TreeCAM: *Decoupling Updates and Lookups in Packet Classification*

Balajee Vamanan and T. N. Vijaykumar
School of Electrical & Computer Engineering

CoNEXT 2011

PURDUE
UNIVERSITY

Presenter: Kyle Wang
2012-06-05

Packet Classification

- ***Packet Classification:***
  - Find highest-priority rule that matches a packet
- Packet classification is key for
  - Security, traffic monitoring/analysis, QoS
- ***Classifier:*** a set of rules

Source IP	Destination IP	Source Port	Dest. Port	Protocol	Action
120…0/24	198…0/2	0:65535	11:17	0xFF/0xFF	Accept
138…1/0	174…0/8	50:10000	0:65535	0x06/0xFF	Deny

Packet classification prevalent in modern routers

Trends in Packet Classification

- Line rates increasing (40 Gbps now, 160 Gbps soon)
- Classifier size (number of rules) increasing
  - Custom rules for VPNs, QoS
- Rules are getting more dynamic too

⇒ Larger classifiers at faster lookup & update rates

- Much work on lookups, but not on updates

Must perform well in lookups and updates at low power

Characteristics of updates

- Two flavors
  - Virtual interfaces: add/remove 10,000s rules per minute
  - QoS: update 10s of rules per flow (milliseconds)
  - For either flavor, update rate (1/ms) << packet rate (1/ns)
- Current approaches
  - Either incur high update effort despite low update rates
    - Eat up memory bandwidth
    - Hold-up memory for long
      - → packet drops, buffering complexity, missed deadlines
  - Or do not address updates
- Recent OpenFlow, online classification → faster updates

Updates remain a key problem

Current Approaches

- TCAM
  - Unoptimized TCAMs search all rules per lookup → high power
  - Modern partitioned TCAMs prune search → reduce power
    - Extended TCAMs [*ICNP 2003*]
  - Physically order rules per priority for fast highest-priority match
    - This ordering fundamentally affects update effort
  - Updates move many rules to maintain order
    - E.g., updates 10 per ms; lookups 1 per 10 ns; 100,000 rules
    - If 10 % updates move (read+write) 10 % rules
      - Updates need 20,000 ops/ms = 0.2 op per 10 ns
      - → 20% bandwidth overhead

High-effort* updates in TCAM *degrade throughput & latency

Current Approaches (Contd.)

- Decision Trees:
  - Build decision trees to prune search per lookup
  - Do not address updates
  - No ordering like TCAMs but updates may cause tree imbalance
    - Imbalance increase lookup accesses
    - Re-balancing is costly

Previous schemes are not good in both lookups and updates

Our Contributions

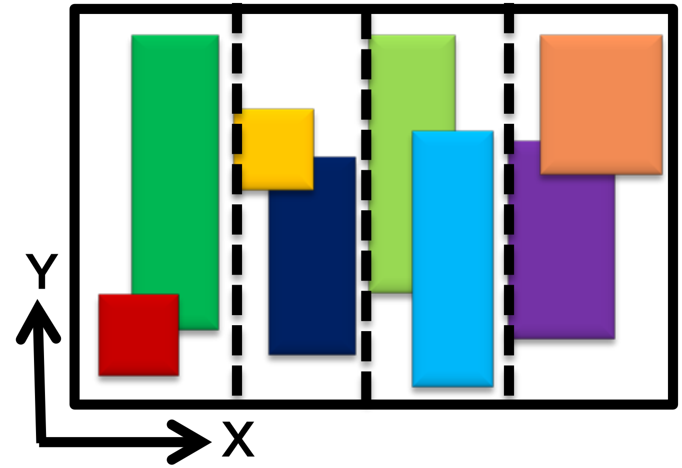
- TreeCAM: Three *novel ideas*
  - *Dual tree versions* to decouple lookups and updates
    - *coarse tree* in TCAM → reduce lookup accesses
      - Tree/TCAM hybrid
    - *fine tree* in control memory → reduce update effort
  - *Interleaved layout* of leaves to cut ordering effort
  - *Path-by-path updates* to avoid hold-up of memory
    - Allow non-atomic updates interspersed with lookups
- Performs well in lookups and updates
  - 6-8 TCAM accesses for lookups
  - Close to ideal TCAM for updates

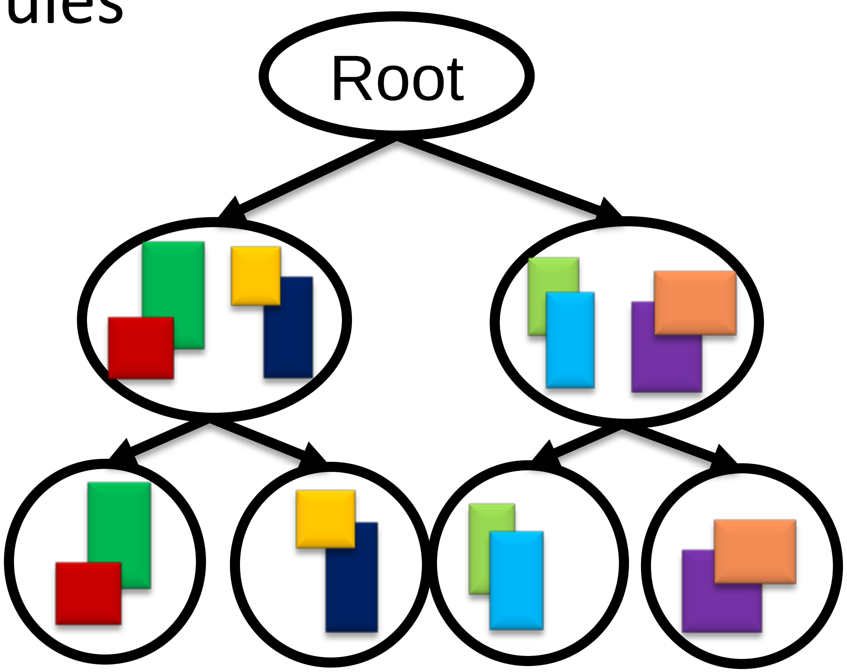
Outline

- Introduction
- Background: Decision Trees
- Dual tree versions
  - Coarse Tree
  - Fine Tree
- Updates using Interleaved Layout
- Path-by-path updates
- Results
- Conclusion

Background: Decision Trees

- Rules are hypercubes in rule space
- Builds decision tree by cutting rule space to separate rules into smaller subspaces (child nodes)
- Stop when a small number of rules at a leaf called binth (e.g., 16)
- Packets traverse tree during classification
- Many heuristics
  - Dimension, number of cuts



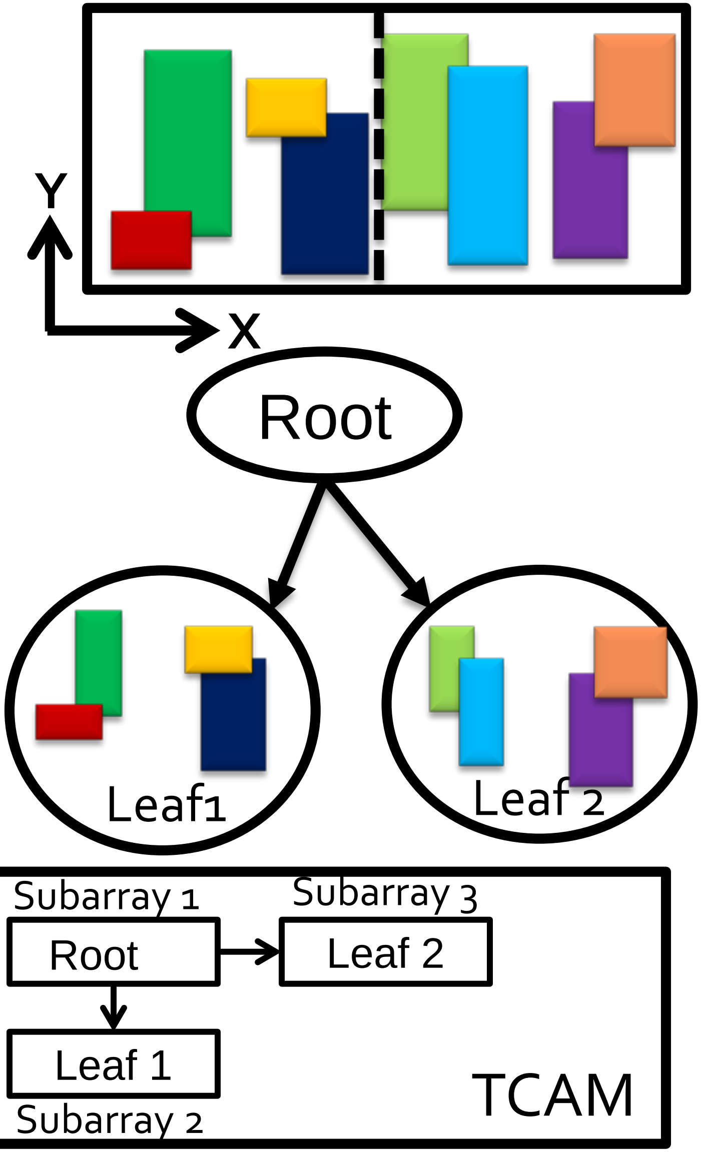


Outline

- Introduction
- Background: Decision Trees
- Dual tree versions
  - Coarse Tree
  - Fine Tree
- Updates using Interleaved Layout
- Path-by-path updates
- Results
- Conclusion

TreeCAM Coarse Tree (Version#1 for lookups)

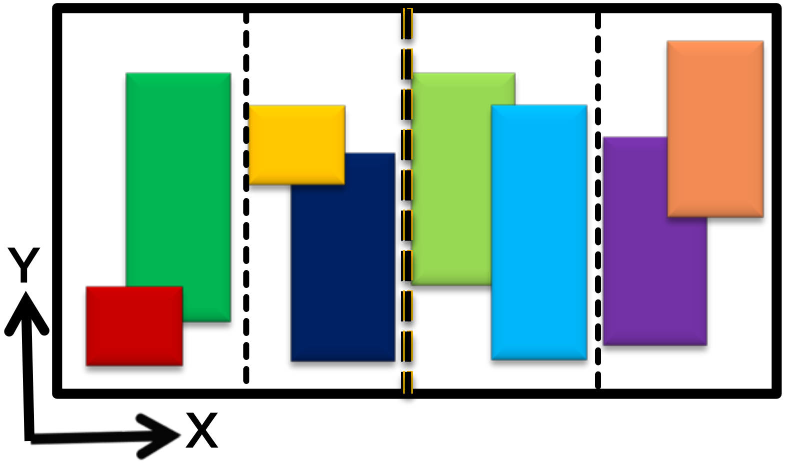
- Idea: partition rules among TCAM subarrays using decision trees
- 4k-entry subarrays → coarse tree with each leaf in a subarray
  - 2-deep tree → fast lookup
- Packets traverse subarrays
- Previous heuristics complicated
- We propose simple sort heuristic
  - Sort rules & cut equally
    - *EffiCuts* → min. rule replication

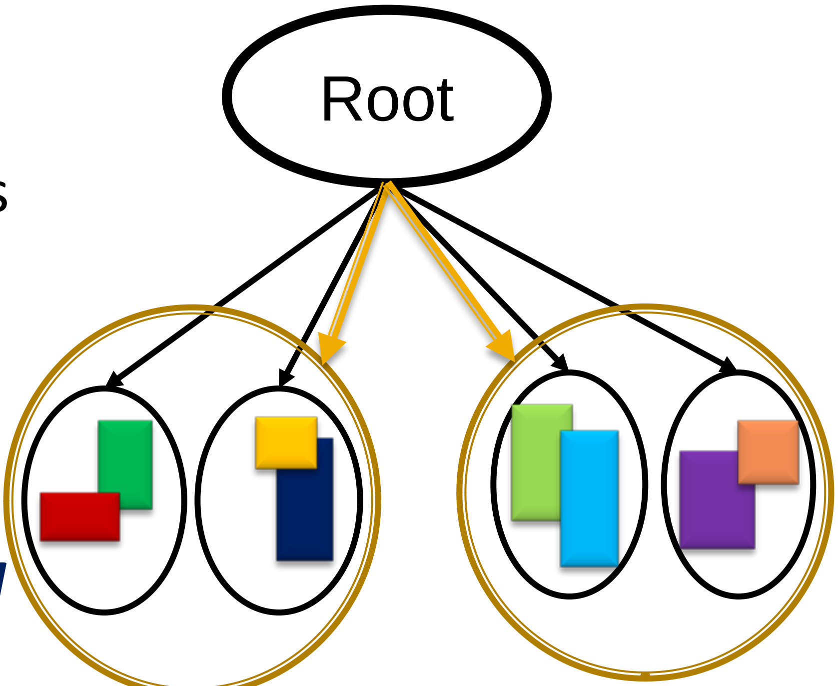


Coarse Trees → Search Pruning (trees) + Parallel Search (TCAM)

TreeCAM Fine Tree (Version#2 for updates)

- Key observation: A packet cannot match multiple leaves → only rules within the same leaf need ordering
- Reduce update effort → Tree with small binth – *fine tree*
  - Not real, just annotations
  - One *coarse-tree* leaf contains some contiguous fine-tree leaves
  - Both trees kept consistent
- Updates slow → store *fine tree* in control memory





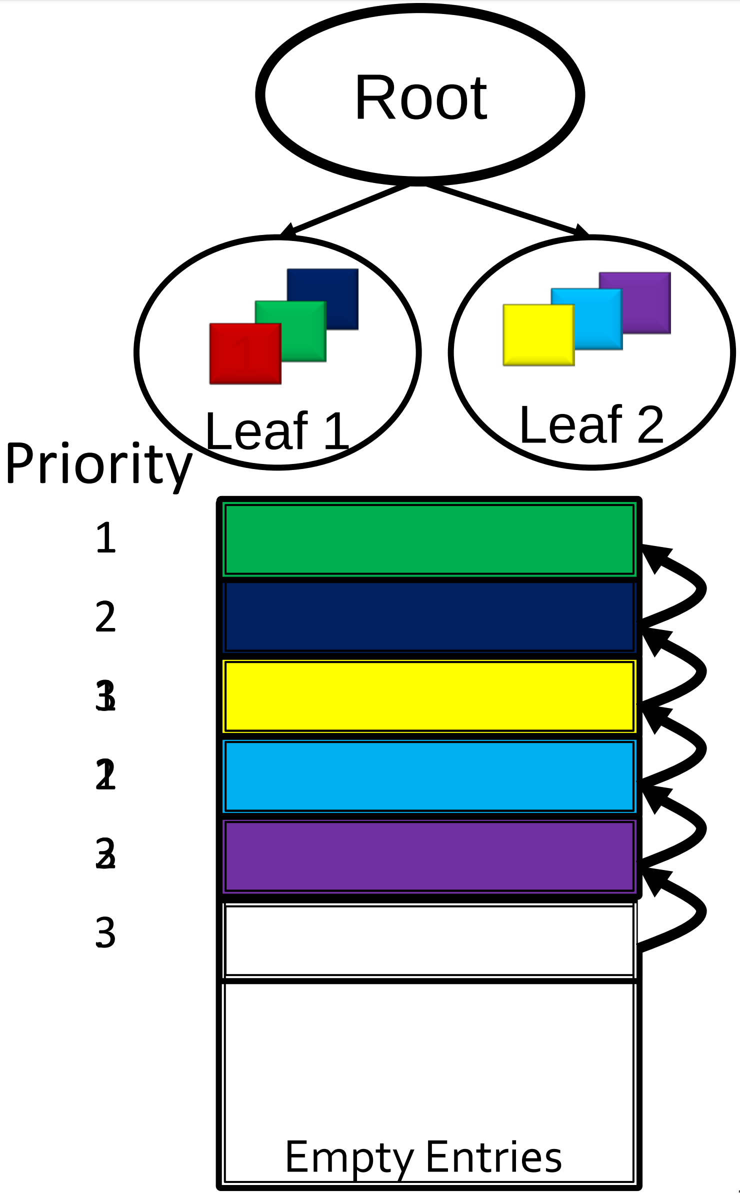
Fine Trees: Small binth, reduced ordering effort in TCAM

Outline

- Introduction
- Background: Decision Trees
- Dual tree versions
  - Coarse Tree
  - Fine Tree
- Updates using Interleaved Layout
- Path-by-path updates
- Results
- Conclusion

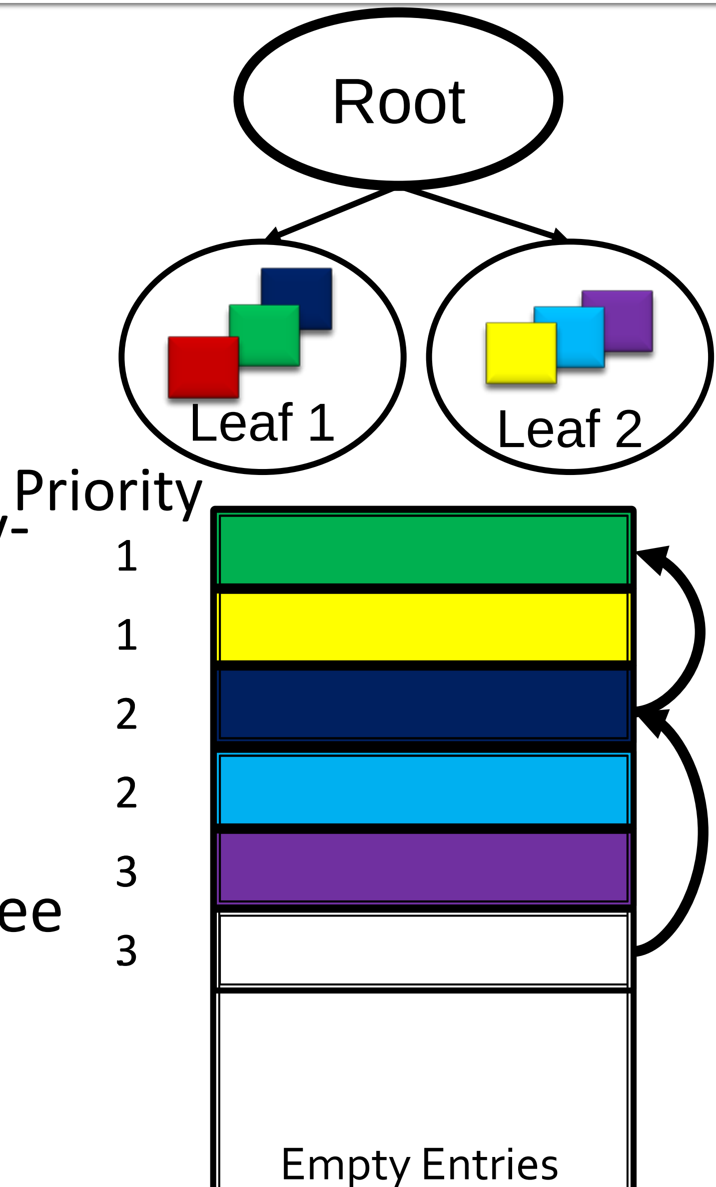
Updates

- Add a rule
  - Create empty space at right priority level via repeated swaps
  - With naïve, contiguous layout of leaves, requires many repeated swaps
    - As many swaps as #rules
- Observation: Only overlapping rules need ordering per priority
  - Too hard in full generality

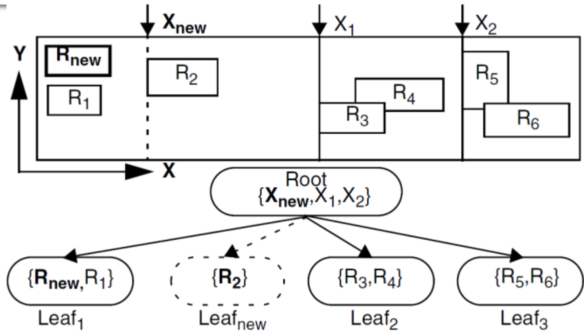


Interleaved Layout

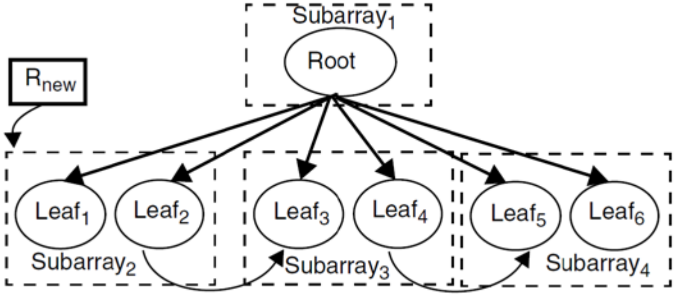
- Insight: Leaves are naturally non-overlapping → only rules in a leaf need to be ordered
  - Trivial unlike general overlap
- *Interleave* rules across all leaves
  - Contiguously place same priority-level rules from all leaves
  - Order only across priority levels
  - Move at most one rule per level
    - binth levels → small for fine tree
    - Update effort ≈ binth swaps



Updates with interleaved layout (cont.)



- Updating for three cases
  - No Re-balancing
    - Empty entries exist in target leaf
  - Local Re-balancing
    - No room in target leaf, displace the borderline rules of target leaf to neighboring leaf, or create a new leaf
  - Global Re-balancing
    - No room in target leaf and subarray, displace rules recursively from one to the next, until empty entries exist in target subarray
- Worst case where rules move across TCAM subarrays
  - Interleaved layout effort ≈ 3*binth *#subarrays swaps
    - ≈ 3*8*25 ≈ 600 swaps (100K rules and 4K rules/subarray)
  - Contiguous layout effort ≈ #rules
    - ≈ 100K swaps (100K rules)



Path-by-path Updates

- Problem: Update moves hold up memory for long
- Make updates non-atomic
  - Packet lookups can be interspersed between updates
- Procedure
  - (1) place the to-be-displaced rule into the back-up TCAM;
  - (2) shrink the boundary of the source subarray;
  - (3) remove (invalidate) the rule from the source;
  - (4) add the rule to the destination subarray;
  - (5) expand the boundary of the destination subarray; and
  - (6) invalidate the back-up TCAM.
- The rule swap in (4) still causes atomic operations, but few
  - 3 (read, write, invalidate) versus 1250 (global re-balancing)

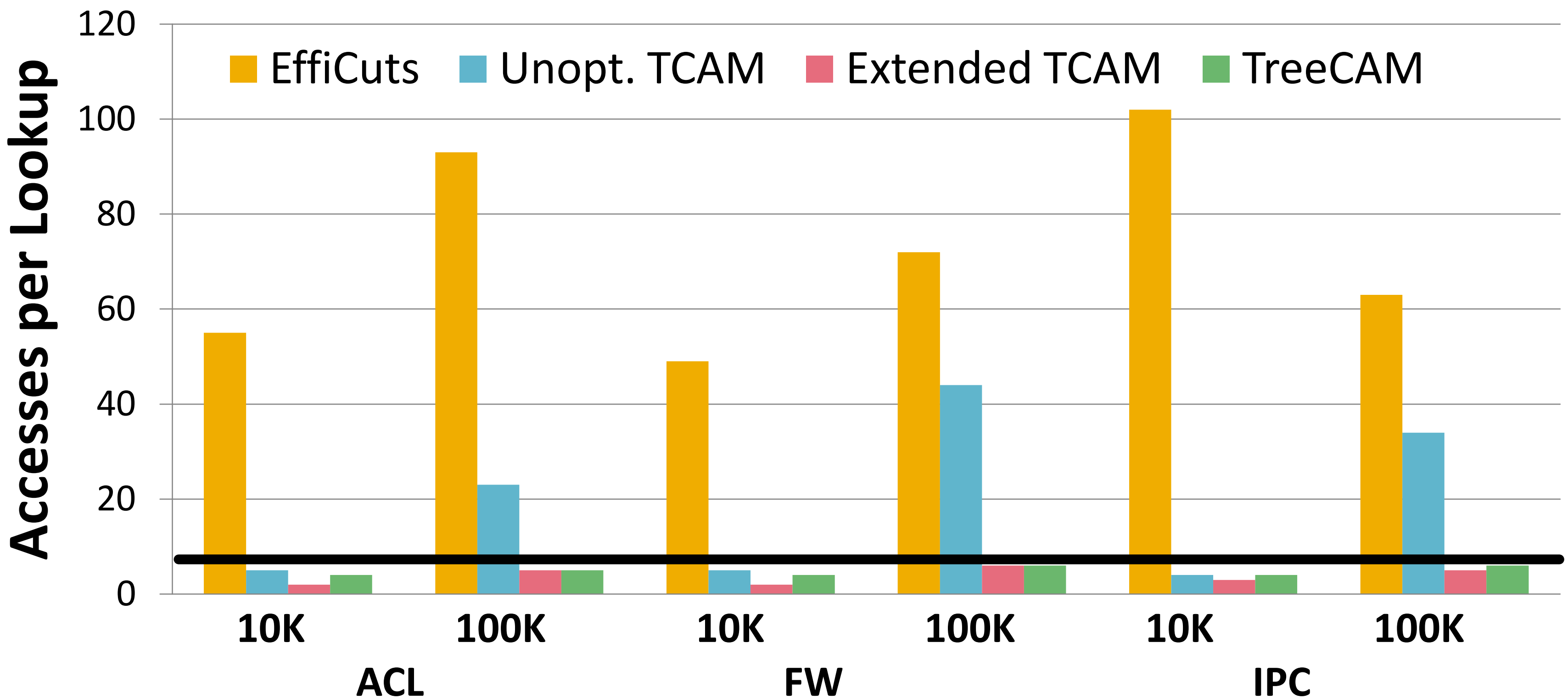
Outline

- Introduction
- Background: Decision Trees
- Dual tree versions
  - Coarse Tree
  - Fine Tree
- Updates using Interleaved Layout
- Path-by-path updates
- Results
- Conclusion

Experimental Methodology

- Key metrics: Lookups & Updates
  - Lookup accesses: #accesses per packet match
  - Update effort: #accesses per one-rule addition
- Software simulator
- EffiCuts, TCAM, and TreeCAM
  - TCAM: Unoptimized & Partitioned TCAM (Extended TCAM)
    - 4K subarrays
  - Decision tree algorithm (EffiCuts)
  - Tree/TCAM hybrid (TreeCAM)
    - Coarse tree binth = 4K, Fine tree binth = 8
- ClassBench generated classifiers – ACL, FW and IPC

Accesses per Lookup



- EffiCuts require many more SRAM accesses than TCAMs
- Extended TCAM and TreeCAM require only at most 8 accesses even for 100,000 rule classifiers
  - Extended TCAM does not handle updates

Update Effort

- Compare TCAM-basic, TCAM-ideal and TreeCAM
  - EffiCuts and Extended TCAM do not discuss updates
- TCAM-basic: periodic empty entries
- TCAM-Ideal: Adapt existing work on longest prefix match for packet classification
  - Identify groups of overlapping rules, and ideally assume
    - Enough empty entries at the end of every group
    - Two groups of overlapping rules DO NOT merge (*ideal*)
- We generate worst case update stream for each scheme
  - Adds rules to the left-most leaf of the largest separable tree
  - Removes rules from the right-most leaf of the tree

Worst-case Update Effort (cont.)

Classifier Type	Classifier Size	TCAM-basic		TCAM-ideal		TreeCAM	
		Empty Slots	Max # TCAM Ops	Max. Overlaps	Max # TCAM Ops	#Sub-arrays	Max # TCAM Ops
ACL	10K	44K	**30K**	67	**134**	3	**91**
	100K	60K	**276K**	166	**332**	19	**684**
FW	10K	68K	**64K**	90	**180**	3	**112**
	100K	82K	**567K**	295	**590**	29	**1069**
IPC	10K	43K	**22K**	62	**124**	1	**24**
	100K	46K	**236K**	137	**274**	11	**385**

TreeCAM is close to Ideal and two orders of magnitude better than TCAM-basic

Conclusion

- Previous schemes do not perform well in both lookups and updates
- TreeCAM uses three techniques to address this challenge:
  - Dual tree versions: Decouples lookups and updates
    - *Coarse trees* for lookups and *fine trees* for updates
  - Interleaved layout bounds the update effort
  - Path-by-path update enables non-atomic updates which can be interspersed with packet lookups
- TreeCAM achieves 6 – 8 lookup accesses and close to ideal TCAM for updates, even for large classifiers (100K rules)

TreeCAM scales well with classifier size, line rate, and update rate