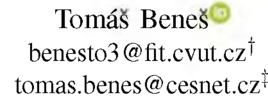
2021 24th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS) | 978-1-6654-3595-6/20/$31.00 ©2021 IEEE | DOI: 10.1109/DDECS52668.2021.9417060

High-speed stateful packet classifier based on TSS algorithm optimized for off-chip memories



Michal Orsák
iorsak@fit.vutbr.cz*
orsak@cesnet.cz‡

Tomáš Beneš
benesto3@fit.cvut.cz†
tomas.benes@cesnet.cz‡

*Faculty of Information Technology, Brno University of Technology, Brno, Czech Republic

†Faculty of Information Technology, Czech Technical University, Prague, Czech Republic

‡CESNET, a.l.e., Prague, Czech Republic

***Abstract*—We present a modular out-of-order architecture for stateful packet classification. The architecture uses DDR4 SDRAM memory to store rules and their state information to support millions of rules. The memory access pattern generated by network traffic significantly degrades the performance of the DDR4. Our architecture contains a cache and aggregation queues to negate this effect. Additionally, the memory subsystem supports a read cancellation and uses an out-of-order pipeline to maximize the main memory's effectiveness further. The rule set update is implemented as a non-blocking operation and can be interleaved with lookup operations without any performance decrease, leading to the same execution time for rule update and rule lookup. The architecture is optimized for the modern data-center's network traffic and a small on-chip memory footprint, making it suitable as an accelerator for the Open vSwitch. As a result, our novel architecture configured with 1 million exact match rules can process traffic up to 202 Gbit/s (300 Mp/s) in average case and 51 Gbit/s (76 Mp/s) in the worst case with the use of a common dual-channel 64 bit DDR4-2666 MHz. It uses fewer FPGA resources (excluding cache memory) than the well-known de facto industry standard Xilinx MIG DDR4 controllers. Our proposed architecture enables commodity FPGA cards commonly equipped with DDR4 to process 100 Gbit/s which results in a significant cost reduction of a 100G SmartNICs.**

***Index Terms*—Open vSwitch, OpenFlow, networking, TSS, external memory, 100G, FPGA, SmartNIC, Out-of-Order, cache, LSU, packet classfication**

I. INTRODUCTION

Complexity and flexibility of the modern networks used, e.g., in data centers, were first introduced by a Software-Defined Networking (SDN) concept. The concept is based on highly configurable devices controlled and configured by an external network controller, usually using OpenFlow (OF) protocol [1]. The OF specifies 47 packet header fields, which can be matched in up to 255 tables by rules with up to $2^{32}-1$ priorities [2]. As the rules might overlap, the potential match complexity can be theoretically enormous. An OF rule also contains per-rule statistics and per-rule actions, which need to be updated/performed for each packet match. Additionally, the rule update rate is much faster than traditional core networks as rule-set changes with every reconfiguration of any service in the network.

978-1-6654-3595-6/21/$31.00 ©2021 IEEE

Open vSwitch (OvS) [3] is an open-source reference and a widely used software switch supporting SDN concept. However, for a 100 Gbit/s networks, it requires at least 16 cores [4] and causes cache spoiling and thus largely degrades the machine's performance.

The OvS utilizes an OpenFlow classifier and two, optionally three levels of software-based caching strategy. The lowest OvS cache level is called Exact Match Cache (EMC), which caches fully expanded rules. The EMC's size is usually set to match the CPU's L1 cache's size and may contain approximately hundreds of items. The second level of cache is called Mega Flow Cache (MFC), and it caches the cross produced OpenFlow rules. The size of this cache is also configurable, and it usually contains tens of hundreds of records in a simple setup. The accelerator API in OvS allows the accelerator to be connected as an EMC or MFC. The EMC is a table with a single key mask, and MFC supports multiple key masks and priorities. The offload of MFC results in fewer up-calls to software as a rule in MFC cover more potential variants of a record for EMC.

For OvS applications, the accelerator needs to store only some rules as it works as a cache. However, the memory capacity requirements are higher than the available FPGAs' internal resources. Additionally, it is also necessary to use some additional internal memory resources for the classification process, such as rule statistics.

We have analyzed data-sets of OvS rules from large data centers and identified minimal requirements of 1 million OF rules. Even though external memories (DDR) can solve the capacity issue, they have many characteristics that complicate their usage. The external memory bandwidth significantly degrades with a random access pattern, and it has a large and non-deterministic latency. Therefore, it is challenging to design a network packet processing architecture, which uses external DDR memories.

This paper presents a novel architecture based on FPGA technology that solves the issues with external memories. Therefore, it provides enough memory capacity (for about 2 millions rules including the counters) with high-performance processing (up to 202 Gbit/s) that is required in a real environment. The proposed architecture can significantly accelerate

Authorized licensed use limited to: Rutgers University. Downloaded on May 17,2021 at 02:38:26 UTC from IEEE Xplore. Restrictions apply.

the OvS application, enabling the deployment of a 100 Gbit/s networking and potential cost reductions.

II. Requirements

Computer networks generally differ in their configurations and traffic. This variability affects requirements on the system design. We have explored existing solutions and analyzed available SDN configurations from real data-centers to estimate the optimal solution for a typical target environment. The OpenFlow matching scheme is designed for flexibility. The current version of OpenFlow (1.5.1) does support 255 tables, each rule has 32b priority, and the matching key may contain up to 47 fields. The complexity of the rule match and the overall number of rules depend on the chosen schema and the network infrastructure. Although there are recommendations on the design of OpenFlow tables [5], the table schema is nearly always a user-specific. Therefore we have reached an large data center operators VMware, Inc and eBay, Inc for their OpenFlow dumps. Based on the provided data from 2015-2020 and example configuration for various data-center software, we identified the four configuration categories, which are shown in Table I. Event thought the number of rules might seem low, each rule may be translated to multiple rules for an accelerator resulting even in several order of magnitude difference.

One of the critical requirement is the maximal network speed supported by the accelerator. However, its performance is measured in packets-per-seconds (PPS). The transition from PPS to network throughput depends on the packet sizes in the deployment. The accelerator used in a data center with larger packets (such as multimedia content streaming) would naturally require lower performance than the datacenter with mainly shorter ones. The Benson et al. [6] focused on the traffic patterns inside the datacenters. Their result shows that the network packet sizes follow a heavy-tail distribution with peaks at 200 B and 1440 B. Therefore, we can assume that the average packet length falls into the interval of the mentioned peaks.

In general, there are four assumptions that our architecture must fulfil to be worth enough compared to the existing solutions:

1) **Packet classifier must be capable of handling at least 100 Gbit/s** at half of the average packet size in the worst case. The current software solution capable of such speed requires at least 16 CPU cores. We estimate that a reasonable CPU load would use maximally two cores.
2) **The system must support frequent incremental rule-set update**, because the longer update significantly decreases overall performance of the Open vSwitch caches.
3) **The rule statistic update collisions must not cause drops in throughput**, because such a events are common in real deployment.
4) **The system must support at least 1 million OpenFlow rules.** This and previous requirements makes use of the traditional hardware architectures for a packet classification infeasible because they rely on on-chip memory which does not have sufficient capacity.

III. Related work

The packet classification is a well-explored area that can be solved with multiple different approaches. Generally, there are four types of algorithms: **1)** dimensional decomposition (such as HSM [7], and BV algorithm [8]), **2)** geometric space partition (such as Efficuts [9] or HyperSplit [10] algorithm), **3)** Trie tree structures (such as Quad-Trie algorithm [11]), and **4)** the predefined filter matching (such as TSS algorithm [12]). Unfortunately, almost no algorithm satisfies the requirements presented in Section II. The algorithms usually require a large number of memory accesses, which precludes their implementation with external memory. Or the principle of algorithm functionality does not allow fast or incremental updates of a rule-set. Among the four presented approaches, only the filter matching based algorithms are suitable for our use-case.

The acceleration of the filter matching based classification algorithm is usually targeted as a part of a software implementation of a virtual switch. Wanf et al. [13] used a Cuckoo hashing principles [14] and achieved a 3.5 times improvement in throughput compared to the previous solution implemented in the openvswitch.

Tseng et al. [15] focused on accelerating the classification by offloading the search algorithm to GPU. By balancing between GPU and CPU, they achieved three times higher throughput of OvS than the optimized CPU only implementation. Similarly, Qiu et al. [16] proposed a GFlow algorithm for GPU based acceleration.

However, all software-based implementations require significant CPU resources for reaching 100 Gbit/s (more than 16 cores in a dpdk based OvS [4]). Allocation of such a high amount of resources only for network stack is expensive; therefore research community focused also on hardware acceleration of packet classification. These architectures could (under certain circumstances) even exceed our defined 100 Gbit/s threshold [17–19]. However, all of the mentioned papers achieve the speed for the price of using only the internal FPGA resources (Bloom Filters, a pipeline of unwrapped decision trees), which unfortunately are very limited and do not support large rule-set.

The support of a large number of rules requires the usage of external memory. Nonetheless, the throughput of external memories usually limits the whole architecture; thus, it does not reach the 100 Gbit/s limit [20, 21]. Unfortunately, the architectures that do achieve 100 Gbit/s [22–24] do not support a fast update of a rule-set. The updates cause a pause in their functionality in order of seconds, which is unthinkable in a production environment. Besides, these architectures require SRAM, TCAM memories, which are more expensive than DRAM. Therefore, we are not aware of any architecture that clams to meet all requirements defined in Section II, which ensures the architecture applicability in modern data centers production environment.

IV. Our architecture

In order to satisfy the requirements mentioned in Section II we designed an out-of-order hardware architecture for an TSS

Authorized licensed use limited to: Rutgers University. Downloaded on May 17,2021 at 02:38:26 UTC from IEEE Xplore. Restrictions apply.

TABLE I: Parameters of real word rule-sets

Category name	Number of rules	Number of OF tables	OF table pass-through graph	Match complexity
Generic ACL	10K-1M	eta 30	Linear pass-through	Mostly exact match
VM Hypervisor	1K - 10K	<30	Small tree	LPM for L4 ports
Service Hypervisor	10K	<30	Small tree	Exact match
Network monitor	30K - 50K	eta 100	DAG with high number of edges	LPM anywhere

classification algorithm utilizing FPGA with an external DDR4 memory. The following sections describe the most important aspects that had to be considered.

A. *External Memory*

The DDR4 SDRAM is commonly present on cheap commodity FPGA cards, and the usage of this type of memory is essential for a supporting millions of rules on such a cards. These memories provide gigabytes of a memory space and high throughput (64b DDR4-2666 170 Gbit/s). However, it has a relatively large latency (70–100 cycles [25]). Additionally, the random access pattern together with switching between read and write mode degrades the throughput to a 24 % [25]. Unfortunately, the hash table lookup does have a random access patterns, and the content maintenance is also a read-modify-write operation. This requires either using multiple memory channels for a single 100G Ethernet port or integrating a large enough cache to filter pattern repetitions. Flow-based traffic analysis [26] suggests that the window between packets in a flow can be overlapped by a cache with a high probability for achievable cache size.

Storing of stateful rule information on-chip can eliminate the need for the frequent write to a memory, which significantly improves the overall performance, and just two DDR4-2666 memory channels are required to satisfy the packet-rate of 100G Ethernet in the worst case. If the rule stateful information is stored in DDR, the four DDR4-2666 channels are required.

B. *Classification Algorithm*

Our architecture implements packet classification algorithm **Tuple Space Search** (TSS) [12]. It has a small number of memory accesses for the classification of SDN-like traffic and a very fast update. It is using hash tables that are easily implemented in hardware architecture. The other alternatives either require a large number of memory operations for its operation or do not support incremental update.

The TSS uses a list of hash tables sorted by the maximum rule priority. Each table is associate with a rule mask, which is used for a key comparison and hashing. The lookup starts at the first table and ends once the next table may not contain the rule with larger priority, or there is no other table. The rule with the highest priority is returned as a result of the lookup operation.

C. *Hashing options*

The primary parameters of a hash table data structure is a hashing scheme and a hash function. The hashing schemes with a high memory efficiency like Cuckoo hashing or Double hashing are traditionally used for on-chip memories where the memory size is limited. In our application, we need to pay close attention to the number of memory transactions as the main limitation. The more memory efficient hash schemes require more memory accesses (e.g. direct hashing [27] vs Cuckoo hashing [14]). The collision probability of direct hashing is approximated by equation $P_{collision} = 1 - e^{(\frac{-k(k-1)}{2N})}$ where k is number of occupied items and N is a total size of the table. That implies that the probability of hash collision for a 1 GB of memory filled with a 147 thousands of rules is 50 %. The memory of hash tables can be shared between the hash tables if the key stored in a hash table is extended with a unique id. Sharing table memories results in a more uniform memory load and better overall memory efficiency in the multi-table scenario. Note that the small on-chip memories may still be utilized to reduce the problem of hash collisions. In our architecture, the hash function is used to distribute rules in the main memory, and it is unrelated to the efficiency of the cache. The work of Hua [28] suggests that the CRC-32 hash function also works for network flows and has efficient hardware implementation as it is LFSR based. Due to the previously mentioned reasons, we decide to use CRC-32 as a main hash function.

D. *Main parts of our architecture*

Our architecture implements packet classification using external memories and TSS algorithm with a per rule statistical counters stored inside the external memory. The architecture is configured for a OvS use-case, where the mostly used rules are offloaded into an accelerator. The rest of the rules may then applied to the unclassified traffic by the software.

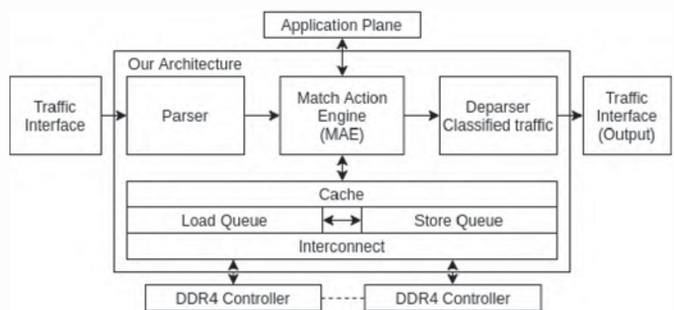


Fig. 1: Diagram of our architecture

Our architecture depicted in Fig. 1 contains parser and deparser, out-of-order TSS pipeline (inside Match Action Engine), Cache, Load/Store queues and DDR memory subsystem. The whole design is non-blocking and pipelined.

The Match Action Engine (MAE) is a component implementing the TSS algorithm and rule statistic handling. The packet classification may finish out-of-order and the memory transactions may also finish out-of-order for a single lookup. The pipelined design presents a large number of hazards and issues which we will address in this section. The inner design and data flow is depicted in Fig. 2.

Authorized licensed use limited to: Rutgers University. Downloaded on May 17,2021 at 02:38:26 UTC from IEEE Xplore. Restrictions apply.

The functional design of the MAE can be separated into several parts. Input handler, Memory subsystem (including Cache and Load/Store queues) and the main Action Pipeline.

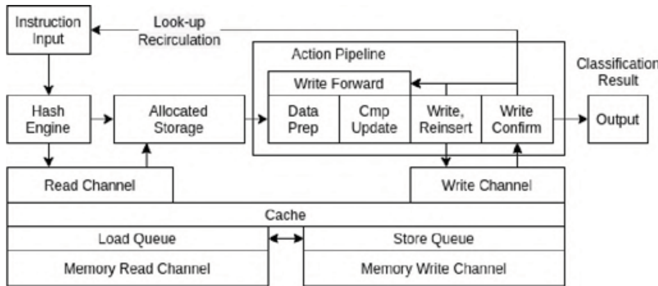


Fig. 2: Diagram of Match Action Engine datapath

The MAE has an input traffic interfaces which always executes Look-up instruction for the first table of the TSS. It also has a service interface that is used for managing the table and monitoring statistics by the application plane. The management of the tables inside the external memory of the accelerator does have many corner cases and hazards that can severely degrade the performance and are costly to remove. One of the examples is an atomic copy of all counters to a software. The pipeline does support of concurrently running updates and the slowdown is caused only by limited throughput of the DDR4. The AMBA AXI4 interface is used for a memory access.

Look-up operation consists of several steps: 1) A computation of the hash from selected fields according to a specification of actually selected table. 2) The read transaction is issued into external memory. 3) The the result of the read transaction is processed inside the Action pipeline. If the key is not found inside the table the action pipeline selects next table according the TSS table chain and recirculates the input to the first step. If the key is found the counters are modified and written back into the memory subsystem and classified packet is dispatched on the output interface.

1) Input Handler: The input handler identifies and parses instructions from both of the interfaces. It computes the hash according to the actually selected table for a transaction. The hash is used to initiate read transactions from the table inside the memory subsystem. When the transaction is dispatched the session with an information for later statistic update is stored in Allocated Storage which also manages the allocation of the tags for a communication over the main bus.

The id of read transaction received from memory subsystem loads a session state from Allocated storage in to a action pipeline, which then computes the state update and potentially deallocates the session from its storage upon operation competition. This allows the action pipeline to process transaction out-of-order thus significantly reducing delays on cache misses.

2) Memory subsystem: Our memory subsystem is designed to increase the possibility that memory transactions will be merged before dispatching to DDR4 SDRAM chip, to maximize the possibilities for Memory-level parallelism (MLP) and to fully utilize memory chips in the 100 % cache miss scenario. Compare to a common cache subsystems widely found in CPUs. It has several unique features. It is designed for latency insensitive applications and to minimize the need for a buffering. Note that the increase of MLP and minimization of buffering are contradictory. We minimize buffer requirements by moving management of transaction context and collision handling to the MAE and Load/Store queues, where we can improve MLP at a much smaller cost.

3) Cache: The cache is an optional part of our architecture. We utilize unused resources inside the FPGA to increase the throughput of our design. The cache works in write-allocate mode and utilizes Tree-PLRU as a cacheline replacement policy. The write-allocate is used to minimize memory access to a cache memory as our application always writes the cacheline because of rule statistics update. The read cancellation happens when a cache line is invalidated by a write transaction, it is forwarded to a Load queue, and it marks all read from selected address to be invalidated. This means that the cache does not need to store information about pending transactions or buffer them. Also, it makes only MAE and Load/Store queue responsible for maintenance of correct store order. As the MAE and Store queue already has the out-of-order infrastructure, the additional resource requirements are minimal, and the cache is significantly simplified.

4) Load/Store queues: The Load/Store queues are essential parts of our architecture. The components are simple from an algorithmic point of view. It only merges transactions that are working with a same address. It implements read and write bypass and write forwarding. The CAMs used to detect address collisions as well as leading zero/one detectors used to allocate in out-of-order transaction logic have high latency, and thus the architecture needs to have a high degree of pipelining.

During the synchronization between write (Store) and read (Load) channel queues, the ongoing read transaction may be canceled by write into the cache. This event causes the pending read transaction to be invalidated, and thus it must be re-executed from the action pipeline. As the read cancellation was caused by cache write and the cache works in write-allocate mode, the cacheline should be present in the cache, and thus, the next read should execute immediately.

5) Action pipeline: Action pipeline is the most complicated part of the parts. It needs to resolve hazards that can occur during the processing by the pipeline itself in an out-of-order manner.

The pipeline receives Out-of-Order read transactions from memory with the attached tag. It then reads the appropriate session from the allocated storage and starts processing the read data. It compares the result with the searched key using a specified mask. In case of a match, it then updates the appropriate statistics of the rule and writes the updated values into the cache. This writing transaction's latency should be minimal because potential cache line flushing uses LSQ, which has minimal write latency. Afterward, the result is sent to the output in case of match. In case of mismatch, the instruction is recirculated to search the next table given by the TSS

Authorized licensed use limited to: Rutgers University. Downloaded on May 17,2021 at 02:38:26 UTC from IEEE Xplore. Restrictions apply.

TABLE II: Overview of similar solutions for packet classification

Name	Rule Size	Rule Storage	Memory	Mpps
Our architecture	64 B	16 GB	off-chip	300
Rozhko [20]	50 B	536 Mbit	off-chip	22.4
Jiang [18]	20 B	4896 kbit	on-chip	112
Qu [19]	50 B	729 kbit	on-chip	462
DPDK 4-core [4]	64 B+	512 kbit	L1 cache	36

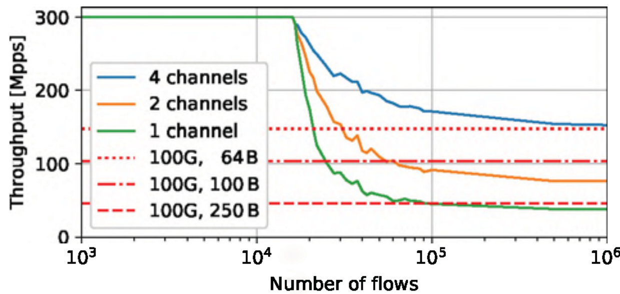


Fig. 3: The worst case performance of our architecture with increasing number of flows

algorithm. If the engine reaches the last table, it outputs the packet as unclassified to be handled by the application plane.

Hazards introduced by searching the same keys in a span of few clocks are solved by implementing write forwarding for the rule statistics, which is enabled by comparing the address and record move and match flags inside the current stage of the pipeline with stages containing previously written data.

V. Performance Analysis

The external memories limit the performance of our architecture. Two 64b DDR4-2666 memory channels are sufficient for a classification of 100G traffic, but it can be scaled up to 4 channels.

A. Results

Table II shows a comparison of our architecture with other similar packet classification solutions. This comparison is the general and optimal use-case where the number of rules does not exceed the caches configured for each solution. The only solutions also using external memories for a networks of similar speeds we are aware of are Rozhko and AccelNet. The AccelNet does not provide similar information as the other architectures. However, it is designed for 40 Gbit/s. Our architecture outperforms all of the previous architectures in terms of rule storage and all of the solutions using external memories in terms of throughput.

To analyze the effectiveness of memory subsystem of our architecture, we simulate the whole architecture with maximal feasible size of the cache (16K cache-lines) for our FPGA chip. Our architecture is mostly limited by the number of DDR channels available. Figure 3 shows an evaluation of the performance for incoming number of flows depending on the number of DDR channel used. In the graph there are also approximate thresholds for shortest, median and half of average packet lengths for 100 Gbit/s network [29, 30]. The simulations implement the worst case scenario of using uniform distribution of incoming packets for given number of flows. Realistically, this scenario is improbable to happen due to the nature of internet traffic. This worst-case scenario shows that after the number of flows exceeds the cache size the performance is decreased to the performance of the DDR channels. Our architecture on high-end FPGA card can handle even the case of the shortest packet lengths. On mid-tier FPGA cards, our architecture can handle $\approx 50\%$ of the worst case, $\approx 80\%$ of the packets with median packet lengths, and 100% of the packets with half of average packet lengths, which satisfies requirements defined in II.

B. Resources

The design has been implemented on `Xilinx xcvu9p` FPGA. The Load and Store buffer is configured to have space for 64 items and the rule contains 64 B packet and byte counter and the last timestamp and a main pipeline with 8 stages. It achieved a frequency of over 300 MHz. The platform has 2x100 Gbit/s Ethernet port, 2x64 bit DDR4-2666 8 GB and it uses OpenFlow 1.5.1-like rule format (64 B in size). The data width of main bus was 512 bit, and the 4-set associative cache can store 16K cachelines. The main cache memory is implemented using Xilinx UltraRAM, the BRAM memories of the cache are used to store tags and LRU flags. Resources are displayed in Table III. The sum of logical resources used is just a fraction of resources consumed by the memory controllers. The cache is configured to perform 1 write and 1 read in a single clock, which result in classification performance of 300 Mp/s with minimal read and write latency of 4 and 5 clock cycles.

TABLE III: Resource utilization and frequency. Implemented for Xilinx Virtex Ultrascale+.

Component	LUTs	FFs	BRAMs	URAMs	F [MHz]
MAE	3525	5989	8	0	300
Cache	1152	3656	23.5	32	300
Load queue	1668	2508	0	0	300
Store queue	4189	4862	17	0	300
Interconnect	331	164	0	0	300
Sum	10865	17179	48.5	32	-
Mem controllers	30462	37172	51	0	300

VI. Conclusions

Modern network infrastructures have tremendous demands on flexibility and configurability. Therefore, developers of network devices are forced to support a high capacity memory storage capacity that can contain a sufficient number of configuration rules. Such rules are internally used for packet classification and define the behavior of the network.

Additionally, as the network traffic increases its volume, required throughput grows, and it is becoming usual in large infrastructures and data-centers to deploy 100 Gbit/s technologies. It is challenging to design a solution with sufficient performance parameters and a high memory capacity.

This paper presented a novel hardware architecture that allows usage of external high capacity memory for above

Authorized licensed use limited to: Rutgers University. Downloaded on May 17,2021 at 02:38:26 UTC from IEEE Xplore. Restrictions apply.

100G applications. Using our proposed hardware design, it is possible to perform packet classification, which is an essential process of OVS, using up to 1.9 million rules. Meanwhile, the design itself is ready for up to 200 Gbit/s processing up to 300 millions packets per second. The comparison of our architecture with previous solutions shows significant improvements for real-world applications. These results seem very promising, and, based on our analysis of available datasets, the performance is sufficient for deployment into the real environment.

Our architecture allows us to continue our research in multiple directions in the future. We believe the performance of our architecture can be further improved by utilizing multiple memory ports on our cache. Additionally, the future work can be focused on the application plane and optimal arrangement of the tables of rules.

ACKNOWLEDGMENT

This work was supported by Technology Agency of the Czech Republic, project Acceleration platform for virtual switches [TH04010193], and also by the Grant Agency of the Czech Technical University in Prague, grant No. SGS20/210/OHK3/3T/18.

REFERENCES

[1] P. Goransson *et al.*, *Software defined networks: a comprehensive approach*, Second edition. Amsterdam ; Singapore: Morgan Kaufmann, 2017, 409 pp., ISBN: 978-0-12-804555-8.

[2] Open Networking Foundation, "OpenFlow switch specification version 1.5.1," [Online]. Available: https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf (visited on 02/08/2021).

[3] Open vSwitch, [Online]. Available: https://www.openvswitch.org/ (visited on 01/24/2021).

[4] Intel Corporation, "Intel® open network platform release 2.1 performance test report," [Online]. Available: https://download.01.org/packet-processing/ONPS2.1/Intel_ONP_Release_2.1_Performance_Test_Report_Rev1.0.pdf.

[5] M. A. Finlayson, "OpenFlow table type patterns," manual, Aug. 15, 2014, [Online]. Available: https://opennetworking.org/wp-content/uploads/2013/04/OpenFlow%20Table%20Type%20Patterns%20v1.0.pdf.

[6] T. Benson *et al.*, "Understanding data center traffic characteristics," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1, pp. 92–99, Jan. 7, 2010, DOI: 10.1145/1672308.1672325.

[7] Bo Xu *et al.*, "HSM: A fast packet classification algorithm," in *19th International Conference on Advanced Information Networking and Applications (AINA'05) Volume 1 (AINA papers)*, vol. 1, Mar. 2005, 987–992 vol.1, DOI: 10.1109/AINA.2005.200.

[8] T. Srinivasan *et al.*, "Scalable and parallel aggregated bit vector packet classification using prefix computation model," in *International Symposium on Parallel Computing in Electrical Engineering (PARELEC'06)*, Sep. 2006, pp. 139–144, DOI: 10.1109/PARELEC.2006.71.

[9] B. Vamanan *et al.*, "EffiCuts: Optimizing packet classification for memory and throughput," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, pp. 207–218, Aug. 16, 2010, DOI: 10.1145/1851275.1851208.

[10] Y. Qi *et al.*, "Packet classification algorithms: From theory to practice," in *IEEE INFOCOM 2009*, Apr. 2009, pp. 648–656, DOI: 10.1109/INFCOM.2009.5061972.

[11] H. Lim *et al.*, "A quad-trie conditionally merged with a decision tree for packet classification," *IEEE Communications Letters*, vol. 18, no. 4, pp. 676–679, Apr. 2014, DOI: 10.1109/LCOMM.2014.013114.132384.

[12] V. Srinivasan *et al.*, "Packet classification using tuple space search," *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 4, pp. 135–146, Oct. 1999, DOI: 10.1145/316194.316216.

[13] Y. Wang *et al.*, "Optimizing open vSwitch to support millions of flows," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, Dec. 2017, pp. 1–7, DOI: 10.1109/GLOCOM.2017.8254754.

[14] R. Pagh *et al.*, "Cuckoo hashing," *Journal of Algorithms*, vol. 51, no. 2, pp. 122–144, May 1, 2004, DOI: 10.1016/j.jalgor.2003.12.002.

[15] J. Tseng *et al.*, "Accelerating open vSwitch with integrated GPU," in *Proceedings of the Workshop on Kernel-Bypass Networks*, Los Angeles CA USA: ACM, Aug. 9, 2017, pp. 7–12, DOI: 10.1145/3098583.3098585.

[16] K. Qiu *et al.*, "GFlow: Towards GPU-based high-performance table matching in OpenFlow switches," in *2015 International Conference on Information Networking (ICOIN)*, Jan. 2015, pp. 283–288, DOI: 10.1109/ICOIN.2015.7057897.

[17] W. Jiang *et al.*, "Large-scale wire-speed packet classification on FPGAs," in *Proceeding of the ACM/SIGDA international symposium on Field programmable gate arrays - FPGA '09*, Monterey, California, USA: ACM Press, 2009, p. 219, DOI: 10.1145/1508128.1508162.

[18] W. Jiang *et al.*, "Scalable packet classification on FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 9, pp. 1668–1680, Sep. 2012, DOI: 10.1109/TVLSI.2011.2162112.

[19] Y. R. Qu *et al.*, "High-performance and dynamically updatable packet classification engine on FPGA," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 1, pp. 197–209, Jan. 2016, DOI: 10.1109/TPDS.2015.2389239.

[20] D. Rozhko *et al.*, "Packet matching on FPGAs using HMC memory: Towards one million rules," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Monterey California USA: ACM, Feb. 22, 2017, pp. 201–206, DOI: 10.1145/3020078.3021752.

[21] D. Firestone *et al.*, "Azure accelerated networking: SmartNICs in the public cloud," in *Proceedings of the 15th USENIX conference on networked systems design and implementation*, ser. NSDI'18, USA: USENIX Association, 2018, pp. 51–64.

[22] V. Puš *et al.*, "Fast and scalable packet classification using perfect hash functions," in *Proceedings of the ACM/SIGDA international symposium on field programmable gate arrays*, ser. FPGA '09, New York, NY, USA: Association for Computing Machinery, 2009, pp. 229–236, DOI: 10.1145/1508128.1508163.

[23] S. Dharmapurikar *et al.*, "Fast packet classification using bloom filters," in *2006 Symposium on Architecture For Networking And Communications Systems*, Dec. 2006, pp. 61–70, DOI: 10.1145/1185347.1185356.

[24] S. Pontarelli *et al.*, "FlowBlaze: Stateful packet processing in hardware," in *16th USENIX symposium on networked systems design and implementation (NSDI 19)*, Boston, MA: USENIX Association, Feb. 2019, pp. 531–548, [Online]. Available: https://www.usenix.org/conference/nsdi19/presentation/pontarelli.

[25] Xilinx, Inc., "UltraScale architecture-based FPGAs memory IP v1.4 LogiCORE IP product guide," 2020, [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/ultrascale_memory_ip/v1_4/pg150-ultrascale-memory-ip.pdf.

[26] M. Žádník, "OPTIMIZATION OF NETWORK FLOW MONITORING," Brno, 2013, [Online]. Available: https://www.vutbr.cz/www/_base/zav/_prace/_soubor/_verejne.php?file%5C_id=136992 (visited on 10/07/2020).

[27] C. Zhiruo *et al.*, "Performance of hashing-based schemes for internet load balancing," in *Proceedings IEEE INFOCOM 2000.*, vol. 1, Mar. 2000, 332–341 vol.1, DOI: 10.1109/INFCOM.2000.832203.

[28] N. Hua *et al.*, "Non-crypto hardware hash functions for high performance networking ASICs," in *2011 ACM/IEEE Seventh Symposium on Architectures for Networking and Communications Systems*, Oct. 2011, pp. 156–166, DOI: 10.1109/ANCS.2011.32.

[29] CAIDA: Center for Applied Internet Data Analysis. Packet length distributions, CAIDA, [Online]. Available: https://www.caida.org/research/traffic-analysis/AIX/plen_hist/index.xml (visited on 02/06/2021).

[30] ——, Packet size distribution comparison between internet links in 1998 and 2008, CAIDA, [Online]. Available: https://www.caida.org/research/traffic-analysis/pkt_size_distribution/graphs.xml (visited on 02/08/2021).

Authorized licensed use limited to: Rutgers University. Downloaded on May 17,2021 at 02:38:26 UTC from IEEE Xplore. Restrictions apply.