Rule Caching in Software-Define Networkings

Supervisor: Prof Weifa Liang
Student: Zhenge Jia, u5433077
Date of presentation: 24th May 2016
Background

Applications, Service program

Northbound Interface: Rest API

Floodlight, Ryu, etc

Southbound Interface: OpenFlow, OpFlex, etc

Physical, Virtual switch (Open vSwitch)
Example of Flow Table

<table>
<thead>
<tr>
<th>Matching Rule</th>
<th>Action</th>
<th>Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Op.</td>
<td>Size</td>
<td>Loca</td>
</tr>
<tr>
<td>=</td>
<td>16</td>
<td>P</td>
</tr>
<tr>
<td>=</td>
<td>16</td>
<td>P</td>
</tr>
</tbody>
</table>

• Each flow table entry contains a set of packet fields to match, an action and an statistics.

When an OpenFlow Switch receives a packet it has never seen before, it sends this packet to the controller.

It can drop the packet. Or it can add a flow entry directing the switch on how to forward similar packets in the future.
Background

TCAM
(Ternary Content-Addressable Memory)

- **Pros**
  - High lookup speed
  - Wildcard matching capability
- **Cons**
  - Power-hungry (100 times more power-consuming per Mbit than SRAM)
  - Expensive (400 times more expensive per Mbit than SRAM)
  - Limited size (Could store limited number of rules)

Rule Caching

- Cache the “important” rule
- Algorithms applied in wired SDN
Related Work

A. Rule-Caching Algorithms for SDN [1]
   - Incremental rule-dependency analysis
   - Splice dependency chain by cover-set
   - Wildcard rules caching algorithm based on cover-set

   - Wildcard rules caching algorithm based on accumulative contribution

C. Classbench: A packet classification benchmark [3]
   - Policy and trace generator
Related Work

Rules Dependency DAG
Construct a directed graph where each rule is a node, and each edge captures a direct dependency between a pair of rules. [1]

Cover-set
A new rule is created to cover many low-weight rules. [1]

Wildcard Rules Caching Algorithm
The algorithm in [2] select the combination of caching rules with highest contribution by layer-by-layer calculation.
# System Model

## Rules dependency DAG

<table>
<thead>
<tr>
<th>Rule</th>
<th>Match</th>
<th>Action</th>
<th>Priority</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>000</td>
<td>Forward to port 1</td>
<td>6</td>
<td>20</td>
</tr>
<tr>
<td>R2</td>
<td>00*</td>
<td>Forward to port 3</td>
<td>5</td>
<td>30</td>
</tr>
<tr>
<td>R3</td>
<td>0**</td>
<td>Forward to port 4</td>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>R4</td>
<td>110</td>
<td>Drop</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>R5</td>
<td>10*</td>
<td>Forward to port 2</td>
<td>2</td>
<td>80</td>
</tr>
<tr>
<td>R6</td>
<td>**0</td>
<td>Forward to port 5</td>
<td>1</td>
<td>90</td>
</tr>
</tbody>
</table>

1. Rules on higher level have higher priority.
2. The dependency exists when the child rule overlap with parent rule in match field.
3. Once the rule is selected, all other dependent rules must be also stored in TCAM.
System Model

Terms

- **Weight**: The volume of traffic matching the rule
- **Cost**: The number of rules that must be installed together
- **Contribution**: The ratio of weight to cost
In each layer, select the combination of rules with the highest accumulative contribution. And get into next layer.

Select children of \( R9 \) and form the \textit{Layer_1}. Sort the rules by individual contribution.
New Wildcard Rule Caching Algorithm

Select the rule with highest individual contribution to add into cache.

Update the TCAM cache set by add the new rule into.

Algorithm 1 Compute accumulated contribution of one rule

1: function CALCULATEDACV(R : Rule, SizeTCAM : integer)
2: if $R.\text{Cost} \leq \text{SizeTCAM}$ then
3: \hspace{1em} $CMACV = R.\text{Contribution}$;
4: \hspace{1em} List = $R.\text{Children}$;
5: \hspace{1em} set = [$R$];
6: else
7: \hspace{1em} Return;
8: end if
9: flag = True;
10: while flag \equiv True do
11: \hspace{1em} combineOne = List.Max(Contribution);
12: \hspace{1em} commonSize = (CombineOne.Parents \cap set).Size;
13: \hspace{1em} combineCost = CombineOne.Cost - commonSize;
14: if combineCost + set.Cost \leq \text{SizeTCAM} then
15: \hspace{1em} $ACV = \frac{\text{combineOne.Weight} + \text{set.Weight}}{\text{combineCost + set.Cost}}$;
16: \hspace{1em} set.add((combineOne);
17: \hspace{1em} combineList = combineOne.Children;
18: \hspace{1em} List.remove(combineList);
19: \hspace{1em} List = List \cup combineList;
20: else
21: \hspace{1em} flag = False;
22: end if
23: if $ACV > CMACV$ then
24: \hspace{1em} CMACV = ACV
25: \hspace{1em} $R.\text{ruleSet} = \text{set}$;
26: end if
27: end while
28: Return CMACV;
29: end function

Break the layer-by-layer calculation. Add children of all selected rules in TCAM for the next loop.
New Wildcard Rule Caching Algorithm

Result

Input dependency DAG

Result under layer-by-layer caching algorithm

Result under new wildcard rules caching algorithm
Future Works

• Test the wildcard rules caching algorithm for large amount of rules.

• Test in real SDN environment
References


Thanks