Meta-algorithms for Software based Packet Classification

Peng He (Institute of Computing Technology, China)
Gaogang Xie(Institute of Computing Technology, China)
Kavé Salamatian (University of Savoie, France)
Laurent Mathy (University of Liège, Belgium)
Agenda

- Motivation
- Key Observations
- Two Meta Methods
  - Memory Consumption Model
  - Characterizing range distribution uniformity
- The AutoPC framework and SmartSplit algorithm
- Experiment Results.
Agenda

- **Motivation**
- **Key Observations**
- **Two Meta Methods**
  - Memory Consumption Model
  - Characterizing range distribution uniformity
- **The AutoPC framework and SmartSplit algorithm**
- **Experiment Results.**
Packet Classification

Packet Classification: find the highest priority rule that matches a packet

Classifier: a set of rules

<table>
<thead>
<tr>
<th>Source IP</th>
<th>Destination IP</th>
<th>Source Port</th>
<th>Destination Port</th>
<th>Protocol</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>120.0.0.0/24</td>
<td>198.12.130.0/2</td>
<td>0:65535</td>
<td>11:17</td>
<td>0xFF/0xFF</td>
<td>Accept</td>
</tr>
<tr>
<td>138.42.83.1/0</td>
<td>174.3.18.0/8</td>
<td>50:10000</td>
<td>0:65535</td>
<td>0x06/0xFF</td>
<td>Deny</td>
</tr>
</tbody>
</table>

Packet classification is key for
- Security
- Traffic monitoring and analysis
- QoS

Packet classification prevalent in modern routers
Two Solutions

- **TCAM based Solutions**
  - fast, deterministic performance.
  - power-hungry, expensive.

- **RAM based (algorithmic) Solutions**
  - Tradeoff between **memory size** and **memory accesses**.
  - Theoretical Bounds
    - $O(\log N)$ speed and $O(N^k)$ space
    - $O(\log^{k-1} N)$ speed and $O(N)$ space
  - All existing algorithms are heuristic algorithms, exploiting special ruleset *structures*. 
Performance Unpredictability

Choose the right algorithm for different ruleset!

### TABLE I: Performance comparison on different rulesets

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Ruleset(size)</th>
<th>Memory size</th>
<th>Mem. accesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>HyperSplit</td>
<td>ACL1_100K</td>
<td>2.12MB</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>ACL2_100K</td>
<td>83MB</td>
<td>43</td>
</tr>
<tr>
<td>EffiCuts</td>
<td>ACL1_100K</td>
<td>3.23MB</td>
<td>65</td>
</tr>
<tr>
<td></td>
<td>ACL2_100K</td>
<td>4.81MB</td>
<td>136</td>
</tr>
</tbody>
</table>

same algorithm, different ruleset,
Our contributions

- Study this performance unpredictability
  - Two ruleset features
- Develop methods to predict performance
  - Two meta methods
- Choose the right algorithm and develop more efficient one
  - The AutoPC framework and SmartSplit Algorithms

Workaround

- Compare all the alternative algorithms, choose one with better performance.
- Need more than 24 hours to build a tree for some rulesets. (INFOCOM 09')
Motivation

Key Observations

Two Meta Methods
- Memory Consumption Model
- Characterizing range distribution uniformity

The AutoPC framework and SmartSplit algorithm

Experiment Results.
Geometric View of rulesets

<table>
<thead>
<tr>
<th>Rule #</th>
<th>Field 1</th>
<th>Field 2</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>111*</td>
<td>*</td>
<td>DROP</td>
</tr>
<tr>
<td>R2</td>
<td>110*</td>
<td>*</td>
<td>PERMIT</td>
</tr>
<tr>
<td>R3</td>
<td>*</td>
<td>010*</td>
<td>DROP</td>
</tr>
<tr>
<td>R4</td>
<td>*</td>
<td>011*</td>
<td>PERMIT</td>
</tr>
<tr>
<td>R5</td>
<td>*</td>
<td>*</td>
<td>PERMIT</td>
</tr>
</tbody>
</table>

Table 2: An example ruleset

Each rule can be viewed as a *hyperrectangle*

Algorithms performs “cuts” or “splits” on the space to reduce the search space.
An Example (HiCuts and HyperSplit)

(a) HiCuts
- Equal sized cutting
- 00*, 01*, 10*, 11*
- R3 R4 R5
- R1 R2 R3 R4
- R5 R5

(b) HyperSplit
- Unequal sized splitting
- [0,13] [14,15]
- R2 R3 R4 R5
- R1 R3 R4
- R5 R5

2 dimension rule-set
Each field has 4 bits

A lot of rule duplications, large memory size
Less rule duplications, smaller memory size

<table>
<thead>
<tr>
<th>Rule #</th>
<th>Field 1</th>
<th>Field 2</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00*</td>
<td>*</td>
<td>DROP</td>
<td></td>
</tr>
<tr>
<td>01*</td>
<td>*</td>
<td>PERMIT</td>
<td></td>
</tr>
<tr>
<td>10*</td>
<td>*</td>
<td>DROP</td>
<td></td>
</tr>
<tr>
<td>11*</td>
<td>*</td>
<td>PERMIT</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: An example ruleset
Non-uniformly distributed ranges

Need 8 equal-sized cuts to separate R1 and R2, however R3, R4 and R5 get duplicated.

The range distribution on Field 1 is not uniform, resulting in more memory accesses or more rule duplications.
Sparse rules and orthogonal structure rules

(a) sparse rules

Rules can be easily separated

(b) orthogonal structure rules

No matter how to cut, rules get duplicated. Memory size increases.
EffiCuts

Untangle the orthogonal structures

Smallest memory size
No rules get duplicated

Table 2: An example ruleset
Orthogonal Structures can be invalid.

Field 1 is enough to separate rules. Orthogonal structures can be ignored.

2 memory accesses
1 memory access
Discussions

- “Orthogonal structures” should be considered, and rules should be eventually split in order to untangle these structure and avoid memory explosion.

- When splitting a ruleset, if a dimension appears that contain only small ranges, it should be used to separate the rules with a single tree.

- Equal-sized cutting becomes more efficient when ruleset ranges are uniform, if not splitting with non-equal sized intervals should be considered.
Agenda

- Motivation
- Key Observations
- Two Meta Methods
  - Memory Consumption Model
  - Characterizing range distribution uniformity
- The AutoPC framework and SmartSplit algorithm
- Experiment Results.
Memory Consumption Model

90% memory is for rule duplications.

R1, R2, R3, R4 are \((small, large)\) rules while R5, R6, R7 and R8 are \((large, small)\) rules.

Key Observations:
Cutting F1 will usually cause \((small, large)\) rules duplicate. Cutting F2, \((large, small)\) rules will get duplicated.

\[ l_s \times Y + s_l \times X \text{ rule duplications} \]
Assume: in each divided space, there are \( \text{binth} \) rules. Half of rules are (large, small) rules and the other half are (small, large) rules.

We need \( \frac{ls}{\text{binth}/2} \) cuts on F1, \( \frac{sl}{\text{binth}/2} \) cuts on F2

In total, we have \( \frac{ls}{\text{binth}/2} \times sl + \frac{sl}{\text{binth}/2} \times ls \)

rule duplications
Memory Consumption Model

- How about (small, small) rules and (large, large) rules?
  - See paper for details.
- How to minimize the estimation error?
  - See paper for details

\[
\begin{align*}
M_{ss} &= ss \times PTR \\
M_{ls} &= ls \times \frac{sl + ss \times \alpha}{binth/2} \times PTR \\
M_{sl} &= sl \times \frac{ls + ss \times (1 - \alpha)}{binth/2} \timesPTR \\
M_{ul} &= ll \times \frac{sl + ss \times \alpha}{binth/2} \times \frac{ls + ss \times (1 - \alpha)}{binth/2} \times PTR
\end{align*}
\]
Characterizing range distribution uniformity

Build a interval tree for the ranges on each field.

Characterizing range distribution by the \textit{shape} of interval tree!
The shape of Interval trees

Quasi-Balanced subtrees

\[ B_{\text{ratio}} = \frac{\text{# Nodes in the } k\text{th level}}{\text{# Nodes in the } (k-1)\text{th level}} \geq 1.5 \]

Balance tree distance \( D \)
Max Balance tree distance \( D_{\text{max}} \)

\[ D_{\text{max}} = 3 \text{ for this tree} \]
The threshold of range distribution uniformity

When $D_{max} \leq \frac{1}{2} \log \frac{\#rules}{binth}$

One should use equal-sized cut based algorithm

See paper for details.
SmartSplit Algorithm

Split the ruleset according to large/small ranges on IP fields

Merge rulesets (ss, sl) or (ss, ls)

Use different algorithms on three rulesets.

Diagram:
- Split the ruleset into large (ll) and small (all) ranges.
- Merge rulesets (ss, sl) or (ss, ls).
- Use different algorithms (al1, al2, al3) on three rulesets (rs1, rs2, rs3).
The *AutoPC* framework

Automatically perform the tradeoff for a given ruleset.
Experiments Results

- We conduct extensive experiments for memory model and compare the performance of AutoPC and SmartSplit.

- The performance of different algorithms is related the logarithm of the memory.

Fig. 7: Average Memory Access Latency and Memory Size

Fig. 8: Cache Misses Rate and Memory size
Experiments Results

- For 60 rulesets of various size, the real memory size vs. the memory estimation
Memory estimation is fast

<table>
<thead>
<tr>
<th>Ruleset</th>
<th>$\log_2 Mem$</th>
<th>Time(s)</th>
<th>$\log_2 Mem$</th>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>acl1_100K</td>
<td>19.7</td>
<td>167</td>
<td>21.4</td>
<td>0.4</td>
</tr>
<tr>
<td>acl2_100K</td>
<td>26.6</td>
<td>234</td>
<td>27.2</td>
<td>0.6</td>
</tr>
<tr>
<td>acl3_100K</td>
<td>28</td>
<td>1794</td>
<td>30</td>
<td>0.7</td>
</tr>
<tr>
<td>acl4_100K</td>
<td>27</td>
<td>1061</td>
<td>29</td>
<td>0.6</td>
</tr>
<tr>
<td>acl5_100K</td>
<td>19</td>
<td>186</td>
<td>18.5</td>
<td>0.4</td>
</tr>
<tr>
<td>ipc1_100K</td>
<td>30</td>
<td>2424</td>
<td>29</td>
<td>0.6</td>
</tr>
<tr>
<td>ipc2_100K</td>
<td>29</td>
<td>1132</td>
<td>28</td>
<td>0.6</td>
</tr>
<tr>
<td>fw1_100K</td>
<td>30</td>
<td>2124</td>
<td>32</td>
<td>1.7</td>
</tr>
<tr>
<td>fw2_100K</td>
<td>30</td>
<td>2568</td>
<td>29</td>
<td>0.8</td>
</tr>
<tr>
<td>fw3_100K</td>
<td>29.5</td>
<td>1148</td>
<td>32</td>
<td>1.9</td>
</tr>
<tr>
<td>fw4_100K</td>
<td>33</td>
<td>6413</td>
<td>34</td>
<td>10</td>
</tr>
<tr>
<td>fw5_100K</td>
<td>30</td>
<td>1891</td>
<td>32</td>
<td>2</td>
</tr>
</tbody>
</table>

**TABLE VI:** Estimated and Actual Memory size of Large rulesets

1000x faster then really building a decision tree
Experiments Results

- Compare SmartSplit and EffiCuts

![Graph showing memory size comparison between EffiCuts and SmartSplit with 10x small memory improvement.]

![Graph showing memory accesses comparison between EffiCuts and SmartSplit with 2~4x faster access.]

Fig. 12: Memory and Accesses for EffiCuts and SmartSplit
Real Evaluation

- On Xeon machines, the SmartSplit is in average 2 times faster.

- AutoPC can choose the right algorithm, which can be 3.8 / 18.9 faster than using EffiCuts or HyperSplit alone.
Conclusion

- Orthogonal structure and non-uniformly distributed ranges have impact on algorithm performance

- Memory size can be roughly estimated by simply counting orthogonal structure rules (memory consumption model)

- Through exploiting uniformity of range distributions, we can improve the performance (SmartSplit)

- Through carefully choose right algorithm, we can automatically achieve better tradeoff between memory size and memory accesses (AutoPC)
Thank you!

hepeng@ict.ac.cn