MegaTurbo: A Scalable FPGA-based Engine for MegaFlow Classifier in Open vSwitch

Sheng Lan*
lansh@pcl.ac.cn
Peking University
Shenzhen Graduate School
Shenzhen, China

Ying Li*
liy25@pcl.ac.cn
Southern University of
Science and Technology
Shenzhen, China

Zhongxian Liang*
liangzhx@pcl.ac.cn
Harbin Institute of
Technology
Shenzhen, China

Wenjun Li†
wenjunli@pku.org.cn
liwj@pcl.ac.cn✉
Pengcheng Laboratory
Shenzhen, China

Yao Xin‡
xinyaogzhu@gzhu.edu.cn
Guangzhou University
Guangzhou, China

Ying Wan
wy25@seu.edu.cn
Southeast University
Nanjing, China

Hui Li
lih64@pkusz.edu.cn
Peking University
Shenzhen Graduate School
Shenzhen, China

Weizhe Zhang*
wzzhang@hit.edu.cn
Harbin Institute of
Technology
Shenzhen, China

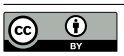
Abstract

Open vSwitch (OVS) is a key component in cloud and data center networks, yet its MegaFlow classifier imposes significant CPU overhead. Existing SmartNIC-based acceleration approaches for the MegaFlow classifier typically employ simplistic hardware offloading techniques, which exhibit limited scalability for dynamic, large-scale flow tables. Motivated by these challenges, we argue that a hardware accelerator specifically tailored for the MegaFlow classifier is necessary, forming the basis of our FPGA-based solution, MegaTurbo. The core innovations of MegaTurbo are threefold: (1) a scalable and hardware-friendly decision-tree based packet classification algorithm, specifically optimized for the structure of MegaFlow rules; (2) a novel hardware architecture incorporating multiple pipelined matching engines, designed to process multiple decision trees generated by the software algorithm in parallel; and (3) a heterogeneous framework composed of CPU and FPGA, which can work together to support online rule updates, with little and bounded impact on rule searching. Experimental results on a Xilinx Virtex UltraScale+ FPGA demonstrate that MegaTurbo achieves a sustained classification throughput of 500 MPPS while supporting dynamic rule updates at 300-500 KUPS on 100K-scale rulesets. These results not only validate the effectiveness of our domain-specific co-design approach, but also highlight the potential of FPGA-based SmartNICs to address the performance bottlenecks of software switches in large-scale cloud and data center networks.

CCS Concepts

• **Hardware → Networking hardware**; **Application specific integrated circuits**; • **Networks → Packet classification**.

*These authors are also affiliated with Pengcheng Laboratory.
†The first three authors are Ph.D. students, and they conducted this work under the supervision of their advisor and the corresponding author of this paper: Wenjun Li.
‡Yao Xin is the co-corresponding author of this paper.



FPGA '26, Seaside, CA, USA

ACM ISBN 979-8-4007-2079-6/2026/02
https://doi.org/10.1145/3748173.3779197

Keywords

SmartNIC, FPGA, Open vSwitch, MegaFlow, Packet Classification

ACM Reference Format:
Sheng Lan, Ying Li, Zhongxian Liang, Wenjun Li, Yao Xin, Ying Wan, Hui Li, and Weizhe Zhang. 2026. MegaTurbo: A Scalable FPGA-based Engine for MegaFlow Classifier in Open vSwitch. In *Proceedings of the 2026 ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '26), February 22–24, 2026, Seaside, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3748173.3779197

1 Introduction

The relentless growth of cloud computing and big data analytics has placed unprecedented demands on the underlying network infrastructure of modern data centers. To meet the requirements for agility, scalability, and centralized management, Software-Defined Networking (SDN) has emerged as the foundational paradigm. Within this framework, the virtual switch—a software-based data plane element—plays a pivotal role in connecting virtual machines and containers. Among these, Open vSwitch (OVS) has emerged as a widely adopted standard, serving as a critical component in major cloud platforms [37]. A central performance-determining element within OVS is the MegaFlow classifier (MFC), an intelligent cache designed to minimize expensive multi-stage packet-matching operations. However, this software-based classifier itself consumes substantial CPU resources, becoming a severe performance bottleneck that limits throughput and increases latency.

To reduce this CPU overhead, a promising approach is to offload the packet classification task to hardware accelerators embedded in SmartNICs [8, 31]. Existing solutions typically employ simplistic hardware offloading techniques and can be broadly categorized into two types: exact-match offloads and TCAM-based offloads. The former, widely adopted in commercial SmartNICs, offloads flow entries into hash tables for precise tuple matching [35, 36]. While effective for cached flows, this approach is fundamentally incapable of handling the wildcard patterns pervasive in MegaFlow rules, severely limiting its applicability. Some high-end SmartNICs utilize Ternary Content-Addressable Memory (TCAM) to support wildcards. However, TCAMs are notoriously expensive, power-hungry, and offer limited capacity (typically a few thousand entries), making them unsuitable for the large-scale, dynamic flow tables in cloud environments [23, 25, 27, 33, 34, 44, 56, 65, 66].

More recently, advanced research efforts have sought to overcome the limitations of earlier FPGA-based designs through more sophisticated architectural approaches. These can be broadly categorized into two classes: run-to-completion (RTC) architectures, such as TcbTree [59] and KickTree [58, 61], and pure pipeline architectures, such as ParaSplit [9] and MbitTree [51]. The former, while offering greater flexibility and ease of supporting dynamic rule updates, suffer from low lookup efficiency. To compensate, they often require multiple processing cores to accumulate performance, leading to high resource utilization. More critically, the non-deterministic nature of the RTC execution model results in variable and unpredictable latency for packet processing. This necessitates a complex and costly result reordering mechanism to restore packet order at the output, inevitably increasing overall latency and design complexity. The latter class—pure pipeline architectures—can achieve high lookup throughput with relatively low resource overhead, but they face significant challenges in supporting incremental rule updates. Importantly, neither architecture is specifically tailored to the distinct structure and frequent update patterns of modern switching abstractions such as the OVS MegaFlow cache, indicating considerable room for optimization.

Motivated by this gap, we argue that overcoming the MegaFlow bottleneck requires a dedicated hardware accelerator that moves beyond generic offloading or classification. A solution must be specifically architected through a holistic algorithm–hardware co-design approach that is optimized for the unique properties of the MegaFlow classifier. This paper presents the design, implementation, and evaluation of MegaTurbo, a novel FPGA-based accelerator that embodies this philosophy. The core innovations are threefold:

- We design a scalable, hardware-friendly decision-tree based packet classification algorithm, MegaTree, that is explicitly tailored to the structural characteristics of MegaFlow rules.
- We design a novel, multi-engine pipelined hardware architecture that enables efficient parallel processing of the multiple decision trees generated by our software algorithm.
- We develop a heterogeneous CPU-FPGA cooperative framework that supports online rule updates with negligible impact on the ongoing packet classification performance.

Through extensive evaluations on a Xilinx Virtex UltraScale+ FPGA, we demonstrate that MegaTurbo achieves a stable classification throughput of 500 MPPS, coupled with a dynamic rule update throughput exceeding 370 KUPS. This classification performance significantly surpasses all existing FPGA-based packet classification solutions, showcasing MegaTurbo's superior capability in handling large-scale, dynamic flow tables. These results not only validate the efficacy of our co-design approach but also highlight the potential of FPGA-based SmartNICs to alleviate critical performance bottlenecks in large-scale cloud and data center networks.

2 Background and Motivation

2.1 MegaFlow Classification Problem in OVS

Open vSwitch (OVS) is widely regarded as the standard virtual switch in cloud and data center environments, its performance critically depending on the MegaFlow cache-a sophisticated classifier that minimizes packet processing overhead through intelligent flow entry caching and aggregation [45]. Algorithmically, the MegaFlow classification is essentially a variant of the classic packet classification problem, actively explored for over twenty years [5, 12, 13, 20, 22, 24, 30, 32, 47–49, 52, 54, 55, 60, 62, 67, 69]. The multi-field packet classification problem can be formally defined as follows: Given a packet $P = (f_1, f_2, \ldots, f_d)$ with d header fields and a ruleset $\mathcal{R} = \{R_1, R_2, \ldots, R_N\}$ where each rule $R_i = (F_1^i, F_2^i, \ldots, F_d^i, Priority_i, Action_i)$ contains field specifications (prefixes, ranges, or exact values), a match occurs when $\forall j \in [1, d], f_j \in F_j^i$. The goal is to find the matching rule with the highest priority: $R^* = argmax\{Priority(R) \mid R \in \mathcal{M}(P)\}$, where $\mathcal{M}(P) = \{R_i \in \mathcal{R} \mid \forall j, f_j \in F_j^i\}$ is the set of all rules matching packet P. The key difference between them is that there is no overlap among the MegaFlow rules, thereby eliminating the need for priority resolution. Table 1 shows an example MegaFlow ruleset with two 4-bit width fields.

Table 1: A two-field example of MegaFlow ruleset

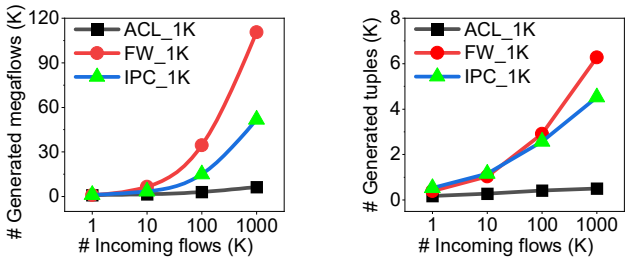
Rule	Field X	Field Y	Action
R1	000*	000*	Action1
R2	00**	001*	Action2
R3	01**	0***	Action3
R4	0***	111*	Action4
R5	*	100*	Action5
R6	10**	000*	Action6
R7	10**	010*	Action7
R8	101*	111*	Action8
R9	11**	0***	Action9
R10	11**	11**	Action10

2.2 MegaFlow Classification Challenge in OVS

While MegaFlow cache enables high-performance packet forwarding by avoiding multi-stage packet-matching operations, it introduces significant challenges that constrain OVS's scalability in modern cloud environments [6, 68, 70, 72]. One very important reason is that MegaFlow classifer employs a classic tuple-based algorithm for its packet classification [37, 48], which requires exhaustive traversal through all tuple combinations and exhibits poor scalability in both dimensionality and rule volume [7, 28, 29].

Empirical evidence from Figure 1 reveals the algorithm's fundamental weakness: the number of generated tuples grows dramatically with increasing flow diversity, creating substantial memory and computational overhead. This tuple explosion forces the tuple-based algorithm to traverse a rapidly growing search space for each packet, creating a severe bottleneck that undermines the cache's efficiency and highlights the urgent need for more scalable classification approaches. Although its performance remains manageable with structured flows like ACLs, where flow aggregation mitigates tuple growth, it deteriorates markedly under diverse traffic patterns, such as those in microservice communications, where cache entries scale nearly linearly with incoming flows.

The operational overhead of the MegaFlow classifier further exacerbates these challenges. The continuous processes of packet classification, flow entry management, and cache maintenance consume substantial CPU and memory resources that would otherwise serve tenant applications. This resource contention becomes particularly severe under adverse traffic conditions, where decreased classification efficiency coincides with increased computational demands. The resulting competition between network functions and business applications not only limits network performance but also diminishes the overall computational capacity of the host system.



(a) Number of flow rules in MFC **(b) Number of tuples in MFC**

Figure 1: MFC evaluation results under varying traffic loads, using OVS TestBench open sourced from NuevoMatch [43].

These dual challenges of performance unpredictability and resource overhead underscore software-based MegaFlow classification's limitations and highlight the need for hardware offloading. As cloud networks scale, migrating this critical functionality to specialized accelerators, such as FPGAs and SmartNICs, grows increasingly compelling. This architectural evolution is essential to achieve deterministic performance, eliminate host resource contention, and meet modern virtualized infrastructure's growing demands.

2.3 FPGA-based Acceleration Paradigms

Existing FPGA acceleration for packet classification as well as other complex packet-processing tasks like regular-expression matching falls into two major paradigms. [10, 11, 14–17, 21, 38, 40–42, 46, 50, 57–59, 61, 63], each with inherent trade-offs that make them suboptimal for the dynamic nature of the MegaFlow cache.

2.3.1 Run-to-Completion (RTC) Architectures. Emerging FPGA-based techniques, exemplified by state-of-the-art designs such as KickTree [58, 61] and TcbTree [59], consolidate all nodes from every level of a tree into a single SRAM. This architecture supports diverse rulesets without reconfiguration and overcomes tree depth limitations, offering significantly greater versatility than fixed-pipeline solutions. A key advantage of this flexibility is the inherent support for dynamic rule updates. However, its fundamental drawback is the sequential access of nodes, which inherently limits throughput. Consequently, achieving high performance often necessitates replication across multiple cores, resulting in poor resource utilization. More critically, the non-deterministic execution latency introduces unpredictable packet reordering, thereby requiring complex and high-latency reordering mechanisms at the output.

2.3.2 Pure Pipeline Architectures. Architectures such as HyperCuts on FPGA [16], HyperSplit on FPGA [38], ParaSplit [9] and MbitTree [51] implement classification as a dedicated fixed-latency pipeline. In this model, tree node information is distributed across multiple SRAMs, each corresponding to a pipeline stage, enabling a result to be output in every cycle. As a result, classification throughput scales linearly with the operating frequency, delivering high performance and deterministic latency with efficient resource usage. However, its static nature is its greatest weakness. The number of pipeline stages is determined by the data structure generated by a specific ruleset, which makes supporting incremental rule updates—a fundamental requirement for the ever-evolving MegaFlow cache—exceptionally difficult without stalling the pipeline or incurring significant performance overhead.

2.4 The Gap and Our Contribution

As discussed, the prevailing hardware paradigms for packet classification, including those implemented in FPGAs, present a stark trade-off. While FPGAs offer programmability, conventional designs based on RTC architectures incur significant reconfiguration overhead that undermines performance determinism, whereas static pipelines prioritize throughput but cannot support the agile updates required by the MegaFlow cache. The core issue persists: even on FPGA platforms, existing design methodologies fail to adequately resolve the fundamental conflict between the dynamic table management inherent to MFC and the static resource allocation of hardware circuits. Crucially, prior FPGA-based implementations have not been architected specifically to accommodate the distinct wildcard patterns and the continuous, fine-grained update stream characteristic of the MegaFlow cache, leading to inefficient resource utilization and limited scalability under realistic OVS workloads.

This landscape underscores the limitation of applying conventional hardware design paradigms to the MFC offload problem and motivates the need for a truly domain-specific, FPGA-native architecture that co-designs the algorithm with the underlying fabric.

Our work, MegaTurbo, bridges this gap by introducing a novel architecture designed from the ground up for FPGA implementation to address this challenge. We combine a hardware-friendly classification algorithm optimized for MegaFlow rules with a hybrid engine that leverages FPGA resources to decouple high-throughput packet processing from low-latency update management. This enables MegaTurbo to deliver pipelined performance while supporting efficient, low-impact online updates—a capability absent in prior FPGA designs. By aligning the architecture with the operational characteristics of OVS, MegaTurbo effectively breaks the traditional flexibility-performance trade-off, demonstrating how a domain-specific FPGA design can overcome conventional limitations to fully leverage the MegaFlow cache's potential.

3 The Proposed MegaTurbo

3.1 Overall Framework

The overall architecture of MegaTurbo, as illustrated in Figure 2, embodies a tightly coupled CPU–FPGA co-design. It operates as a heterogeneous system where the host CPU running the standard OVS stack collaborates with a custom FPGA accelerator.

The workflow begins with the *Flow Classifier Building Agent* on the CPU side. This agent compiles the flow rules in MegaFlow classifier, which are originally established by the OVS slow path, into the hardware-optimized MegaTree data structure. This data structure serves a dual purpose: it is used to generate the hardware configuration files that build the pipeline trees on the FPGA, and it defines the actual content (i.e., node and leaf data) that is loaded into the pipeline memories via a dedicated interface.

The heart of the acceleration is the *Hardware Classification Accelerator* on the FPGA. This accelerator is composed of a large number of parallel processing *Pipelines* (e.g., *Pipeline 1* to *Pipeline N*). Each pipeline is configured to process packets against a specific segment of the ruleset. Incoming packets are directed to these pipelines, where their headers are processed in parallel. The hardware efficiently traverses the preloaded MegaTree structures and outputs the matching result, which is then used for forwarding decisions.

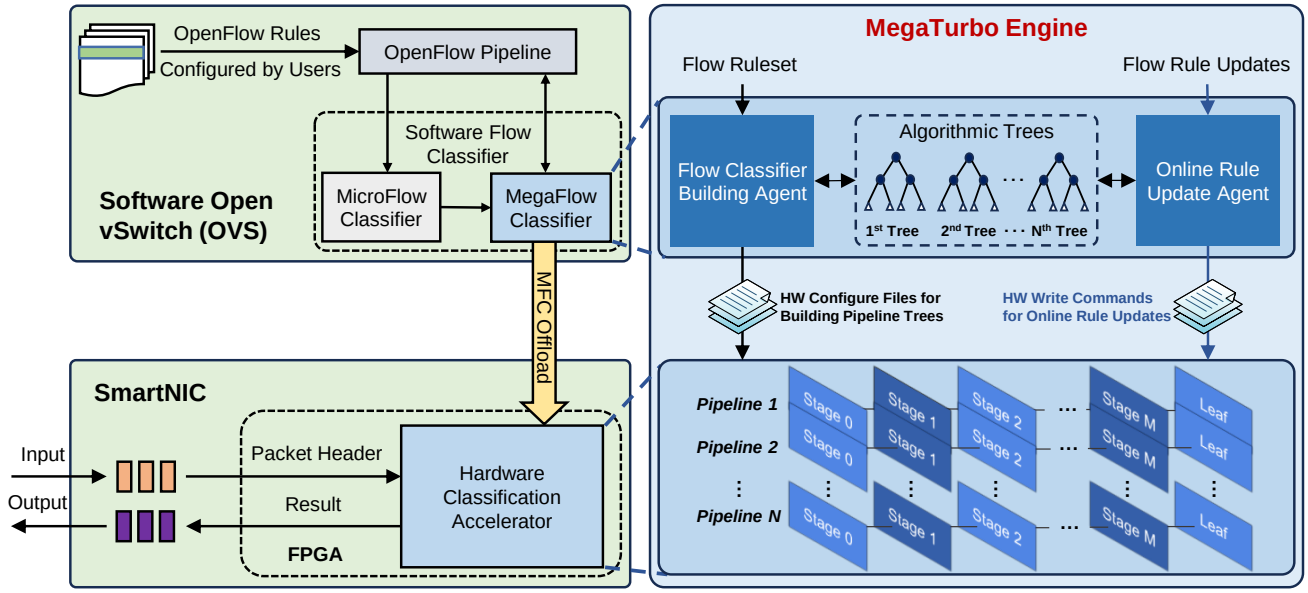


Figure 2: System architecture of MegaTurbo.

Crucially, to natively support the dynamic nature of the OVS MegaFlow cache, the *Online Rule Update Agent* manages incremental rule updates. It computes differences in the MegaTree data structure and generates corresponding configuration updates, which are delivered to the FPGA to modify specific pipelines on-the-fly. This mechanism ensures that high classification throughput is sustained with minimal impact, while supporting a substantial dynamic rule update throughput, making MegaTurbo suitable for large-scale, dynamic cloud environments.

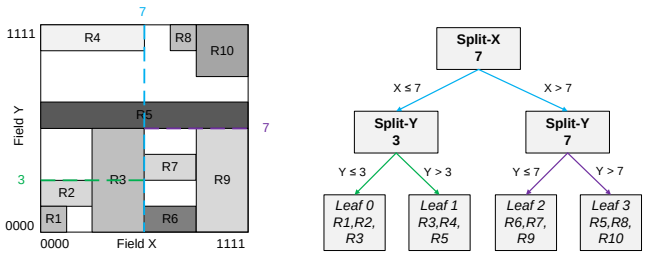
3.2 Algorithm Foundation: HyperSplit

MegaTurbo employs the well-known HyperSplit algorithm [39] as the foundational method for building decision trees. The geometric representation in Figure 3a visually demonstrates the distribution of rules in the field space, while the decision tree in Figure 3b , built from the ruleset in Table 1, illustrates this process: the algorithm recursively splits rules at each node using a cutpoint T that equally partitions the current ruleset into two subsets of balanced size (i.e., $X \leq T$ and $X > T$). Partitioning continues until the number of rules in a node falls below a predefined threshold *binth*, at which point the node becomes a leaf containing the final rule subset.

The selection of HyperSplit is well-suited to MegaFlow classification primarily because the inherent non-overlapping property of MegaFlow rules significantly reduces rule replication during tree construction, enabling MegaTurbo to achieve effective classification of large-scale rulesets (e.g., 100K rules) using only a modest number of parallel trees in our system.

3.3 The MegaFlow-Dedicated Algorithm

To overcome the limitations of existing packet classification algorithms when applied to the MegaFlow cache, we designed MegaTree, a scalable and hardware-friendly algorithm specifically tailored for the structure and dynamics of MegaFlow rules. The core objective of MegaTree is to transform a complete set of MFC flow rules into



(a) Geometric view of rules **(b) Generated decision-tree**

Figure 3: HyperSplit for example rules in Table 1 (binth=4).

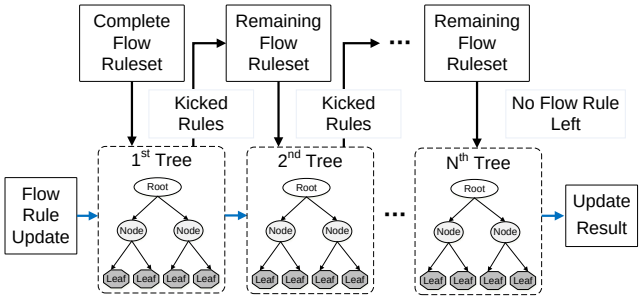


Figure 4: MegaFlow classifier algorithm (i.e., MegaTree). A rule is "kicked" under two conditions: (1) if a split would cause replication, and (2) if the destination leaf node has already reached its maximum rule capacity.

a collection of compact, balanced, and independent decision trees that can be efficiently processed in parallel by the FPGA hardware. As illustrated in Figure 4, the algorithm construction involves two key iterative operations: *Non-Overlap Split* and *Rule Kicking*. The pseudo-code of MegaTree can be found in Algorithm 1, and we have released the source code of MegaTree on the Zenodo[2]/Website[1].

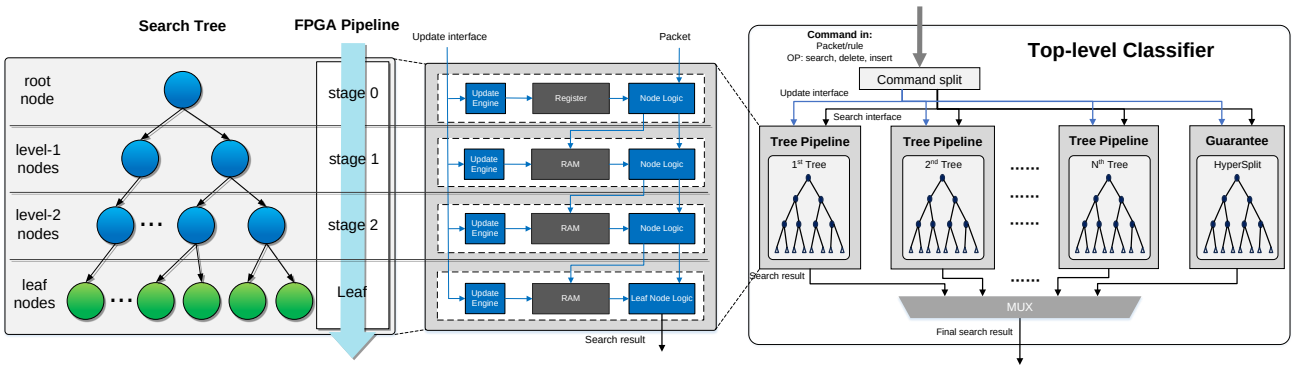


Figure 5: Hardware architecture.

Algorithm 1: Construction of MegaTree

Data: Max tree depth *maxDepth*, leaf bin size *binth*
Input: Rule set R
Output: Decision Trees $roots[n]$

$roots \leftarrow \emptyset$
$currentRules \leftarrow R$
while $currentRules \neq \emptyset$ **do**
  $remainingRules \leftarrow \emptyset$
  $newRoot \leftarrow$ ConstructMegaTree($currentRules$, $remainingRules$)
  $roots \leftarrow roots \cup \{newRoot\}$
  $currentRules \leftarrow remainingRules$
return $roots$
Function *ConstructMegaTree(rules, res)*:
  $root \leftarrow$ CreateNode($rules$)
  $queue \leftarrow \{root\}$
  while $queue \neq \emptyset$ **do**
    $node \leftarrow$ Dequeue($queue$)
    if $node.depth \geq maxDepth$ **then**
      Move excess rules beyond $binth$ to res
      $node.isLeaf \leftarrow true$
      continue
    Find optimal split using HyperSplit: select dimension d, threshold t
    Calculate $leftRules$, $rightRules$, $kickedRules$ for split (d, t)
    if *No valid split* ($leftRules = \emptyset$ **or** $rightRules = \emptyset$) **then**
      Move excess rules beyond $binth$ to res
      Create single-branch child; continue
    $node.dim \leftarrow d$
    $node.threshold \leftarrow t$
    Add $kickedRules$ to res
    if $leftRules \neq \emptyset$ **then**
      Create left child; Enqueue($queue$, $leftChild$)
    if $rightRules \neq \emptyset$ **then**
      Create right child; Enqueue($queue$, $rightChild$)
  return $root$

3.3.1 Algorithmic Construction Process. The MegaTree algorithm is designed to compile the complete MegaFlow ruleset into a forest of compact and balanced decision trees optimized for hardware matching. The construction is an iterative process. Starting with the full flow ruleset, each iteration applies the HyperSplit algorithm to partition the rules. The key to achieving hardware efficiency lies in the *Non-Overlap Split* strategy: during partitioning, rules that would require replication across multiple decision tree branches (e.g., R5 and R4 in Figure 8), together with the portion of rules in the current node that exceeds the threshold *binth* (e.g., R8 in Figure 8), are "kicked out" and deferred to a subsequent iteration. This guarantees that all rules in the current tree are unique, thus avoiding duplication and resulting in a compact and efficient structure. A critical optimization goal during this process is to balance the depth of the generated trees, ensuring that the maximum depth across all trees is as consistent as possible. This uniformity is essential for achieving predictable, low-latency traversal in the parallel hardware pipelines, as it prevents any single tree from becoming a processing bottleneck. The process repeats on the remaining set of "kicked" rules until no rules are left, ultimately generating a collection of independent trees.

3.3.2 Hardware-Optimized Design. The use of the HyperSplit algorithm as the core splitting method for building decision trees is pivotal, as it provides direct control over the maximum tree depth during partitioning, allowing the construction of trees that are inherently balanced and depth-bounded. This controllability is essential for tailoring the data structure to the constraints of the hardware pipelines. In parallel, the configurable maximum tree depth as an independent parameter during tree construction provides direct control over the worst-case traversal latency. These mechanisms ensure predictable and balanced processing latency across all parallel matching engines, preventing any single pipeline from becoming a bottleneck.

3.3.3 Seamless Support for Dynamic Updates. The algorithm natively supports the dynamic nature of the OVS MegaFlow cache. A core enabler for this is the strict non-replication of rules—each rule is stored exactly once within a single tree and node, unlike approaches such as ParaSplit [9] that replicate rules across multiple trees and nodes. This fundamental property ensures that any insertion or deletion of a rule affects only a single localized point in the data structure. Consequently, the MegaTree algorithm can identify the specific tree and node impacted by a change and recompute only

the necessary branch in microsecond-scale time to generate the corresponding hardware configuration. This granular and incremental update mechanism, managed by the software agent, produces minimal and bounded configuration changes for the hardware, ensuring that packet classification throughput remains high even during the most frequent cache updates.

3.4 Detailed Hardware Architecture Design

The MegaTurbo hardware engine is designed as a purely pipelined high-throughput classifier that directly maps the forest of decision trees generated by the MegaTree algorithm onto parallel physical resources. The core design, depicted in Figure 5, is centered on multiple independent *Tree Pipelines* that operate concurrently. A fundamental principle of this design is to enforce constant and deterministic latency for every packet through the system, regardless of the actual path it takes within a decision tree. This section details the key components and the data flow for both packet classification and rule updates.

3.4.1 Overall Pipeline Architecture. The top-level classifier is composed of N identical *Tree Pipelines*, each dedicated to processing one tree from the MegaTree forest. This multi-pipeline architecture is the foundation of our design's scalability and performance. An incoming packet header is broadcast to all pipelines for simultaneous processing. Importantly, each pipeline is designed with a fixed number of stages, corresponding to the configured maximum tree depth. Even if a packet would logically reach a leaf node before the maximum depth according to the MegaTree, it must still traverse all pipeline stages in hardware, progressing through the remaining stages as a "bubble" or empty operation. This ensures that every packet takes exactly the same number of clock cycles to traverse each pipeline. This deterministic latency is paramount for building a stable and predictable system, as it prevents timing variations that could complicate result collection and system synchronization.

This classifier exemplifies our application-algorithm-architecture co-design: since the MegaFlow classifier inherently contains no overlapping rules, each packet will find at most one match across all parallel trees. This domain-specific property allows the final stage to employ a simple selector instead of the complex priority resolution logic required in traditional multi-pipeline architectures.

3.4.2 Structure of a Single Tree Pipeline. Each *Tree Pipeline* is a deeply pipelined engine that mirrors the structure of its assigned decision tree. The pipeline is divided into multiple stages, corresponding to the levels of the tree (e.g., root node, level-1 nodes, level-2 nodes, leaf nodes).

- Node Table RAMs with Dual Read Ports: At the heart of each stage is a True Dual-Port (TDP) RAM block that stores the node data. A key feature of our design is that each of the two independent read ports of the TDP RAM serves a separate and complete tree search pipeline. This allows two packets to be processed through the same set of rules concurrently and asynchronously, effectively doubling the classification throughput without doubling the memory footprint—a crucial advantage given the scarcity of on-chip memory in FPGAs.
- Node Logic: Each stage contains dedicated *Node Logic* responsible for comparing the incoming packet header against the node data fetched from the RAM. Based on the comparison result, the logic determines the address of the next node in the subsequent level's RAM.
- Update Engine: Each RAM block is associated with a dedicated *Update Engine*. These engines process commands received via the *Update Interface*, the engine first determines whether the target node resides in its local RAM. Only when a match is confirmed does it proceed to modify the corresponding node data, without stalling the packet processing path. This decentralized update mechanism is key to achieving high and non-blocking classification throughput.

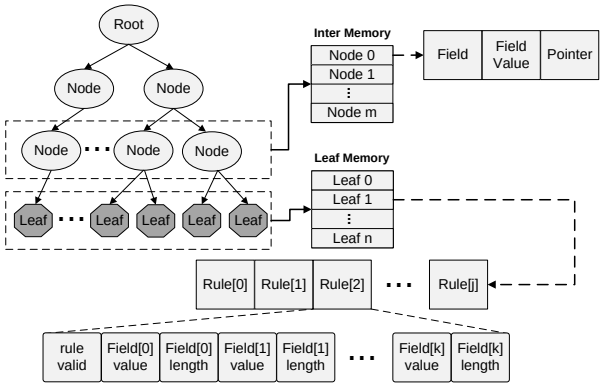


Figure 6: Storage organization of search tree.

3.5 Hardware Tree Node Data Structure

The MegaTree hardware node structure is optimized for high-speed parallel access and efficient memory utilization, while maintaining extensibility to accommodate diverse OVS rulesets. As illustrated in Figure 6, The separation of internal and leaf nodes into distinct memory blocks enables parallel access and efficient resource utilization on the FPGA.

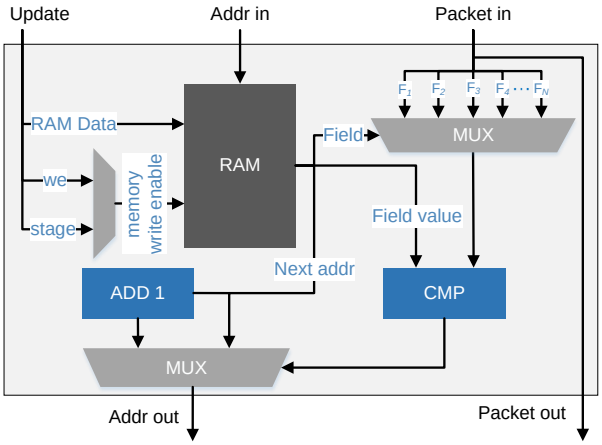
3.5.1 Internal Node Structure. Internal nodes are stored in the *Inter Memory* and contain the decision logic for packet traversal. Each internal node consists of three key fields:

- **Field Identifier**: Specifies which packet header field to examine.
- **Field Value**: The comparison value used for branching decisions.
- **Pointer**: Address of the next child node in the pipeline hierarchy.

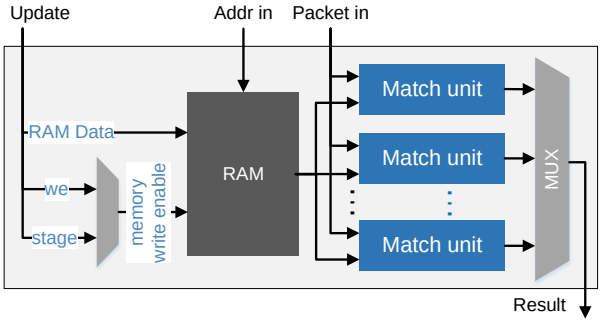
This compact structure enables single-cycle decision making at each pipeline stage, with the field value determining whether packets should be routed to the left or right child node based on the comparison result.

3.5.2 Leaf Node Structure with Parallel Matching. Leaf nodes reside in the *Leaf Memory* and are designed to store multiple rules within a single memory address, enabling parallel rule matching. Each leaf node accommodates multiple rules (i.e., *Rule[0]* to *Rule[j]*) and includes:

- **Rule Valid Bits**: Bitmask indicating which rule entries contain valid rules.
- **Field Values**: Complete specification of each rule's matching criteria across k fields.
- **Field Lengths**: Corresponding prefix lengths for each field, supporting wildcard matching.



(a) The inter module architecture



(b) The leaf module architecture

Figure 7: Architectures of inter module and leaf module.

3.6 Module Architecture

Figure 7 illustrates the detailed architecture of the two fundamental modules in our pipeline design: the inter module for internal tree nodes and the leaf module for terminal nodes.

3.6.1 Inter Module. As shown in Figure 7a, each inter module implements an internal node of the decision tree. The module receives configuration data and packet headers, and processes them through a multi-stage comparison logic. The core architecture employs field extraction units that select multiple fields (F_1 through F_N) from the incoming packet header, which are then compared in parallel against pre-configured thresholds stored in the module's RAM. Based on the comparison results, the next node address is generated through an address calculation unit that either adds an offset or selects from predefined paths. The inter module maintains deterministic latency through its pipelined design, while the write-enable control signal supports dynamic rule updates with minimal impact on packet processing.

3.6.2 Leaf Module. Figure 7b presents the leaf module design, which terminates the tree traversal and produces final classification results. The architecture features multiple parallel match units that concurrently evaluate different rule conditions, with a result aggregation unit that selects the appropriate output from the active match units. A dedicated update interface supports online modifications to the ruleset.

3.7 Dynamic Rule Update Mechanism

A fundamental requirement for a practical MegaFlow classifier is the ability to handle frequent flow rule updates while maintaining high packet classification throughput. MegaTurbo meets this requirement through a sophisticated non-blocking, interleaved update mechanism that seamlessly integrates rule modifications with ongoing packet processing.

3.7.1 Update Interface and Command Processing. The update mechanism operates through a dedicated path that connects the *Online Rule Update Agent* to the hardware pipelines. When rule modifications occur, the update agent computes the necessary changes to the MegaTree structure and generates corresponding memory access sequences (the addresses and data to be written in) for online rule updates, a process that is completed within microseconds. These commands are delivered directly to the specific pipelines affected by the changes, enabling precise targeting of individual pipeline elements without disturbing unrelated processing elements.

3.7.2 Non-Blocking Interleaved Update Mechanism. The core of the update mechanism lies in the interleaved execution of write operations with normal packet processing. The system strategically inserts write instructions into gaps between packet processing cycles, utilizing idle memory port bandwidth to complete rule updates without pausing any pipeline. This design enables Port B (the second port in the TDP RAM) to serve read requests for one pipeline while switching to write mode during specific cycles to perform update operations, then immediately resuming normal read service—making the entire process transparent to packet processing.

3.7.3 Bounded Latency and Throughput Impact. This interleaved approach ensures that update operations have a strictly bounded and predictable impact on system performance. Since write instructions are seamlessly integrated into the normal packet processing flow, there is no pipeline bubble or stalling penalty. Since each update involves modifications to only a few nodes, the impact on overall throughput is negligible.

3.7.4 Fault Tolerance and Fallback Mechanism. To ensure reliable classifier construction and rule updates, MegaTurbo incorporates a fallback mechanism where a pre-configured empty pipeline serves as a guaranteed placement for new rules when insertion fails — whether during initial tree building or subsequent dynamic updates due to insufficient space in existing trees, thus maintaining system availability without interrupting packet processing.

3.8 A Running Example

To illustrate the operation of the MegaTurbo system, we present a concrete example that traces the journey of flow rules from software data structures to hardware pipeline configuration. This example demonstrates how the MegaTree algorithm transforms a set of rules into an optimized decision tree structure, and how this structure is subsequently deployed onto the FPGA for high-speed classification.

3.8.1 Tree Construction in Software. As shown in the first layer of Figure 8, the MegaTree algorithm begins with a set of rules in Table 1. It applies the HyperSplit method with *Non-Overlap Split* and *Rule Kicking* strategies to build a balanced decision tree. The construction creates multiple splitting nodes based on field

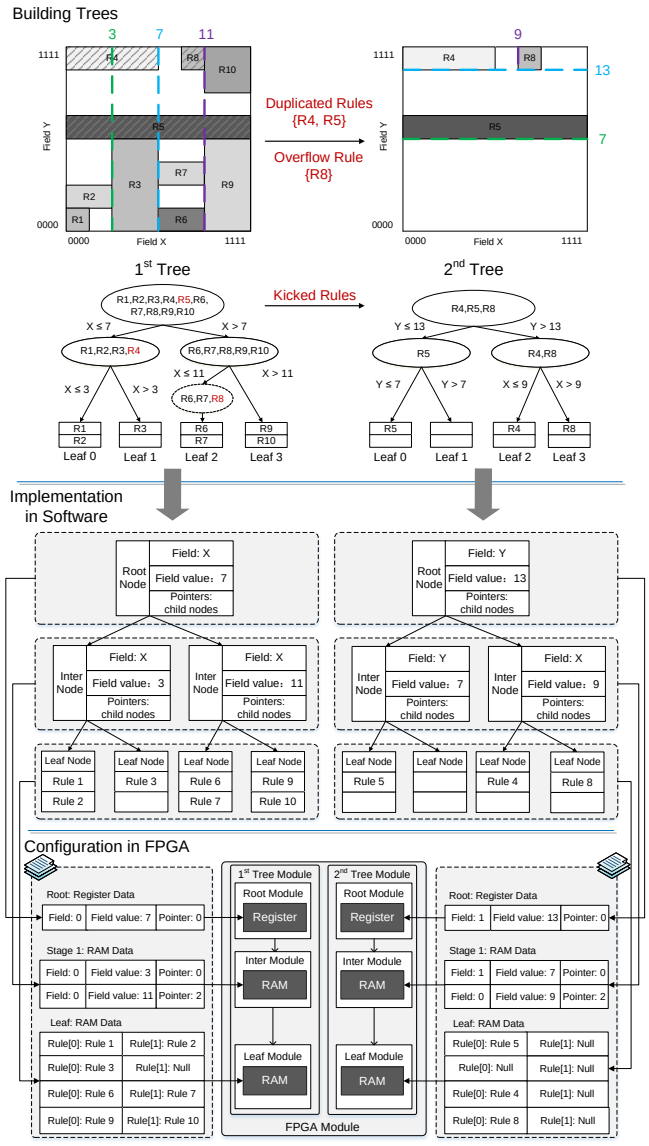


Figure 8: A running example for rules in Table 1 (binth=2).

X values, with threshold comparisons such as $X \leq 7$ at the root node, followed by $X \leq 3$ and $X \leq 11$. Specifically, rules $R5$ and $R4$ are kicked out during the first and second splits due to potential rule replication, while $R8$ is kicked out when a leaf node reaches the maximum depth (2) while containing more rules than the *binth* threshold (2). The kicking mechanism ensures proper distribution of conflicting rules and eliminates duplicates in leaf nodes.

3.8.2 Software Data Structure Implementation. The second layer of Figure 8 shows the software representation of the decision trees. Each node in the tree is implemented as a structured data element containing: field identifier, comparison value, and pointers to child nodes. The *Root Node* and *Internal Nodes* perform the branching decisions, while the *Leaf Nodes* store the actual rules that will be applied when packet traversal reaches them. The software maintains this complete representation to enable dynamic updates and regeneration of hardware configuration files when the ruleset changes.

3.8.3 FPGA Hardware Configuration. The third layer of Figure 8 illustrates the hardware instantiation, which maps the software decision tree directly onto the MegaTurbo pipeline architecture. The 1st *Tree Module* implements our example tree across multiple pipeline stages: the *Root Module* (Stage 0) stores the root node comparison value (7) in registers for immediate access, subsequent *Internal Modules* (Stages 1) store internal node data in Block RAMs with field values {3, 11}, and the *Leaf Module* (final stage) contains the actual rules distributed across different memory locations. A parallel 2nd *Tree Module* demonstrates the system's ability to handle multiple trees concurrently, implementing additional rules like *Rule 8*. This modular design allows packets to flow through the pipeline stages, with each stage contributing to the classification decision until a final rule match is identified in the leaf stage.

4 Implementation Results

4.1 Experiment Setup

4.1.1 Hardware Platform. Our evaluation platform consists of a server with an Intel Xeon Platinum 8575C processor and 30GB memory, running Ubuntu 22.04. The accelerator is built on a Xilinx Virtex UltraScale+ VU9P FPGA. The MegaTurbo design is implemented in Vivado 2022.2, targeting a clock frequency of 250 MHz.

4.1.2 Rulesets and Workloads. Three types of rulesets are generated by ClassBench [53] for performance evaluation: Access Control List (ACL), Firewall (FW), and IP Chain (IPC). Specifically, ACL and FW each comprise five distinct rulesets with 100K rules, while IPC includes two rulesets at the same scale, resulting in a total of twelve representative rulesets. The 100K MegaFlow rules and accompanying traces are generated using the OVS TestBench open sourced by the NuevoMatch paper [43].

4.1.3 Implementation Configuration. The Virtex UltraScale+ VU9P FPGA provides substantial on-chip memory resources including abundant Block RAMs and UltraRAMs. This memory-rich architecture enables the implementation of multiple parallel processing pipelines, each containing complete decision tree structures for high-throughput packet classification. The design entry was developed using Verilog HDL and implemented with Vivado 2022.2, employing the $Performance\_Explore$ strategy for implementation to achieve optimal timing closure. The tree depth is set to 11 layers with a binth of 8, and 8 *Tree Pipelines* are implemented.

4.1.4 Evaluation Methodology. We employ multiple metrics to comprehensively evaluate system performance:

Classification Throughput: Measured in millions of packets per second (MPPS), reflecting the raw packet processing capability.

Update Throughput: Measured in thousands of updates per second (KUPS), representing the system's capacity to handle dynamic rule modifications.

Resource Utilization: FPGA resource consumption including LUTs, Registers, BRAMs, and LUTRAMs.

Maximum Frequency: The highest achievable operating frequency of the implemented design, measured in MHz.

Table 2: Implementation result for different rulesets

Ruleset	CLB LUTs (1182240)	CLB Registers (2364480)	BRAM (2160)	LUTRAM (591840)	Max Frequency
ACL1_100K	3.23%	2.05%	29.17%	0.39%	250.90 MHz
ACL2_100K	3.23%	2.05%	29.17%	0.39%	250.94 MHz
ACL3_100K	3.23%	2.05%	29.17%	0.39%	250.94 MHz
ACL4_100K	3.23%	2.05%	29.17%	0.39%	250.94 MHz
ACL5_100K	3.23%	2.05%	29.17%	0.39%	250.94 MHz
FW1_100K	3.23%	2.05%	29.17%	0.39%	250.94 MHz
FW2_100K	3.23%	2.05%	29.17%	0.39%	250.94 MHz
FW3_100K	3.23%	2.05%	29.17%	0.39%	250.94 MHz
FW4_100K	3.53%	2.31%	31.71%	0.21%	250.38 MHz
FW5_100K	3.53%	2.29%	31.71%	0.21%	254.39 MHz
IPC1_100K	3.23%	2.05%	29.17%	0.39%	250.94 MHz
IPC2_100K	3.54%	2.32%	31.71%	0.21%	251.26 MHz

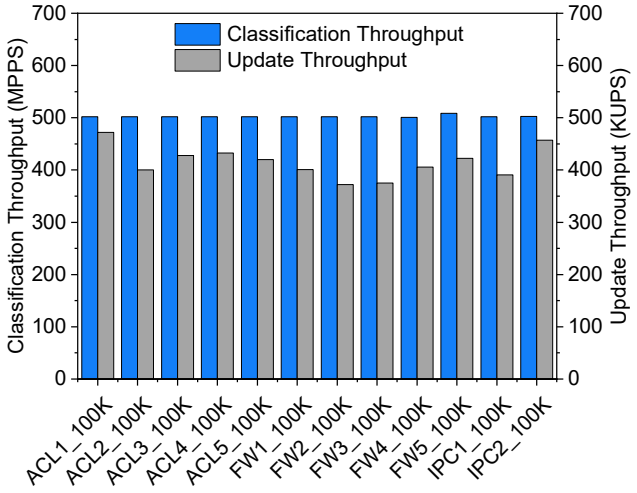


Figure 9: Throughput for 100K rulesets.

4.2 Resource Utilization

Table 2 presents the detailed resource utilization of MegaTurbo in all 12 MegaFlow rulesets on the Xilinx Virtex UltraScale+ VU9P FPGA. The results demonstrate consistent and efficient resource usage patterns in different rule types and configurations.

4.2.1 Logic Resource Utilization. The utilization of CLB LUTs remains highly stable across most rulesets, ranging from 3.23% to 3.54% of the available resources. Similarly, CLB register utilization shows minimal variation, maintaining between 2.05% and 2.32%. This consistency across different rule types (ACL, FW, and IPC) demonstrates the robustness of our architecture in handling diverse classification patterns. The consistently low LUTRAM utilization (0.39% for most rulesets and 0.21% for others) not only indicates an efficient memory organization, but also alleviates routing congestion and timing pressure, while simultaneously reserving substantial resources for future functional expansions.

4.2.2 Memory Resource Utilization. Block RAM utilization shows two distinct patterns: 29.17% for the majority of rulesets and 31.71% for FW4, FW5, and IPC2 rulesets. This variation reflects the different memory requirements of various rule characteristics while maintaining overall efficiency. The consistent BRAM usage within each category demonstrates predictable memory footprint regardless of specific rule compositions.

4.2.3 Timing Performance. All implementations successfully achieve timing closure with maximum operating frequencies ranging from 250.38 MHz to 254.39 MHz, comfortably exceeding our target frequency of 250 MHz. The minimal frequency variation across different rulesets (standard deviation of 1.07 MHz) highlights the deterministic nature of our pipeline architecture and its insensitivity to ruleset variations.

The results confirm that MegaTurbo maintains stable resource utilization and timing performance across diverse classification workloads, making it suitable for deployment in production environments with varying traffic patterns.

4.3 Performance Evaluation

4.3.1 Classification Throughput. As shown in Figure 9, evaluation across all twelve 100K MegaFlow rulesets demonstrates that MegaTurbo achieves a sustained classification throughput of approximately 500 MPPS, with remarkable consistency across diverse ACL, FW, and IPC rule types. This performance stability underscores the robustness of our pipeline architecture, attributable to its deterministic fixed-depth design and efficient parallel processing across multiple tree pipelines.

The achieved throughput of 500 MPPS not only meets but exceeds modern high-speed network requirements, maintaining peak performance even with complex rule configurations. This positions MegaTurbo as a compelling solution for next-generation data centers, ensuring both high performance and consistent quality of service regardless of specific ruleset characteristics.

4.3.2 Incremental Update Throughput. The incremental update throughput of MegaTurbo is evaluated under extreme conditions to measure its peak capability. Specifically, one classification pipeline is temporarily disabled to dedicate all resources to continuous update operations, creating a worst-case scenario for update performance measurement. As shown in Figure 9, under these intensive conditions, the system achieves update throughput ranging from approximately 300 KUPS to 500 KUPS across different 100K rulesets, with most rulesets clustering in the higher performance range, far exceeding the actual requirement for flow table update performance in real OVS system (the MegaFlow cache in real OVS system is typically limited to 200K entries) [19, 37, 71].

This stress test demonstrates MegaTurbo's robust capability to handle intensive update workloads when required. The measured throughput represents the upper bound of the system's update capacity, showcasing its ability to sustain substantial update rates even under maximum load. In practical deployment scenarios where classification and update operations coexist, the system maintains a balanced performance profile while ensuring uninterrupted packet processing during rule modifications.

Table 3: Comparison of decision tree based approaches on FPGA

Approach	Ruleset Scale	Device	Resource Consumption				Classification Throughput (MPPS)	Microsecond-scale On-the-fly Update
			LUT	Registers	BRAM	URAM		
Proposed MegaTurbo	100K	Ultrascale+ VU9P	26828	49367	630	0	501.88	✓
HACL [57]		UltraScale+ XCVU35P	190811	335203	416	156	280.74	×
KickTree_Systolic [58]		Ultrascale+ VU9P	506670	550812	1836	936	121.40	✓
TcbTree [59]		Ultrascale+ VU9P	66834	111719	990	792	45.6	✓
KickTree_Parallel [61]		Ultrascale+ VU9P	607596	819039	1815	762	172.9	✓
MBitTree on FPGA [51]		Virtex-7 XC7V690T	37828	75656	818	0	175.9	×
REC [3]	10K	Virtex-5 XC5VFX200T	7044	\	173	0	323.5	×
		Virtex-6 XC6VLX760	7044	\	173	0	388.2	×
UTPC [18]		Stratix III EP3SE260H780	40070	\	852	0	433	×
D^2BS [64]		Virtex-5 XC5VSX240T	\	\	\	\	263.7	×
HyperCuts on FPGA [17]		Virtex-5 XC5VFX200T	10307	\	407	0	250.7	×
ParaSplit [9]		Virtex-5 XC5VSX240T	48380	\	399	0	200.4	×
CubeCuts [4]		Virtex-5 XC5VFX200T	45656	\	195	0	368.8	×
HyperSplit on FPGA [38]		Virtex-6 XC6VLX760	2988	5976	103	0	230.5	×

4.4 Comparison With Related Work

Table 3 presents a comprehensive comparison between MegaTurbo and state-of-the-art decision-tree based packet classification approaches implemented on FPGAs, with MegaTurbo evaluated using the ACL1_100K ruleset. The evaluation demonstrates MegaTurbo's significant advantages in both performance and resource efficiency.

In the 100K ruleset category, MegaTurbo achieves the highest classification throughput of 501.88 MPPS while consuming the least hardware resources among all comparable approaches. Specifically, our design utilizes only 26,828 LUTs and 49,367 registers, representing 5.3% and 9.0% of KickTree_Parallel's consumption respectively. Notably, a primary driver behind the resource enhancement lies in the fact that the throughput of MegaTurbo is supported by a single core, whereas that of KickTree_Parallel is provided by approximately six cores. Furthermore, MegaTurbo requires only 630 BRAMs and 0 URAMs, substantially less than other 100K-scale solutions. This exceptional resource efficiency directly translates to better scalability and lower power consumption.

Compared to KickTree variants, MegaTurbo delivers 2.9~4.1× higher throughput while reducing LUT usage by 47~95%. When compared to TcbTree, our solution provides 11× higher performance with 60% fewer LUTs and 36% less BRAM consumption. Even against smaller-scale 10K ruleset approaches, MegaTurbo maintains competitive advantages, outperforming most designs in throughput while handling ten times larger rulesets.

Notably, MegaTurbo successfully supports dynamic rule updates—a feature lacking in several high-performance designs such as HACL and MBitTree. Although other pure-pipeline approaches, including HyperSplit on FPGA and ParaSplit, have claimed support for dynamic updates, they suffer from significant rule replication issues when handling large-scale rule sets. This leads to substantial computational overhead in software preprocessing, making it difficult to meet the real-time requirements of SDN dynamic updates. For instance, evaluations show that the average rule replication ratio can reach thousands of times in such algorithms [26]. Moreover, these approaches have not provided detailed implementation schemes or corresponding evaluation results for dynamic updates. This combination of high throughput, minimal resource footprint, and practical update capability positions MegaTurbo as a comprehensive solution for modern data center requirements, effectively addressing the scalability challenges that limit existing approaches in large-scale deployment scenarios.

5 Conclusion

This paper presents MegaTurbo, an innovative FPGA-based accelerator that effectively overcomes the performance limitations of the MegaFlow classifier in Open vSwitch. To the best of our knowledge, MegaTurbo represents the first decision-tree based FPGA accelerator specifically designed for the MegaFlow classifier. Our solution employs a co-design methodology integrating algorithm and hardware optimization, featuring three core contributions: the hardware-friendly MegaTree classification algorithm specifically designed for MegaFlow rules, a parallel multi-pipeline architecture guaranteeing deterministic latency, and a CPU-FPGA collaborative framework enabling efficient dynamic updates. Comprehensive evaluation demonstrates that MegaTurbo achieves exceptional performance on large-scale rulesets. The system's consistent performance across diverse rule types establishes MegaTurbo as a robust and scalable solution for modern data center networks, successfully addressing the critical challenge of maintaining both high performance and flexibility in dynamic cloud environments.

Acknowledgments

We thank reviewers for their constructive comments. This work was carried out at Pengcheng Laboratory under the guidance of corresponding authors Wenjun Li and Yao Xin, and was supported in part by the National Key Research and Development Program of China (2025YFE0200100), the Major Key Project of Peng Cheng Laboratory (PCL2025A07), the National Natural Science Foundation of China (62372123, 62102203), the Young Top-notch Talent Project of Guangdong Province (2023TQ07X362, 2023TQ07X004), and the Basic Research Enhancement Program (2021-JCJQ-JJ-0483).

References

[1] Our Website. https://wenjunli.com/MegaTurbo.

[2] Our Zenodo. https://doi.org/10.5281/zenodo.17697648.

[3] Yeim-Kuan Chang, Han-Chen Chen, and Gerard Parr. 2019. Fast Packet Classification using Recursive Endpoint-Cutting and Bucket Compression on FPGA. *Comput. J.* 62, 2 (2019), 198–214.

[4] Yeim-Kuan Chang and Yu-Hsiang Wang. 2012. CubeCuts: A Novel Cutting Scheme for Packet Classification. In *Proceedings of the IEEE International Conference on Advanced Information Networking and Applications Workshops (AINA)*.

[5] H Jonathan Chao and Bin Liu. 2007. *High performance switches and routers*. John Wiley & Sons.

[6] Levente Csikor, Dinil Mon Divakaran, Min Suk Kang, Attila Kőrösi, Balázs Sonkoly, Dávid Haja, Dimitrios P Pezaros, Stefan Schmid, and Gábor Rétvári. 2019. Tuple space explosion: A denial-of-service attack against a software packet classifier. In *Proceedings of the ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*.

[7] James Daly, Valerio Bruschi, Leonardo Linguaglossa, Salvatore Pontarelli, Dario Rossi, Jerome Tollet, Eric Torng, and Andrew Yourtchenko. 2019. TupleMerge: Fast Software Packet Processing for Online Packet Classification. *IEEE/ACM Transactions on Networking* 27, 4 (2019), 1417–1431.

[8] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, et al. 2018. Azure Accelerated Networking: SmartNICs in the Public Cloud. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[9] Jeffrey Fong, Xiang Wang, Yaxuan Qi, Jun Li, and Weirong Jiang. 2012. ParaSplit: A scalable architecture on FPGA for terabit packet classification. In *Proceedings of the IEEE Annual Symposium on High-Performance Interconnects (HOTI)*.

[10] Thilan Ganegedara, Weirong Jiang, and Viktor K. Prasanna. 2014. A Scalable and Modular Architecture for High-Performance Packet Classification. *IEEE Transactions on Parallel and Distributed Systems* 25, 5 (2014), 1135–1144.

[11] Thilan Ganegedara and Viktor K Prasanna. 2012. StrideBV: Single chip 400G+ packet classification. In *Proceedings of the IEEE International Conference on High Performance Switching and Routing (HPSR)*.

[12] Pankaj Gupta and Nick McKeown. 1999. Packet classification on multiple fields. In *Proceedings of the ACM Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*.

[13] Pankaj Gupta and Nick McKeown. 1999. Packet classification using hierarchical intelligent cuttings. In *Proceedings of the IEEE Annual Symposium on High-Performance Interconnects (HOTI)*.

[14] Weirong Jiang and Viktor K Prasanna. 2009. Field-split parallel architecture for high performance multi-match packet classification using FPGAs. In *Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*.

[15] Weirong Jiang and Viktor K Prasanna. 2009. A FPGA-based parallel architecture for scalable high-speed packet classification. In *Proceedings of the IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*.

[16] Weirong Jiang and Viktor K Prasanna. 2009. Large-scale wire-speed packet classification on FPGAs. In *Proceedings of the ACM/SIGDA international symposium on field-programmable gate arrays (FPGA)*.

[17] Weirong Jiang and Viktor K Prasanna. 2012. Scalable Packet Classification on FPGA. *IEEE Transactions on Very Large Scale Integration Systems* 20, 9 (2012), 1668–1680.

[18] Alan Kennedy and Xiaojun Wang. 2014. Ultra-High Throughput Low-Power Packet Classification. *IEEE Transactions on Very Large Scale Integration Systems* 22 (2014), 286–299.

[19] Maciej Kuźniar, Peter Perešíni, and Dejan Kostić. 2015. What you need to know about SDN flow tables. In *Proceedings of the Springer International Conference on Passive and Active Network Measurement (PAM)*.

[20] TV Lakshman and Dimitrios Stiliadis. 1998. High-speed policy-based packet forwarding using efficient multi-dimensional range matching. In *Proceedings of the ACM Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*.

[21] Shiming Lei, Chao Wang, Haijie Fang, Xi Li, and Xuehai Zhou. 2016. SCADIS: A Scalable Accelerator for Data-Intensive String Set Matching on FPGAs. In *Proceedings of the IEEE Trustcom/BigDataSE/ISPA*.

[22] Wenjun Li, Dagang Li, Yongjie Bai, Wenxia Le, and Hui Li. 2019. Memory-efficient recursive scheme for multi-field packet classification. *IET Communications* 13, 9 (2019), 1319–1325.

[23] Wenjun Li, Dagang Li, Xinwei Liu, Ting Huang, Xianfeng Li, Wenxia Le, and Hui Li. 2019. A power-saving pre-classifier for TCAM-based IP lookup. *Computer Networks* 164 (2019), 106898.

[24] Wenjun Li and Xianfeng Li. 2013. HybridCuts: A scheme combining decomposition and cutting for packet classification. In *Proceedings of the IEEE Annual Symposium on High-Performance Interconnects (HOTI)*.

[25] Wenjun Li, Xianfeng Li, and Hui Li. 2017. MEET-IP: Memory and energy efficient TCAM-based IP lookup. In *Proceedings of the International Conference on Computer Communication and Networks (ICCCN)*.

[26] Wenjun Li, Xianfeng Li, Hui Li, and Gaogang Xie. 2018. CutSplit: A Decision-Tree Combining Cutting and Splitting for Scalable Packet Classification. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*.

[27] Wenjun Li, Xinwei Liu, Wenxia Le, Hui Li, and Huayu Zhang. 2019. A practical range encoding scheme for TCAMs. In *Proceedings of the ACM Conference of the ACM Special Interest Group on Data Communication (SIGCOMM), Posters and Demos*.

[28] Wenjun Li, Tong Yang, Yeim-Kuan Chang, Tao Li, and Hui Li. 2019. TabTree: A TSS-assisted Bit-selecting Tree Scheme for Packet Classification with Balanced Rule Mapping. In *Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*.

[29] Wenjun Li, Tong Yang, Ori Rottenstreich, Xianfeng Li, Gaogang Xie, Hui Li, Balajee Vamanan, Dagang Li, and Huiping Lin. 2020. Tuple Space Assisted Packet Classification With High Performance on Both Search and Update. *IEEE Journal on Selected Areas in Communications* 38, 7 (2020), 1555–1569.

[30] Eric Liang, Hang Zhu, Xin Jin, and Ion Stoica. 2019. Neural Packet Classification. In *Proceedings of the ACM Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*.

[31] Will Lin, Yizhou Shan, Ryan Kosta, Arvind Krishnamurthy, and Yiying Zhang. 2024. SuperNIC: An FPGA-based, cloud-oriented SmartNIC. In *Proceedings of the ACM/SIGDA international symposium on field-programmable gate arrays (FPGA)*.

[32] Yuxi Liu, Yao Xin, Wenjun Li, Haoyu Song, Ori Rottenstreich, Gaogang Xie, Weichao Li, and Yi Wang. 2022. HybridTSS: A Recursive Scheme Combining Coarse- and Fine- Grained Tuples for Packet Classification. In *Proceedings of the ACM Asia-Pacific Workshop on Networking (APNet)*.

[33] Cong Luo, Chuhao Chen, Hao Mei, Ruyi Yao, Ying Wan, Wenjun Li, Sen Liu, Bin Liu, and Yang Xu. 2022. BubbleTCAM: bubble reservation in SDN switches for fast TCAM update. In *Proceedings of the IEEE/ACM International Symposium on Quality of Service (IWQoS)*.

[34] Chad R. Meiners, Alex X. Liu, and Eric Torng. 2010. *Hardware based packet classification for high speed Internet routers*. Springer.

[35] Mellanox. 2024. Mellanox Adapters Programmer's Reference Manual (PRM). Website. https://network.nvidia.com/files/doc-2020/ethernet-adapters-programming-manual.pdf.

[36] Nvidia. 2024. ConnectX NICs. Website. https://www.nvidia.com/en-us/networking/ethernet-adapters/.

[37] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, , et al. 2015. The design and implementation of open vSwitch. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[38] Yaxuan Qi, Jeffrey Fong, Weirong Jiang, Bo Xu, Jun Li, and Viktor Prasanna. 2010. Multi-dimensional packet classification on FPGA: 100 Gbps and beyond. In *Proceedings of the International Conference on Field-Programmable Technology (FPT)*.

[39] Yaxuan Qi, Lianghong Xu, Baohua Yang, Yibo Xue, and Jun Li. 2009. Packet classification algorithms: From theory to practice. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*.

[40] Yun R Qu and Viktor K Prasanna. 2015. High-performance and dynamically updatable packet classification engine on FPGA. *IEEE Transactions on Parallel and Distributed Systems* 27, 1 (2015), 197–209.

[41] Yun R Qu, Hao H Zhang, Shijie Zhou, and Viktor K Prasanna. 2015. Optimizing many-field packet classification on FPGA, multi-core general purpose processor, and GPU. In *Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*.

[42] Yun R Qu, Shijie Zhou, and Viktor K Prasanna. 2013. High-performance architecture for dynamically updatable packet classification on FPGA. In *Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*.

[43] Alon Rashelbach, Ori Rottenstreich, and Mark Silberstein. 2022. Scaling Open vSwitch with a Computational Cache. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[44] Yaniv Sadeh, Ori Rottenstreich, and Haim Kaplan. 2022. Optimal weighted load balancing in TCAMs. *IEEE/ACM Transactions on Networking* 30, 3 (2022), 985–998.

[45] Nick Shelly, Ethan J Jackson, Teemu Koponen, Nick McKeown, and Jarno Rajahalme. 2014. Flow caching for high entropy packet fields. In *Proceedings of the ACM workshop on Hot topics in software defined networking (HotSDN)*.

[46] Qilong Shi, Chengjun Jia, Wenjun Li, Zaoxing Liu, Tong Yang, Jianan Ji, Gaogang Xie, Weizhe Zhang, and Minlan Yu. 2024. BitMatcher: Bit-level counter adjustment for sketches. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*.

[47] Sumeet Singh, Florin Baboescu, George Varghese, and Jia Wang. 2003. Packet classification using multidimensional cutting. In *Proceedings of the ACM Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*.

[48] Venkatachary Srinivasan, Subhash Suri, and George Varghese. 1999. Packet Classification using Tuple Space Search. In *Proceedings of the ACM Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*.

[49] Venkatachary Srinivasan, George Varghese, Subhash Suri, and Marcel Waldvogel. 1998. Fast and Scalable Layer Four Switching. In *Proceedings of the ACM Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*.

[50] Mingqian Sun, Guangwei Xie, Fan Zhang, Wei Guo, Xitian Fan, Tianyang Li, Li Chen, and Jiayu Du. 2024. PTME: A Regular Expression Matching Engine Based on Speculation and Enumerative Computation on FPGA. *ACM Transactions on Reconfigurable Technology and Systems* 18, 1 (2024), 1–28.

[51] Jing Tan, Gaofeng Lv, and GuanJie Qiao. 2021. MBitTree: A fast and scalable packet classification for software switches. In *Proceedings of the IEEE Annual Symposium on High-Performance Interconnects (HOTI)*.

[52] David E Taylor. 2005. Survey and taxonomy of packet classification techniques. *Comput. Surveys* 37, 3 (2005), 238–275.

[53] David E Taylor and Jonathan S Turner. 2007. ClassBench: A packet classification benchmark. *IEEE/ACM Transactions on Networking* 15, 3 (2007), 499–511.

[54] Balajee Vamanan, Gwendolyn Voskuilen, and TN Vijaykumar. 2010. EffiCuts: Optimizing Packet Classification for Memory and Throughput. In *Proceedings of the ACM Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*.

[55] George Varghese and Jun Xu. 2022. *Network Algorithmics: An interdisciplinary approach to designing fast networked devices.* Morgan Kaufmann.

[56] Ying Wan, Haoyu Song, Yang Xu, Chuwen Zhang, Yi Wang, and Bin Liu. 2021. Adaptive Batch Update in TCAM: How Collective Optimization Beats Individual Ones. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*.

[57] Yao Xin, Chengjun Jia, Wenjun Li, Ori Rottenstreich, Yang Xu, Gaogang Xie, Zhihong Tian, and Jun Li. 2024. A Heterogeneous and Adaptive Architecture for Decision-tree-based ACL Engine on FPGA. *IEEE Trans. Comput.* 74, 1 (2024), 263–277.

[58] Yao Xin, Wenjun Li, Chengjun Jia, Xianfeng Li, Yang Xu, Bin Liu, Zhihong Tian, and Weizhe Zhang. 2024. Recursive Multi-Tree Construction With Efficient Rule Sifting for Packet Classification on FPGA. *IEEE/ACM Transactions on Networking* 32, 2 (2024), 1707–1722.

[59] Yao Xin, Wenjun Li, Guoming Tang, Tong Yang, Xiaohe Hu, and Yi Wang. 2022. FPGA-Based Updatable Packet Classification Using TSS-Combined Bit-Selecting Tree. *IEEE/ACM Transactions on Networking* 30, 6 (2022), 2760–2775.

[60] Yao Xin, Wenjun Li, Gaogang Xie, Yang Xu, and Yi Wang. 2022. Updatable Packet Classification on FPGA with Bounded Worst-Case Performance. In *Proceedings of the IEEE Annual Symposium on High-Performance Interconnects (HOTI)*.

[61] Yao Xin, Wenjun Li, Gaogang Xie, Yang Xu, and Yi Wang. 2023. A Parallel and Updatable Architecture for FPGA-Based Packet Classification With Large-Scale Rule Sets. *IEEE Micro* 43, 2 (2023), 110–119.

[62] Yao Xin, Yuxi Liu, Wenjun Li, Ruyi Yao, Yang Xu, and Yi Wang. 2021. KickTree: A recursive algorithmic scheme for packet classification with bounded worst-case performance. In *Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*.

[63] Chengcheng Xu, Shuhui Chen, Jinshu Su, Siu-Ming Yiu, and Lucas CK Hui. 2016. A survey on regular expression matching for deep packet inspection: Applications, algorithms, and hardware platforms. *IEEE Communications Surveys & Tutorials* 18, 4 (2016), 2991–3029.

[64] Baohua Yang, Jeffrey Fong, Weirong Jiang, Yibo Xue, and Jun Li. 2012. Practical multituple packet classification using dynamic discrete bit selection. *IEEE Trans. Comput.* 63, 2 (2012), 424–434.

[65] Ruyi Yao, Cong Luo, Xuandong Liu, Ying Wan, Bin Liu, Wenjun Li, and Yang Xu. 2021. MagicTCAM: A Multiple-TCAM Scheme for Fast TCAM Update. In *Proceedings of the IEEE International Conference on Network Protocols (ICNP)*.

[66] Ruyi Yao, Cong Luo, Hao Mei, Chuhao Chen, Wenjun Li, Ying Wan, Sen Liu, Bin Liu, and Yang Xu. 2023. Colue: Collaborative tcam update in sdn switches. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*.

[67] Sorrachai Yingchareonthawornchai, James Daly, Alex X Liu, and Eric Torng. 2018. A Sorted-Partitioning Approach to Fast and Scalable Dynamic Packet Classification. *IEEE/ACM Transactions on Networking* 26, 4 (2018), 1907–1920.

[68] Xinyi Zhang, Qianrui Qiu, Zhiyuan Xu, Peng He, Xilai Liu, Kavé Salamatian, Changhua Pei, and Gaogang Xie. 2025. NPC: Rethinking Dataplane through Network-aware Packet Classification. In *Proceedings of the ACM Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*.

[69] Jincheng Zhong, Ziling Wei, Shuang Zhao, and Shuhui Chen. 2023. TupleTree: A High-Performance Packet Classification Algorithm Supporting Fast Rule-Set Updates. *IEEE/ACM Transactions on Networking* 31, 5 (2023), 2027–2041.

[70] Annus Zulfiqar, Ali Imran, Venkat Kunaparaju, Ben Pfaff, Gianni Antichi, and Muhammad Shahbaz. 2025. Gigaflow: Pipeline-Aware Sub-Traversal Caching for Modern SmartNICs. In *Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[71] Annus Zulfiqar, Ben Pfaff, Gianni Antichi, and Muhammad Shahbaz. 2025. Kairo-Incremental View Maintenance for Scalable Virtual Switch Caching. In *Proceedings of the ACM Conference of the ACM Special Interest Group on Data Communication (SIGCOMM), Posters and Demos*.

[72] Annus Zulfiqar, Ben Pfaff, William Tu, Gianni Antichi, and Muhammad Shahbaz. 2023. The slow path needs an accelerator too! *ACM SIGCOMM Computer Communication Review* 53, 1 (2023), 38–47.