Packet Classification: From Theory to Practice

Jun Li
Most contributions from Yaxuan Qi and many other students of mine

Tsinghua Univ., Beijing, China
Outline

- Packet Classification Introduction
- Review of Classic Algorithms
- Our Recent Advancement
- Conclusion and Discussion
Network Security Gateway

OSI
- L1
- L2
- L3
- L4
- ...
- L7

AV/AS
- packet filtering
- session filtering
- flow filtering
- file filtering

XML/SOAP, etc.
- app. filtering
- deep inspection
- stateful inspection

FW

FW + VPN + IDS/IPS + AV/AS + …

Content Filtering

LB
router
switch
The Packet Classification Problem

Definition:

- Given \( N \) rules, find the action associated with the highest priority rule matching an incoming packet.

Applications:

- Access control
- Quality of service
- Traffic engineering
- Intrusion detection
- ...

<table>
<thead>
<tr>
<th>Rule</th>
<th>Field 1 (sIP)</th>
<th>Field 2 (dPort)</th>
<th>Field F (protocol)</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>166.111.72.50/21</td>
<td>80</td>
<td>UDP</td>
<td>Deny</td>
</tr>
<tr>
<td>2</td>
<td>166.168.3.0/24</td>
<td>53</td>
<td>TCP</td>
<td>Accept</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>N</td>
<td>0.0.0.0/0</td>
<td>0-65535</td>
<td>ANY</td>
<td>Drop</td>
</tr>
</tbody>
</table>
Given a classifier $C$ with $N$ rules, $R_j$, $1 \leq j \leq N$, where $R_j$ consists of three entities:

- **Range expressions**: $R_j[i]$, $1 \leq i \leq d$, on each of the $d$ header fields.

- **Priority**: $\text{pri}(R_j)$, indicating the priority of the rule in the classifier. Commonly, $1^{\text{st}}$ policy has the highest priority, $N^{\text{th}}$ policy (normally deny all) has the lowest.

- **An action**: referred to as $\text{action}(R_j)$. In firewall, usually there is a default policy as the last policy that matches and denies all.
Problem Definition II

- For an incoming packet $P$ with the header considered as a $d$-tuple of points $(P_1, P_2, ..., P_d)$, the *d-dimensional packet classification problem* is to find the rule $R_m$ with the highest priority among all the $N$ rules that match the $d$-tuple.

  - i.e., $\text{pri}(R_m) > \text{pri}(R_j), \forall j \neq m, 1 \leq j \leq N$, such that $P_i$ matches $R_j[i], 1 \leq i \leq d$. In firewall, this is the first matching policy.
  
  - $R_m$ is called the best matching rule for packet $P$, therefore $\text{action}(R_m)$ is applied to packet $P$. 
Problem Complexity: Theoretically

- **Computational Geometry**
  - Point Location among $N$ non-overlapping hyper-rectangles in $F$ dimensions
  - Takes either $O(\log N)$ time with $O(N^F)$ space or $O(N)$ space with $O(\log^{F-1} N)$ time
  - E.g. $N=1000$, $F=4$: 1000G memory or 1000 times access

- **De-overlapping**
  - Each field need up to $(2N-1)$ non-overlapping regions to represent $N$ rules. How about $F$ fields?

- **Range-to-Prefix**
  - Each rule with ranges in $[0, 2^W-1]$ becomes up to $(2^W-2)^F$ rules. How about $N$ rules?
Geometric Interpretation in 2D

e.g. (128.16.46.23, *)

e.g. (144.24.0.0/16, 120.58.64.0/24)
Problem Complexity: Practically

Fortunately
- Few application reaches the worst case bound
- Real-life rule sets have some inherent data-structures

<table>
<thead>
<tr>
<th>Rule Sets</th>
<th># rules</th>
<th># non-over ranges in each field (theoretical)</th>
<th># non-over ranges in sIP (practical)</th>
<th># non-over ranges in dIP (practical)</th>
<th># non-over ranges in sPT (practical)</th>
<th># non-over ranges in dPT (practical)</th>
<th># non-over rectangles (theoretical)</th>
<th># non-over rectangles (practical)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FW1</td>
<td>269</td>
<td>539</td>
<td>100</td>
<td>111</td>
<td>23</td>
<td>77</td>
<td>8.44 x 10^{16}</td>
<td>1.97 x 10^7</td>
</tr>
<tr>
<td>FW1-100</td>
<td>92</td>
<td>185</td>
<td>19</td>
<td>45</td>
<td>20</td>
<td>48</td>
<td>1.17 x 10^9</td>
<td>8.21 x 10^5</td>
</tr>
<tr>
<td>FW1-1K</td>
<td>791</td>
<td>1583</td>
<td>221</td>
<td>314</td>
<td>23</td>
<td>75</td>
<td>6.28 x 10^{12}</td>
<td>1.20 x 10^9</td>
</tr>
<tr>
<td>FW1-5K</td>
<td>4653</td>
<td>9307</td>
<td>3429</td>
<td>5251</td>
<td>23</td>
<td>77</td>
<td>7.50 x 10^{15}</td>
<td>3.19 x 10^10</td>
</tr>
<tr>
<td>FW1-10K</td>
<td>9311</td>
<td>18623</td>
<td>6496</td>
<td>10961</td>
<td>24</td>
<td>80</td>
<td>1.20 x 10^{17}</td>
<td>1.71 x 10^13</td>
</tr>
<tr>
<td>ACL1</td>
<td>752</td>
<td>1505</td>
<td>539</td>
<td>100</td>
<td>23</td>
<td>77</td>
<td>5.13 x 10^{12}</td>
<td>9.67 x 10^6</td>
</tr>
<tr>
<td>ACL1-100</td>
<td>98</td>
<td>197</td>
<td>100</td>
<td>111</td>
<td>23</td>
<td>77</td>
<td>1.51 x 10^9</td>
<td>4.03 x 10^5</td>
</tr>
<tr>
<td>ACL1-1K</td>
<td>916</td>
<td>1833</td>
<td>197</td>
<td>314</td>
<td>23</td>
<td>75</td>
<td>1.13 x 10^{13}</td>
<td>1.32 x 10^9</td>
</tr>
<tr>
<td>ACL1-5K</td>
<td>4415</td>
<td>8831</td>
<td>221</td>
<td>314</td>
<td>23</td>
<td>75</td>
<td>6.08 x 10^{15}</td>
<td>2.42 x 10^9</td>
</tr>
<tr>
<td>ACL1-10K</td>
<td>9603</td>
<td>19207</td>
<td>3429</td>
<td>5251</td>
<td>23</td>
<td>77</td>
<td>1.36 x 10^{17}</td>
<td>1.90 x 10^9</td>
</tr>
<tr>
<td>IPC1</td>
<td>1550</td>
<td>3101</td>
<td>6496</td>
<td>10961</td>
<td>24</td>
<td>80</td>
<td>9.25 x 10^13</td>
<td>3.40 x 10^9</td>
</tr>
<tr>
<td>IPC1-100</td>
<td>99</td>
<td>159</td>
<td>100</td>
<td>111</td>
<td>23</td>
<td>77</td>
<td>1.57 x 10^9</td>
<td>9.49 x 10^5</td>
</tr>
<tr>
<td>IPC1-1K</td>
<td>938</td>
<td>1877</td>
<td>2377</td>
<td>4604</td>
<td>59</td>
<td>94</td>
<td>1.24 x 10^{13}</td>
<td>1.70 x 10^9</td>
</tr>
<tr>
<td>IPC1-5K</td>
<td>4460</td>
<td>8921</td>
<td>886</td>
<td>2125</td>
<td>59</td>
<td>94</td>
<td>6.33 x 10^{15}</td>
<td>1.03 x 10^10</td>
</tr>
<tr>
<td>IPC1-10K</td>
<td>9037</td>
<td>18075</td>
<td>2377</td>
<td>4604</td>
<td>59</td>
<td>94</td>
<td>1.07 x 10^{17}</td>
<td>6.07 x 10^10</td>
</tr>
</tbody>
</table>

Note: sIP, dIP, sPT, and dPT are source IP, destination IP, source Port and destination Port. FW, ACL, IPC are firewall policies, access control lists, and IP chain rules.
Outline

- Packet Classification Introduction
- Review of Classic Algorithms
- Our Recent Advancement
- Conclusion and Discussion
Existing Work: Basic Ideas

- Divide-and-Conquer
  - **Space Decomposition**
    - Decompose the search space into multiple subspaces using a set of axis-orthogonal hyper-planes. Each sub-space is associated with a sub-set of rules.
  - **Recursion Scheme**
    - Recursively apply the space decomposition, the original problem is divided into a series of sub-problems with smaller search space and fewer rules.
Example Rule Set

2-field rules with overlap
R1 has the highest priority
R5 is the default rule

2 Points
P1: (00, 10) → R4
P2: (11, 10) → R5

<table>
<thead>
<tr>
<th></th>
<th>X-field</th>
<th>Y-field</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>11</td>
<td>00</td>
</tr>
<tr>
<td>R2</td>
<td>1*</td>
<td>01</td>
</tr>
<tr>
<td>R3</td>
<td>0*</td>
<td>0*</td>
</tr>
<tr>
<td>R4</td>
<td>0*</td>
<td>1*</td>
</tr>
<tr>
<td>R5</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>
Packet Classification Algorithms

- Hierarchical Trie
- Set-pruning Trie
- Grid-of-Trie
- Bit Vector
- HiCuts
- Cross-producting
- ABV
- HyperCuts
- RFC
- AFBV
- AggreCuts
- HSM
Hierarchical Tries

- Back-tracking search: $O(Wd)$ time
- Each rule appears only once: $O(N)$ space
Set-pruning Tries

- No back-tracking search: $O(dW)$ time
- Rules may appear multiple times: $O(Nd)$ space
Grid of Tries

- No back-tracking search: $O(dW)$ time
- Rules appear only once: $O(N)$ space
- Only work for 2-D scheme
Packet Classification Algorithms

- Hierarchical Trie
- Set-pruning Trie
- Grid-of-Trie
- Bit Vector
- ABV
- AFBV
- HiCuts
- HyperCuts
- AggreCuts
- Cross-producing
- RFC
- HSM
Bit Vector

- Field-independent search: $O(dW+N)$ time
- Bitmap Storage: $O(dN^2)$ space
- Bitmap comparison is time-consuming
Packet Classification Algorithms

- Hierarchical Trie
- Set-pruning Trie
- Grid-of-Trie
- Bit Vector
- ABV
- AFBV
- HiCuts
- HyperCuts
- AggreCuts
- Cross-producing
- RFC
- HSM
HiCuts

Equal-sized space partition
Field-dependent search
Linear search at leaf-node
Child-node merge
Full-covered region check
HyperCuts and AggreCuts

HyperCuts: multi-dimensional cuttings

AggreCuts: pointer array compression
Packet Classification Algorithms

- Hierarchical Trie
- Bit Vector
- HiCuts
- Set-pruning Trie
- ABV
- HyperCuts
- Grid-of-Trie
- AFBV
- AggreCuts
- Cross-producting
- RFC
- HSM
Cross-producting

- Max#segments: $2N+1$
- CP table size: $O(N^d)$
- Binary search: $O(d \times \log(N))$
HSM

Packet

Binary search

sIP

dIP

sPT

dPT

Segment points

AMT

AMT: Address Mapping Table

PMT

PMT: Port Mapping Table

PLT

PLT: Policy Lookup Table

Cross-producting table
Summary: Taxonomy

Prefix Search Algorithms:
- Hierarchical Trie
  - Set-pruning Trie
    - Grid-of-Trie
  - Bit Vector
    - ABV
    - AFBV

Range Search Algorithms:
- HiCuts
  - HyperCuts
  - Cross-producting
- RFC
  - AggreCuts
  - HSM

Decomp

<table>
<thead>
<tr>
<th>Decomp \ Recur</th>
<th>rule-based</th>
<th>equal-sized</th>
</tr>
</thead>
<tbody>
<tr>
<td>field-independent</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>field-independent</td>
<td>?</td>
<td></td>
</tr>
</tbody>
</table>

Our recent advancement

equal-sized space decomp

rule-based space decomp
Outline

- Packet Classification Introduction
- Review of Classic Algorithms
- Our Recent Advancement
- Conclusion and Discussion
HyperSplit: Motivation

- High-speed classification
  - Speed is the most important performance metric
  - Speed should be bounded in the worst case
- Modest memory storage
  - Memory storage cannot exceed the overall system memory size
  - Modest memory storage enables the use of fast memory technology
HyperSplit: Ideas

- Rule-based space decomposition
  - Binary splitting: $O(\log(N))$ search time
  - Intelligently select the splitting point
    - Heuristic-1: select the mid-value point
    - Heuristic-2: select the mid-segment point
    - Heuristic-3: select the weighted mid-segment point

- Field-dependent recursion scheme
  - Always select the most discriminative field to apply decomposition
    - For Heuristic-1 and Heuristic-2: select the field with the largest number of segments
    - For Heuristic-3: select the field with minimized average weight

- Termination conditions
  - There’re less than $\text{Thresh}$ rules in current search space
  - The current search space is fully covered by all the current rules
HyperSplit: Example

Data structure design and optimization also important, see paper.
Data-set and Test-bed

- **Algorithms**
  - HyperSplit-1, HyperSplit-8, HiCuts-1, HiCuts-8 and HSM

- **Data-set**
  - WUSTL Evaluation of Packet Classification Algorithms
    - 100~10K real-life 5-tuple firewall, ACL and IP Chain rules
    - [http://www.arl.wustl.edu/~hs1/PClassEval.html](http://www.arl.wustl.edu/~hs1/PClassEval.html)

- **Test-bed**
  - Memory access, usage, and preprocessing time: 2.0GHz dual-core with 4GB DDRII memory running Ubuntu 8.04LTS
  - Throughput: Cavium OCTEON 3860 multi-core processor running in “Simple Executive” mode
  - SmartBit packet generator
Cavium OCTEON 3860

4x RGMII
4x RGMII
Boot/flash GPIO 2xUART
64 bit 133MHz

Packet Interface 0
Packet Interface 1
Misc I/O
PCI-X
TCP Unit
Compress/Decomp
Secure Vault

Packet
Scheduler/
Sync Order
Packet Input
I/O Bridge
Packet Output

Hyper Access Low Latency Memory Controller

16x RegEx Engines

Packet
Security
MIPS64 r2 Integer Core
32K Lcache
8K Dcache
2K Write Buffer

16 cnMIPS64 cores

 PACKET
Security
MIPS64 r2 Integer Core
32K Lcache
8K Dcache
2K Write Buffer

1 MB Shared L2 Cache
Hyper Access Memory Controller

Optional 2x18-bit RLDRAM2

DDR2 upto 667 MHz
72 or 144 bit wide
4GBx DIMMs max
Memory Access

- **HyperSplit-1 vs. HSM**: 20~50% less access
- **HyperSplit-1 vs. HiCuts-1**: 50~80% less access
- **HyperSplit-8 vs. HiCuts-8**: 10~30% less access
Memory Usage

- **HyperSplit-1 vs. HSM**
  - 1~2 orders less memory;
  - HSM fails for fw1-5k, fw1-10k and ipc1-10k (due to 4GB+ memory usage)
- **HyperSplit-1 vs. HiCuts-1**
  - 1~2 orders less memory;
  - HiCuts-1 fails for fw1-5k, fw1-10k and ipc1-10k (due to 4GB+ memory usage)
- **HyperSplit-8 vs. HiCuts-8**
  - 1~2 orders less memory; successful for all rule-sets.
Pre-processing: Intel Core2 duo 2.0GHz, 4G DDRII, Ubuntu8.04 LTS

HyperSplit-1 vs. HSM
- 1~4 orders less time
- HSM fails for fw1-5k, fw1-10k and ipc1-10k (e.g. 24 hours for fw1-10k, and failed)

HyperSplit-1 vs. HiCuts-1
- 1~2 orders less time
- HiCuts-1 fails for fw1-5k, fw1-10k and ipc1-10k (e.g. 4 hours for fw1-10k, and failed)

HyperSplit-8 vs. HiCuts-8
- About 1 order less time
Throughput

- 64B packet-size test with different # of cores:
  - HyperSplit: 6.4/4.2Gbps with 16 cores; HSM: 2.4Gbps; HiCuts: 1.1/3.2Gbps
- Variable packet-size test with 16 cores:
  - HyperSplit: 8Gbps with 128B+ packets; HSM: 256B+; HiCuts: 1024B+/256B+
Outline

- Packet Classification Introduction
- Review of Classic Algorithms
- Our Recent Advancement
- Conclusion and Discussion
Conclusion and Discussion

- **Conclusion**
  - Theoretically: Explicit worst-case search time $O(\log N)$
  - Practically: 6.4Gbps on OCTEON3860, apply to all data sets @WUSTL

- **Discussion**
  - Adaptive to different memory hierarchy rather than the L2-DRAM coherent memory system?
  - Policy-based switching rather than routing?
  - Application identification rather than flow classification?
References


Network Security Lab, RIIT
Tsinghua University
Our Recent Publications


Thanks!

Questions?