Algorithmic TCAMs: Implementing Packet Classification Algorithms in Hardware

Pedro Reviriego, Gil Levy, Matty Kadosh and Salvatore Pontarelli

***Abstract*— The adoption of cloud computing and the deployment of new services require more and more computing and networking resources in large datacenters. In particular, there is a need for faster rich featured switches. Today, switching integrated circuits can handle more than 50 Tb/s and are expected to reach 100 Tb/s next year. This trend is likely to continue approaching Petabit capacities in a few years. At the same time, switches need to support more advanced packet classification features. This poses many challenges to the designers of switching integrated circuits creating a need for innovations to move forward. One of those challenges is to support programmable packet classification with large rule sets at wire speed using limited silicon footprint. Unfortunately, traditional solutions like Ternary Content Addressable Memories (TCAMs) have high cost in area and power and therefore, are not a viable option for large on-chip rule sets and new alternatives are needed. In the last two decades, researchers have extensively studied the problem of packet classification proposing many algorithms based on standard memories, mostly targeting software implementations. These packet classification algorithms could be a solution to enable large on-chip classification rule sets using standard memories. However, porting the standard packet classification algorithms to a hardware implementation is not a trivial task, due to the different requirements and the different kinds of memory and computing resources available on the ASIC switching chips. This paper describes a packet classification design tailored for high-speed switching integrated circuits, which is currently used in the NVIDIA® Mellanox® Spectrum® switching ASIC product line, illustrating how to solve the mismatch between the usual software-based classification algorithms and the specific requirements of a hardware implementation.**

***Index Terms*— Packet classification, Switches, ASICs.**

I. Introduction

THE design of state of the art switching chips has always been a challenge as they need to handle very large data rates and support advanced features [1]. In the last decade, the complexity that designers face has increased exponentially due to the introduction of programmability as a requirement for the switches [2] and to the need for larger speeds driven by the hyperscalers. The throughput of a top of the rack switch increased more than tenfold in the last decade and it is expected to keep growing reaching the petabit range during this decade[1]. This happens at a time when the downscaling of the microelectronic technology that has traditionally enabled part of the gains in speed and scale of switching chips is slowing down. This forces designers to innovate at the algorithmic or architectural level to find more efficient solutions [3]. In this context, industry has a need for innovations and a good option would be to leverage the extensive academic research on packet processing carried out in the last two decades.

One of the challenges for switch design is to implement packet classification at line rates for large sets of rules. This has traditionally been done with Ternary Content Addressable Memories (TCAMs) [4]. However, TCAMs have a large cost in terms of silicon footprint and power consumption that limits their scalability to support large tables. TCAMs also lack flexibility as differently from SRAM/DRAM they cannot be used to store other types of data, they can only store rules for matching. This means that it is of interest to minimize the use of TCAMs in programmable switching ASICs. Indeed, for exact match they can be replaced by standard SRAM or DRAM memories using efficient hashing schemes [2],[5]. Unfortunately, those hashing schemes are not applicable to more complex packet classifiers that include don't care bits in the rules on arbitrary positions. Therefore, it would be of interest to implement general packet classifiers that are built on top of the exact matching hashing implementations. This approach would enable both scalability, since standard memories are cheaper than TCAM, and flexibility since the same memory can be used both for exact matching, generic data storage, and for packet classification. For this, packet classification algorithms developed for software implementations could be a starting point. However, there are major differences between hardware and software implementations, and an efficient solution can be developed only after a detailed analysis of the impact of the implementation of packet classification algorithms in hardware. In our case, this study led to propose a modified packet classification algorithm that is well suited for a hardware implementation highly compatible with a high-end ASIC switching chip.

The proposed design efficiently combines on-chip SRAMs and TCAM to efficiently support the matching of general rules which is a significant improvement over existing architectures that use SRAM only for exact match rules while all other rules have to be stored in the TCAM. The proposed hybrid SRAM/TCAM architecture has also several novel optimizations that have enabled reaching 51.2 Tb/s of throughput on a single device while supporting up to 512K packet classification rules.

The following section presents an overview of existing packet classification algorithms proposed by academia discussing the challenges and limitations when trying to use them for silicon implementation. Then in section III, the proposed architecture is described highlighting the

[1]https://elegantnetwork.github.io/posts/A-Summary-of-Network-ASICs/

modifications introduced in the algorithms to make them efficient for hardware implementation. The section also provides some evaluation results to show the effectiveness of the proposed scheme. Finally, the conclusions are presented in section IV.

II. PACKET CLASSIFICATION ALGORITHMS

A. *Existing Solutions*

Most of the proposed packet classification algorithms focus on software implementations running on commodity processors. This is due several reasons. The first one is a disconnect between the networking and the microelectronics research communities in academia. It is unusual for both faculty and students to be knowledgeable in both areas. A second reason is that evaluating algorithms in software can be done with standard computing equipment while evaluation on silicon requires access to Electronic Design Automation (EDA) tools, ASIC libraries and ultimately fabrication and testing of the device. This further reduces the ability of students or small university groups to evaluate silicon implementations of networking algorithms. A third reason is that algorithms implemented in software can be easily integrated in existing packet processing frameworks like DPDK or VPP [6] and used by the community. This is not yet possible for silicon implementations. There are initiatives to promote open hardware, but they are still in their infancy, and it remains to be seen if they will ultimately succeed. A notable exception is the NetFPGA initiative that has successfully linked networking with FPGAs [7]. However, even in this case, the number of packet classification algorithms mapped to NetFPGA is very small. Finally, the brute-force TCAM based hardware approach provides very good performance in terms of throughput and support for rules with wildcards, which was difficult to achieve with algorithmic approaches. However, in the last few years the limitations related to scalability and power consumption have driven the development of packet classification algorithms in hardware.

Problem Statement: the goal of packet classification is to classify packets by applying a set of rules to the header fields of a packet. Each rule R_i has n components, and the j-th component of the rule, referred to as $R_{i,j}$ is a match expression (i.e. an exact value, prefix or range matching) on the j-th header field. A packet p matches a rule R_i if all the n header fields of p match the corresponding $R_{i,j}$. If a packet p matches multiple rules, the matching rule with the highest priority is returned. Table 1 presents an example of a packet classifier with n=2 header fields and Figure 1 depicts the geometric view of the ruleset. Two rules are compatible or overlap, when there are packets that can match both rules. For example, a packet with field values [1111,1001] would match both R_1 and R_3. Instead, other rules do not overlap like R_1 and R_2 and no packet can match both. Therefore, after matching, the search can be restricted to the rules that overlap with the matched rule and have higher priority. For each rule, the compatible rules with higher priority can be precomputed and when there are no such rules, a match on that rule is final and there is no need to compare with other rules.

Packet classification is a hard problem to solve in its generic form, thus researchers have tried to exploit the features of real-life classifiers to reduce the complexity of the problem, proposing a wide range of solutions. The most used metrics to evaluate the performance of packet classification algorithms are [8]:

- Throughput: the number of packets per second that the algorithm can classify. The target for the ASIC implementation is above 1 billion of packets per second. The throughput mainly depends on the number of memory accesses needed per packet lookup.
- Memory footprint: the ruleset representation in memory depends on the classification algorithm used. Since high throughputs require fast on-chip SRAM memories, and these memories cost silicon area, this is an important metric for an ASIC implementation.
- Update speed: a change in the ruleset requires to update the data structure. Depending on the algorithm, this can correspond to the update of a few entries or to the reconstruction of the whole data structure. The latter kind of algorithms has update times that are significantly higher than the former. Since we target fast updates, we need a classification algorithm that does not require data structure reconstruction for updates.

Table 1: example of a 2-field classifier

Rule	Field 1	Field 2
R_1	111*	*
R_2	110*	*
R_3	*	1001
R_4	01**	10**

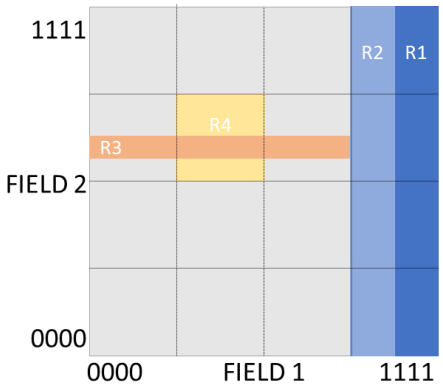

Figure 1. Geometric representation of the classifier of Table 1

A widely accepted categorization of packet classification algorithms is to split them in decision tree-based and partition-based [9],[10]. These algorithms achieve different performances with respect to the above-mentioned metrics.

Decision tree-based schemes. Algorithms like HyperCut [11] and SmartSplit [12] classify a packet recursively cutting the n-dimensional space of the ruleset to obtain a small subset of rules compatible with the packet to classify. These subsets correspond to the leaves of the tree. Therefore, the tree traversal operation selects a small subset of rules, which are tested against the incoming packet, and the matching rule with the

highest priority is selected as output. The performances of these algorithms are excellent both in terms of throughput and memory footprint, but they cannot support fast updates since the insertion of a new rule requires a costly tree update [10]. This is a sufficient motivation to exclude these algorithms as candidates for a hardware implementation. The complexity of realizing a tree-based structure in a system with a fixed memory allocation (i.e. it is hard to manage the pointers composing a tree in an ASIC implementation) further suggested to avoid these methods for our target implementation.

Partition based schemes. These algorithms split a ruleset into multiple subsets. For each subset, a bitmask (also called rule pattern) is used to translate the generic rule match expression into an exact match lookup. The first work proposing this approach was Tuple Space Search (TSS) [13]. With this algorithm, the lookup requires to check all the subsets and selects the matching one with higher priority. Instead, the update operation requires only an insertion in one subset. Taking as example the classifier of Table 1, TSS creates 3 subsets ($\{R_1,R_2\},\{R_3\},\{R_4\}$) with masks ([1110,0000], [0000,1111],[1100,1100]). This permits to store the following masked rules {[1110,0000],[1100,0000]},{[0000,1001]} {[0100,1000]} in the three tables of TSS. Since the number of subsets can be high, several algorithms have been proposed to improve the throughput of partition-based schemes. As a complete survey of these algorithms is out of the scope of this paper, here we just mention TupleMerge (TM) [10], which not only represents one of the state-of-the-art algorithms for packet classification but is also related to the ASIC implementation that we developed[2]. The idea of TM is to merge subsets with different bitmasks in the same table. The bitmasks share some common bits that are used to select the table in which the rules are stored. While this approach reduces the number of tables, and therefore increases the throughput, it has as drawback the so-called "collision problem". A collision occurs when two rules with different bitmasks are stored in the same table. A search in this table is no more a pure exact match since the rule pattern does not mask all the don't care bits. Taking again the example of Table 1, a possible merge is between R_3 and R_4 applying as mask [0000,1100]. This merge also creates a collision. Suppose that a packet p=[0101,**10**01] arrives. The masked bits to access to the ruleset are [0000,**10**00], and both R_3 and R_4 match this subset. Therefore, only comparing the original packet p with both R_3 and R_4 it is possible to select the final rule. This means that several rules may have to be checked on a single subset.

B. Differences and challenges for silicon implementation

As discussed in the previous subsection, most algorithms proposed by academia target a software implementation. Therefore, the throughput supported is that of one or a few a NIC cards. This can be today at most around 1 Tb/s. Instead, a silicon implementation for a switch must support the traffic of many ports thus requiring a throughput that is orders of magnitude larger. For example, close to 100 Tb/s. Therefore, the performance requirements are completely different and indeed more challenging.

[2] We developed and patented our algorithm before TM, but for sake of clarity we compare it with this algorithm as it is the closest to our work.

Another important difference is that on a server, DRAM memory is abundant. Instead, in many switching ASIC designs, no external memory is used and only the one on-chip is available. This severely reduces the memory available that should be allocated and used very carefully. On the other hand, an ASIC implementation provides more flexibility to instantiate several memories that can be accessed in parallel whereas on a server, the memory configuration is given. Similarly, on an ASIC we can also use TCAM blocks if needed, something that is not possible on a commodity server.

These significant differences in performance and in the underlying hardware architecture imply that algorithms targeted to a software implementation will not be optimal for an ASIC implementation. Therefore, transferring the ideas and algorithms to switching ASICs is not straightforward and will benefit from the collaboration of industry and academia. The main contribution of this paper is to present a scalable packet classification algorithm that is suitable for ASIC implementation and that has been implemented and is currently in used in the NVIDIA® Mellanox® Spectrum® switching ASIC product line.

III. PROPOSED ALGORITHM AND IMPLEMENTATION

A. Architecture

Since exact match lookups are needed for example for Ethernet switching, and as discussed before they can be efficiently implemented using standard memories, it would be beneficial to build the packet classification engine on top of that. That is, use an algorithm that performs several exact matches, like for example Tuple Space Search (TSS) [13]. However, to support large throughputs, one of the goals of the design is to complete the classification of a packet in a few exact matches. For example, an average value that does not exceed four and a worst case that does not exceed eight. This limits the applicability of most existing software algorithms. For example, in TSS the worst case can be significantly larger than eight as we can easily have more than one hundred rule patterns in large rule sets. Furthermore, an algorithm derived from TSS also provides fast insertion times, since the insertion of rules only requires an insertion in a hash table.

To achieve this design goal and provide an efficient silicon implementation, an architecture that combines a small TCAM with an optimized version of the TSS algorithm has been designed. The overall architecture is illustrated in Figure 2. It can be seen that it is formed by two main blocks: a hash based exact match engine and a TCAM block. The exact match engine implements an optimized version of TSS for the patterns that have more rules while the ones with few rules are stored in the TCAM. This enables us to keep the number of accesses to the exact match engine low while storing most of the rules in this engine and thus on standard memories. Instead, the least frequent patterns are stored in the more costly TCAM. Those account for the majority of the patterns but only for a small fraction of the rules. Therefore, by placing them on a small TCAM, the overall design is much faster than that of a traditional TSS and the TCAM cost is kept low. Note that this would not be possible in a computer as there are not TCAM blocks, and only standard memories can be used.

In our proposed architecture, packets are first checked in the exact match engine and if a match that is final (there are no overlapping rules with higher priority) is found, there is no need to check the TCAM. Instead, if no match is found or a match is found but there can be a higher priority match on the TCAM, it is subsequently checked. This reduces the bandwidth needed on the TCAM as only a fraction of the packets would access it reducing power consumption and enabling further optimizations and cost reductions in the TCAM design.

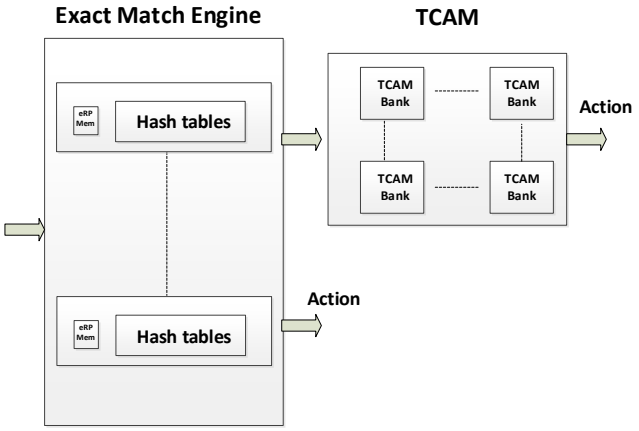

Figure 2. Proposed architecture for packet classification

Unfortunately, this hybrid architecture may still need too many accesses to store most of the rules on the exact match engine (thus reducing the size of the TCAM). Therefore, the TSS algorithm is extended to cover more rules on each exact match lookup. This is done by introducing the concept of extended Rule Pattern (eRP). The idea is to cover several patterns on a single exact match lookup, in a way similar to the approach proposed in TupleMerge [10]. However, differently from TupleMerge, we merge only patterns rules that differ only in a few bits. This choice has two benefits: (i) it limits the collision effect and (ii) permits to represent the rule inside the hash table with a limited memory overhead. As a simple example, let us consider a rule pattern formed by the destination IPv4 address with a /16 mask and a second one that corresponds to the IPv4 address with a /18 mask. Then both can be stored by hashing on the /16 bits and adding a delta to cover the last two bits. This delta is an additional field in the entry that extends the data to be compared beyond the mask used for the hashing. The delta, composed by an offset and a few bits mask, permits to merge several RP in the same eRP while requiring a very small additional memory footprint (less than 2 bytes). This approach enables us to cover more rules with fewer exact match lookups. The downside is that the number of hash collisions may increase and needs to be adequately managed by the exact match engine implementation. This can be easily done when the number of collisions is limited as discussed in [5],[10].

The concept of the eRP is illustrated in Figure 3. In this case, rules have only eight bits and we have three patterns for the rules, where 'u' are bits that are either 0 or 1 in the rule and 'x' are bits that are wildcards. For example, a rule like '101xxx00' would correspond to rule pattern 1 (RP_1) and instead rule '0010xx00' to rule pattern 2 (RP_2). In the traditional TSS algorithm, each of the three rule patterns would require an exact match lookup. Instead, in our implementation, rule patterns that are different only on a few bits are merged into a single extended rule pattern and checked with a single exact match lookup. For example, in the Figure, the three rule patterns are merged into a single eRP. This is done by creating an eRP mask that has an 'x' on every bit for which at least one of the RPs that form the eRP has an 'x'. That mask is then used for the exact match hashing as illustrated also in the Figure by setting those 'x' bits to zero before computing the hash. The logic needed to compare the key with the stored entries has to be extended to cover the different rule patterns but that is something that can be done at low cost in a silicon implementation. The use of eRPs can significantly reduce the number of exact match lookups needed as will be seen in the following.

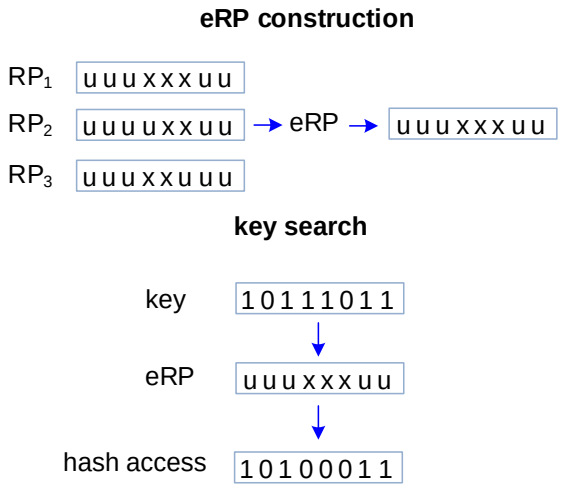

Figure 3. Illustration of the concept of extended Rule Patterns (eRPs). The 'x' corresponds to wildcard bits and the 'u' to bits that are either 0 or 1 in the rule

A number of optimizations were implemented to further improve performance. Here we describe only two of them.

Early stop: The first optimization exploits the observation that in most of the examined rulesets most of the rules are not overlapping. This means that a match in an eRP can stop the search since there are not compatible rules with higher priority. This information can be used for example to minimize the number of eRPs checked for each packet lookup. In fact, the developed system can configure the order in which the eRPs are checked so that the eRPs that are more frequently matched are placed first. This can be done during operation by measuring the number of hits per eRP and adjusting the order based on the results.

Pruning: The second optimization analyzes for each rule, which are the compatible eRPs (i.e. the ones with overlapping rules with higher priority) and stores this information together with the rule as a vector of compatible eRPs. When a match is found on a rule, the engine knows on which eRPs there can be rules with higher priority and can use this information to prune the eRPs that need to be checked after the match. These optimizations enabled significant reductions in the average number of eRPs checked per packet.

File	*Number of rules*	*Number of Rule Patterns*	*1 eRP*	*2 eRPs*	*3 eRPs*	*4 eRPs*	*8 eRPs*
acl1	39964	412	52.55%	65.44%	75.40%	79.39%	87.62%
acl2	39807	403	26.66%	35.39%	45.68%	52.62%	69.75%
acl3	39654	458	27.44%	39.76%	48.87%	57.98%	71.12%
acl4	27513	178	42.68%	52.98%	63.51%	71.80%	83.70%
fw1	38975	504	36.06%	45.34%	57.32%	64.09%	80.14%
fw2	39480	135	20.06%	39.54%	44.71%	48.98%	71.63%
fw3	38268	417	27.97%	47.15%	58.40%	66.02%	84.14%
fw4	37529	139	25.78%	39.40%	49.76%	54.81%	86.78%
fw5	37952	487	27.33%	38.93%	55.31%	65.70%	77.16%
ipc1	40004	43	28.03%	50.93%	69.85%	73.74%	92.99%

Table 2. Classbench generated rule databases and extended rule pattern coverage in percentage

B. Performance Evaluation

Rule coverage. To evaluate the benefits of the proposed packet classification architecture, it was evaluated both using rule sets generated with Classbench [14] and with real rulesets deployed in the field by customers and made available for this project by non-disclosure agreements (NDAs). Here we restrict the discussion to the Classbench rulesets to comply with the NDAs. Classbench generates packet classification databases that should resemble typical real packet classification databases. In more detail, Classbench generates databases that correspond to:

- Access Control List (ACL): standard format for security, VPN, and NAT filters for firewalls and routers (enterprise, edge, and backbone).
- Firewall (FW): proprietary format for specifying security filters for firewalls.
- IP Chain (IPC): decision tree format for security, VPN, and NAT filters for software-based systems.

Using Classbench a set of 10 databases were generated with 30 to 40 thousand rules. The rules generated by Classbench have the following format: Source IP, Destination IP, Source Port, Destination Port, Protocol, Flags. This corresponds to the classical five tuple extended with 16-bit flags making a total of 120 bits per rule. The flag bits model for example TCP flags that are used to determine if a packet is the start of a connection. The details of the generated rule sets are summarized in Table 2. This includes the number of rules and rule patterns. It can be seen that the number of rule patterns is in most cases larger than a hundred. This number of exact match lookups per packet would be clearly unacceptable for our target silicon design. The table also shows the percentage of rules covered when using different numbers of eRPs. It can be seen that with 8 eRPs most rules are covered. Those would be stored in the exact match engine. Therefore, a compromise can be made between the number of exact match lookups needed per packet and coverage (that determines the TCAM size). In particular, it seems that using a TCAM of size ¼ that of the rule set can be sufficient in most cases. This is a very significant reduction that enables a scaling of the rule sets that can be supported. If further reductions on the TCAM size are needed, a larger number of eRPs can be used at the cost of increasing the number of exact match lookups per packet. Therefore, by implementing a generic architecture that supports a programmable number of eRPs, different trade-offs can be made depending on the rule set.

Finally, it is important to note that the proposed packet classification algorithm is used by the NVIDIA® Mellanox® Spectrum® switching ASIC family since its second generation and is deployed today in many datacenters and enterprise environments. In most of those deployments, the rule coverage is better than the one reported in Table 2 for the synthetic Classbench databases, thus proving that in practical systems rules tend to have a regular structure that can be exploited by the proposed architecture.

Update speed. Adding or removing a rule stored in an eRP requires simply accessing the corresponding table and performing an insertion removal on it. In our implementation cuckoo hash tables are used [5] that support more than ten thousand updates per second. This is sufficient in most applications as rule changes are not frequent. Adding or removing eRPs is more complex but it can be done gradually while the switch continues to operate normally and thus is not critical. For example, when creating a new eRP, rules from the TCAM can be moved one by one to the new eRP and lookups will still match the rules regardless of whether they are already in the new eRP or still in the TCAM.

Throughput. The time needed to process a packet depends on the number of eRPs that have to be checked. The search order is optimized so that eRPs with more packet matches are searched first. The search order is thus adapted to the traffic but changing the order only requires updating the early stop and pruning vectors of the rules which can be easily computed. On the other hand, making the search order traffic aware reduces the number of checks significantly. As an example, the results for the same databases are summarized in Table 3 when using 8 eRPs and a random packet per rule. It can be observed that significant reductions are achieved by using the frequency-based search order versus the expected 4.5 eRP checks of a non-optimized

search order. Further reductions can be achieved with the two optimizations (early stop and pruning) used in the proposed design, so that the average number of eRPs checked is closer to one.

The throughput of the hash tables can be scaled by using multiple banks. Instead, that is not easily done for the TCAM which can become the bottleneck of the system. The absolute value of the throughput depends on many factors such as the packet size, the technology node of the ASIC, the speed of the memories, or the optimizations used. However, to give an idea of the scalability of the proposed architecture, it has been implemented on the Spectrum-4 device with a 51.2 Tb/s throughput.

File	***Average Number of eRPs checked***
acl1	2.6
acl2	2.5
acl3	2.5
acl4	2.5
fw1	2.7
fw2	3.1
fw3	3.1
fw4	2.8
fw5	3.1
ipc1	2.7

Table 3. Average number of eRPs checked per packet when using a traffic aware search order

Circuit area and power. Another important metric is the overhead needed in the exact match engine to support eRPs. This includes the additional logic for matching entries against all the rule patterns that form the eRP and also the management of the collisions in the hash created by rules that map to the same hash value on different eRPs. The architecture described has been implemented in several advanced technology nodes and the additional circuit area was negligible compared to the plain exact match engine. In fact, this small overhead was completely outweighed by the reduction in TCAM size, and the additional flexibility provided by the proposed architecture. In particular, for the databases considered of approximately forty thousand rules, the area and power of the proposed architecture using a ten thousand entries TCAM and a thirty thousand entries SRAM reduces the area and power by a factor of two approximately compared to a forty thousand entries TCAM[3].

Scalability. The use of the proposed algorithmic TCAM enables switching ASICs to support a much larger number of TCAM-like rules, especially when the rules have a regular structure with most concentrated on a few eRPs. This is commonly the case in datacenters. In those scenarios, scaling factors of 10x are easily achieved.

[3] Further details cannot be provided due to confidentiality restrictions.

IV. Conclusion and future work

In this paper, we have described a packet classification architecture for high-speed silicon implementation. The design process was driven by the need to support large rule sets and storing most of the rules in standard memories while supporting very high speeds. To achieve this, innovations were introduced and optimized for implementation resulting in a hybrid solution that stores most rules in standard memories using hash based exact matching and the rest on a small TCAM. The exact matching engine is augmented so that each lookup can cover more rules than in existing implementations. The evaluation results over different packet classification rule sets show that the proposed architecture can indeed store most rules in standard memories and complete packet classification with a small number of exact match lookups. Additionally, the overhead introduced in terms of silicon area over a plain exact match implementation is very small. The end-result is an architecture that has introduced an unprecedented scale in terms of the number of packet classification rules that can be supported on-chip in switching ASICs and that is currently deployed in many datacenters.

References

[1] R. Rojas-Cessa, “Interconnections for Computer Communications and Packet Networks,” CRC Press. 2017.

[2] P. Bosshart, G. Gibb, H. S, Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, “Forwarding metamorphosis: fast programmable match-action processing in hardware for SDN,” in Proc. of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM), pp. 99-110, 2013.

[3] J. L. Hennessy and D. A. Patterson, “A New Golden Age for Computer Architecture,” in Communications of the ACM 62, 2, pp. 48–60, 2019.

[4] F. Yu, R.H. Katz and T.V. Lakshman, “Efficient multimatch packet classification and lookup with TCAM,” IEEE Micro, vol. 25, no. 1, pp. 50-59, 2005.

[5] G. Levy, S. Pontarelli and P. Reviriego, "Flexible Packet Matching with Single Double Cuckoo Hash," in IEEE Communications Magazine, vol. 55, no. 6, pp. 212-217, June 2017.

[6] D. Barach, et al. "High-speed software data plane via vectorized packet processing." IEEE Communications Magazine 56.12 (2018): 97-103.

[7] N. Zilberman, Y. Audzevich, G. Covington and A. Moore, “NetFPGA SUME: Toward 100 Gbps as research commodity,” IEEE Micro, vol. 34, pp. 32-41, 2014.

[8] P. Gupta and N. McKeown. "Algorithms for packet classification," IEEE Network 15.2, 2001, pp. 24-32.

[9] D. E Taylor. 2005. Survey and taxonomy of packet classification techniques. ACM Computing Surveys (CSUR) 37, 3 (2005), 238–275.

[10] J. Daly et al. “Tuplemerge: Fast software packet processing for online packet classification,” IEEE/ACM transactions on networking 27.4, pp. 1417-1431, 2019.

[11] S. Singh, et al. "Packet classification using multidimensional cutting." Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communications. 2003.

[12] P. He et al. “Meta-algorithms for software-based packet classification,” 2014 IEEE 22nd International Conference on Network Protocols. IEEE, 2014.
[13] V. Srinivasan, S. Suri and G. Varghese "Packet classification using tuple space search," Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication. 1999.
[14] D.E. Taylor and J.S. Turner "ClassBench: a packet classification benchmark," INFOCOM, pp.2068,2079 vol. 3, 13-17 March 2005.

BIOGRAPHIES

Pedro Reviriego (revirieg@it.uc3m.es) holds a master and a Ph.D degrees in telecommunications engineering, both from the Universidad Politécnica de Madrid. He is currently with the Universidad Carlos III de Madrid. He previously worked for LSI Corporation (now part of Broadcom) on the development of Ethernet transceivers, and for Teldat implementing routers and switches. His current research interests are high speed packet processing and probabilistic data structures.

Gil Levy (gill@nvidia.com) holds a BSCE in Electrical engineering from the Tel-Aviv University. He is currently working at Nvidia on Ethernet switches architecture. He previously worked for Marvell and Broadlight (now Broadcom) on the development of Ethernet switches and network processors for metro and data center. His focus is high-speed packet processing probabilistic data structures, packet buffering, and telemetry.

Matty Kadosh is a principal architect at Nvidia. His areas of interest range from IP networking, Network programmability, HW acceleration, switch and NIC silicon HW architecture, P4 and Linux kernel. He worked in the development and architecture of the NOS and drivers of several Nvidia products and in particular on the last generation Nvidia switch and NIC ASICs. Matty Kadosh holds B.Sc in Computer Science from the University of Haifa, Israel.

Salvatore Pontarelli (salvatore.pontarelli@uniroma1.it) is currently Tenure-track Assistant Professor at Department of Computer Science, Sapienza University of Rome. Before joining Sapienza, he worked as senior researcher with the CNIT, the National Inter-University Consortium for Telecommunications. He also held several research positions at Telecom Paristech, University of Bristol, Italian Space Agency and University of Rome Tor Vergata. His research interest focuses on the design of high-speed hardware architectures for programmable network devices.