A Sorted Partitioning Approach to High-speed and Fast-update OpenFlow Classification

Sorrachai Yingchareonthawornchai,
James Daly, Alex X. Liu, Eric Torng

November 9th, 2016
Motivation

• Software-defined networking:
  • Performance bottleneck: lookup process

• OpenFlow places new demands on packet classifiers
  • Incremental Update ( ~ 1 microseconds per update)
  • Dimensional Scalability ( 5 in traditional, up-to 45 fields in OpenFlow 1.5 and higher)

• Traditional packet classification focus on fast classification
  • but not for so fast on update

• Current practices (e.g., Open vSwitch) use Tuple Space Search (TSS)
  • which has fast update, but slow classification
New challenges for OpenFlow packet classification,
(1) Dynamism of Rules
(2) High number of dimensions/fields

*Figures from [Pankaj Gupta, et al, Algorithms for Packet Classification]*
Related Work

• Decision Trees (HiCuts, HyperCuts, HyperSplit)
  • Fast classification, but slow update (rule replication)

• Partitioning Methods
  • Fast update, but slow classification (too many partitions) (Tuple Space Search)
  • Slow update, fast classification (SAX-PAC, Independent Set)

• Decision Trees + Partitioning (EffiCuts, SmartSplit)
  • Slow update
<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Classification</th>
<th>Update</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Search</td>
<td>$O(dn)$</td>
<td>$O(d)$</td>
<td>$O(dn)$</td>
</tr>
<tr>
<td>Tuple Space Search</td>
<td>$O(dT)$</td>
<td>$O(d)$</td>
<td>$O(dn)$</td>
</tr>
<tr>
<td>Decision Trees</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
<td>$\Omega(n^2)$</td>
</tr>
</tbody>
</table>

**Challenge:** achieving both (1) fast packet classification and (2) fast rule update

$d$ – number of fields  
$n$ – number of rules  
$T$ – number of sub-partitions
<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Classification</th>
<th>Update</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Search</td>
<td>$O(dn)$</td>
<td>$O(d)$</td>
<td>$O(dn)$</td>
</tr>
<tr>
<td>Tuple Space Search Search</td>
<td>$O(dT)$</td>
<td>$O(d)$</td>
<td>$O(dn)$</td>
</tr>
<tr>
<td>Decision Trees</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
<td>$\Omega(n^2)$</td>
</tr>
<tr>
<td>Proposed: PartitionSort</td>
<td>$O(dT' + T' \log n)$</td>
<td>$O(dT' + T' \log n)$</td>
<td>$O(dn)$</td>
</tr>
</tbody>
</table>

$d$ – number of fields

$n$ – number of rules

$T$ – number of sub-partitions
<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Classification</th>
<th>Update</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Search</td>
<td>$O(dn)$</td>
<td>$O(d)$</td>
<td>$O(dn)$</td>
</tr>
<tr>
<td>Tuple Space Search</td>
<td>$O(dT)$</td>
<td>$O(d)$</td>
<td>$O(dn)$</td>
</tr>
<tr>
<td>Decision Trees</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
<td>$\Omega(n^2)$</td>
</tr>
<tr>
<td>Proposed: PartitionSort</td>
<td>$O(dT' + T' \log n)$</td>
<td>$O(dT' + T' \log n)$</td>
<td>$O(dn)$</td>
</tr>
</tbody>
</table>

fast classification and fast update if $T'$ is small
Current state of the art

- Tuple Space Search
- Decision Trees
Current state of the art

- Decision Trees
- Proposed
- Tuple Space Search

<table>
<thead>
<tr>
<th>Classification Time</th>
<th>Update Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
</tr>
</tbody>
</table>
Proposed Approach

Define “Sortable” Rules

\[ r_1 <_f r_2 <_f \ldots <_f r_n \]

Fast search and update with linear space
1. Ruleset Sortability

2. Multi-dimensional Interval Tree (MITree)

3. Sortable Ruleset Partitioning

4. Online Partitioning
Definition: Given field order F, rule 1 < rule 2 if on projection on F[1]
   If non-intersecting, interval 1 < interval 2
   If identical, rule 1 < rule2 on projection on of next field recursively
   Else, rule 1 | | rule 2

Definition: Rules are sortable if there exists field ordering that has total order
Ruleset Sortability: Lexical Field Order Comparison

\[ X(r_2) = X(r_3) \]

\[ r_1 \prec_{XY} r_2 \prec_{XY} r_3 \]

\[ r_1 \parallel r_2 \]
Sortable Ruleset Partitioning
Sortable Ruleset Partitioning

• Objective: minimize number of sortable subsets

• Main Idea:
  • repeatedly find the maximal sortable subset

• Challenges:
  • d! field ordering!
  • Need an efficient algorithm for maximal sortable subset

• Proposed:
  • Greedy field selection: repeatedly find a field with maximum sortability
    • For each unselected field, sortability = number of maximum non-intersecting intervals
      • Pick a field with maximum sortability
Example

(a) Selected field: {}

\[
\begin{array}{c}
\text{X:} & r_1 & r_2 & r_3 & r_4 & r_5 & r_6 & r_7 \\
\text{Y:} & r_1 & r_2 & r_3 & r_4 & r_5 & r_6 & r_7
\end{array}
\]
Example

(a) Selected field: {}

\[
\begin{array}{c}
\text{X:} \\
\hline
\text{Partition 1:} & \text{Partition 2:} \\
\hline
r_1 r_2 & r_6 r_7 \\
\hline
\text{MWIS: 4} \\
\end{array}
\]

\[
\begin{array}{c}
\text{Y:} \\
\hline
r_5 & r_7 & r_3 r_6 \\
\hline
r_1 & r_4 r_2 \\
\hline
\text{MWIS: 3} \\
\end{array}
\]
Example

(a) Selected field: {}

Partition 1

Partition 2

MWIS: 4

(b) Selected field: x

Partition 2

Partition 1

MWIS: 4

(c) Selected fields: xy with \{r_1, r_2, r_7, r_6\} in total order xy
Properties of GreedyFieldSelection

• Discover “good” field order and
• Find maximal sortable subset simultaneously
• Complexity: $O(d^2 n \log n)$
Online Sortable Ruleset Partitioning

• How to perform update?
  • Given rule r, test for each tree if r can insert
  • If not, create a new tree of the rule r

• Problem:
  • Number of trees can be large

• Ad-hoc solution:
  • periodically repartition using the previous algorithm

• Not acceptable in OpenFlow packet classification!
Online Sortable Ruleset Partitioning

• Proposed Solution: *Adaptive Field Ordering*
  • Given rule r, test for each tree if r can insert
  • Once inserted, if tree size is less than T, run sortable partitioning on this subset to find a better ordering

• This is fast due to small size of tree (T = 10)
• Work in completely online manner
• No need to know a sequence of rules a priori
Experimental Results

• Compare with SmartSplit, TupleSpace, SAX-PAC
• Classification Time, Update Time, Memory usage, Construction Time
• Internal comparison: Offline vs. Online Partition

• Run over 500 rule sets consisting of ACL, FW, IPC from ClassBench
  • Ranging number from 1k, 2k, 4k, 8k, 16k, 32k, 64k
  • Ranging dimensions from 5, 10, 15, 20
  • Each size has multiple of 5 rule sets

• Classify 1m packets, update 1m intermixed requests
• We run online construction by default
On average, PartitionSort is 7.2x faster on classification, but 1.7x slower on update.

On average, PartitionSort can classify 4.5 Mpps with 0.65 microsecond update time.
Conclusion

Classification Time vs. Update Time

- SmartSplit
- PartitionSort
- Tuple Space Search
Final Note

• PartitionSort is available on Github  https://github.com/sorrachai/PartitonSort

• Simulation and benchmark are available including implementations of other packet classifiers
  • Packet generator (by Classbench API)
  • Rule generator (by Classbench API)
  • Packet Classification Simulator (New)

• You can implement your own packet classifier and compare!

• Questions?
Appendix
PartitionSort vs. SaxPac