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

Packet Classification

A Little Review on Related Work
Packet Classification

Key for policy enforcement in packet forwarding

- Firewall, QoS, OpenFlow, P4, etc.

An example OpenFlow 1.0 classifier/flow table (12-tuple)

<table>
<thead>
<tr>
<th>#</th>
<th>Ingress Port</th>
<th>Ether src</th>
<th>Ether dst</th>
<th>Ether type</th>
<th>VLAN id</th>
<th>VLAN priority</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_1 )</td>
<td>3</td>
<td>*</td>
<td>*</td>
<td>2048</td>
<td>*</td>
<td>*</td>
<td>Action_1</td>
</tr>
<tr>
<td>IP src</td>
<td>IP dst</td>
<td>IP proto</td>
<td>IP ToS bits</td>
<td>TCP/UDP Src Port</td>
<td>TCP/UDP Dst Port</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15.25.70.8/30</td>
<td>18.15.125.3/28</td>
<td>0x11/0xff</td>
<td>1</td>
<td>1024 : 65535</td>
<td>80</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Existing Solutions

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

رى

- Our proposed TabTree: A hybrid approach

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

School of Electronic and Computer Engineering, Peking University
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

- **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
A Little Review on TSS

- **TSS (Tuple Space Search) for packet classification**
  - Partition rules into a set of hash tables based on prefix length

<table>
<thead>
<tr>
<th>Rule #</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>A1</td>
</tr>
<tr>
<td>R2</td>
<td>110*</td>
<td>*</td>
<td>A2</td>
</tr>
<tr>
<td>R3</td>
<td>*</td>
<td>010*</td>
<td>A3</td>
</tr>
<tr>
<td>R4</td>
<td>*</td>
<td>011*</td>
<td>A4</td>
</tr>
<tr>
<td>R5</td>
<td>01**</td>
<td>10**</td>
<td>A5</td>
</tr>
<tr>
<td>R6</td>
<td>*</td>
<td>*</td>
<td>A6</td>
</tr>
</tbody>
</table>

Tuple 1: (3, 0)  
Priority: 6  
 Tuple 2: (0, 3)  
Priority: 4  
 Tuple 3: (2, 2)  
Priority: 2  
 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

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.

HiCuts [HotI]
- HyperCuts [SIGCOMM]
- EffiCuts [SIGCOMM]
- HybridCuts [HotI]
- PartitionSort [ICNP]
- MC-SBC [ANCS]
- CutSplit [INFOCOM]
- ByteCuts [INFOCOM]
- NeuroCuts [SIGCOMM]
- ModularPC [INFOCOM]
- HyperSplit [INFOCOM]
- ParaSplit [HotI]
- SmartSplit [ICNP]
- School of Electronic and Computer Engineering, Peking University
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?

School of Electronic and Computer Engineering, Peking University
Part 3: Proposed Algorithm

TabTree

TSS-assisted bit-selecting Tree

School of Electronic and Computer Engineering, Peking University
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]

School of Electronic and Computer Engineering, Peking University
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

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

TABLE I
Example Rule Set with Two IPv4 Address Fields

<table>
<thead>
<tr>
<th>rule id</th>
<th>priority</th>
<th>src_addr field</th>
<th>dst_addr field</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>14</td>
<td>228.128.0.0/9</td>
<td>0.0.0.0/0</td>
<td>action1</td>
</tr>
<tr>
<td>R2</td>
<td>13</td>
<td>223.0.0.0/9</td>
<td>0.0.0.0/0</td>
<td>action2</td>
</tr>
<tr>
<td>R3</td>
<td>12</td>
<td>0.0.0.0/1</td>
<td>175.0.0.0/8</td>
<td>action3</td>
</tr>
<tr>
<td>R4</td>
<td>11</td>
<td>0.0.0.0/1</td>
<td>225.0.0.0/8</td>
<td>action4</td>
</tr>
<tr>
<td>R5</td>
<td>10</td>
<td>0.0.0.0/2</td>
<td>225.0.0.0/8</td>
<td>action5</td>
</tr>
<tr>
<td>R6</td>
<td>9</td>
<td>128.0.0.0/1</td>
<td>123.0.0.0/8</td>
<td>action6</td>
</tr>
<tr>
<td>R7</td>
<td>8</td>
<td>128.0.0.0/1</td>
<td>37.0.0.0/8</td>
<td>action7</td>
</tr>
<tr>
<td>R8</td>
<td>7</td>
<td>0.0.0.0/0</td>
<td>123.0.0.0/8</td>
<td>action8</td>
</tr>
<tr>
<td>R9</td>
<td>6</td>
<td>178.0.0.0/7</td>
<td>0.0.0.0/0</td>
<td>action9</td>
</tr>
<tr>
<td>R10</td>
<td>5</td>
<td>0.0.0.0/1</td>
<td>172.0.0.0/7</td>
<td>action10</td>
</tr>
<tr>
<td>R11</td>
<td>4</td>
<td>0.0.0.0/1</td>
<td>226.0.0.0/7</td>
<td>action11</td>
</tr>
<tr>
<td>R12</td>
<td>3</td>
<td>128.0.0.0/1</td>
<td>120.0.0.0/7</td>
<td>action12</td>
</tr>
<tr>
<td>R13</td>
<td>2</td>
<td>128.0.0.0/2</td>
<td>120.0.0.0/7</td>
<td>action13</td>
</tr>
<tr>
<td>R14</td>
<td>1</td>
<td>128.0.0.0/1</td>
<td>38.0.0.0/7</td>
<td>action14</td>
</tr>
</tbody>
</table>

Partition

TABLE II
Partitioned Rules with Small Dst_addr Field

<table>
<thead>
<tr>
<th>rule id</th>
<th>src_addr (T_{src_addr} = 2^{25})</th>
<th>dst_addr (T_{dst_addr} = 2^{25})</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1-32th bits</td>
<td>33-39th bits</td>
</tr>
<tr>
<td>R3</td>
<td>00000000000000000000000000000110</td>
<td>10101111000000000000000000000000</td>
</tr>
<tr>
<td>R4</td>
<td>00000000000000000000000000000110</td>
<td>11100000000000000000000000000000</td>
</tr>
<tr>
<td>R5</td>
<td>00000000000000000000000000000110</td>
<td>11100000000000000000000000000000</td>
</tr>
<tr>
<td>R6</td>
<td>00000000000000000000000000000110</td>
<td>01111010000000000000000000000000</td>
</tr>
<tr>
<td>R7</td>
<td>00000000000000000000000000000110</td>
<td>00100100000000000000000000000000</td>
</tr>
<tr>
<td>R8</td>
<td>00000000000000000000000000000110</td>
<td>01111010000000000000000000000000</td>
</tr>
<tr>
<td>R9</td>
<td>00000000000000000000000000000110</td>
<td>10101111000000000000000000000000</td>
</tr>
<tr>
<td>R10</td>
<td>00000000000000000000000000000110</td>
<td>11100000000000000000000000000000</td>
</tr>
<tr>
<td>R11</td>
<td>00000000000000000000000000000110</td>
<td>01111010000000000000000000000000</td>
</tr>
<tr>
<td>R12</td>
<td>00000000000000000000000000000110</td>
<td>01111010000000000000000000000000</td>
</tr>
<tr>
<td>R13</td>
<td>00000000000000000000000000000110</td>
<td>01111010000000000000000000000000</td>
</tr>
<tr>
<td>R14</td>
<td>00000000000000000000000000000110</td>
<td>00100100000000000000000000000000</td>
</tr>
</tbody>
</table>
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
  
  \[
  \text{costFunc}(b \text{ bits}) = \sqrt{\frac{\sum_{i=1}^{n} (x_i - x)^2}{n}}, \text{ where } x = \frac{M}{n} \quad (1)
  \]

- **Greedy strategy: good and fast**
  - A local optimal solution, where the “good” bits are selected one by one recursively
  
  \[
  \text{imbalance} (\text{bit } v) = |\#\text{ruleLChild} - \#\text{ruleRChild}| \quad (2)
  \]
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 (\(#\text{rules} \leq binth\) or PSTSS (\(#\text{rules} > binth\) to facilitate tree constructions.)
A Working Example

- An example rule set with two IPv4 address fields

<table>
<thead>
<tr>
<th>Rule id</th>
<th>src_addr field</th>
<th>dst_addr field</th>
</tr>
</thead>
<tbody>
<tr>
<td>R₁</td>
<td>228.128.0.0/9</td>
<td>0.0.0.0/0</td>
</tr>
<tr>
<td>R₂</td>
<td>223.0.0.0/9</td>
<td>0.0.0.0/0</td>
</tr>
<tr>
<td>R₃</td>
<td>0.0.0.0/1</td>
<td>175.0.0.0/8</td>
</tr>
<tr>
<td>R₄</td>
<td>0.0.0.0/1</td>
<td>225.0.0.0/8</td>
</tr>
<tr>
<td>R₅</td>
<td>0.0.0.0/2</td>
<td>225.0.0.0/8</td>
</tr>
<tr>
<td>R₆</td>
<td>128.0.0.0/1</td>
<td>123.0.0.0/8</td>
</tr>
<tr>
<td>R₇</td>
<td>128.0.0.0/1</td>
<td>37.0.0.0/8</td>
</tr>
<tr>
<td>R₈</td>
<td>0.0.0.0/0</td>
<td>123.0.0.0/8</td>
</tr>
<tr>
<td>R₉</td>
<td>178.0.0.0/7</td>
<td>0.0.0.0/1</td>
</tr>
<tr>
<td>R₁₀</td>
<td>0.0.0.0/1</td>
<td>172.0.0.0/7</td>
</tr>
<tr>
<td>R₁₁</td>
<td>0.0.0.0/1</td>
<td>226.0.0.0/7</td>
</tr>
<tr>
<td>R₁₂</td>
<td>128.0.0.0/1</td>
<td>120.0.0.0/7</td>
</tr>
<tr>
<td>R₁₃</td>
<td>128.0.0.0/2</td>
<td>120.0.0.0/7</td>
</tr>
<tr>
<td>R₁₄</td>
<td>128.0.0.0/1</td>
<td>38.0.0.0/7</td>
</tr>
</tbody>
</table>
A Working Example

Two partitioned subsets, where threshold $T = (2^{25}, 2^{25})$

The 1st subset with small destination address field

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

The 2nd subset with small source address field

<table>
<thead>
<tr>
<th>Rule id</th>
<th>src_addr field (1-32th rule bits)</th>
<th>dst_addr field (33-64th rule bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td>1110010 01*********</td>
<td>***********</td>
</tr>
<tr>
<td>$R_2$</td>
<td>1101111 10*********</td>
<td>***********</td>
</tr>
<tr>
<td>$R_9$</td>
<td>1011001 *********</td>
<td>0*********</td>
</tr>
</tbody>
</table>
A Working Example

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

Selected Bits: 37th & 39th

Leaf Node: Linear Search
Leaf Node: PSTSS Search

School of Electronic and Computer Engineering, Peking University
Part 4: Evaluation

Preliminary Evaluation

Experiment Conclusion

School of Electronic and Computer Engineering, Peking University
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/
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.
For simplicity, we think traversing a decision tree node, a rule or a tuple table as one memory access.
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

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.