HybridTSS: A Recursive Scheme Combining Coarse- and Fine-Grained Tuples for Packet Classification

Yuxi Liu[1,2], Yao Xin[2], Wenjun Li[3,2], Haoyu Song[4], Ori Rottenstreich[5], Gaogang Xie[6,7], Weichao Li[2] and Yi Wang[1,2]

[1]Southern University of Science and Technology, China, [2]Peng Cheng Laboratory, China, [3]Harvard University, USA, [4]Futurewei, USA, [5]Technion, Israel, [6]CNIC, CAS, China, [7]UCAS, China

Outline

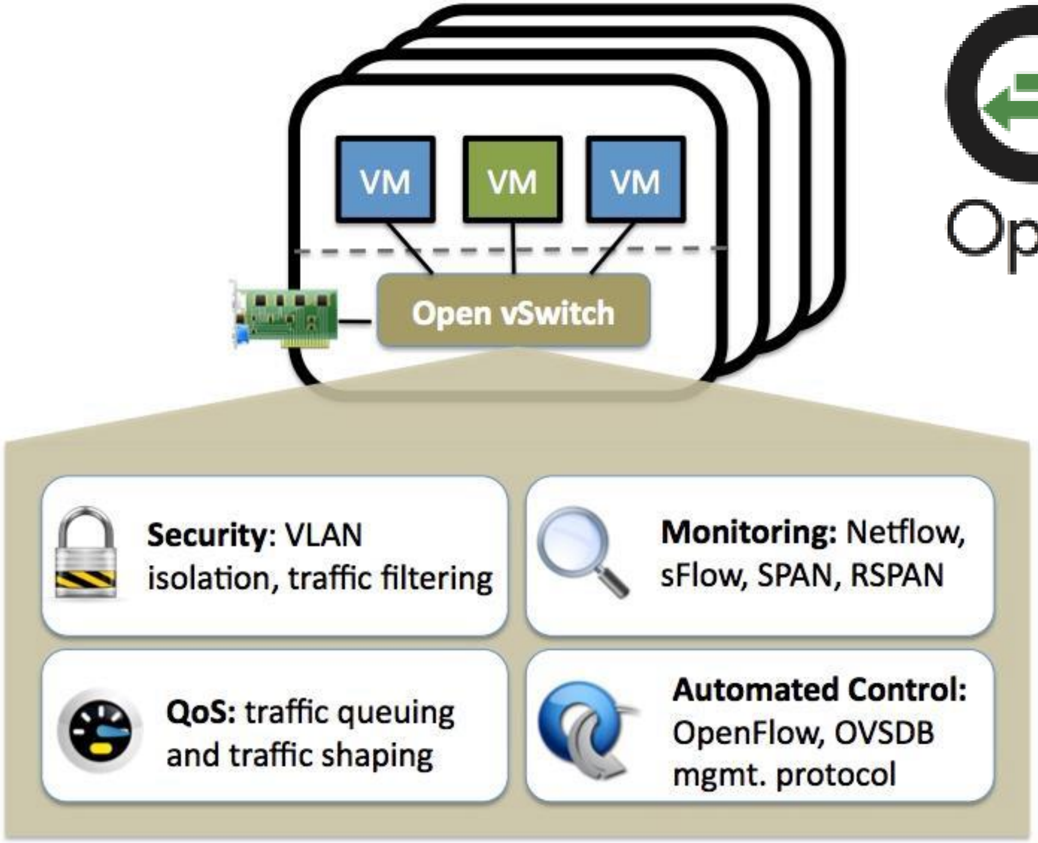
- Background & Motivation
- Proposed HybridTSS
- Experimental Evaluation
- Conclusion and Future Work

Outline

- ❖ Background & Motivation
- ❖ Proposed HybridTSS
- ❖ Experimental Evaluation
- ❖ Conclusion and Future Work

Review on Open vSwitch (OVS)

➢ **Two paths in OVS:** Slow path with OpenFlow tables + Fast path with cache tables



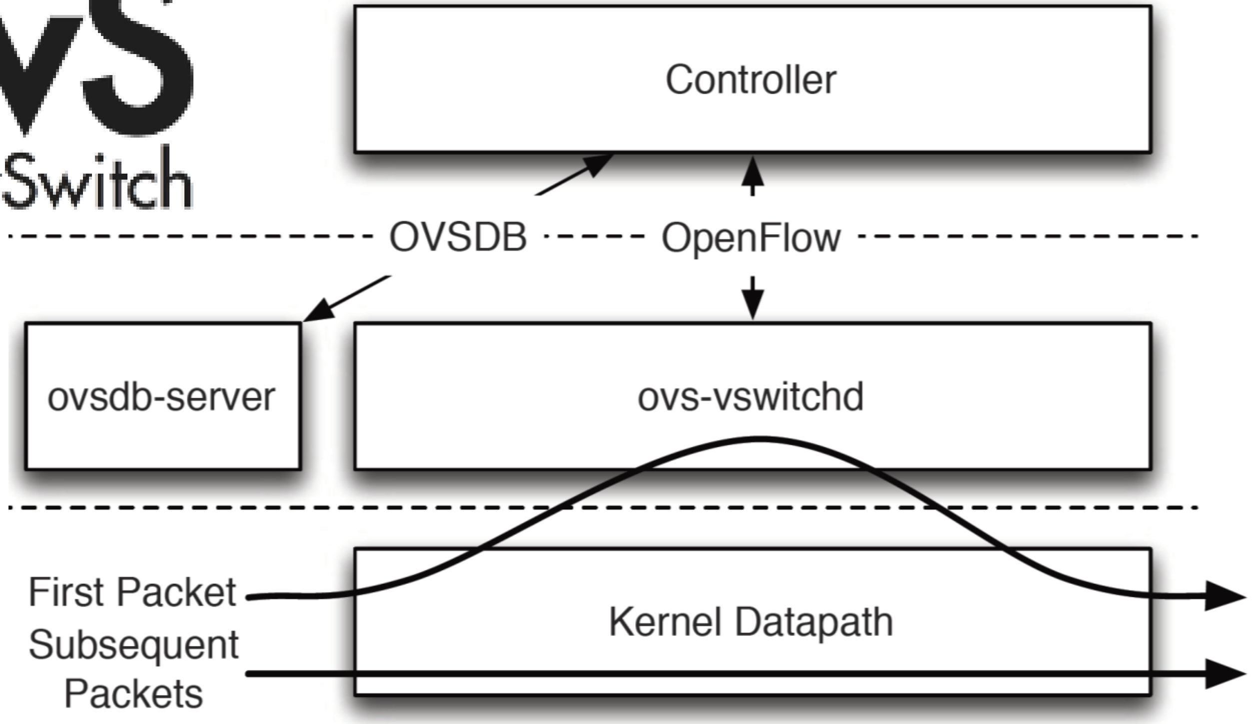


Figure 1. Open vSwitch architecture [1]

[1] Ben Pfaff and et al. The design and implementation of Open vSwitch. In USENIX NSDI 2015.

Packet Classification in Open vSwitch

➢ **Key for OpenFlow rule table lookup and MegaFlow cache table lookup**

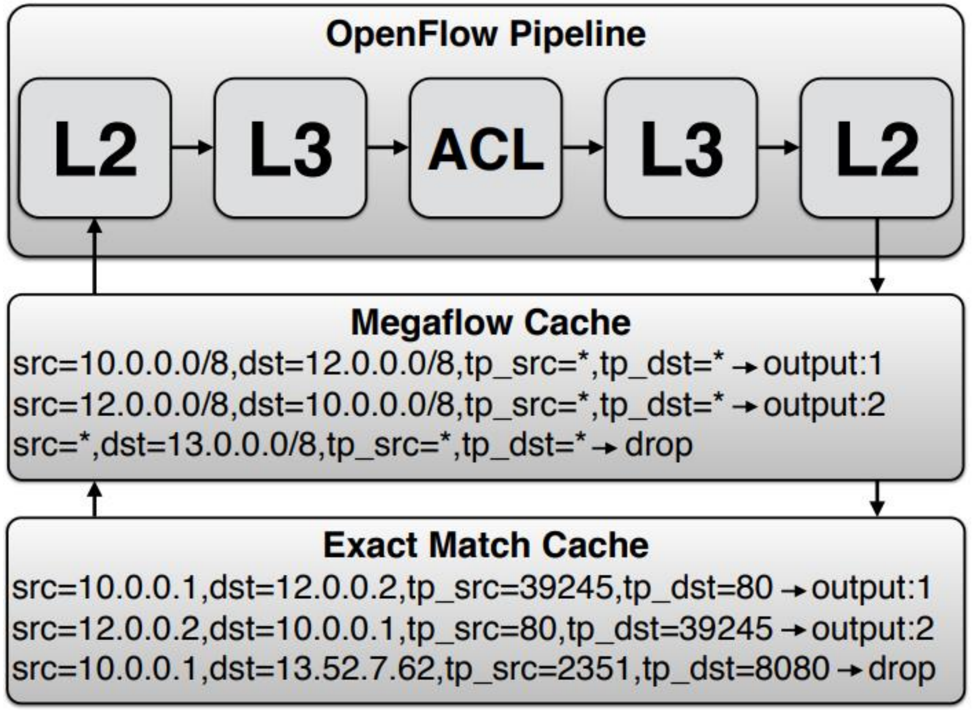


Figure 2. Cache hierarchy in OVS [2]

Slow path with OpenFlow **rule tables**

Fast path with Cache tables (EMC+**MegaFlow**)

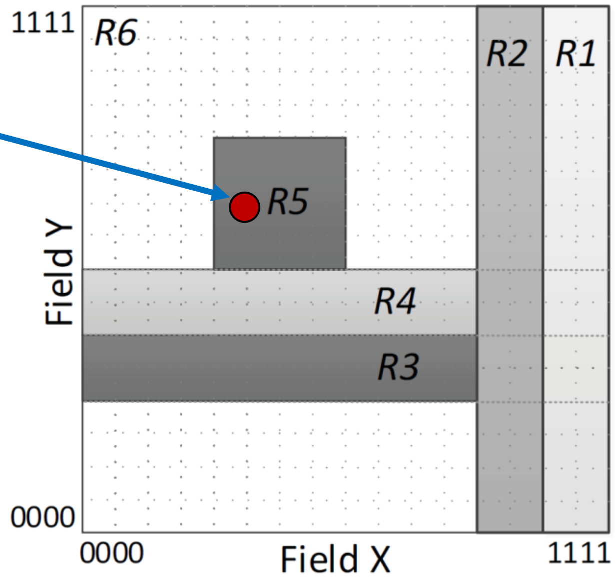
[2] Nick Shelly and et al. Flow Caching for High Entropy Packet Fields. In ACM HotSDN 2014.

Review on the Packet Classification Problem

➢ Algorithmic table lookup ←→ Geometric point location (~NP hard)

Rules	Field X	Field Y	*Action*
R1	111*	*	*action1*
R2	110*	*	*action2*
R3	*	010*	*action3*
R4	*	011*	*action4*
R5	**01****	**10****	***action5***
R6	*	*	*action6*

e.g., Packet P_i
<0101,1010>

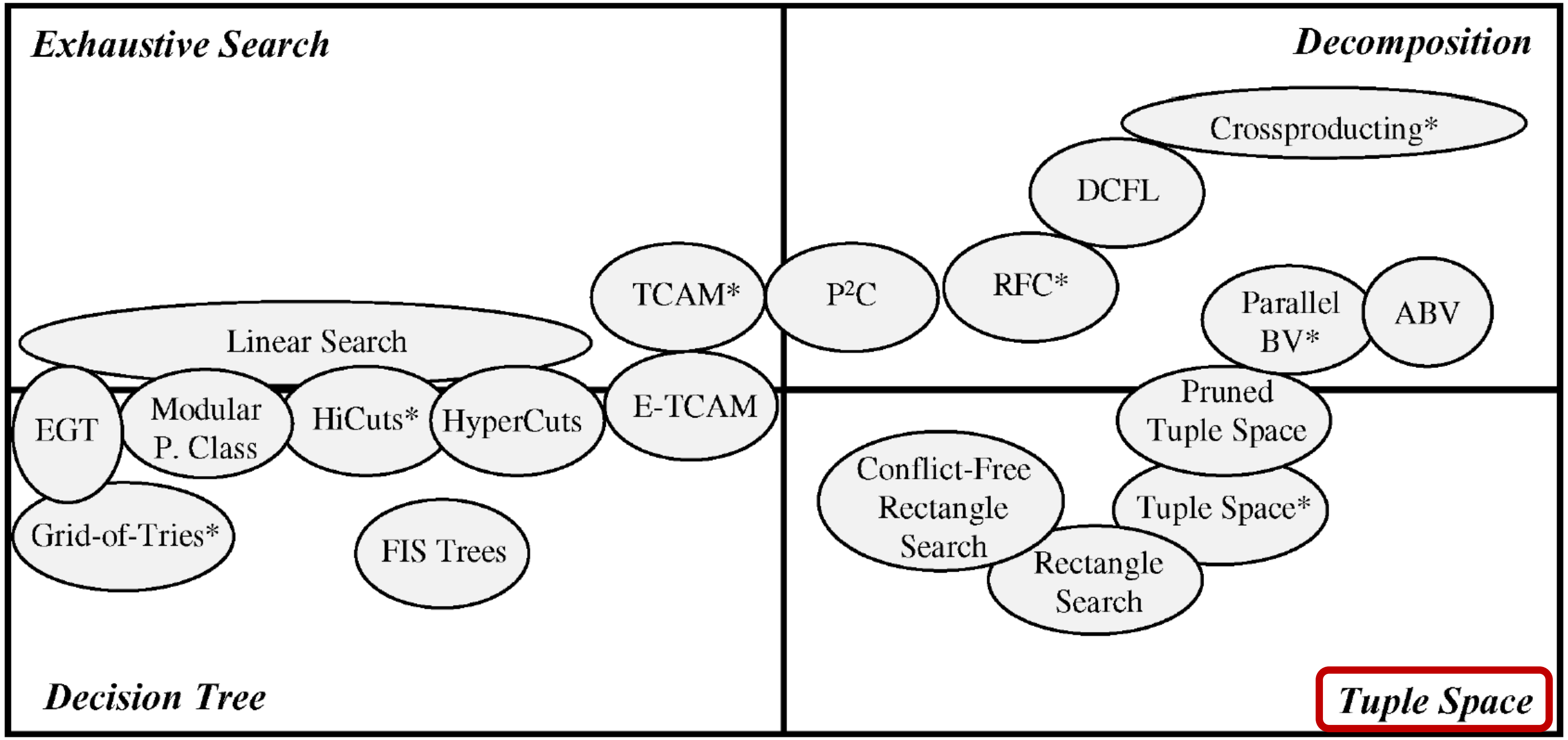


➢ Metrics for multi-field packet classification

- Time: Throughput, Memory access, Construction time
- Space: Memory consumption
- Others: Updatable, More fields, Larger classifier, Power consumption, etc.

Review on Existing Solutions

- **Well-known taxonomy from David E. Taylor[3]**



TSS can support fast rule updates

[3] D.E.Taylor, "Survey and Taxonomy of Packet Classification Techniques," ACM Computing Surveys, 37(3):238-275, 2005.

- **Packet classification in OVS: A variant of Tuple Space Search(TSS)**

Review on TSS and State-of-the-art

- **Tuple Space Search(TSS)[4]**
  - Construct tuple based on prefix
  - Use Cuckoo Hash to lookup rules
- **TupleMerge(TM)[5]**
  - Construct coarse-grained tuple
  - Use Cuckoo Hash to lookup rules
- **Comparison**
  - TM effectively reduces the number of tuples
  - TM has more hash collisions within each tuple
  - Update may cause split tuple in TM
  - Update need O(n) to locate the tuple
- **Common weakness**
  - Too many tuples accessed in one query

Rule	Field A	Field B	TSS	TupleMerge
R_1	000	111	(3, 3)	(3, 3)
R_2	011	10*	(3, 2)	(2, 2)
R_3	01*	101	(2, 3)	
R_4	01*	11*	(2, 2)	
R_5	1**	10*	(1, 2)	(1, 0)
R_6	110	***	(3, 0)	
R_7	1**	***	(1, 0)	
R_8	***	***	(0, 0)	(0, 0)

[4] Venkatachary Srinivasan and et al. Packet Classification using Tuple Space Search. In ACM SIGCOMM 1999.
[5] James Daly and et al. TupleMerge: Fast software packet processing for online packet classification. IEEE/ACM Transactions on Networking 27, 4 (2019), 1417–1431.

Motivation

1. Fewer tuples, Higher throughput!

- Q1. How to reduce the number of tuples?
- Q2. How to reduce the hash collisions?

2. Global consideration, top-down structure

Reinforcement Learning(RL) do well in this puzzle

3. Recursive TSS Construction

From Coarse-Grained tuples to Fine-Grained tuples

Outline

- ❖ Background & Motivation
- ❖ Proposed HybridTSS
- ❖ Experimental Evaluation
- ❖ Conclusion and Future Work

Framework of HybridTSS

- **Key Idea:** HybridTSS avoids tuple explosion in original TSS by recursively partitioning rules into multi-layer tuples from top to bottom, **aided by reinforcement learning**(RL)

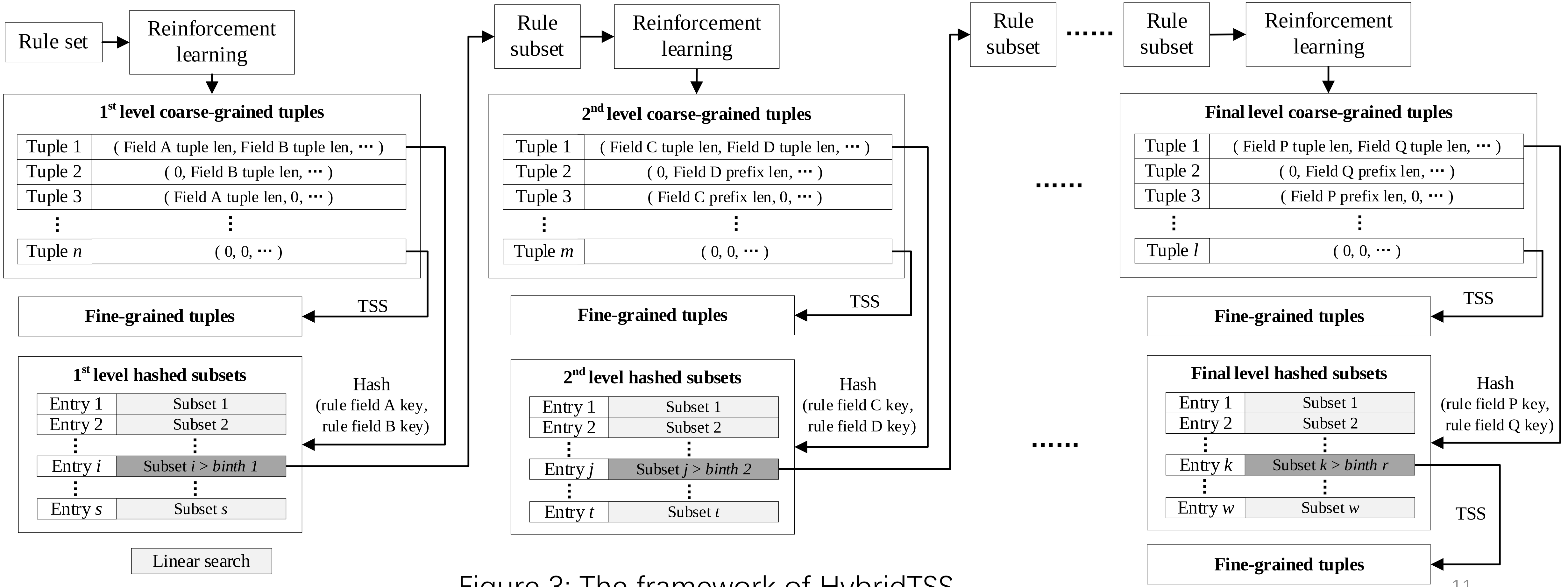


Figure 3: The framework of HybridTSS

Before RL Module…

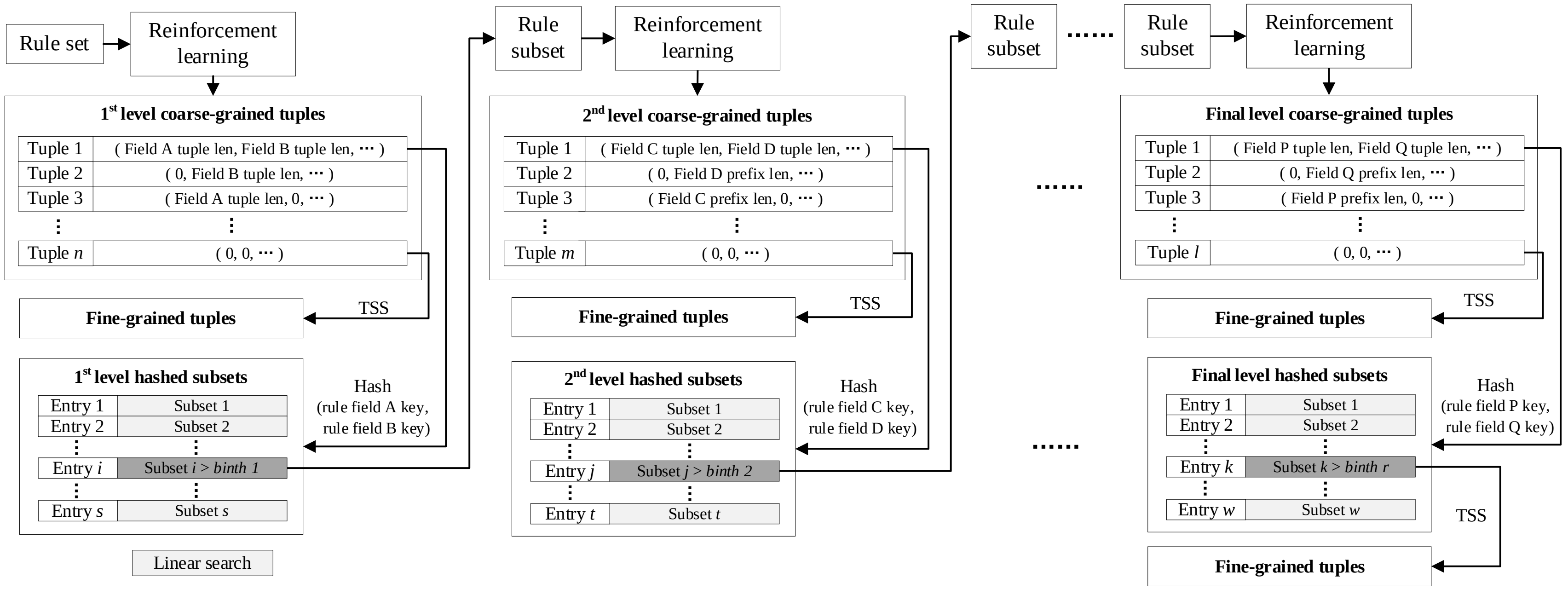
➢ Definition of <u>terminal node</u> and <u>non-terminal node</u> in the framework

✓ Non-Terminal Node

- $\#rules > binth$
- Do next action/construct Tuple Space
- Consume more memory
- Exist better solution

✓ Terminal Node

- $\#rules \leq binth$
- Linear Search is better
- Almost no optimization



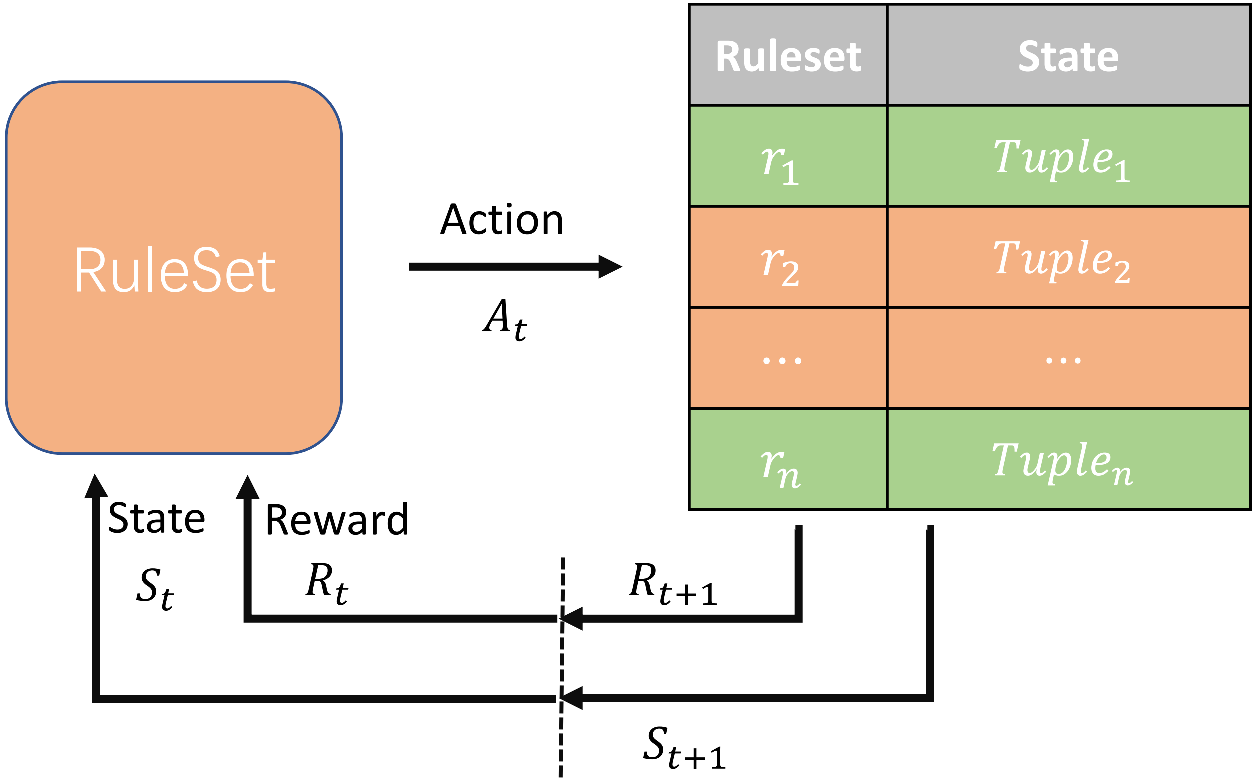
RL Target: More Terminal Leaf Node, Less Non-Terminal Leaf Node!

RL Module in HybridTSS

➢ Adapting RL for generating Tuple Space

- Defining Observation & Action Space
- Defining the appropriate Reward
- Reduce the hash collisions in coarse-grained tuples

Non-Terminal Node
Terminal Node



RL Challenges and Solutions

➢ Challenge 1: Defining Observation & Action Space

Observation Space

- Use Tuple Space to represent State
- Dynamic Programming

Each rule belongs to a unique Tuple Space.
Each tuple Space corresponds to a unique ruleset.

Action Space

- Select Fixed dimension in Each level
- Pruning

Different levels with different dimensions.

Ruleset	State
r_0	$s_0(0,0,0,0)$

Level 1 action a_1

Src_IP, Dst_IP

Ruleset	State
r_1	$s_1(x_1, y_1, 0,0)$
r_2	$s_2(x_2, 0,0,0)$
r_3	$s_3(0, y_3, 0,0)$
r_4	$s_4(0,0,0,0)$

Level 2 action a_2

Src_Port, Dst_Port

…… …

RL Challenges and Solutions

➢ Challenge 2: Determine the reward

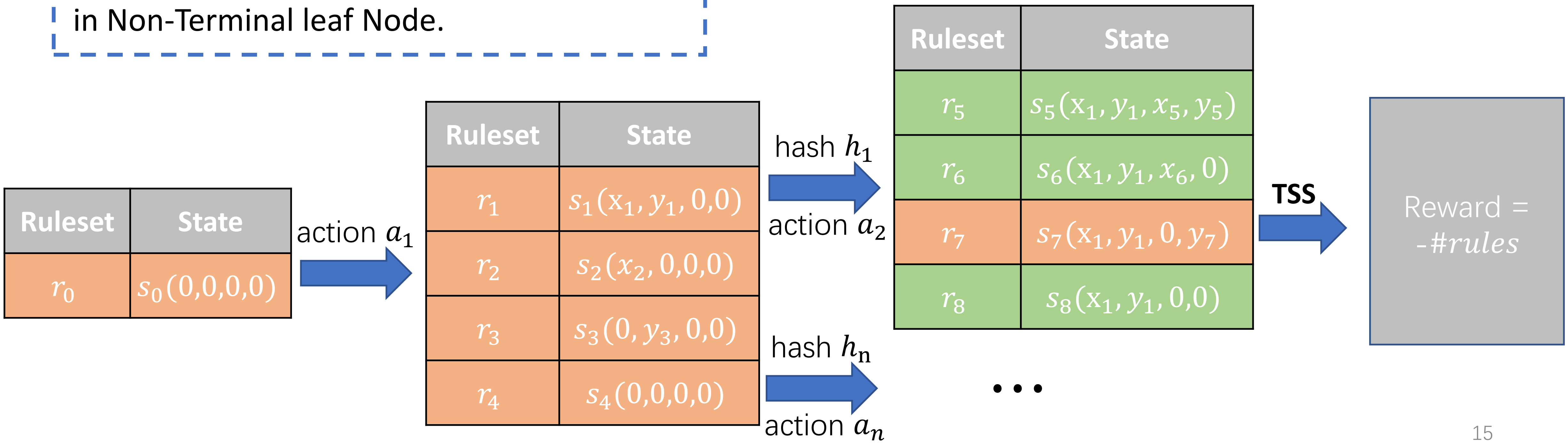
- Non-Terminal leaf Node may cause multiple hashes
- Using Bellman expectation equation to update Q-Table

$$Q^{\pi} = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | s_t, a_t]$$

Non-Terminal Node

Terminal Node

Target: Minimize the total number of rules in Non-Terminal leaf Node.

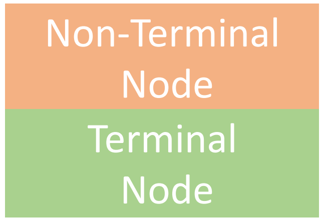


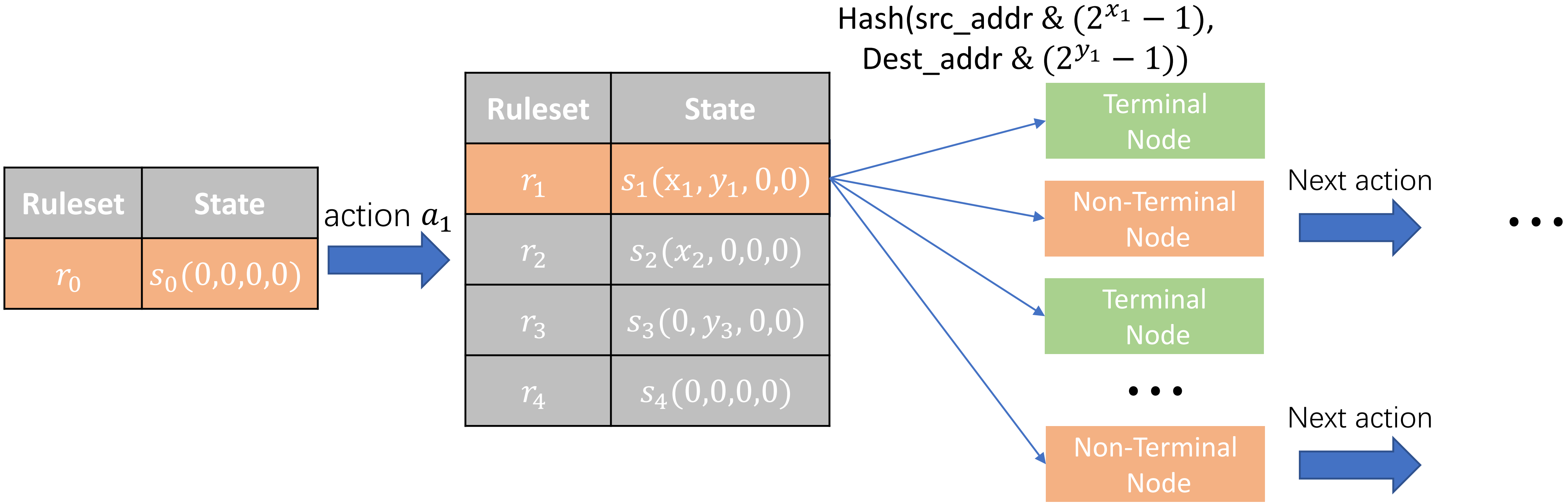
RL Challenges and Solutions

➢ Challenge 3: Reduce the Hash Collisions

Recursive TSS Construction

- Make full use of information after each action
- Hash to separate rules into subset





A Working Example of HybridTSS

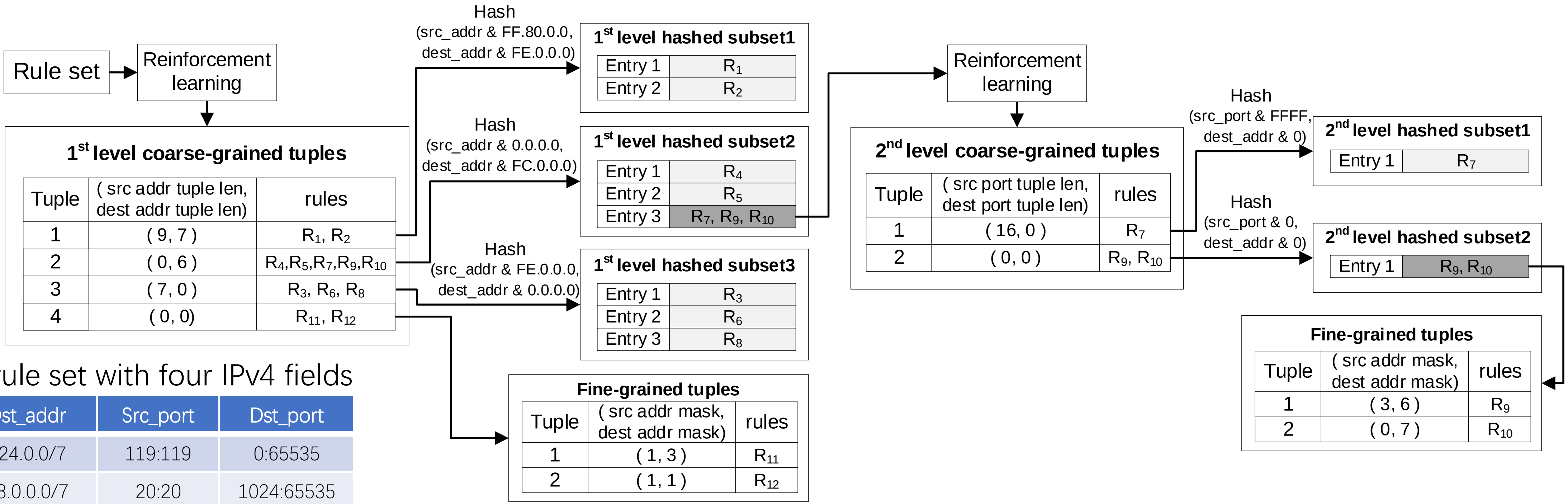


Table 1. Example rule set with four IPv4 fields

ID	Src_addr	Dst_addr	Src_port	Dst_port
R_1	228.128.0.0/9	124.0.0/7	119:119	0:65535
R_2	223.0.0.0/9	38.0.0.0/7	20:20	1024:65535
R_3	175.0.0.0/8	0.0.0.0/1	53:53	0:65535
R_4	128.0.0.0/1	37.0.0.0/8	53:53	1024:65535
R_5	0.0.0.0/2	225.0.0.0/9	123:123	0:65535
R_6	123.0.0.0/8	128.0.0.0/1	0:65535	0:65535
R_7	0.0.0.0/1	255.0.0.0/8	25:25	0:65535
R_8	246.0.0.0/7	0.0.0.0/0	0:65535	53:53
R_9	160.0.0.0/3	252.0.0.0/6	0:65535	0:65535
R_{10}	0.0.0.0/0	254.0.0.0/7	0:65535	0:65535
R_{11}	0.0.0.0/1	224.0.0.0/3	0:65535	23:23
R_{12}	128.0.0.0/1	128.0.0.0/1	0:65535	0:65535

Figure 4: A working example of HybridTSS, with the binths = 1 and the MAX recursion level = 2

PS: Range port fields are simply transformed to Longest Common Prefixes (LCP) [6] for RL in this example

[6] Yeim-Kuan Chang. 2006. A 2-level TCAM architecture for ranges. IEEE Transactions on Computers. 55, 12 (2006), 1614–1629.

Outline

- ❖ Background & Motivation
- ❖ Proposed HybridTSS
- ❖ **Experimental Evaluation**
- ❖ Conclusion and Future Work

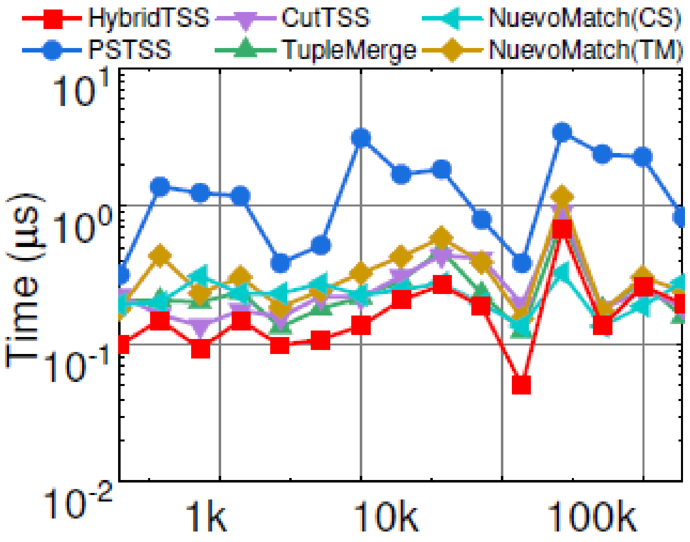
Experimental Evaluation

- **Rule Sets**
  - ClassBench[7]: Generate ACL & FW & IPC based on 12 seed files, with 1K & 10K & 100K
- **Compared with**
  - Classification performance: PSTSS[1], TupleMerge[5], CutTSS[8], NuevoMatch[9]
  - Update performance: PSTSS, TupleMerge, CutTSS
- **The source code of this paper can be downloaded from**
  - http://www.wenjunli.com/HybridTSS
  - https://www.github.com/wenjunpaper/HybridTSS

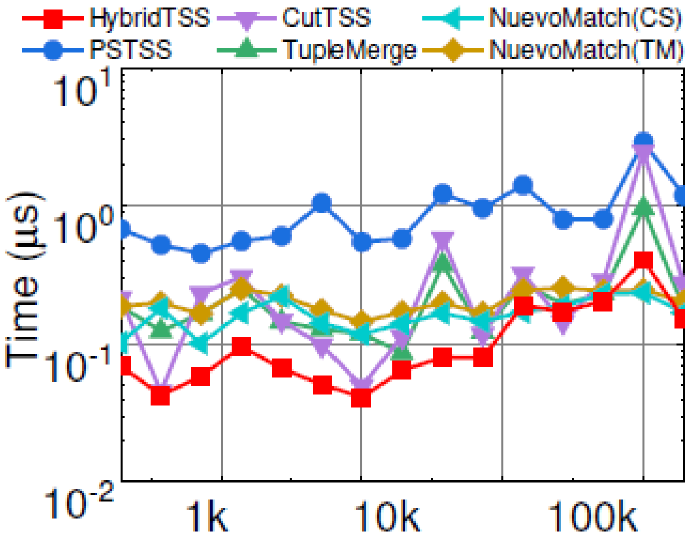
[7] David E Taylor and Jonathan S Turner. 2007. ClassBench: A packet classification benchmark. IEEE/ACM Transactions on Networking 15, 3 (2007), 499–511.
[8] Wenjun Li and et al. 2020. Tuple Space Assisted Packet Classification with High Performance on Both Search and Update. IEEE Journal on Selected Areas in Communications 38, 7 (2020), 1555–1569.
[9] Alon Rashelbach, Ori Rottenstreich, and Mark Silberstein. A Computational Approach to Packet Classification. In ACM SIGCOMM, 2020.

Classification Performance

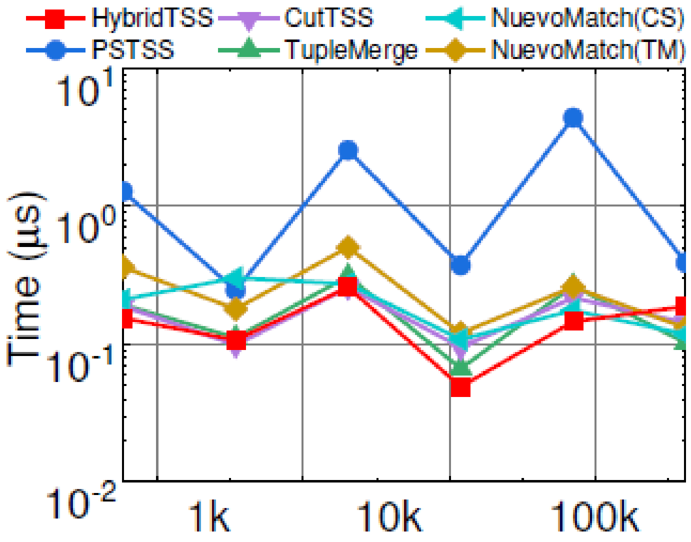
- **Average classification time of one packet on three types of rule sets with different sizes**
  - Achieve $7.76\times, 10.09\times, 8.03\times$ speed up in terms of classification time compare to PSTSS
  - Achieve $1.92\times, 1.54\times,\ 1.82\times, 1.81\times$ speed up compare to CutTSS, TupleMerge, NuevoMatch(TM), NuevoMatch(CS)



(a) ACL



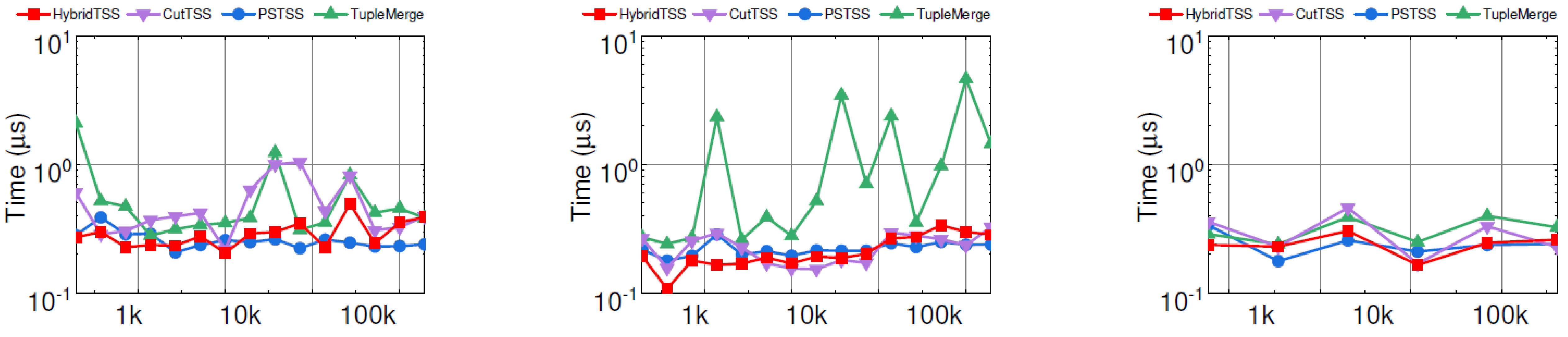
(b) FW



(c) IPC

Update Performance

- **Average update time of one rule on three types of rule sets with different sizes**
  - Achieve $0.96\times, 1.45\times, 1.44\times$ speed up compare to PSTSS, CutTSS, TupleMerge



(a) ACL (b) FW (c) IPC

Outline

- ❖ Background & Motivation
- ❖ Proposed HybridTSS
- ❖ Experimental Evaluation
- ❖ Conclusion and Future Work

Conclusion & Future Work

- **Summary HybridTSS**
  - Adopt RL method to build a small number of coarse-grained tuples
  - From coarse-grained tuple hashed into subset
  - Achieve higher throughput and fast updates

- **Future Work**
  - Adopt new ML/RL approaches for globally balanced tuple partitioning
  - Combine packet classification with flow cache
  - Integrated to OVS and offload to FPGA

Thanks

Q&A