power efficiency.

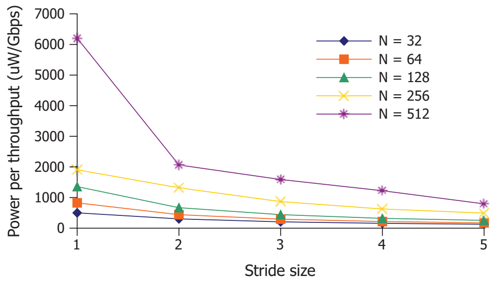

Fig. 5. Power per unit throughput vs. stride size

D. Resource Consumption

As mentioned in the previous section, the proposed architecture is logic oriented. On FPGA, the unit of logic resource is a *slice*. We observed the resource consumption based on the percentage of slices utilized. Figure 6 illustrates the results. Comparing Figure 5 and Figure 6, one can observe similar variation in the two figures. This is because our architecture is purely logic based and the reported power is based on logic power consumption.

In the case of $k = 1$ and $N = 512$, almost all the logic resources available on the considered FPGA is consumed. In such a scenario, two approaches can be taken to alleviate the resource limitation: 1) Use a larger stride size or 2) Exploit block RAM. A practical option would be to use a larger stride size considering all other performance metrics.

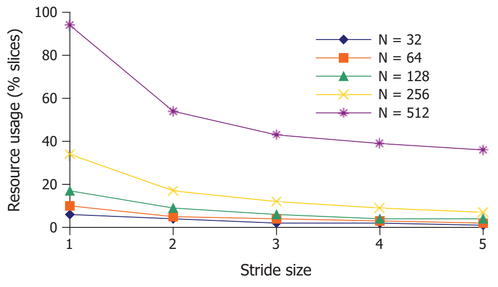

Fig. 6. Resource (logic) usage vs. stride size

E. Comparison with Existing Approaches

We compare the worst case performance of several existing solutions to illustrate the benefits of StrideBV. Table II summarizes this comparison. For this evaluation, we considered a 5-field classification ruleset with 512 rules for all the schemes. For [14], [8] and [17], we scaled the performance to the state-of-the-art technology considering a 18 Mbit TCAM running at 250 MHz consuming 35 W [9]. General observation is that

TABLE II
PERFORMANCE COMPARISON

Approach	Memory req.	Throughput	Power efficiency
TCAM-SSA [17]	13 Bytes/rule	20 Gbps	5150 μW/Gbps
BV-TCAM [14]	82.9 Bytes/rule	80 Gbps	2350 μW/Gbps
FSBV [8]	29 Bytes/rule	100 Gbps	1710 μW/Gbps
StrideBV ($k = 3$)	104 Bytes/rule	393 Gbps	1480 μW/Gbps
StrideBV ($k = 4$)	156 Bytes/rule	407 Gbps	1223 μW/Gbps

even though the TCAM only approach (TCAM-SSA) achieves high memory efficiency, the power consumption is significantly high compared with the SRAM based architectures (BV-TCAM, FSBV, StrideBV).

The memory efficiency of StrideBV is the lowest in Table II due to the parallel pipelined implementation. Note that the other schemes will have worse memory efficiencies when instantiated parallely, to achieve 400 Gbps. Further, StrideBV's memory efficiency can be improved by using small stride size. For example, strides of $1, 2, 3, 4$ and 5 will have memory efficiencies of $26, 26, 34.7, 52$ and 83.2 bytes per rule, respectively, per pipeline. This memory efficiency will come at the cost of increased pipeline length and lower power efficiency. While memory efficiency can be adjusted depending on the requirement, the aforementioned memory efficiencies are guaranteed for a given ruleset. This clearly distinguishes StrideBV from other approaches. For example, the BV-TCAM approach can potentially require $O(N)$ TCAM and FSBV may require $O(w_i)$ TCAM and $O(w_i \times N)$ SRAM storage per field. On the other hand, the storage requirement of StrideBV bounded by $\Theta(N \times W \times 2^k/k)$.

Nevertheless, the throughput and power efficiency achieved by StrideBV is unmatched by existing solutions and its salient characteristic of being independent of ruleset features makes it a unique packet classification solution.

V. CONCLUSION AND FUTURE WORK

We presented *StrideBV*, a packet classification solution that does not depend on the ruleset features and operates at 400G+ throughput levels. To the best of our knowledge, this is the first 400G+ packet classification solution available on a single chip. While achieving high throughput, the proposed solution is, on the average, $2.5\times$ power efficient compared with state-of-the-art solutions. We extended an algorithm proposed earlier as Field-Split Bit-Vector (FSBV) to the complete packet header to achieve these performance improvements. In StrideBV, we introduce *stride* access instead of bit-by-bit inspection of the header to reduce pipeline length of the proposed architecture.

The performance evaluated on Field Programmable Gate Array (FPGA) shows that using larger stride sizes yield better performance while consuming more memory. However, considering the abundant resources available on state-of-the-art FPGA, memory consumption does not become a major concern. We demonstrate this by implementing 4 and 6 parallel pipelines on a single chip. Performance of the proposed architecture is stable for large stride sizes, guaranteeing 400G+ throughput for real-life rulesets.

Note that as the ruleset size increases, the throughput decreases gradually. In such cases, the proposed architecture can be parallelized at fine grained levels to sustain high-performance. Further, ruleset partitioning to alleviate signal routing issues (to improve performance) on FPGA is also possible. For the proposed architecture, it is assumed that a rule can be represented as a ternary string. Hence, in order to support range search, range-to-prefix conversion has to take place. As future work, we intend to include range search capabilities in StrideBV and explore the potential of this scheme to be implemented as a fully parallelized architecture.

REFERENCES

[1] Alcatel-Lucent. Alcatel-lucent fp3 400g network processor. http://www.alcatel-lucent.com/fp3/.

[2] M. Attig and G. 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.

[3] Cisco. Evaluating and enhancing green practices with cisco catalyst switching. http://www.cisco.com.

[4] M. Faezipour and M. Nourani. Wire-speed tcam-based architectures for multimatch packet classification. *Computers, IEEE Transactions on*, 58(1):5 –17, jan. 2009.

[5] P. Gupta and N. 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.

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

[7] G. 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.

[8] W. Jiang and V. 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.

[9] D. Perino and M. Varvello. A reality check for content centric networking. In *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*, ICN '11, pages 44–49, New York, NY, USA, 2011. ACM.

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

[11] K. Scarfone and P. Mell. Guide to intrusion detection and prevention systems (idps). Recommendations of the National Institute of Standards and Technology Special Publication 800-94.

[12] S. Singh, F. Baboescu, G. Varghese, and J. 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.

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

[14] H. Song and J. 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.

[15] Xilinx. Virtex-6 lxt fpgas. http://www.xilinx.com/products/silicon-devices/fpga/virtex-6/lxt.htm.

[16] Xilinx. Xilinx xcell journal. http://www.xilinx.com/publications/xcellonline/.

[17] F. Yu, T. Lakshman, M. Motoyama, and R. Katz. Ssa. In *Architecture for networking and communications systems, 2005. ANCS 2005. Symposium on*, pages 105 –113, oct. 2005.