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

Packet Classification in Open vSwitch

- Key for OpenFlow rule table lookup and MegaFlow cache table lookup

![Diagram of OpenFlow Pipeline and MegaFlow Cache]

Figure 2. Cache hierarchy in OVS [2]

Review on the Packet Classification Problem

Algorithmic table lookup $\leftrightarrow$ Geometric point location ($\sim$NP hard)

<table>
<thead>
<tr>
<th>Rules</th>
<th>Field X</th>
<th>Field Y</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>111*</td>
<td>*</td>
<td>action1</td>
</tr>
<tr>
<td>R2</td>
<td>110*</td>
<td>*</td>
<td>action2</td>
</tr>
<tr>
<td>R3</td>
<td>*</td>
<td>010*</td>
<td>action3</td>
</tr>
<tr>
<td>R4</td>
<td>*</td>
<td>011*</td>
<td>action4</td>
</tr>
<tr>
<td>R5</td>
<td>01**</td>
<td>10**</td>
<td>action5</td>
</tr>
<tr>
<td>R6</td>
<td>*</td>
<td>*</td>
<td>action6</td>
</tr>
</tbody>
</table>

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]**

  ![Decision Tree Diagram]

  Exhaustive Search:
  - Linear Search
  - TCAM*
  - P^2C
  - RFC*
  - E-TCAM
  - Modular P. Class
  - HiCuts*
  - HyperCuts
  - Grid-of-Tries*
  - FIS Trees

  Decomposition:
  - DCFL
  - Parallel BV*
  - ABV
  - Crossproducng*
  - Pruned Tuple Space
  - Tuple Space*
  - Conflict-Free Rectangle Search
  - Rectangle Search

- **TSS can support fast rule updates**


- **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

<table>
<thead>
<tr>
<th>Rule</th>
<th>Field A</th>
<th>Field B</th>
<th>TSS</th>
<th>TupleMerge</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td>000</td>
<td>111</td>
<td>(3, 3)</td>
<td>(3, 3)</td>
</tr>
<tr>
<td>$R_2$</td>
<td>011</td>
<td>10*</td>
<td>(3, 2)</td>
<td></td>
</tr>
<tr>
<td>$R_3$</td>
<td>01*</td>
<td>101</td>
<td>(2, 3)</td>
<td>(2, 2)</td>
</tr>
<tr>
<td>$R_4$</td>
<td>01*</td>
<td>11*</td>
<td>(2, 2)</td>
<td></td>
</tr>
<tr>
<td>$R_5$</td>
<td>1**</td>
<td>10*</td>
<td>(1, 2)</td>
<td>(1, 0)</td>
</tr>
<tr>
<td>$R_6$</td>
<td>110</td>
<td>***</td>
<td>(3, 0)</td>
<td>(1, 0)</td>
</tr>
<tr>
<td>$R_7$</td>
<td>1**</td>
<td>***</td>
<td>(1, 0)</td>
<td></td>
</tr>
<tr>
<td>$R_8$</td>
<td>***</td>
<td>***</td>
<td>(0, 0)</td>
<td>(0, 0)</td>
</tr>
</tbody>
</table>

➢ **Common weakness**
  - Too many tuples accessed in one query

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
### Rule set

<table>
<thead>
<tr>
<th>Tuple 1</th>
<th>(Field A tuple len, Field B tuple len, ...)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuple 2</td>
<td>(0, Field B tuple len, ...)</td>
</tr>
<tr>
<td>Tuple 3</td>
<td>(Field A tuple len, 0, ...)</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Tuple n</td>
<td>(0, 0, ...)</td>
</tr>
</tbody>
</table>

#### 1st level coarse-grained tuples

- **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)**

### Reinforcement learning

#### Hash

- (rule field A key, rule field B key)

#### Rule subset

- Entry 1: Subset 1
- Entry 2: Subset 2
- ... Entry i: Subset i > binth 1
- Entry j: Subset j > binth 2
- Entry t: Subset t

- Hash: (rule field C key, rule field D key)

#### 2nd level coarse-grained tuples

- Tuple 1: (Field C tuple len, Field D tuple len, ...)
- Tuple 2: (0, Field D prefix len, ...)
- Tuple 3: (Field C prefix len, 0, ...)
- ... Tuple m: (0, 0, ...)

#### Final level coarse-grained tuples

- Tuple 1: (Field P tuple len, Field Q tuple len, ...)
- Tuple 2: (0, Field Q prefix len, ...)
- Tuple 3: (Field P prefix len, 0, ...)
- ... Tuple l: (0, 0, ...)

### Reinforcement learning

#### Final level hashed subsets

- Entry 1: Subset 1
- Entry 2: Subset 2
- ... Entry k: Subset k > binth r
- Entry w: Subset w

#### TSS

- Hash: (rule field P key, rule field Q key)

### Final level coarse-grained tuples

- Tuple 1: (Field A tuple len, Field B tuple len, ...)
Definition of terminal node and non-terminal node in the framework

✓ Non-Terminal Node
  • \#rules > binth
  • Do next action/construct Tuple Space
  • Consume more memory
  • Exist better solution

✓ Terminal Node
  • \#rules ≤ binth
  • Linear Search is better
  • Almost no optimization

RL Target: More Terminal Leaf Node, Less Non-Terminal Leaf Node!
Adapting RL for generating Tuple Space

- Defining Observation & Action Space
- Defining the appropriate Reward
- Reduce the hash collisions in coarse-grained tuples
# RL Challenges and Solutions

## Challenge 1: Defining Observation & Action Space

### Observation Space
- Use Tuple Space to represent State
- Dynamic Programming

### Action Space
- Select Fixed dimension in Each level
- Pruning

---

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

### Different levels with different dimensions.

<table>
<thead>
<tr>
<th>Ruleset</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_0$</td>
<td>$s_0(0,0,0,0)$</td>
</tr>
</tbody>
</table>

- **Level 1**
  - Action $a_1$

<table>
<thead>
<tr>
<th>Rule</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1$</td>
<td>$s_1(x_1, y_1, 0,0)$</td>
</tr>
<tr>
<td>$r_2$</td>
<td>$s_2(x_2, 0,0,0)$</td>
</tr>
<tr>
<td>$r_3$</td>
<td>$s_3(0, y_3, 0,0)$</td>
</tr>
<tr>
<td>$r_4$</td>
<td>$s_4(0,0,0,0)$</td>
</tr>
</tbody>
</table>

- **Level 2**
  - Action $a_2$

<table>
<thead>
<tr>
<th>Rule</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$s_5(0,0,0,0)$</td>
</tr>
</tbody>
</table>

**Src_IP, Dst_IP**

**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

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

**Target:** Minimize the total number of rules in Non-Terminal leaf Node.
Challenge 3: Reduce the Hash Collisions

Recursive TSS Construction

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

### Ruleset State

<table>
<thead>
<tr>
<th>Ruleset</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_0$</td>
<td>$s_0(0,0,0,0)$</td>
</tr>
<tr>
<td>$r_1$</td>
<td>$s_1(x_1, y_1, 0,0)$</td>
</tr>
<tr>
<td>$r_2$</td>
<td>$s_2(x_2, 0,0,0)$</td>
</tr>
<tr>
<td>$r_3$</td>
<td>$s_3(0, y_3, 0,0)$</td>
</tr>
<tr>
<td>$r_4$</td>
<td>$s_4(0,0,0,0)$</td>
</tr>
</tbody>
</table>

Hash(src_addr & $(2^{x_1} - 1)$, Dest_addr & $(2^{y_1} - 1)$)
A Working Example of HybridTSS

Table 1. Example rule set with four IPv4 fields

<table>
<thead>
<tr>
<th>ID</th>
<th>Src_addr</th>
<th>Dst_addr</th>
<th>Src_port</th>
<th>Dst_port</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>228.128.0.0/9</td>
<td>124.0.0/7</td>
<td>119:119</td>
<td>0:65535</td>
</tr>
<tr>
<td>R2</td>
<td>223.0.0.0/9</td>
<td>38.0.0/7</td>
<td>20:20</td>
<td>1024:65535</td>
</tr>
<tr>
<td>R3</td>
<td>175.0.0.0/8</td>
<td>0.0.0/1</td>
<td>53:53</td>
<td>0:65535</td>
</tr>
<tr>
<td>R4</td>
<td>128.0.0.0/1</td>
<td>37.0.0/8</td>
<td>53:53</td>
<td>0:65535</td>
</tr>
<tr>
<td>R5</td>
<td>0.0.0.0/2</td>
<td>225.0.0.0/9</td>
<td>123:123</td>
<td>0:65535</td>
</tr>
<tr>
<td>R6</td>
<td>123.0.0.0/8</td>
<td>128.0.0.0/1</td>
<td>0:65535</td>
<td>0:65535</td>
</tr>
<tr>
<td>R7</td>
<td>0.0.0.0/1</td>
<td>255.0.0.0/8</td>
<td>25:25</td>
<td>0:65535</td>
</tr>
<tr>
<td>R8</td>
<td>246.0.0.0/7</td>
<td>0.0.0/0</td>
<td>0:65535</td>
<td>53:53</td>
</tr>
<tr>
<td>R9</td>
<td>160.0.0.0/3</td>
<td>252.0.0.0/6</td>
<td>0:65535</td>
<td>0:65535</td>
</tr>
<tr>
<td>R10</td>
<td>0.0.0.0/0</td>
<td>254.0.0.0/7</td>
<td>0:65535</td>
<td>0:65535</td>
</tr>
<tr>
<td>R11</td>
<td>0.0.0.0/1</td>
<td>224.0.0.0/3</td>
<td>0:65535</td>
<td>23:23</td>
</tr>
<tr>
<td>R12</td>
<td>128.0.0.0/1</td>
<td>128.0.0.0/1</td>
<td>0:65535</td>
<td>0:65535</td>
</tr>
</tbody>
</table>

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

Outline

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

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)
➢ **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.
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