北京大學
PEKING UNIVERSITY

TabTree: A TSS-assisted Bit-selecting Tree Scheme for Packet Classification with Balanced Rule Mapping

Wenjun Li[*,†,‡], Tong Yang[‡], Yeim-Kuan Chang[§], Tao Li[⁋] and Hui Li[*,†]

[*]School of Electronic and Computer Engineering, Peking University,
[†]Peng Cheng Laboratory, [‡]EECS, Peking University, [§]NCKU, [⁋]NUDT

ACM/IEEE ANCS 2019

Cambridge, UK, September 24, 2019

Outline

- Background
- Motivation
- Proposed Algorithm
- Evaluation
- Conclusion

Part 1： Background

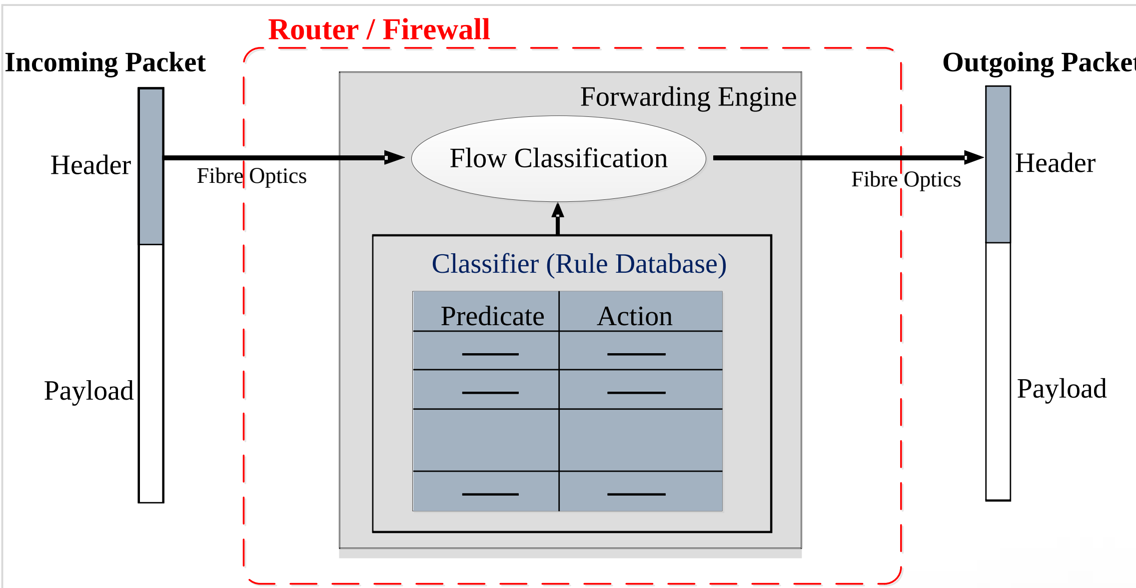
1

Packet Classification

A Little Review on Related Work

Packet Classification

- **Key for policy enforcement in packet forwarding**
  - Firewall, QoS, OpenFlow, P4, etc.



exact match

match range

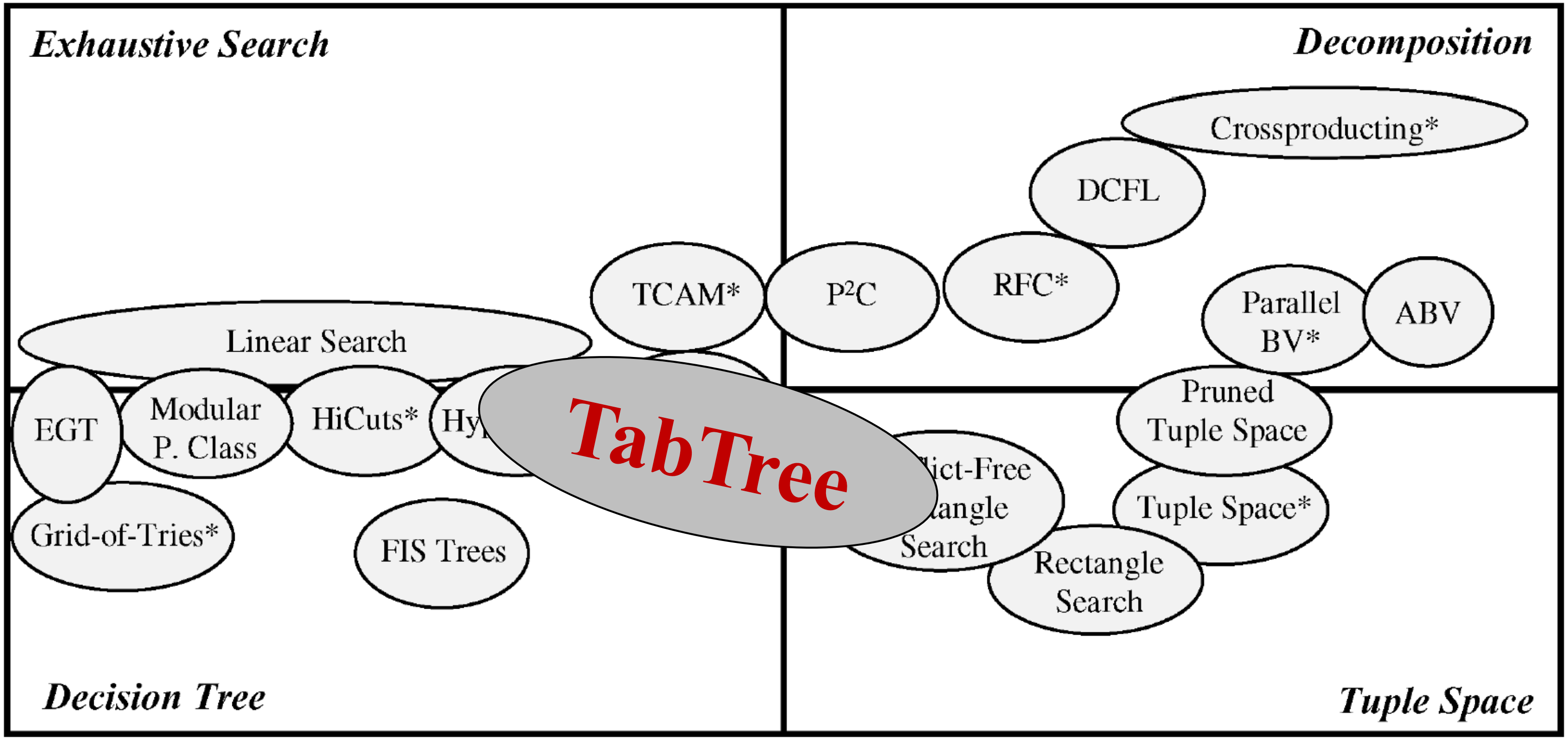
match any

#	An example OpenFlow 1.0 classifier/flow table (12-tuple)						Action
r_1	**Ingress Port**	**Ether src**	**Ether dst**	**Ether type**	**VLAN id**	**VLAN priority**	$Action_1$
	3	*	*	*2048*	*	*	
	IP src	**IP dst**	**IP proto**	**IP ToS bits**	**TCP/UDP Src Port**	**TCP/UDP Dst Port**	
	15.25.70.8/30	*18.15.125.3/28*	*0x11/0xff*	*1*	*1024 : 65535*	*80*	

…

Existing Solutions

- Well-known taxonomy from David E. Taylor[CSUR 2005]



Notes: adjacent techniques are related; hybrid techniques overlap quadrant boundaries; $*$ *denotes a seminal technique*

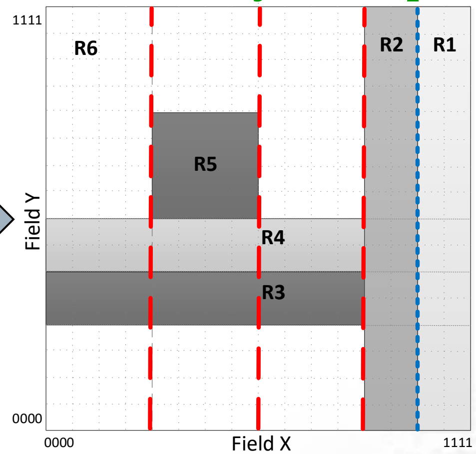
- Our proposed TabTree: A hybrid approach

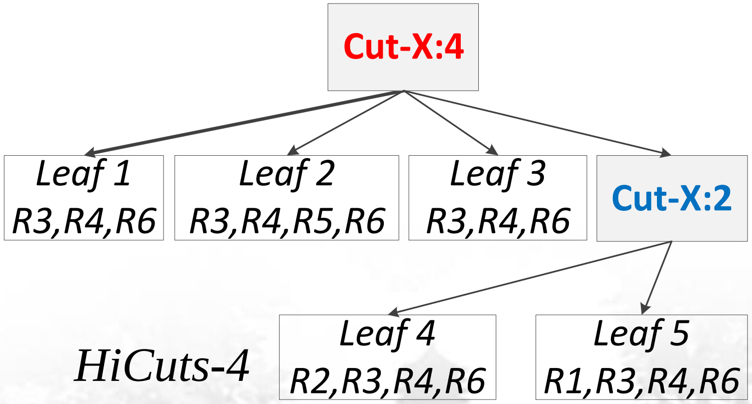
A Little Review on Decision Tree

Decision-tree construction in packet classification

- 1. Rule table matching ↔ Point location in geometric space
- 2. Partition the searching space into sub-spaces recursively
  - Root node: Whole searching space containing all rules
  - Internal node: #rule covered by sub-space > a predefined number of rules
  - Leaf node: #rule covered by sub-space <= a predefined number of rules

Rule #	Field X	Field Y	*Action*
R1	111*	*	*A1*
R2	110*	*	*A2*
R3	*	010*	*A3*
R4	*	011*	*A4*
R5	01**	10**	*A5*
R6	*	*	*A6*



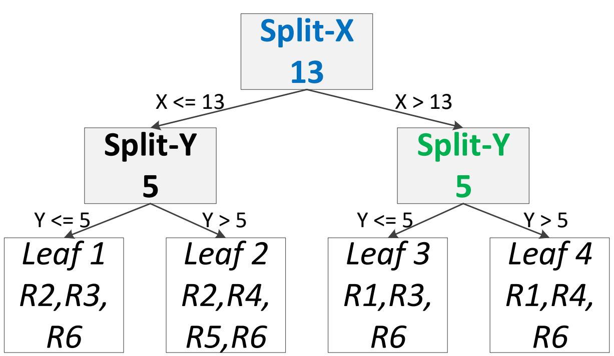


Two major threads of building decision-trees

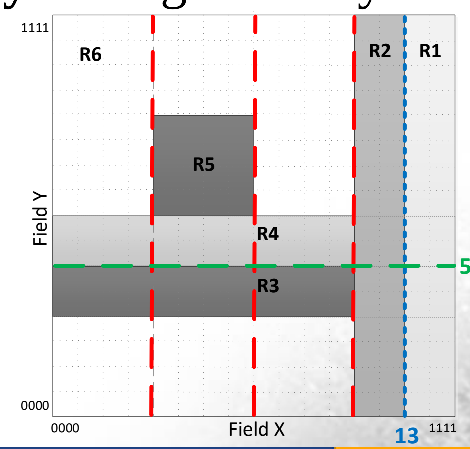
- Equal-sized cutting & Equal-dense splitting

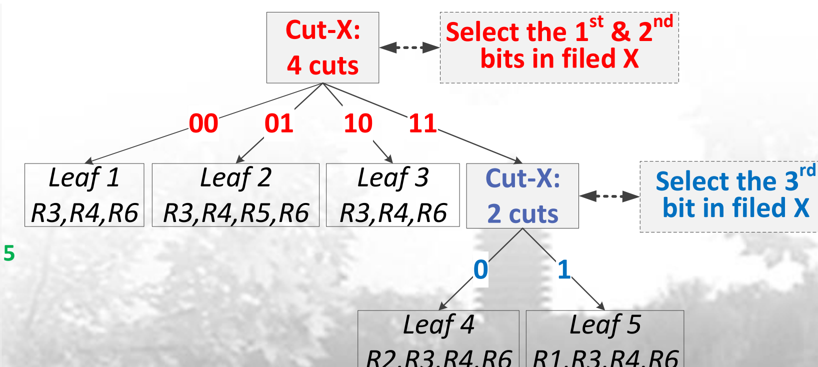
Two Major Threads in Decision-trees

- **Equal-dense splitting based point-comparing**
  - Unequal-sized sub-spaces containing nearly equal number of rules
  - e.g., HyperSplit, ParaSplit, SmartSplit, PartitionSort, etc.
- **Equal-sized cutting based bit-selecting**
  - Separate the searching space into many equal-sized sub-spaces
  - Two major threads based on bit-selecting methods
    - Select **orderly** from the most to the least significant bits, such as HiCuts
    - Select **discretely** among arbitrary field bits, such as ModularPC



HyperSplit-4





HiCuts-4

A Little Review on TSS

TSS (Tuple Space Search) for packet classification

- Partition rules into a set of hash tables based on prefix length

Rule #	Field X	Field Y	*Action*
R1	111*	*	*A1*
R2	110*	*	*A2*
R3	*	010*	*A3*
R4	*	011*	*A4*
R5	01**	10**	*A5*
R6	*	*	*A6*

Rule #	Field X	Field Y	*Action*
R1	111*	*	*A1*
R2	110*	*	*A2*

Tuple 1: (3, 0)
Priority: 6

Rule #	Field X	Field Y	*Action*
R3	*	010*	*A3*
R4	*	011*	*A4*

Tuple 2: (0, 3)
Priority: 4

Rule #	Field X	Field Y	*Action*
R5	01**	10**	*A5*

Tuple 3: (2, 2)
Priority: 2

Rule #	Field X	Field Y	*Action*
R6	*	*	*A6*

Tuple 4: (0, 0)
Priority: 1

PSTSS (Priority Sorting TSS) used in Open vSwitch

- Introduce a pre-computed priority for each tuple space, so that each search can terminate as soon as a match is found

TSS can separate rules into subsets without any replications.

Part 2：Motivation

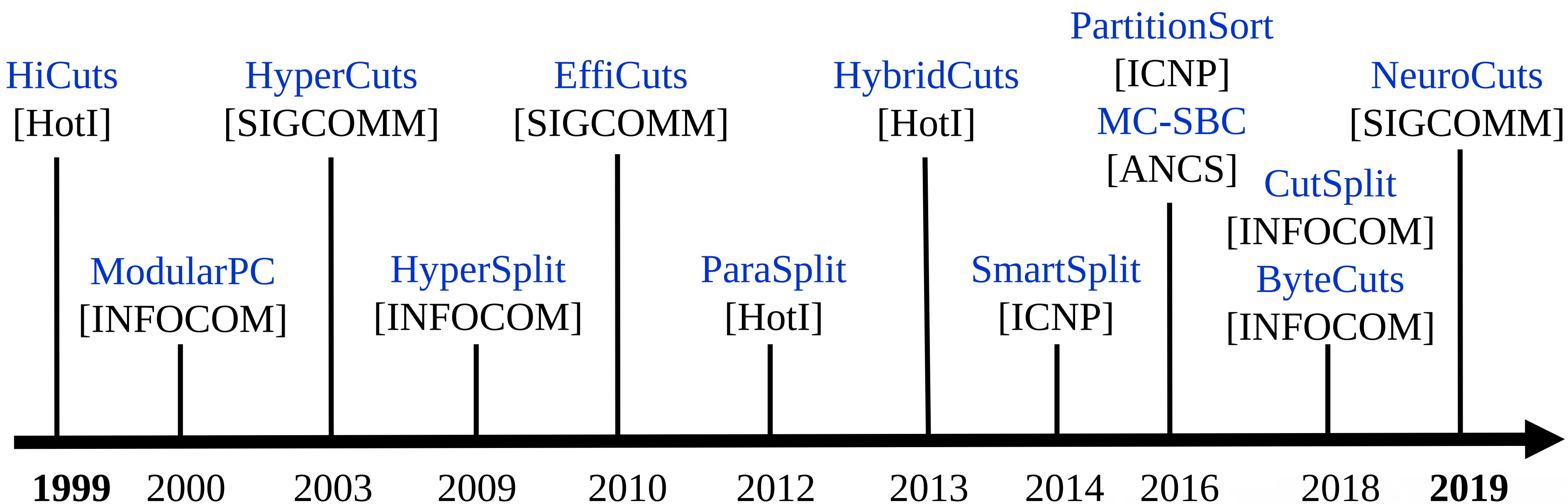
2

Well studied

Why yet another decision tree?

Why Yet Another Decision Tree?

- Well studied: The PAST two decades?

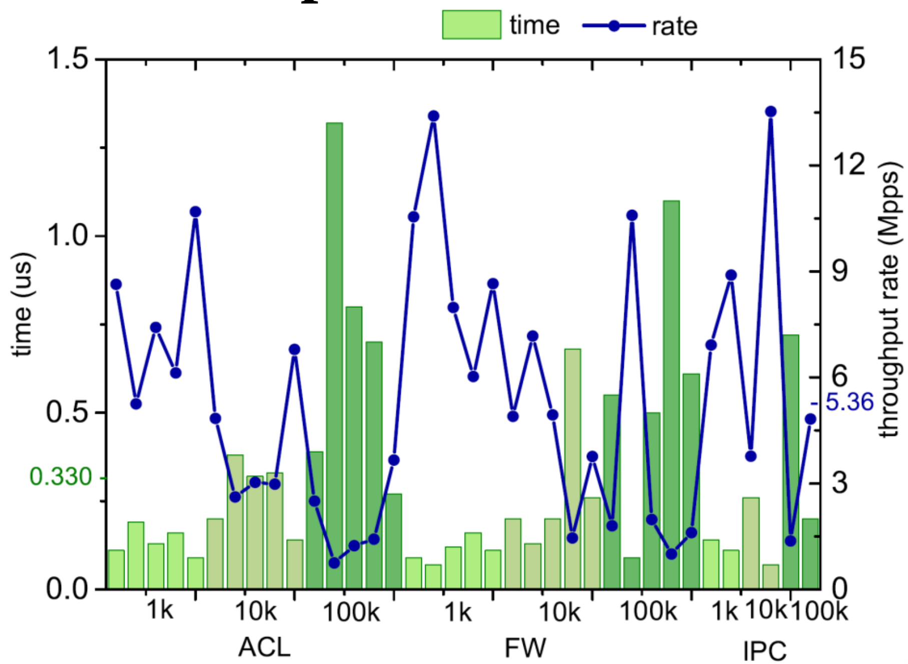


- But still far away from SDN: The LOST two decades!
  - The popular Open vSwitch still uses a variant of TSS (proposed in **1999**) for its table lookups, which is less efficient than decision trees on lookups. The primary reason is its good support for fast rule updates.

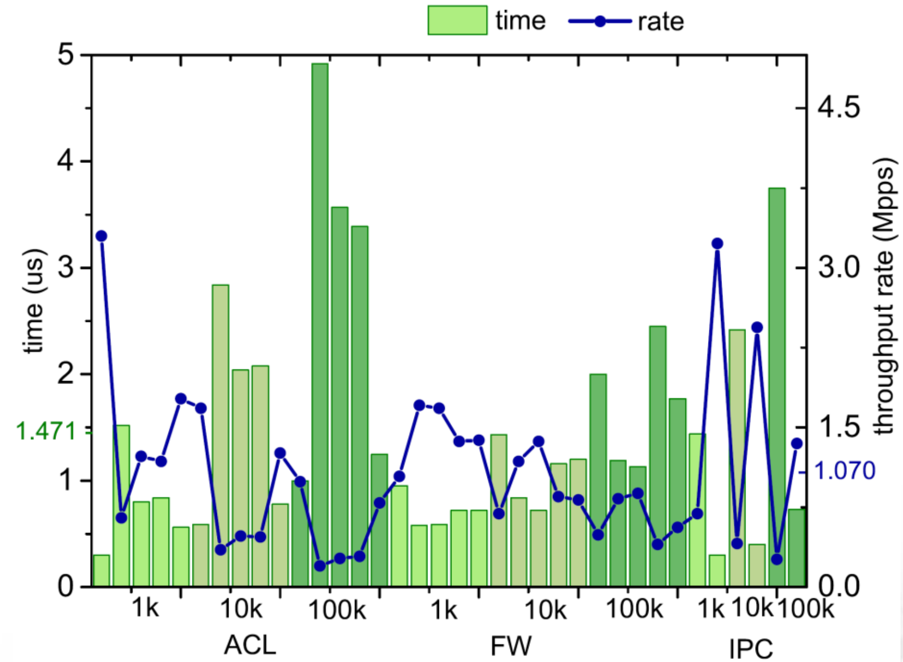
What is the Performance of Software based Packet Classifications?

- Classification performance without caching: A few Gbps

➢ CutSplit[INFOCOM 2018]



➢ PSTSS[NSDI 2015]



Thus, software based packet classifications are also still far away from high performance network.

Thus, Can We…

- **Motivation 1: for rule updates**
  - **Can we use decision trees for packet classification in OVS?**
    - **Can we build trees that also achieve high performance on updates?**
      - ✓ **Can we avoid rule replications in decision trees completely?**

- **Motivation 2: for FPGA acceleration**
  - **Can we use FPGA to accelerate packet classification in OVS?**
    - **Can we build trees that are favorable for FPGA implementations?**
      - ✓ **Can we build decision trees that are balanced and depth bounded?**

Can we design a tree scheme for packet classification in SDN, which is not only suitable for fast rule updates, but also desirable for FPGA implementations and optimizations?

Part 3：Proposed Algorithm

3

TabTree

TSS-assisted bit-selecting Tree

Ideas & Challenges

Decision tree

- ✓ **Pros:** Fast packet classification
- ✓ **Cons:** Slow rule update

TSS

- ✓ **Pros:** Fast rule update
- ✓ **Cons:** Slow packet classification

To foster the strengths and circumvent the weaknesses of decision tree and TSS, the idea directly perceived is to design a heterogeneous framework that can take advantage of both decision tree and TSS approaches:

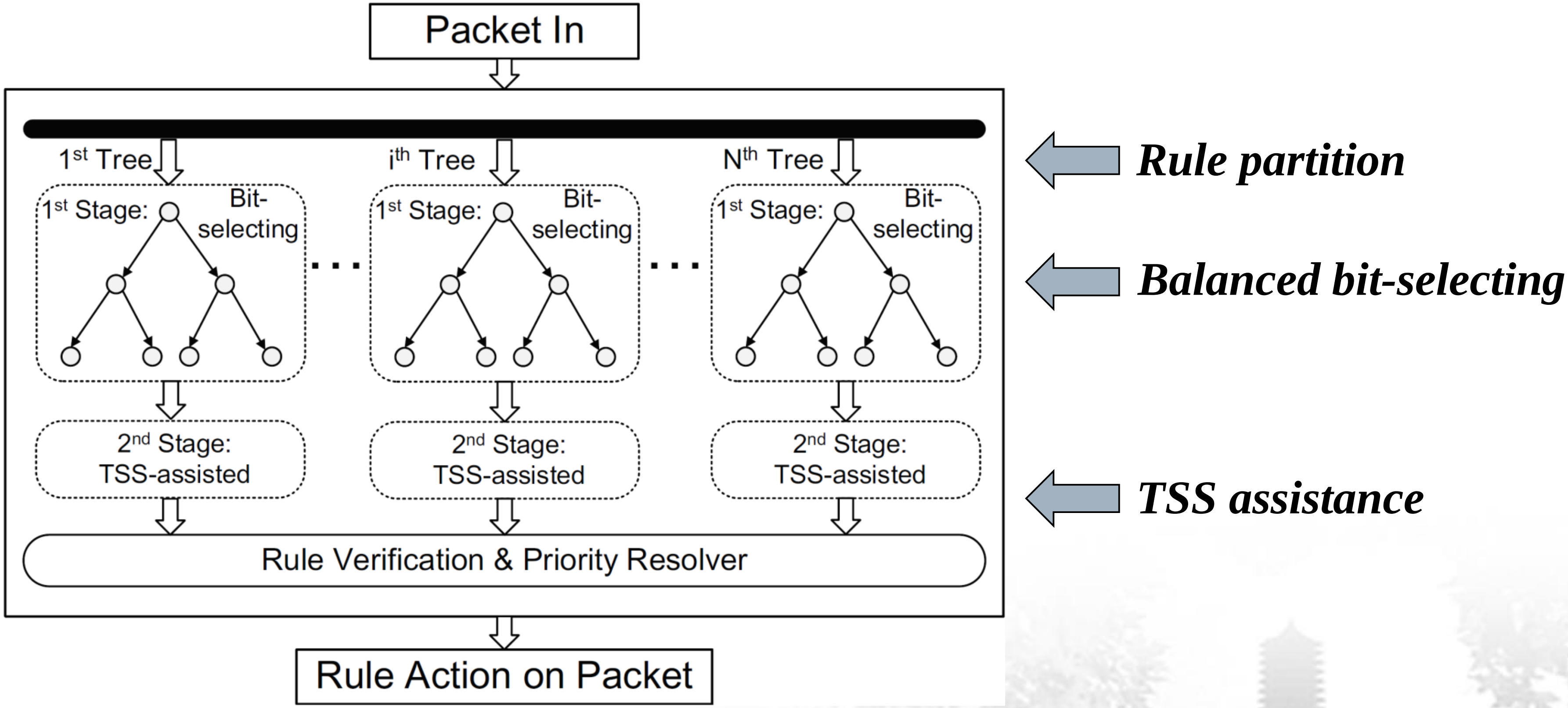
TSS-assisted Tree

Difficulties and challenges

- 1) **Low memory footprint**: to be accommodated into the small Block RAM;
- 2) **Avoid rule replication**: to support fast rule updates;
- 3) **Balanced tree**: to reduce memory accesses for high-throughput;
- 4) **Bounded tree**: to be suitable for pipeline optimizations on FPGA.

The Framework of TabTree

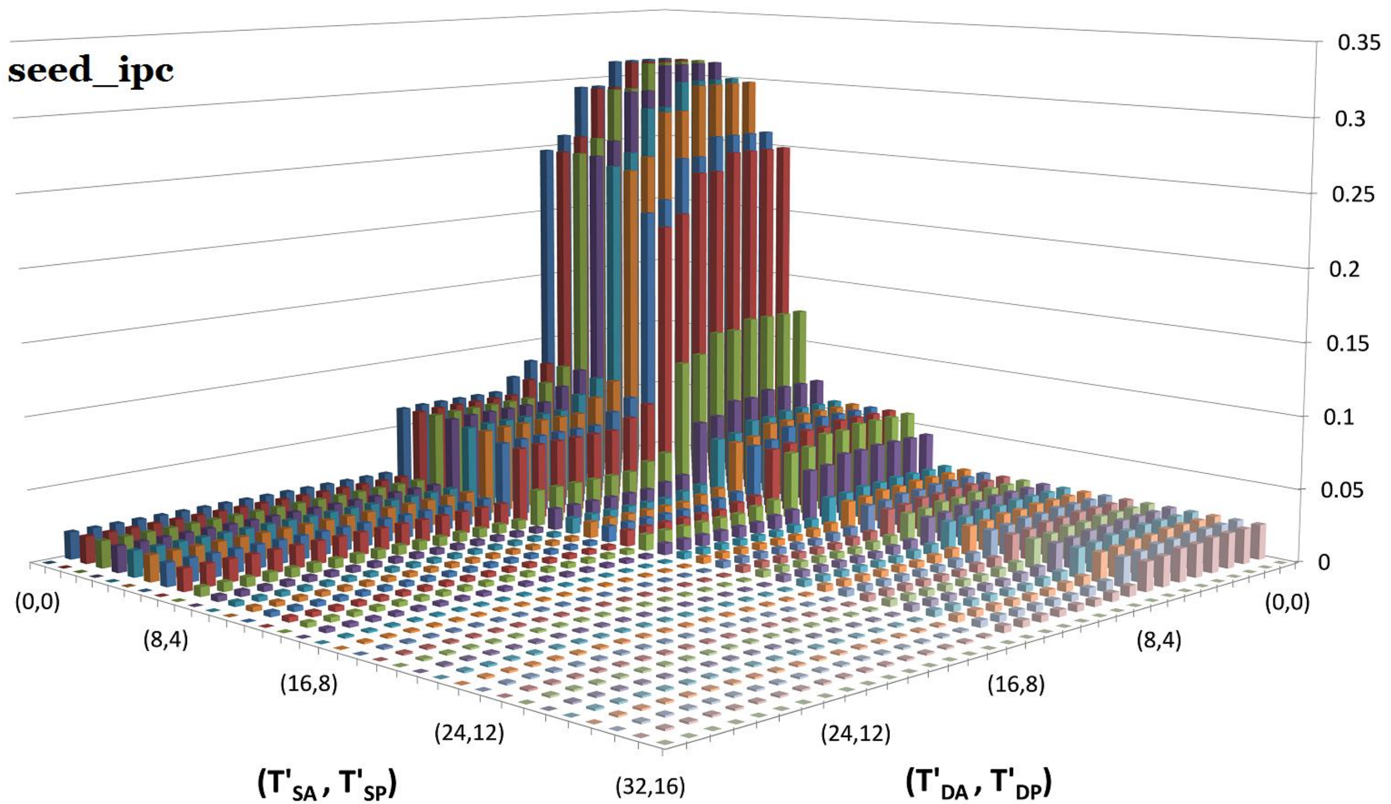
- **A two-stage framework with heterogeneous algorithms**



- **Key Steps of TabTree**
  - 1) Rule partition; 2) Balanced bit-selecting; 3) TSS assistance

Step 1: Rule Partition

- **Most rules have at least one *small field*** [HybridCuts]



- **Partition rules into subsets based on *small fields*** [CutSplit]

Observations on partitioned rules

- **There are a few *selectable bits* in *small rule fields***
  - For a W-bit wide field F_i with the threshold value of 2^K, F_i is a *small field* if and only if there are no wildcard (*) at its most significant *W-K* bits, we call these *W-K* bits as *selectable bits*.
  - For *small range fields*: False range encoding, refer to the paper

TABLE I
EXAMPLE RULE SET WITH TWO IPV4 ADDRESS FIELDS

rule id	priority	src_addr field	dst_addr field	*action*
R_1	14	228.128.0.0/9	0.0.0.0/0	*action1*
R_2	13	223.0.0.0/9	0.0.0.0/0	*action2*
R_3	12	0.0.0.0/1	175.0.0.0/8	*action3*
R_4	11	0.0.0.0/1	225.0.0.0/8	*action4*
R_5	10	0.0.0.0/2	225.0.0.0/8	*action5*
R_6	9	128.0.0.0/1	123.0.0.0/8	*action6*
R_7	8	128.0.0.0/1	37.0.0.0/8	*action7*
R_8	7	0.0.0.0/0	123.0.0.0/8	*action8*
R_9	6	178.0.0.0/7	0.0.0.0/1	*action9*
R_{10}	5	0.0.0.0/1	172.0.0.0/7	*action10*
R_{11}	4	0.0.0.0/1	226.0.0.0/7	*action11*
R_{12}	3	128.0.0.0/1	120.0.0.0/7	*action12*
R_{13}	2	128.0.0.0/2	120.0.0.0/7	*action13*
R_{14}	1	128.0.0.0/1	38.0.0.0/7	*action14*

Partition →

TABLE II
PARTITIONED RULES WITH SMALL DST_ADDR FIELD

rule	src_addr ($T_{src\_addr}= 2^{25}$)	dst_addr ($T_{dst\_addr}= 2^{25}$)	
id	1-32th bits	**33-39th**	40-64th bits
R_3	0*******************************	**1010111**	1************************
R_4	0*******************************	**1110000**	1************************
R_5	00******************************	**1110000**	1************************
R_6	1*******************************	**0111101**	1************************
R_7	1*******************************	**0010010**	1************************
R_8	********************************	**0111101**	1************************
R_{10}	0*******************************	**1010110**	*************************
R_{11}	0*******************************	**1110001**	*************************
R_{12}	1*******************************	**0111100**	*************************
R_{13}	10******************************	**0111100**	*************************
R_{14}	1*******************************	**0010011**	*************************

- **Each *selectable bit* can map rules into at most two rule subsets without any rule replications**

Step 2: Balanced Bit-selecting

The key is how to select the most distinguishing *selectable bits* in each tree node, so that rules can be mapped into its children nodes in the most balanced fashion.

- **Brute force strategy: optimal but slow**
  - Find at most b bits at one-time from *selectable bits*, which partition rules into $n = 2^b$ subsets in the most balanced fashion

$$costFunc(b\ bits) = \sqrt{\frac{\sum_{i=1}^{n}(x_i - x)^2}{n}}, where\ x = \frac{M}{n} \quad (1)$$

- **Greedy strategy: good and fast**
  - A local optimal solution, where the "good" bits are selected one by one recursively

$$imbalance(bit\ v) = |\#ruleLChild - \#ruleRChild| \quad (2)$$

Step 3: TSS Assistance

- **Stop bit-selecting progress in one of the following cases**
  - tree depth achieves the predefined maximum value
  - number of rules in the tree node is less than *binth*
  - remaining unselected rule bits share same values and cannot separate rules from each other
  - further bit-selecting will led to rule replications due to wildcards
- **Resort to other more effective methods for the following tree constructions**
  - After balanced pre-mappings, the number of rules in the terminal nodes (i.e., leaf nodes) has been significantly reduced
  - To exploit this favorable property, we use linear search (#rules ≤ *binth*) or PSTSS (#rules > *binth*) to facilitate tree constructions.

A Working Example

- An example rule set with two IPv4 address fields

Rule id	src_addr field	dst_addr field		*Rule id*	src_addr field	dst_addr field
R_1	228.128.0.0/9	0.0.0.0/0		R_8	0.0.0.0/0	123.0.0.0/8
R_2	223.0.0.0/9	0.0.0.0/0		R_9	178.0.0.0/7	0.0.0.0/1
R_3	0.0.0.0/1	175.0.0.0/8		R_{10}	0.0.0.0/1	172.0.0.0/7
R_4	0.0.0.0/1	225.0.0.0/8		R_{11}	0.0.0.0/1	226.0.0.0/7
R_5	0.0.0.0/2	225.0.0.0/8		R_{12}	128.0.0.0/1	120.0.0.0/7
R_6	128.0.0.0/1	123.0.0.0/8		R_{13}	128.0.0.0/2	120.0.0.0/7
R_7	128.0.0.0/1	37.0.0.0/8		R_{14}	128.0.0.0/1	38.0.0.0/7

A Working Example

- Two partitioned subsets, where threshold T = (2^{25}, 2^{25})
  - The 1st subset with small destination address field

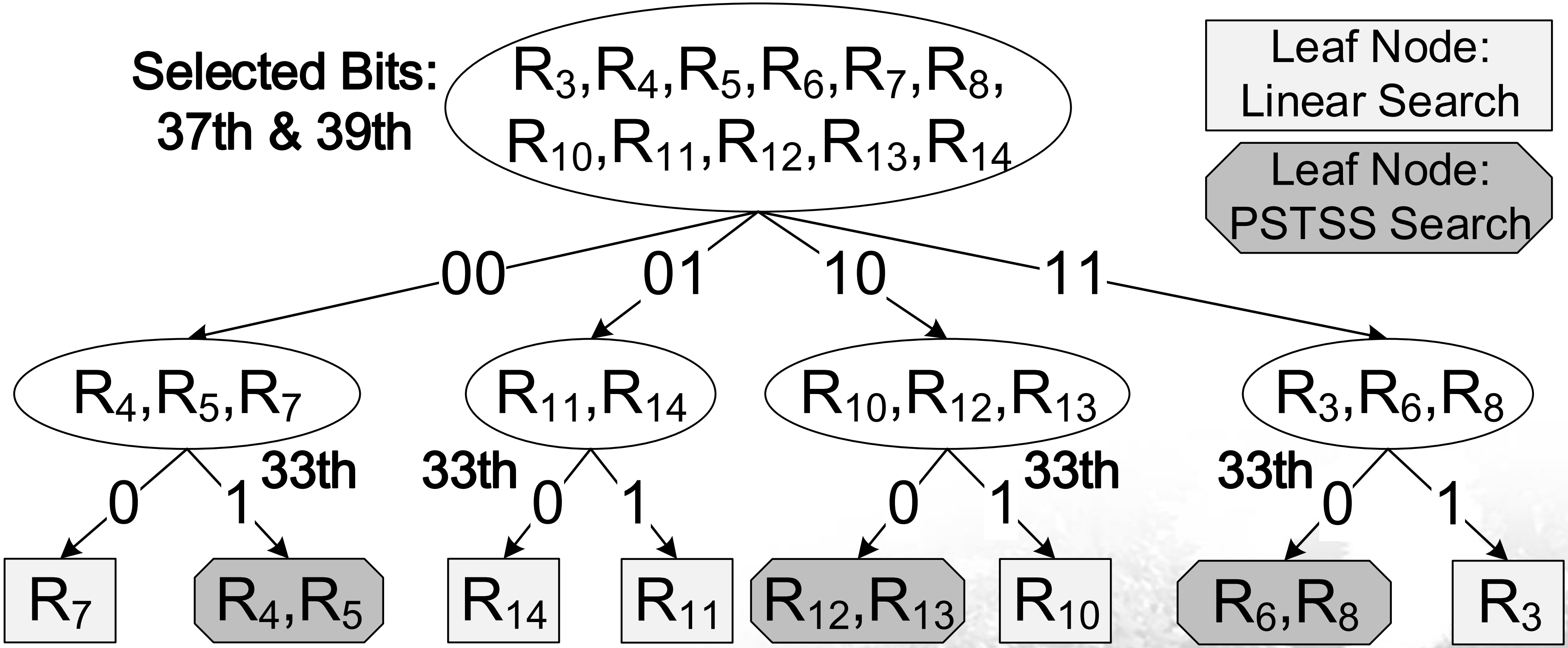
Rule id	src_addr field (1-32th rule bits)	dst_addr field (33-64th rule bits)
R_3	0*******************************	**1010111** 1************************
R_4	0*******************************	**1110000** 1************************
R_5	00******************************	**1110000** 1************************
R_6	1*******************************	**0111101** 1************************
R_7	1*******************************	**0010010** 1************************
R_8	********************************	**0111101** 1************************
R_{10}	0*******************************	**1010110** *************************
R_{11}	0*******************************	**1110001** *************************
R_{12}	1*******************************	**0111100** *************************
R_{13}	10******************************	**0111100** *************************
R_{14}	1*******************************	**0010011** *************************

  - The 2nd subset with small source address field

Rule id	src_addr field (1-32th rule bits)	dst_addr field (33-64th rule bits)
R_1	**1110010** 01***********************	********************************
R_2	**1101111** 10***********************	********************************
R_9	**1011001** *************************	0*******************************

A Working Example

- TSS-assisted decision tree for 11 rules in the 1st subset



Part 4： Evaluation

4

Preliminary Evaluation

Experiment Conclusion

Experimental Setup

- **Tested with**
  - ClassBench
    - Generate ACL & FW & IPC 1k, 10k, 100K
    - Generate 12 rule sets based on 12 seed files
- **Compared with**
  - PSTSS: used in Open vSwitch for flow table lookups
  - CutSplit: the latest cutting based decision tree
  - PartitionSort: the latest splitting based decision tree
- **Primary metrics**
  - Number of subsets
  - Memory footprint
  - Memory access
  - Update performance

Our implementation of TabTree will be available in http://wenjunli.com/TabTree/

#Subsets & Memory Footprint

TABLE III
AVERAGE SUBSETS & MEMORY FOOTPRINT

Algorithms	rules	#Subsets			Memory footprint (MB)		
		1k	10k	100k	1k	10k	100k
TabTree	ACL	3.8	3.8	3.4	0.03	0.25	2.34
	FW	4	4	4	0.03	0.21	2.44
	IPC	3.5	3.5	3.5	0.03	0.28	2.31
PSTSS	ACL	144.4	230.2	267	0.05	0.44	4.66
	FW	69.8	95.6	99.8	0.04	0.45	4.31
	IPC	114.5	164	185.5	0.04	0.48	4.92
PartitionSort	ACL	11	21.6	26.8	0.05	0.49	5.22
	FW	19.4	24.4	34.4	0.05	0.51	4.88
	IPC	10	11.5	12	0.05	0.54	5.57
CutSplit	ACL	3.8	3.8	3.4	0.04	1.28	11.52
	FW	4	4	4	0.04	4.17	18.29
	IPC	3.5	3.5	3.5	0.04	3.29	26.86

Even for rule sets up to 100k entries, TabTree can still construct decision trees in a few MBytes, small enough to be accommodated into the Block RAM of middle-end FPGAs, such as Xilinx Virtex-7 FPGAs.

Memory Access

- For simplicity, we think traversing a decision tree node, a rule or a tuple table as one memory access

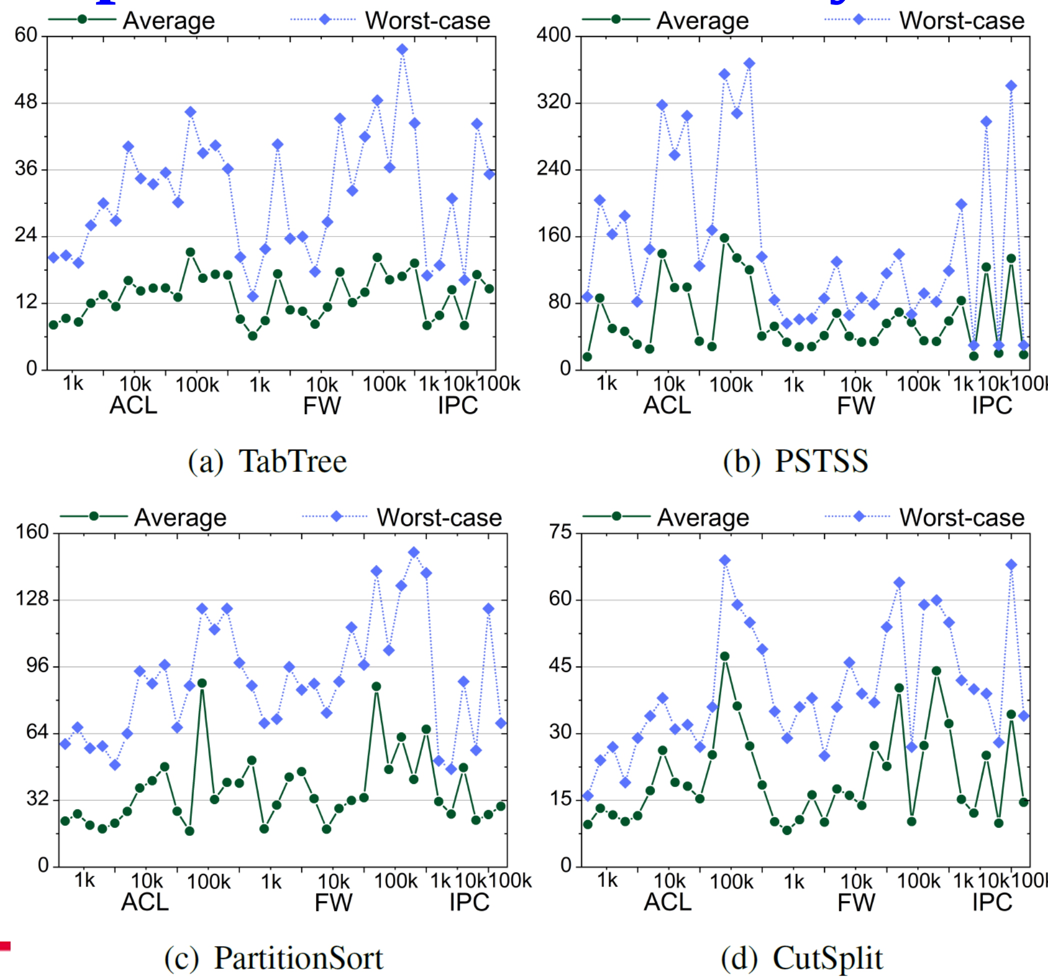


Fig. 5. Numbers of memory accesses.

Update Performance

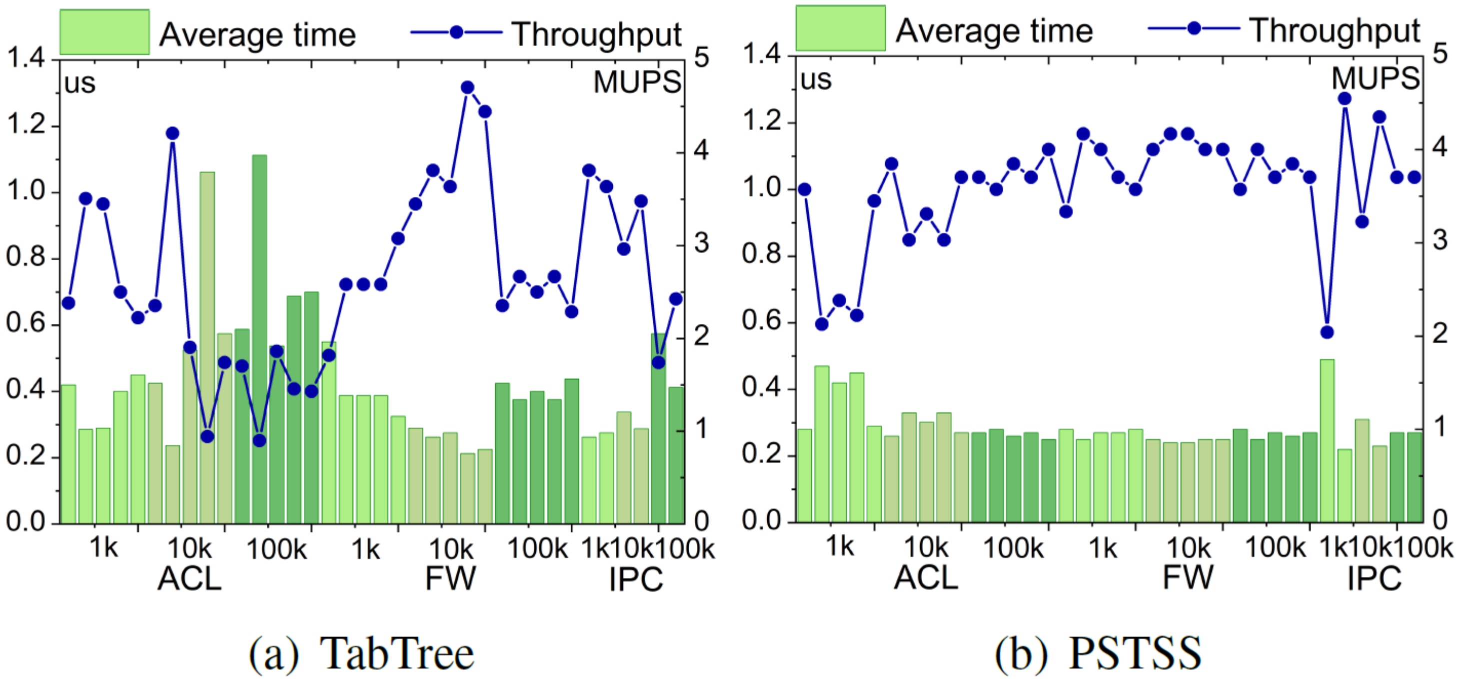


Fig. 6. Update performance.

Preliminary experimental evaluations show that, a very limited number of shallow trees can be generated with linear memory consumption in TabTree, which is also suitable for fast rule updates. More evaluations on FPGA will be given in our future work.

Part 5： Conclusion

5

Conclusion

Future Work

Conclusion

- **TabTree (TSS-assisted bit-selecting Tree)**
  - A framework consisting of heterogeneous algorithms
  - Novel observations on *small fields*
  - Two heuristic balanced bit-selecting
  - TSS to assist decision tree constructions

- **Future Work**
  - Self-adaptive rule partition instead of based on *small fields*
  - Self-adaptive bit-selecting instead of heuristic algorithms
  - Design rule caching algorithm for TabTree
  - Implement TabTree on FPGA

Thank you！

Web: http://www.wenjunli.com

E-mail: wenjunli@pku.edu.cn

BTW: I am now seeking a postdoctoral position after 2020.
Feel free to contact with me if you have a suitable position.