CATCAM: Constant-time Alteration Ternary CAM with Scalable In-Memory Architecture

Dibei Chen
Tsinghua University
chendb17@mails.tsinghua.edu.cn

Zhaoshi Li
Tsinghua University
lizhaoshi@tsinghua.edu.cn

Tianzhu Xiong
Southeast University
tianzhu_xiong@seu.edu.cn

Zhiwei Liu
Tsinghua University
liuzhiwei@tsinghua.edu.cn

Jun Yang
Southeast University
dragon@seu.edu.cn

Shouyi Yin
Tsinghua University
yinsy@tsinghua.edu.cn

Shaojun Wei
Tsinghua University
wsj@tsinghua.edu.cn

Leibo Liu*
Tsinghua University
liulb@tsinghua.edu.cn

***Abstract*—TCAM (Ternary Content-Addressable Memory) is the essential component for high-speed packet classification in modern hardware switches. However, due to its relatively slow update process, recent advances in Software-Defined Network (SDN) regard them as the bottleneck to the agile deployment of network services. Rule installation in commodity switches suffers from non-deterministic delays, ranging from a few milliseconds to nearly half a second. The crux of the problem is that TCAM prioritizes rules based on physical addresses. Corresponding entries have to be reallocated according to the priority of an incoming rule, such that the insertion delay grows linearly with the number of existing rules in a TCAM.**

In this paper, we present *Constant-time Alteration Ternary CAM* (CATCAM) that can accomplish both lookup queries and update requests for packet classification in a few nanoseconds. The key to fast update is to decouple rule priorities from physical addresses. We propose a matrix-based priority encoding scheme that records the priority relation between rules and can be implemented in 8T SRAM arrays with the emerging Processing In-Memory (PIM) technique. CATCAM also comes with a hierarchical architecture to scale out, its interval-based scheduling scheme guarantees deterministic update performance in all scenarios. CATCAM is developed under full-custom design in the 28 nm process. Evaluation across benchmark workloads shows that CATCAM provides at least three orders of magnitude speedup over state-of-the-art TCAM update algorithms and offers equivalent search capability to conventional TCAM while incurring 0.3% power and 20% area overhead.

***Index Terms*—packet classification, ternary content-addressable memory, processing in-memory, hardware accelerator**

I. Introduction

Packet classification is a key algorithm in computer networking. The goal of packet classification is to match the header fields of an incoming packet against a set of pre-defined rules, then take the corresponding action associated with matched rules [12]. It is widely used in network infrastructure to enable many network services including packet forwarding, firewalls, access control, traffic monitoring, load balancing and Quality of Service (QoS). Emerging Software-Defined Networking (SDN) [4], [39] calls for the agile deployments of network services, thereby efficient and flexible packet classification is crucial.

However, recent measurements on commercial switches from various vendors show that the performance of updating rules is far from satisfying for SDN deployments [17], [23]. Today's hardware switches only support less than one hundred rule updates per second [28], which is far from enough for dynamic fine-grained policies. Unpredictable rule installation latency up to milliseconds causes switches to fall behind network events even periodically or randomly stop responding to requests. To make matters worse, packets may drop or be forwarded incorrectly during seemingly responsive rule updates. In this case, the consistency of network states may not be guaranteed, where security concerns arise [32], [44]. On the other hand, software switches running on commodity servers are a potential alternative since they can update the rule table at a much faster rate [17]. However, they cannot match the performance of hardware switches, which remain two orders of magnitude faster in terms of packet processing [5].

Ternary Content-Addressable Memory (TCAM) is the bottleneck of hardware switches towards the agile SDN deployments [23]. To allow a single rule to match multiple types of headers with the same action, rules are specified with wildcards (masks). TCAM leverages specialized memory architecture to compare input search data against all stored wildcarded rules in parallel [40], offering fast and constant lookup time. Therefore, despite being power-hungry and expensive, TCAM has been an indispensable component of lookup tables in commodity switches or routers.

Nevertheless, TCAM suffers from expensive and inflexible update operations. Given the header of a packet, when multiple rules with wildcards are matched, the one with the highest priority is selected. Conventional TCAM stores rules from top to bottom in decreasing order of priority [41], i.e., an entry located at a higher physical address has higher priority. Therefore, when multiple matches occur, the entry with the highest address is selected. However, the insertion of a single rule could lead to a substantial amount of moves for existing entries in a TCAM to maintain priority order [46]. Such overhead was not a major problem when network rulesets were

*Corresponding author: Leibo Liu (liulb@tsinghua.edu.cn)

978-1-7281-7383-2/20/$31.00 ©2020 IEEE
DOI 10.1109/MICRO50266.2020.00038

almost static, but they are dynamically reconfigured in SDN. Although state-of-the-art TCAM updating algorithms [15], [43] utilize the minimum dependency graph to minimize entry movements for rule insertion, whose computation is complex and time-consuming, they still incur $O(n)$ update cost in the worst case, where n is the number of existing rules in TCAM.

In this paper, we present *Constant-time Alteration Ternary CAM* (CATCAM) that can accomplish both lookup queries and update requests for packet classification in constant time. Our insight is that rule priorities can be decoupled from physical addresses if we encode the priority relation between rules separately in a *priority matrix*. During lookup, after matching against all rules in a *match matrix*, matched rules go through the priority matrix to determine the one with the highest priority. During update, because of the decoupling, new rules can be written to any empty slot without consideration of relative priority among rules.

Since lookup into the priority matrix involves large amounts of logic operations, we observe that the Processing In-Memory (PIM) paradigm is a perfect match to realize CATCAM at a low cost. We customize the decoding and control logics of 8T SRAM arrays so that priority decision is performed in-place. Existing SRAM designs are also challenged since our encoding scheme demands column-wise write during rule insertion, which is not supported in conventional SRAM arrays. We learn from a dual-voltage scheme to directly support column-wise write from the circuit level. Furthermore, 8T SRAM arrays can also be repurposed to implement the match matrix as an alternative to conventional 16T TCAM.

As the cloud platforms are scaling up to support an increasing number of network services, underlying switches have to support more complex and larger rulesets. Usually, a ruleset is broken into several subtables so that each subtable can be stored in a TCAM block. Matched rules from different subtables are arbitrated according to pre-defined priorities among subtables. However, this method scales poorly in terms of update performance. Fortunately, the idea of decoupling rule priorities from physical addresses can also be applied to scale out CATCAM subtables. By introducing a secondary priority matrix that encodes the priority relation of subtables, CATCAM builds a hierarchical architecture that can be fully pipelined during lookup. It ensures that at most two extra SRAM arrays are accessed per query, rendering the energy overhead negligible. During update, empty subtables can be assigned on demand. With each subtable covers a dynamic range of rule priorities, CATCAM can scale out efficiently while keeping the update cost low and deterministic.

The major contributions of this paper are as follows.

- We propose a matrix-based priority encoding scheme to decouple rule priorities from physical addresses. By replacing the priority encoder of conventional TCAM with a priority matrix, $O(1)$ time rule update is achieved in CATCAM.
- We propose a hierarchical scalability scheme that uses a secondary priority matrix to encode the priority relation between subtables. By dynamically segmenting the range of rule priorities into non-overlapping intervals and associating them with new subtables at runtime, CATCAM can accommodate hundreds of thousands of rules.
- We implement CATCAM in-memory with customized 8T SRAM arrays. Evaluation across benchmark workloads shows that it provides at least three orders of magnitude speedup over state-of-the-art TCAM update algorithms and offers equivalent search capability to conventional TCAM while incurring 0.3% power and 20% area overhead.

The rest of this paper is structured as follows. Section 2 introduces background on SDN, TCAM and PIM. Section 3 proposes the matrix-based priority encoding scheme. Section 4 proposes the hierarchical scalability scheme. Section 5 and 6 present the in-memory implementation of CATCAM. Section 7 and 8 discuss the evaluation methodology and report the experimental results. Section 9 discusses the related works and Section 10 concludes the paper.

II. Background

A. Network Switches in SDN

In Software-Defined Network (SDN), a logically centralized controller named the control plane manages the network traffic by installing lookup table rules in the underlying switches named the data plane. The data plane then matches the header fields of incoming packets and takes the corresponding action associated with the matched rules. A main advantage brought by SDN is dynamic network reconfiguration with frequently updated fine-grained policies (over 42 fields defined in OpenFlow specification). Since network states can respond faster to network events, SDN enables a wide range of virtualization functionalities such as failure recovery [38], traffic engineering [20] and real-time migration [34]. Therefore, unlike traditional algorithmic packet classification with a focus on fast table lookup, SDN-enabled switches have a much higher demand for rule update [31], [54]. A recent benchmark with software Open vSwitch handles 42,000 rule updates per second [28]. On the contrary, today's hardware switches only support around 40 to 50 rule updates per second [17]. Systematic experiments on commodity OpenFlow switches [29] report that the divergence between switch data plane installation and control plane acknowledgment increases with the number of rules in the flow table, as shown in

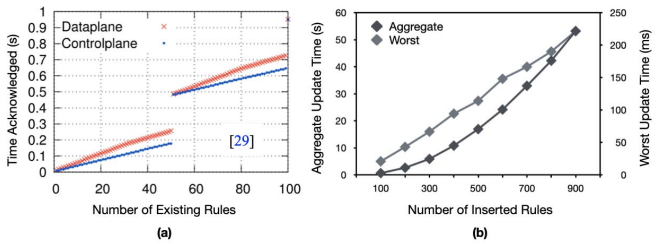


Fig. 1. (a) Divergence between data plane and control plane in HP 5406zl during rule installation. (b) Rule insertion time in a naive TCAM.

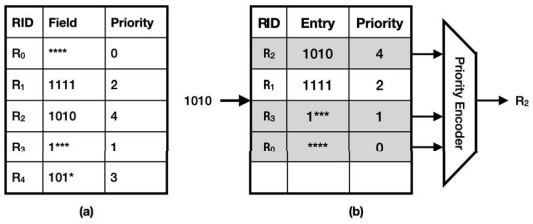


Fig. 2. The wildcarded rules are prioritized in a TCAM for lookup.

Fig. 1(a). Unpredictable rule installation latency up to 300 ms causes switches to fall behind network events even periodically or randomly stop responding to requests. Network states are inconsistent during this period and packets may drop or be forwarded incorrectly.

B. TCAM Primer

Content-Addressable Memory (CAM) compares input search data against every stored entry (rule) and returns the address location of the matched entries. It allows parallel lookups to its entire memory. *Ternary* in TCAM means except for 0s and 1s, TCAM also stores "don't care" bits, denoted as $*$, that match both 0 and 1, thus adding flexibility to the match. However, the ambiguity of wildcards may cause multiple rules to be matched with a given input. In that case, priorities are assigned to rules for disambiguation. Fig. 2(a) shows a ruleset with five rules ($R_0 \sim R_4$) and their priorities. Here, R_0, R_3 and R_4 are specified with the wildcard *. The larger the priority number, the higher the rule priority.

To represent the ruleset, TCAM stores rules in decreasing order of priority, as shown in Fig. 2(b) how $R_0 \sim R_3$ are arranged. In many network applications, a single highest priority match is required. Since an entry located at a higher physical address has higher priority, a *priority encoder* [16] chooses the one with the highest priority by simply detecting the most significant 1 in the bit vector indicating matched entries. For example, in Fig. 2(b), an input 1010 matches R_2, R_3, R_0, resulting a *match vector* 1011. The priority encoder reports R_2 because R_2 has the highest physical address among the matched entries. In this way, TCAM can perform lookup of packet classification in constant time.

However, the time complexity of TCAM update grows linearly with the number of stored entries. For example, if R_4 in Fig. 2(a) is to be inserted to the TCAM in Fig. 3(a), the firmware first determines an appropriate physical address according to its priority, i.e., it should be placed between entry 1010 and 1111. Then all entries below are shifted sequentially to make room for R_4, as shown in Fig. 3(b).

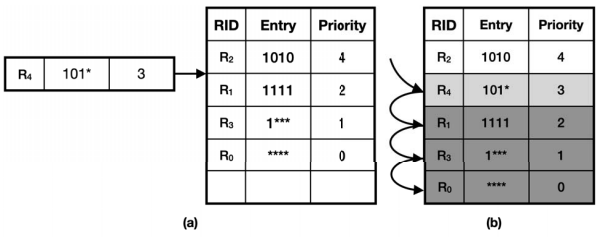


Fig. 3. The insertion of a new rule causes existing entries to reallocate.

To demonstrate the impact, we model a naive TCAM and assess its update process. The frequency is set to 400 MHz along with a capacity of 1000 entries, which is close to those adopted in a commodity switch. It starts with an empty TCAM and rules from benchmark workloads (section VII) are inserted. Here, the update time is proportional to the number of moves it takes to accommodate a new rule. Fig. 1(b) plots the aggregate and worst update time with an increasing number of rules to be inserted. As can be seen, the update time per rule grows linearly with the number of inserted rules. In the worst case, it takes more than 200 ms, meaning almost every existing entry has to be rewritten. On a 40 Gbps link, such delay is equivalent to the arrival of more than 15 million packets of 64 bytes. Thus, updating a TCAM is similar to the insertion sort that takes $O(n)$ time, where n is the number of inserted rules. Taking into account the amount of work the switch firmware does to schedule the update, the latency could reach hundreds of milliseconds. Packets are matched with the outdated ruleset during this period, which accounts for the undesired switch behavior reported in Fig. 1(a).

C. Processing In-Memory

Today the von Neumann architecture suffers from the memory-wall problem: memory bandwidth cannot satisfy modern data-intensive systems. The Processing In-Memory (PIM) paradigm is an emerging solution to overcome the memory wall. The idea is to physically bring the computation into the memory, thus eliminates the need to move data back and forth between memory and computing units. They are under extensive study for data-intensive applications [1] like neural network [9], [45], automata [49], graph analytics [60], sequence alignment [26], [35], sparse distributed memory [24], etc.

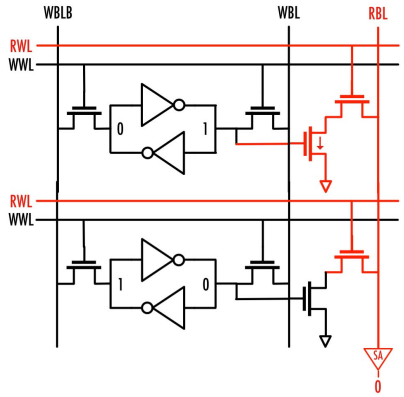


Fig. 4. In-memory logic operation explained. Two rows (RWLs) are activated simultaneously. RBL is initially pre-charged to '1'. If any of the activated bits is '1', RBL discharges and is sensed as '0'. Otherwise, RBL stays high and is sensed as '0'. A *nor* operation is therefore performed.

PIM repurposes memory arrays like SRAM into massive vector computing units. Usually, for an SRAM array, only one word-line is activated at a time to access the corresponding row. It is observed that when two or more word-lines are activated simultaneously, the shared bit-lines can be sensed to perform in-place computation on the data stored in the activated words. Fig. 4 shows two 8T SRAM cells. If both RWLs are activated, the RBL can be sensed to produce the result of nor on the stored bits. Compared to conventional 6T SRAM, by decoupling the read and write path, 8T SRAM prevents data corruption due to multi-row access [2], which allows more word-lines to be activated at the same time. The entire memory array, therefore, morphs into massive vector computing units whose operand sizes or numbers are orders of magnitude larger than conventional SIMD units.

III. Matrix-based Priority Encoding

According to our analysis in section II-B, the crux of the problem is that TCAM orders rule priorities according to physical addresses, hence does not explicitly encode priority information. If we manage to decouple rule priorities from physical addresses, new rules can be written to any empty slot. To this end, we propose the matrix-based priority encoding scheme (section III-A). With this scheme, CATCAM replaces the priority encoder for conventional TCAM with a *priority matrix*, as shown in Fig. 5. During lookup, after matching against all rules, the match vector will traverse the priority matrix to filter out the rule with the highest priority and generate the report vector (section III-B). While inserting a new rule, the priority matrix will be updated along with the match matrix (section III-C).

A. Encoding Scheme

If we want to decouple priorities from physical addresses, a straightforward way is to store the priorities along with the stored rules. Assuming we have stored n rules, selecting the highest priority rule requires $O(n^2)$ integer comparisons considering multiple rules can be matched. Since rulesets typically contain thousands of rules, $O(n^2)$ comparators are unrealistic to be implemented in parallel on hardware.

Instead, our scheme encodes the binary relation of the relative order between rules in the *priority matrix* of Fig. 5. Each rule in the match matrix correlates with a row and a column in the priority matrix. A boolean element of the priority matrix $P_{ij} = true$ means the corresponding rule of row i has a higher priority than the corresponding rule of column j. In our case, $P_{32} = true$ indicates R_2 has a higher priority than R_1 while $P_{14} = false$ indicates R_0 has a lower priority than R_3.

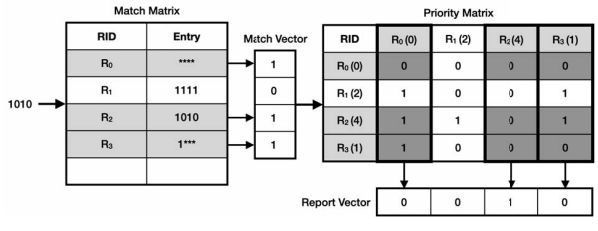


Fig. 5. Priority relation of rules from Fig. 2 is encoded in the priority matrix. Input 1010 is compared in the match matrix first, the resulting match vector is fed to the priority matrix. The arrows beneath the priority matrix denote a nor operation is performed on the shaded elements of the corresponding column. The true in the report vector indicates R_2 has the highest priority.

B. Priority Decision

To filter out the rule with the highest priority, the priority matrix is traversed. We leverage the insight that if the ith rule has the highest priority among all matched rules, its priority has to be higher than any other matched rules. That is to say, any other matched rule has a lower priority than it, i.e., for any matched rule j, $P_{ji} = false$.

CATCAM iterates all matched columns with this insight to choose the matched rule with the highest priority. Intuitively, according to the match vector, the matched rows and columns are selected to compose a submatrix, as shown in the shaded part of Fig. 5. Then each column of the submatrix conducts the reductive nor. If any matched rule has a higher priority, which suggests there is a $true$ in the column, the result of nor is $false$. Therefore, the resulting bit indicates whether the corresponding rule of this column has the highest priority. In other words, all rows of the submatrix conduct bit-wise nor. Since no matched rules for a header have the same priority, the resulting one-hot *report vector* whose ith bit is $true$ indicates the ith matched rule has the highest priority.

C. Rule Update

Since priority order is no longer subject to physical addresses, new rules can be inserted into any empty slot in the match matrix. During insertion, the priority of the new rule is compared to stored rules with $O(n)$ comparators. Such resource cost is much more affordable on hardware than previous $O(n^2)$ comparators. According to the allocated address in the match matrix for the new rule, the corresponding row and column is to updated in the priority matrix, as shown in Fig. 6. CATCAM does not need to reallocate an empty slot by moving other entries like conventional TCAM, thus features $O(1)$ time insertion.

CATCAM updates also include rule deletion and modification. Deletion simply invalidates the corresponding rule.

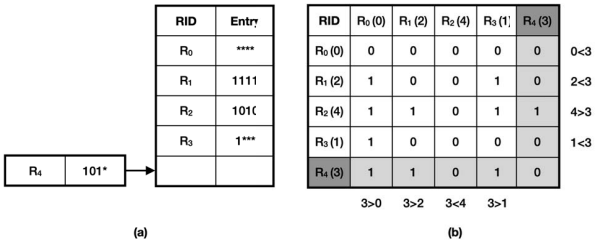


Fig. 6. R_4 from the scenario in Fig. 3 is inserted into CATCAM. While the rule is inserted to the empty slot in the match matrix (a), the priority matrix is updated according to its address and priority (b).

Modification can be processed by deleting the original rule then inserting its new version.

IV. Hierarchical Scalability

To support increasingly complex network services, CATCAM has to find an efficient approach to scale out. To accommodate hundreds of thousands of rules, large and complex rulesets should be partitioned into multiple subtables so that each subtable can be stored in one memory block. Similar to the priority encoder of TCAM, a straightforward scheme to scale out is to manually define a global priority order according to the placement of TCAM subtables. But this scheme will further deteriorate the update performance since existing rules may have to be moved among subtables to reallocate slots for new rules. Designing an efficient and scalable partition scheme is nontrivial, especially when taking both lookup and update overhead into account.

Fortunately, the idea of decoupling rule priorities from physical addresses can be borrowed to scale out CATCAM subtables. The priority matrix described in section III eliminates rule reallocations within a subtable. Similarly, it can be applied to avoid rule reallocations among subtables. A secondary priority matrix, named the *global priority matrix*, is devised to encode the priority relation of CATCAM subtables at runtime. In this way, CATCAM scales out by segmenting the range of rule priorities into non-overlapping intervals and dynamically associating each subtable with an interval, which will be discussed in section IV-A. Then section IV-B will discuss how rule insertion is handled in different scenarios and section IV-C will discuss the necessity of the proposed encoding scheme.

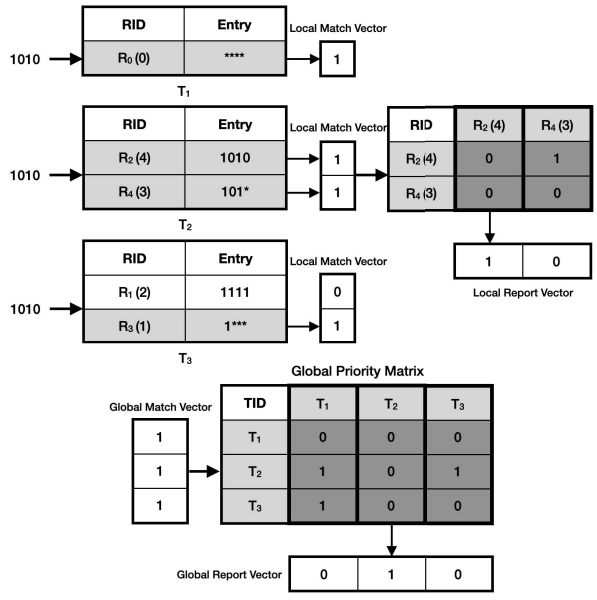


Fig. 7. Input 1010 is searched in three CATCAM subtables then arbitrated by the global priority matrix.

A. Global Priority Matrix

Since rule priorities are generally specified as a set of integers (e.g., a 16-bit field for OpenFlow rules), the entire range of rule priorities can be segmented into non-overlapping intervals. Each subtable is associated with an interval. By storing the rule whose priority belongs to a certain interval in the associated subtable, it is ensured that all the rules in a subtable have higher (or lower) priorities than any other subtable. The priority relation between subtables is therefore established and can be encoded in the global priority matrix using the same scheme as before. For example, we assume that rules from Fig. 2 are assigned to three subtables, as shown in Fig. 7. According to their priority intervals, the order of subtables can be expressed as $T_2 > T_3 > T_1$, which is encoded in the global priority matrix.

Fig. 7 shows how an input header is searched in three subtables then arbitrated by the global priority matrix. The search is performed hierarchically. To begin with, the input header is broadcast to each subtable to obtain the local match vector. Intuitively, the matched rule with the highest priority must be among the highest priority subtable with matched rules. Consequently, a *global match vector*, whose ith bit is $true$ indicates there exist matched rules in the ith subtable, is fed to the global priority matrix. Similar to priority decision in the local priority matrix, the global priority matrix generates the one-hot *global report vector* indicating which subtable with matched rules has the highest priority. After arbitration, the chosen local match vector is sent to the corresponding local priority matrix to generate the final report vector. In our case, the global priority matrix chooses T_1 to report its rule with the highest priority.

B. Rule Insertion Scheduling

The hierarchical design seems to work for existing TCAM - if only lookup is considered. However, rulesets are not static. Constrained by the linearity of the priority encoder, the order of subtables is fixed, so does the possible position for the new rule. Rule insertion may cause a substantial amount of moves between existing rules, some of which may even cross different subtables. In the worse case, if the rule is to be inserted into the first subtable while the empty slot is located in the last subtable, rule reallocation will happen between every adjacent subtables.

To enable flexible rule placement during insertion, the key is to allocate empty slots on demand. Apart from the flexibility that allows rules to be inserted into any empty slot in a subtable, in light of different update scenarios, CATCAM proposes an interval-based scheduling scheme along with the hierarchical encoding scheme. Each subtable maintains the maximum priority among its rules. In this way, the entire range of possible priorities is segmented into a series of non-overlapping intervals by these maximum priorities. A new rule first locates its interval range according to its priority. Since intervals are non-overlapping, each priority only belongs to one of them, whose associated subtable is the one to be inserted.

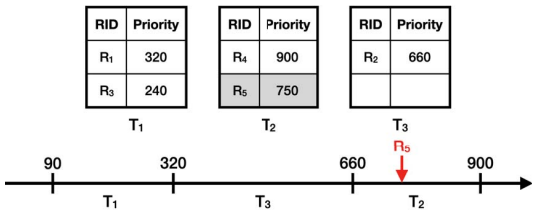


Fig. 8. According to its priority in the interval range, R_5 is inserted into T_2.

By checking the availability of relevant subtables, rule insertion can be classified into three scenarios. *First*, the subtable to be inserted is available. In this case, the rule can be safely inserted into any empty slot. The interval range stays the same as the boundary remains unchanged, as shown in Fig. 8.

Second, if the subtable is already full, an existing rule in the current subtable has to be evicted to make room for the new rule. Since this rule will eventually be inserted into another subtable, to keep the priority relation between subtables, it should be the one with the highest or lowest priority. In our case, we choose the former. The new rule, therefore, takes the place of the evicted rule. As for the evicted rule, its priority must be within the following interval range, so it will be reinserted if the associated subtable is available. The interval range is adjusted due to rule reallocation while the priority relation between subtables is kept, as shown in Fig. 9.

Third, when the next subtable for the evicted rule is full as well, a naive approach may lead to the "reallocation chain". Suppose there are k subtables in the hierarchy, it will incur $O(k)$ rule reallocation. Note that because of the global priority matrix, the priority relation between subtables is no longer related to the placement of subtables. To break the "chain", an empty subtable is assigned to accommodate the evicted rule, whose priority should be between the original subtable and the one that follows. Similar to rule insertion, the update of the global priority matrix can be done by comparing the maximum priority of the new subtable (i.e., the priority of the evicted rule) against other subtables. As a result, the interval range of the original subtable is segmented into two with their order encoded in the global priority matrix, as shown in Fig. 10. Meanwhile, the new subtable will accommodate further evicted rules from the original subtable. Such flexibility enables CATCAM to scale from a default subtable with the entire priority range to hundreds of subtables with segmented intervals. Given that at most one rule is reallocated in any scenario, rule insertion in the hierarchy can be completed in $O(1)$ time, which proves the scalability of CATCAM.

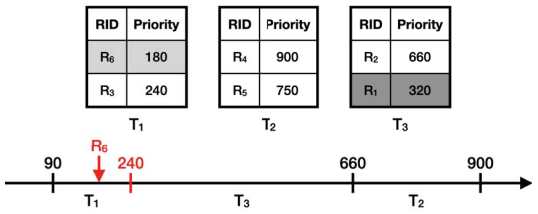


Fig. 9. R_6 is inserted into T_1. Since T_1 is full, R_1 is reallocated to T_3. The interval range of T_1 is adjusted accordingly.

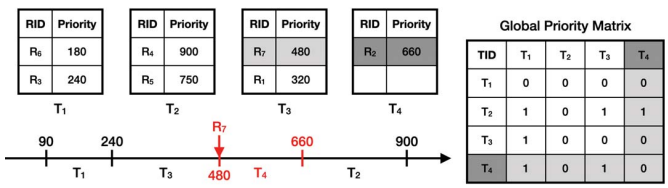


Fig. 10. R_7 is inserted into T_3. Since T_2 is full, T_4 is assigned to accommodate R_2 evicted from T_3. The global priority matrix is updated accordingly so that $T_3 < T_4 < T_2$.

C. Trace Maximum Priority

A significant step of the scheduling scheme is to maintain the maximum priority of each subtable. Only saving the present maximum priority is not enough since it has to be updated when the rule with the highest priority is evicted or deleted. However, directly recording the second-highest priority can be costly: a sorted list or a min-max heap of priorities has to be maintained in case of further eviction. To avoid complexity, CATCAM locates the rule with the highest priority in a subtable with a simple insight. Assume all the rules in a subtable are matched during lookup, the rule to report is the one with the highest priority. By manually setting the local match vector to all $true$, the maximum priority can be updated according to the result of the priority decision.

V. In-Memory Implementation

A small-scale priority matrix could be easily implemented as an array of registers. But even for a moderate-scale ruleset containing hundreds of rules, the timing of such register array would become extremely hard to constrain considering large amounts of nor operations performed during both lookup and update. Fortunately, the emerging PIM technique provides massive bit-level parallelism, which makes it possible to implement a large-scale priority matrix at a low cost.

This section introduces how CATCAM can be realized in 8T SRAM arrays. Section V-A describes the mapping of the priority matrix to 8T SRAM arrays and how the matrix-based priority scheme is performed. Since conventional SRAM only supports row-wise write, to update the priority matrix column-wise during rule insertion, a dual-voltage write scheme is adopted, as discussed in section V-B. The match matrix derived from 8T SRAM arrays is introduced in section V-C as an alternative to the conventional 16T TCAM design.

A. Data Layout

Taking into account the similarity between the priority decision and PIM paradigm, the mapping from the priority matrix to 8T SRAM arrays is obvious. The priority decision process can be simplified as bit-wise nor among rows (section III-B), which matches the bit-line nor operation enabled by the PIM

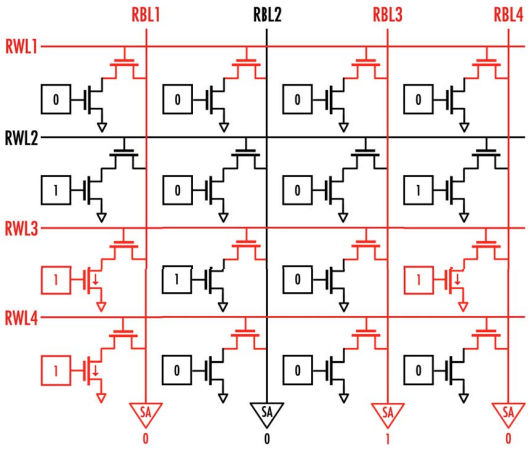


Fig. 11. The priority matrix in Fig. 5 is mapped to an 8T SRAM array. Since R_0, R_2 and R_3 are matched, RB1/RBL3/RBL4 are pre-charged first, then RWL1/RWL3/RWL4 are activated. Transistors in red are on. The output of the sense amplifiers (SA) indicates that R_2 has the highest priority. The write path is ignored for readability.

technique in 8T SRAM arrays (section II-C). In this way, each priority matrix is mapped to an 8T SRAM array of the same dimension, and each boolean element is stored in a bit-cell. In Fig. 11, the priority matrix of Fig. 5 is mapped to a 4×4 array.

To perform priority decision, the match vector is applied as the input to the priority matrix. Given that entries in the priority matrix are arranged the same order as the match matrix or the match vector, if the ith bit in the match vector is $true$, the ith read word-line (RWL) is activated. Only the read bit-lines (RBL) corresponding to matched entries are pre-charged beforehand. If any of the bit-cells connected to the ith RBL carries a '1', the RBL discharges, meaning other matched entries have a higher priority than the ith entry. Meanwhile, if the jth entry is not matched, the jth RBL is grounded to '0'. Therefore, the row-wise nor of the priority decision in Fig. 5 is performed, whose results are sensed on the RBLs, hence the output of the 8T SRAM array is the one-hot report vector.

B. Column-wise Write

However, in terms of rule insertion, existing SRAM designs are challenged since the update of the priority matrix requires writing both a row and a column (section III-A). Since conventional SRAM only features row-wise write, i.e., single row write at a time, the update of a column has to write all rows sequentially, which incurs $O(n)$ time-complexity.

To meet the $O(1)$ time requirement, we adopt a dual-voltage scheme [21] to support column-wise write. The idea is to write all '1's and '0's in the data separately. To do so, the data is applied to the WWLs instead of the conventional WBLs while the address determines the column to be written, as shown in Fig. 12. The write word-lines (WWL) for bit-cells where '1's ('0's) have to be written are enabled, and the write bit-line (WBL)/write bit-line bar (WBLB) for the column are driven to write '1's ('0's). To prevent data corruption

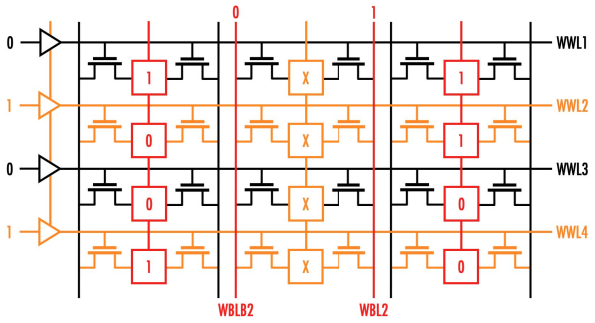


Fig. 12. '1's are being written to the second column. The cross-couple voltage of the second column is lowered and WWL2/WWL4 are under-driven, colored in orange. WBL2/WBLB2 for the second column are driven strongly. The read path is ignored for readability.

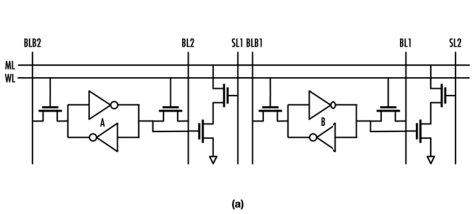
in other columns due to multi-row access, the WWLs are under-driven to bias against the pseudo-write. Accordingly, the cross-couple voltage of the column-under-write is lowered to allow writing with low WWL voltage. Data in other columns are protected by keeping their cross-couple voltage high. Meanwhile, common row-wise write of SRAM remains intact.

Column-wise write takes two cycles to write '1's and '0's in the data respectively. Combined with one row-wise write, the update overhead of the priority matrix is three cycles per rule, which can be completed in $O(1)$ time.

C. Match Matrix Design

The *match matrix* of Fig. 5 has the same function as a TCAM excluding the priority encoder. We observe that TCAM is essentially a form of PIM: massive comparisons are performed in parallel within memory array without retrieving any data. Building up from it, by swapping the RWLs and RBLs of the original 8T SRAM cell, the 8T SRAM array can also be repurposed to implement the TCAM-like match matrix to facilitate the integration of CATCAM.

Fig. 13(a) shows an equivalent design to the 16T TCAM cell composed of two transposed 8T SRAM cells. It uses the same encoding scheme as TCAM for storing and searching data, as shown in Fig. 13(b). To represent the ternary states of stored data, each memory cell contains two bits. Ternary bit 0, 1, and $*$ is encoded as 10, 01, and 00, respectively. They are stored separately in two 8T SRAM cells marked A, B in Fig. 13.



Stored Data	A	B
0	1	0
1	0	1
Don't Care	0	0
Search Data	**SL1**	**SL2**
0	0	1
1	1	0

(b)

Fig. 13. The bit-cell design and encoding scheme for the match matrix.

During lookup, the encoded search string is applied to the search-lines (SL) while the match-lines (ML) output the matching results. Note that the opposite encoding scheme for stored and search words. If the input bit is 1, lookup is conducted by setting SL1 high and SL2 low. In this case, the pre-charged ML will not be pulled down if the stored bit is 1 or $*$. It will discharge only if A = 1, which means a mismatch to the stored bit 0. Similarly, the input bit 0 is compared by setting SL1 low and SL2 high. As a result, $xnor$ between the input bit and the stored bit is performed.

The match matrix organizes words row-wise like SRAM. Bit-cells can be read or written thorough the 6T part. An entry is matched if its stored word matches every bit of the search string, as the bit-wise $xnor$s are wire anded on the MLs. The result is sensed to generate the match vector.

VI. Overall Architecture

Fig. 14 shows the overall architecture of CATCAM. It is composed of a bunch of subtables, a global priority matrix, a metadata cache and a task scheduler. Each subtable includes a match matrix and a priority matrix. The hierarchical architecture is interconnected through the global bus, where each subtable is connected to the centralized control logics. Requests consisting of lookups and updates are sent to the task scheduler. For lookups, queries are broadcast to all subtables for entry matching. According to the arbitration from the global priority matrix, one of these subtables proceeds with further priority decision to report the match. For updates, rule insertion is scheduled according to the metadata of subtables. Apart from crucial subtable metadata such as maximum priorities, vacancies and successor pointers, all the priorities of existing rules in each subtable are kept in a priority store in the metadata cache. During insertion, the priority of the rule is forwarded to the priority store and compared against existing rules of the subtable to generate the priority vector for update. The assignment of a subtable is done in a similar fashion. For example, to reallocate a given rule, its fields in the match matrix have to be read out along with its priority in the priority store before it is reinserted into the next subtable.

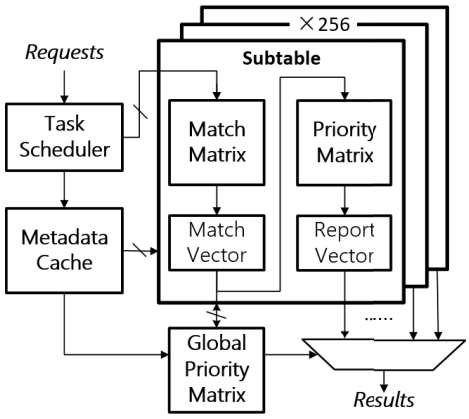


Fig. 14. The overview of CATCAM architecture.

Although the local priority matrix and the global priority matrix can be queried simultaneously to reduce latency, in favor of energy efficiency, only one local priority matrix will be queried after the arbitration of the global priority matrix. Furthermore, CATCAM features a three-stage pipeline for lookup (entry matching, global priority decision, local priority decision) so the extra cycle will not affect overall throughput. In practice, lookup queries are interspersed with update requests. Since rule update is atomic and may vary in different scenarios, a FIFO is used to buffer the requests. As discussed in IV-B, both of them can be responded in $O(1)$ time instead of further blocking lookup queries.

VII. Evaluation Methodology

In this section, we first describe our benchmark workloads, followed by an explanation of the experimental setup, and finally give the system configuration of CATCAM.

Packet classification workloads. We evaluate the proposed scheduling scheme of CATCAM with ClassBench [50], a widely used benchmark for packet classification. It generates various types of rulesets, including Access Control List (ACL), Firewall (FW), and IP Chain (IPC), with different sizes ranging from 1K to 40K. For fair comparison, we generate all the synthetic rulesets each containing 1K, 10K, 20K, 40K entries. Although ClassBench itself does not specify an update stream, we follow the approach in prior studies and randomly selecting rules in the ruleset to update. Rule insertion and deletion account for half each in the test traces so that the total number of entries remains unchanged. Since rule insertion in CATCAM is based on priorities, they are extracted and fed to the scheduler. We issue 1K update requests to observe the average performance.

Experimental setup. To estimate the delay, power and area of the priority matrix and the match matrix, we adopt the standard 8T SRAM bit-cell for the TSMC 28 nm technology. The nominal voltage for this process is 0.9 V. Compared to a regular memory compiler, we custom the decoding and control logics for in-memory operations. An additional column decoder and a common header switch is added to support column-wise write in the priority matrix. For the match matrix, the original bit-cell is modified into its transposed counterpart based on a SPICE model, which proves to have similar delay and energy features while still meet push-rule to be compact. All the modifications are verified with Design Rule Checking (DRC) and Layout Versus Schematic (LVS), and Monte Carlo simulations show a stability of more than six sigma robustness. Memory IPs are instantiated from memory arrays then integrated with peripheral control logics synthesized from System Verilog to implement CATCAM.

System configuration. Table I presents the key parameters of memory IPs in CATCAM. The size and number of match matrices and priority matrices are determined according to the requirements of dRMT [6], a state-of-the-art architecture for programmable switches. dRMT is programmed with P4 [4], a programming language for protocol-independent packet processing. It uses a 4K bits packet header vector consisting of

TABLE I
MEMORY PARAMETERS

Match Matrix				Priority Matrix			
Size	Computing delay	Energy	Area	Size	Compute delay	Energy	Area
256×160	585 ps	0.78 fJ/bit	0.039 mm^2	256×256	505 ps	0.59 fJ/bit	0.031 mm^2
Incremental	Access delay	Read energy	Write energy	Incremental	Access delay	Read energy	Write energy
63.3 fJ	461 ps	26.7 pJ	35.6 pJ	148.6 fJ	479 ps	22.7 pJ	30.3 pJ
Number of Match Matrix				Number of Priority Matrix			
4×256				$256 + 1$			

224 fields. From this vector, dRMT extracts fields to up to 640 bits of match search keys to fit into TCAM.

The match matrix of a subtable is organized into four SRAM subarrays which can be queried in parallel. Each subarray features 160 bits width and 256 bits depth to form a total width of 640 bits. During lookup, a 640 bits key is broken into four 160 bits keys and sent to four subarrays respectively, whose results are bitwise anded to generate the match vector. Then the match vector is fed to the priority matrix, which consists of a 256×256 SRAM array. In addition, considering OpenFlow specifies the priority level of flow entry with a 16-bit field, a 256×16 register file is utilized as the priority store for a subtable. The prototype of CATCAM consists of 256 subtables that can accommodate 64K entries. In this way, the total capacity of CATCAM is 40 Mb, which is comparable to a commercial off-the-shelf TCAM [37]. A global priority matrix, also made up of a 256×256 SRAM array, is connected with these subtables. Peripheral control logics are evaluated by taking this configuration into consideration. For simplicity, we do not take into account the impact of wires interconnecting subtables and control logics.

VIII. RESULTS

In this section, we first analyze the overall performance of CATCAM, followed by comparisons to TCAM update and packet classification algorithms, and finally compare power and area overhead to existing TCAM designs.

A. Overall Performance

The overall performance of CATCAM is dictated by the clock frequency and Cycles Per Request (CPR). Since CATCAM is an in-memory architecture, memory operations account for most of the overhead during the process, therefore the clock period is mainly determined by the memory delay while CPR depends on the number of memory operations involved. The computing delay of the match matrix is the time taken from the input of the encoded header fields to the output of the match vector. Similarly, the computing delay of the priority matrix is the time taken from the match vector to the report vector. The access delay is the time to read or write a word row-wise and doubles for column-wise write. Based on these parameters, we set the operating frequency of CATCAM to 500 MHz, leaving enough margin for the control logics in each cycle.

TABLE II
CATCAM METRICS

Frequency	500 MHz
Power	16.7 W
Area	48.8 mm^2
Capacity	40 Mb
Configuration	$(160b \times 4) \times 256 \times 256$
Lookup Rate	500 MOPS
Update Rate	100 MOPS

Since lookup is fully pipelined to CATCAM, its amortized cost per request is 1 cycle. Rule deletion is trivial and takes 1 cycle to invalidate the corresponding entries. The update of the match matrix takes 1 cycle, the update of the priority matrix takes 3 cycles considering both row-wise write and column-wise write are required. Since they can happen in parallel, the cost of a rule insertion is 3 cycles. In case of rule reallocation between subtables, existing rules have to be read out first, adding an extra cycle. Different subtables of CATCAM can be updated simultaneously so both rules can be inserted at the same time. In case of the assignment of a new table, the update of the global priority matrix can be overlapped with that of the local priority matrix. Finally, updating the maximum priority of the subtable takes one more cycle. Therefore at most 5 cycles are considered for any update request. According to further benchmarking that will be discussed later, 28% of update requests can be accomplished in 3 cycles with the remaining 72% in 5 cycles, indicating an average CPR of 4.4. In conclusion, CATCAM supports 500 million lookup queries or 100 million update requests per second.

B. Comparisons of Update and lookup Performance

Table III and Table IV show the update performance of CATCAM in comparison to state-of-the-art TCAM update algorithms across the same benchmarks. FastRule (FR) [43], RuleTris (RT) [53], Partial Order Theory (POT) [15] and TreeCAM [52] are the latest TCAM firmware optimizations focusing on incremental update. Both of them utilize the minimum dependency graph, a kind of Directed Acyclic Graph (DAG) which describes rule dependency, to decrease TCAM entry movements during update. Therefore, two criteria

TABLE III
COMPARISON OF TCAM/CATCAM UPDATE COST

Ruleset	Size	Naive	FastRule	RuleTris	Partial Order Theory		TreeCAM		CATCAM	
		Average	Average	Average	Average	Maximum	Average	Maximum	Average	Maximum
ACL	1K	500	1.6	0.9	1.2	10	6.3	49	0.1	1
	10K	6000	1.6	1	1.2	30	7.3	57	0.35	1
	20K	NA	1.7	1	1.3	24	4.5	57	0.4	1
FW	1K	470	1.9	0.9	4	14	11	41	0.1	1
	10K	3000	2.3	1	4.8	19	6.5	57	0.35	1
	20K	NA	2.1	1.1	5	24	5.3	57	0.4	1
IPC	1K	470	1.6	0.9	1.1	6	12.6	57	0.1	1
	10K	6600	1.7	1	1.7	19	9.6	57	0.35	1
	20K	NA	1.6	1.1	1.2	23	4.5	57	0.4	1

TABLE IV
COMPARISON OF FIRMWARE TIME

Ruleset	Size	Naive	FR	RT	POT	CATCAM
ACL	1K	300 ms	35 us	0.5 ms	7 us	6.4 ns
	10K	3500 ms	35 us	400 ms	57 us	7.4 ns
	20K	NA	50 us	1500 ms	149 us	7.6 ns
FW	1K	280 ms	35 us	0.6 ms	11 us	6.4 ns
	10K	1800 ms	35 us	500 ms	215 us	7.4 ns
	20K	NA	50 us	2000 ms	573 us	7.6 ns
IPC	1K	300 ms	40 us	0.5 ms	7 us	6.4 ns
	10K	3500 ms	40 us	400 ms	70 us	7.4 ns
	20K	NA	40 us	2000 ms	277 us	7.6 ns

concerning the performance, the firmware time and the update cost, are reported separately in Table IV and Table III.

The total update time of TCAM-based solutions is the aggregation of the firmware time and the TCAM update time, while for CATCAM the "firmware time" in Table IV refers to the total update time derived from the update cost in Table III. The extra computation effort prior works spend to generate the dependency graph and calculate the update sequence is measured in the *firmware time*. For example, they need a compiler to convert each update into DAG and a scheduler to convert DAG update back to TCAM entry movements. Then the TCAM/CATCAM *update cost* is the number of rule reallocations. For CATCAM, it is either 0 or 1 depending on the availability of subtables. For others, it is the number of entry movements in the update sequence. Since the actual time to conduct rule updates in TCAM is the number of TCAM moves times the clock period of the specific TCAM (e.g., 2.5 ns for 400 MHz), in Table III update time from their original papers are normalized to TCAM moves to remove the impact of hardware differences. For software-based Open vSwitch, by default, it allows 65,000 rules with nearly instantaneous rule installation, which is close to what is reported in a previous benchmark [28].

From Table III and Table IV it can be observed that although prior works manage to reduce the TCAM update cost, their performance is hindered by the prohibitive firmware time. Because they are still based on the premise that TCAM sorts rule priorities by physical addresses, they exploit the dependency relationship with the newly inserted entry to minimize the number of rule reallocations. However, the compilation of minimum dependency information is complicated and time-consuming, especially for large rulesets, which makes up most of the firmware time. Their performance relies on the characteristics of rulesets and is non-deterministic among updates. In contrast, CATCAM inserts rules based on their priorities rather than dependencies. With the encoding scheme, rule priorities and subtable priorities are decoupled from their physical addresses, ensuring that at most one rule is reallocated for an update. The algorithm/architecture co-design eliminates the firmware overhead and guarantees $O(1)$ time update latency. Therefore, its performance is deterministic and not subject to ruleset characteristics.

However, due to the exhaustion of empty entries, the ratio of rule reallocation rises as the number of rules in the table increases. We evaluate the worst case by generating as many rules and feeding them to CATCAM until an insertion fails. Statistics show that for our prototype with 64K entries, it can accommodate around 50K rules (inflation due to range expansion excluded), with an occupancy of 78%. About 28% of them are inserted without rule reallocation, bringing an average update time of 8.9 ns. Overall, CATCAM provides an overwhelming speedup between three orders of magnitude and six orders of magnitude over previous optimal algorithms.

Since our goal is to achieve fast update without jeopardizing lookup performance, we compare CATCAM with existing solutions for packet classification in terms of throughput. As shown in Fig. 15, since the match matrix shares the same architecture as TCAM, CATCAM can offer equivalent search capability. Prior studies have shown that TCAM is an order of magnitude faster than Open vSwitch and HALO [57], its cache accelerator in tuple space search due to its excellence in wildcard match.

These comparisons reveal a fundamental trade-off between

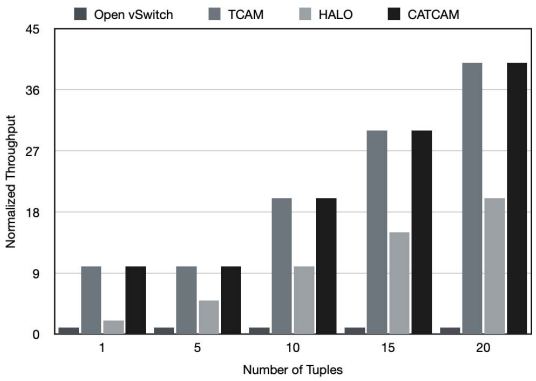


Fig. 15. Comparison of lookup performance.

lookup speed and update effort in packet classification. The hardware-based solution like TCAM features rapid parallel lookup but involves expensive and inflexible update operations while the software-based approach like Open vSwitch is more scalable but limited in lookup efficiency. To the best of our knowledge, CATCAM is the first work to achieve $O(1)$ time update while maintaining $O(1)$ time lookup, making it a promising candidate for future SDN switches.

C. Energy Consumption and Area Overhead

This section discusses the energy consumption and area overhead of CATCAM. Although the real energy consumption of CATCAM is subject to the actual workload for lookup, it mainly depends on two factors. *First*, the number of valid entries to match in a subtable. Each valid entry corresponds to an ML in the match matrix, which is pre-charged during lookup. *Second*, the number of matched entries in a subtable. Similarly, each matched entry is related to a pre-charged RBL and an activated RWL in the priority matrix. Such incremental cost is also reported in Table I. Fig. 16 shows the energy consumption of the match matrix and the priority matrix with an increasing number of valid and matched entries in a subtable. Since the cost of the control logics inside memory is amortized over entries, the energy consumption per rule is decreasing as the number of entries increase. In case of a fully loaded and matched subtable, the energy cost is 0.78 fJ per ternary bit for the match matrix and 0.59 fJ per bit for the priority matrix. However, in practice, only two priority

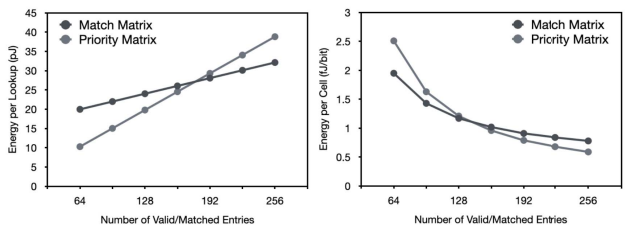


Fig. 16. Energy consumption with increasing valid/matched entries in a subtable.

TABLE V
COMPARISON WITH EXISTING TCAM DESIGNS

	CATCAM	Jeloka [21]	Nil [37]	Arsovski [3]
Technology	28 nm	28 nm	28 nm	32 nm
Bit cell	16T	12T	16T	16T
Area/cell	0.71 um^2	0.304 um^2	0.625 um^2	n.a.
Frequency	500 MHz	370 MHz	400 MHz	1 GHz
Energy/search	0.78 fJ/bit	0.74 fJ/bit	n.a.	0.58 fJ/bit
Array size	256×160	32×64	$4k \times 80$	128×128

matrices in CATCAM are visited per query and they happen in different pipeline stages, meaning at most two priority matrices are active at the same time. Considering the large amount of match matrices in CATCAM, the energy consumed by priority matrices is nearly negligible.

The maximum power consumption of CATCAM is 16.7 W, of which the match matrices consume 16.4 W while the priority matrices consume 0.1 W. The area overhead of CATCAM is 48.8 mm^2, of which the match matrices cover 40.2 mm^2 while the priority matrices cover 8.1 mm^2. Therefore, the priority matrices only add 0.3% power consumption and 20% area overhead towards the budget of match matrices. Experiments also show that the peripheral control logics account for less than 1% in terms of energy or area. Putting it all together, in a fully loaded CATCAM, the energy and area cost per ternary bit is 0.782 fJ and 0.71 um^2.

Table V compares CATCAM to some taped-out TCAM designs. For the record, since the match matrix in CATCAM has the same function as a TCAM, CATCAM is also compatible with existing TCAM designs. The reason we design our own match matrix is for the evaluation purpose. From Table V we can safely conclude that our evaluation of the match matrix is reasonable. Compared to the best case in Table V, CATCAM incurs 34% energy and 134% area overhead. Since the transistor density of conventional TCAM is limited by the power density, it is profitable for CATCAM to trade area overhead for flexible update, which lowers the power density of current TCAM designs.

IX. RELATED WORKS

This section introduces related works on the packet classification algorithms, especially on their recent resurgence driven by SDN. To enable high-performance packet classification, prior works propose both software-based and hardware-based packet classification. The primary challenge is simultaneously achieving fast lookup and fast update. TCAM with the conventional priority encoder (PE) features $O(1)$ lookup and $O(n)$ insertion, where n is the number of stored rules.

In the past, fast update was not as important as fast lookup for statically configured networks. Some researches essentially ignored update to drastically fasten lookup [47]. Tuple Space Search (TSS) [48] was the precursor to trade the lookup time for fast update. It partitioned the ruleset based on the pattern of wildcards used for each field (tuple) and group all rules that

share the same tuple. TSS features $O(d)$ lookup and $O(1)$ insertion, where d is the number of tuples, but may have performance concern for large rulesets due to tuple expansion.

The rapid deployment feature of SDN [39], exemplified by OpenFlow [33] and Network Function Virtualization (NFV) [13], emphasizes the importance of fast update, which becomes the major design goal of packet classification algorithms nowadays.

Software-based packet classification emerges as the mainstream solution to fast update. The *de novo* framework Open vSwitch (OvS) [42] for networking in virtual environments like multi-tenancy data centers, combines TSS with hash tables and software-based caches, can handle up to 65,000 updates per second with CPUs [28]. Optimizations on both CPU architectures, like DDIO [19] and SR-IOV [8], and OS infrastructures, like DPDK [18], are proposed to exploit modern multi-core processors for packet classifications. Recent researches on software-based approaches implement the rulesets as decision tree [56] or similar divide-and-conquer data structure [7]. Apart from the rule replication problem during update, traversal of complicated data structures may incur long search paths. In general, current best practice in software-based packet classification algorithms achieves $O(d + logn)$ lookup, insertion and deletion in average cases.

In terms of **hardware-based packet classification**, prior studies typically explore three approaches.

Optimizations on TCAM. Although TCAM is expensive, power-hungry and slow in update, their extremely fast lookup ensures that it is still the *de facto* hardware solution to high-speed packet classification [5], [6]. Numerous researches explore how to exploit TCAM for rapid deployment. Some of them seek to reduce the number of insertions required for updated rulesets. For example, modular composition compilers such as CoVisor [22] and RuleTris [53] minimize the number of rule updates sent to switches through eliminating redundant updates; and Dionysus [23] reduces multi-switch policy update latency caused by suboptimal scheduling. Others aim to reduce the number of moves required by insertion. Priority-Decision [51] utilizes the length information of the matched patterns to eliminate the need for PEs, but is limited to the longest prefix matching. Existing TCAM solutions aim to decrease in rule moves through the interleaved layout of empty entries [46] or utilizing the minimum dependency graph to avoid unnecessary moves during insertion [43]. But the former approaches waste precious TCAM space without improving the worst-case complexity, and the latter one has to calculate the dependency graph of rules, which is prohibitively time-consuming. To reconcile the need between lookup and update, TreeCAM [52] proposes dual versions of decision trees to decouple lookup and update, and perform well in both lookup efficiency and update effort. CacheFlow [27] features a hierarchical flow table design between TCAM and SRAM to scale up, with cover-set rules to decrease unnecessary flow entry movements between them. Again, the extra computing effort to the dependency relation of rulesets amortized over insertions will deteriorate the overall performance.

RRAM-based TCAM. TCAM based on the emerging non-volatile memory technology - Resistive Random Access Memory (RRAM) benefits from the small cell size, excellent scalability and fast access speed [58]. Associative Processing (AP) [25], [55] with RRAM-based TCAM is also a promising PIM paradigm that combines data storage and data processing, and supports massively parallel SIMD operations in-situ by decomposing arbitrary computations into a sequence of primitive memory operations [59]. Although RRAM technology can be used to implement CATCAM with many advantages including native column-wise write support, our main concern is the limited resistive memory endurance (10^{12}) [36], which fails within hours for our design featuring high-speed update. However, the number is likely to grow in the future [10], we thereby leave RRAM-based CATCAM for future exploration.

Hardware-accelerated software-based solutions. Recent trends on domain-specific accelerators also address the packet classification algorithms. The parallelism of existing hardware accelerators such as GPUs [11], [14], FPGAs [30] and ASICs [57] are exploited to offload the software-based packet classification. These methods share the limitation of pure software-based solutions in that their lookup performance is far from satisfying.

X. Conclusion

Motivated by the recent resurgence of researches on the inefficient update for packet classification algorithms, this paper reinvents the underlying mechanism for TCAM. The emerging PIM paradigm provides a new perspective on this problem. By decoupling rule priorities from physical addresses, we propose the matrix-based priority encoding scheme that can be implemented in-memory with 8T SRAM arrays. With the hierarchical scalability, we propose CATCAM that supports up to hundreds of thousands of rules while enabling $O(1)$ time lookup and update. Evaluation shows that CATCAM performs at least three orders of magnitude faster than other TCAM update solutions with reasonable power and area overhead.

CATCAM not only provides fast update and better scalability but also gives better responsive guarantee than TCAM. These features can greatly simplify the design of SDN control plane for hardware switches. In this way, CATCAM may serve to drive the evolution of network architecture.

Moreover, we hope this work would shed light on the new research direction for Processing In-Memory and make a broader impact on the PIM technique to more than data-intensive applications.

Acknowledgments

We thank the anonymous reviewers and the shepherd for their thoughtful comments. This work is supported in part by the National Natural Science Foundation of China (Grant No. 61834002, No. 61672317), and in part by the National Key R&D Program of China (Grant No. 2018YFB2202101), and in part by the National Science and Technology Major Project of the Ministry of Science and Technology of China(Grant No. 2018ZX01027101-002).

REFERENCES

[1] S. Aga, S. Jeloka, A. Subramaniyan, S. Narayanasamy, D. Blaauw, and R. Das, "Compute caches," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2017, pp. 481–492.

[2] A. Agrawal, A. Jaiswal, C. Lee, and K. Roy, "X-sram: Enabling in-memory boolean computations in cmos static random access memories," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 12, pp. 4219–4232, 2018.

[3] I. Arsovski, T. Hebig, D. Dobson, and R. Wistort, "A 32 nm 0.58-fj/bit/search 1-ghz ternary content addressable memory compiler using silicon-aware early-predict late-correct sensing with embedded deep-trench capacitor noise mitigation," *IEEE journal of solid-state circuits*, vol. 48, no. 4, pp. 932–939, 2013.

[4] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[5] 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," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 99–110, 2013.

[6] S. Chole, A. Fingerhut, S. Ma, A. Sivaraman, S. Vargaftik, A. Berger, G. Mendelson, M. Alizadeh, S.-T. Chuang, I. Keslassy *et al.*, "drmt: Disaggregated programmable switching," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017, pp. 1–14.

[7] J. Daly, V. Bruschi, L. Linguaglossa, S. Pontarelli, D. Rossi, J. Tollet, E. Torng, and A. Yourtchenko, "Tuplemerge: Fast software packet processing for online packet classification," *IEEE/ACM transactions on networking*, vol. 27, no. 4, pp. 1417–1431, 2019.

[8] Y. Dong, X. Yang, J. Li, G. Liao, K. Tian, and H. Guan, "High performance network virtualization with sr-iov," *Journal of Parallel and Distributed Computing*, vol. 72, no. 11, pp. 1471–1480, 2012.

[9] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaaauw, and R. Das, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 383–396.

[10] K. Eshraghian, K.-R. Cho, O. Kavehei, S.-K. Kang, D. Abbott, and S.-M. S. Kang, "Memristor mos content addressable memory (mcam): Hybrid architecture for future high performance search engines," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 8, pp. 1407–1417, 2010.

[11] Y. Go, M. A. Jamshed, Y. Moon, C. Hwang, and K. Park, "Apunet: Revitalizing {GPU} as packet processing accelerator," in *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, 2017, pp. 83–96.

[12] P. Gupta and N. McKeown, "Algorithms for packet classification," *IEEE Network*, vol. 15, no. 2, pp. 24–32, 2001.

[13] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, 2015.

[14] S. Han, K. Jang, K. Park, and S. Moon, "Packetshader: a gpu-accelerated software router," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, pp. 195–206, 2010.

[15] P. He, W. Zhang, H. Guan, K. Salamatian, and G. Xie, "Partial order theory for fast tcam updates," *IEEE/ACM Transactions on Networking*, vol. 26, no. 1, pp. 217–230, 2017.

[16] C.-H. Huang, J.-S. Wang, and Y.-C. Huang, "Design of high-performance cmos priority encoders and incrementer/decrementers using multilevel lookahead and multilevel folding techniques," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 1, pp. 63–76, 2002.

[17] D. Y. Huang, K. Yocum, and A. C. Snoeren, "High-fidelity switch models for software-defined network emulation," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 2013, pp. 43–48.

[18] Intel Corporation, "Data Plane Developer Kit (DPDK)." [Online]. Available: https://software.intel.com/en-us/networking/dpdk

[19] ——, "Intel® Data Direct I/O Technology." [Online]. Available: https://www.intel.com/content/www/us/en/io/data-direct-i-o-technology.html

[20] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined wan," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 3–14, 2013.

[21] S. Jeloka, N. B. Akesh, D. Sylvester, and D. Blaauw, "A 28 nm configurable memory (tcam/bcam/sram) using push-rule 6t bit cell enabling logic-in-memory," *IEEE Journal of Solid-State Circuits*, vol. 51, no. 4, pp. 1009–1021, 2016.

[22] X. Jin, J. Gossels, J. Rexford, and D. Walker, "Covisor: A compositional hypervisor for software-defined networks," in *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, 2015, pp. 87–101.

[23] X. Jin, H. H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer, "Dynamic scheduling of network updates," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 539–550, 2014.

[24] M. Kang, E. P. Kim, M.-s. Keel, and N. R. Shanbhag, "Energy-efficient and high throughput sparse distributed memory architecture," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2015, pp. 2505–2508.

[25] R. Kaplan, L. Yavits, and R. Ginosar, "Prins: Processing-in-storage acceleration of machine learning," *IEEE Transactions on Nanotechnology*, vol. 17, no. 5, pp. 889–896, 2018.

[26] R. Kaplan, L. Yavits, R. Ginosar, and U. Weiser, "A resistive cam processing-in-storage architecture for dna sequence alignment," *IEEE Micro*, vol. 37, no. 4, pp. 20–28, 2017.

[27] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "Cacheflow: Dependency-aware rule-caching for software-defined networks," in *Proceedings of the Symposium on SDN Research*, 2016, pp. 1–12.

[28] M. Kuźniar, P. Perešíni, and D. Kostić, "What you need to know about sdn flow tables," in *International Conference on Passive and Active Network Measurement*. Springer, 2015, pp. 347–359.

[29] M. Kuźniar, P. Perešíni, D. Kostić, and M. Canini, "Methodology, measurement and analysis of flow table update characteristics in hardware openflow switches," *Computer Networks*, vol. 136, pp. 22–36, 2018.

[30] B. Li, K. Tan, L. Luo, Y. Peng, R. Luo, N. Xu, Y. Xiong, P. Cheng, and E. Chen, "Clicknp: Highly flexible and high performance network processing with reconfigurable hardware," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 1–14.

[31] H. Li, T. Huang, T. Yang, W. Li, and G. Zhang, "A fast flow table engine for open vswitch with high performance on both lookups and updates," in *Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos*, 2019, pp. 125–127.

[32] R. Mahajan and R. Wattenhofer, "On consistent updates in software defined networks," in *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, 2013, pp. 1–7.

[33] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[34] M. Moshref, M. Yu, A. Sharma, and R. Govindan, "vcrib: Virtualized rule management in the cloud," in *Presented as part of the*, 2012.

[35] A. Nag, C. Ramachandra, R. Balasubramonian, R. Stutsman, E. Giacomin, H. Kambalasubramanyam, and P.-E. Gaillardon, "Gencache: Leveraging in-cache operators for efficient sequence alignment," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 334–346.

[36] J. Nickel, "Memristor materials engineering: From flash replacement towards a universal memory," in *Proceedings of the IEEE IEDM Advanced Memory Technology Workshop*, 2011, pp. 1–3.

[37] K. Nii, T. Amano, N. Watanabe, M. Yamawaki, K. Yoshinaga, M. Wada, and I. Hayashi, "A 28nm 400mhz 4-parallel 1.6 gsearch/s 80mb ternary cam," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. IEEE, 2014, pp. 240–241.

[38] B. Niven-Jenkins, D. Brungard, M. Betts, N. Sprecher, and S. Ueno, "Requirements of an mpls transport profile," 2009.

[39] Open Networking Foundation, "SDN Technical Specifications." [Online]. Available: https://www.opennetworking.org/software-defined-standards/specifications/

[40] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (cam) circuits and architectures: A tutorial and survey," *IEEE journal of solid-state circuits*, vol. 41, no. 3, pp. 712–727, 2006.

[41] R. Panigrahy and S. Sharma, "Sorting and searching using ternary cams," *IEEE Micro*, vol. 23, no. 1, pp. 44–53, 2003.

[42] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The design and implementation of open vswitch," in *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, 2015, pp. 117–130.

[43] K. Qiu, J. Yuan, J. Zhao, X. Wang, S. Secci, and X. Fu, "Fastrule: Efficient flow entry updates for tcam-based openflow switches," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 484–498, 2019.

[44] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 323–334, 2012.

[45] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[46] D. Shah and P. Gupta, "Fast updating algorithms for tcam," *IEEE Micro*, vol. 21, no. 1, pp. 36–47, 2001.

[47] 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*, 2003, pp. 213–224.

[48] V. Srinivasan, S. Suri, and G. Varghese, "Packet classification using tuple space search," in *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, 1999, pp. 135–146.

[49] A. Subramaniyan, J. Wang, E. R. Balasubramanian, D. Blaauw, D. Sylvester, and R. Das, "Cache automaton," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, 2017, pp. 259–272.

[50] D. E. Taylor and J. S. Turner, "Classbench: A packet classification benchmark," *IEEE/ACM transactions on networking*, vol. 15, no. 3, pp. 499–511, 2007.

[51] H.-J. Tsai, K.-H. Yang, Y.-C. Peng, C.-C. Lin, Y.-H. Tsao, M.-F. Chang, and T.-F. Chen, "Energy-efficient tcam search engine design using priority-decision in memory technology," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 3, pp. 962–973, 2017.

[52] B. Vamanan and T. Vijaykumar, "Treecam: decoupling updates and lookups in packet classification," in *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies*, 2011, pp. 1–12.

[53] X. Wen, B. Yang, Y. Chen, L. E. Li, K. Bu, P. Zheng, Y. Yang, and C. Hu, "Ruletris: Minimizing rule update latency for tcam-based sdn switches," in *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2016, pp. 179–188.

[54] T. Yang, A. X. Liu, Y. Shen, Q. Fu, D. Li, and X. Li, "Fast openflow table lookup with fast update," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 2636–2644.

[55] L. Yavits, S. Kvatinsky, A. Morad, and R. Ginosar, "Resistive associative processor," *IEEE Computer Architecture Letters*, vol. 14, no. 2, pp. 148–151, 2014.

[56] S. Yingchareonthawornchai, J. Daly, A. X. Liu, and E. Torng, "A sorted partitioning approach to high-speed and fast-update openflow classification," in *2016 IEEE 24th International Conference on Network Protocols (ICNP)*. IEEE, 2016, pp. 1–10.

[57] Y. Yuan, Y. Wang, R. Wang, and J. Huang, "Halo: accelerating flow classification for scalable packet processing in nfv," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2019, pp. 601–614.

[58] Y. Zha and J. Li, "Liquid silicon-monona: A reconfigurable memory-oriented computing fabric with scalable multi-context support," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 214–228, 2018.

[59] ——, "Hyper-ap: Enhancing associative processing through a full-stack optimization," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 846–859.

[60] Y. Zhuo, C. Wang, M. Zhang, R. Wang, D. Niu, Y. Wang, and X. Qian, "Graphq: Scalable pim-based graph processing," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 712–725.