A Computational Approach to Packet Classification

Alon Rashelbach    Ori Rottenstreich    Mark Silberstein
Technion, Israel

SIGCOMM 2020
A Brief about Packet Classification (1/3)

- Access Control
- Quality of Service
- Packet Forwarding
- Firewalls
A Brief about Packet Classification (2/3)

<table>
<thead>
<tr>
<th>Src IP</th>
<th>Dst IP</th>
<th>Src Port</th>
<th>Dst Port</th>
<th>Action</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.10.*</td>
<td>8.8.**.*</td>
<td>0-65535</td>
<td>80</td>
<td>Port 1</td>
<td>3</td>
</tr>
<tr>
<td>10.0.20.*</td>
<td>8.8.7.*</td>
<td>0-65535</td>
<td>443</td>
<td>Port 2</td>
<td>2</td>
</tr>
<tr>
<td>10.0.<em>.</em></td>
<td>8.8.7.1</td>
<td>0-65535</td>
<td>0-65535</td>
<td>Drop</td>
<td>1</td>
</tr>
</tbody>
</table>
## A Brief about Packet Classification (3/3)

### Incoming Packets

<table>
<thead>
<tr>
<th>Src IP</th>
<th>Dst IP</th>
<th>Src Port</th>
<th>Dst Port</th>
<th>Action</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.10.*</td>
<td>8.8.*</td>
<td>0-65535</td>
<td>80</td>
<td>Port 1</td>
<td>3</td>
</tr>
<tr>
<td>10.0.20.*</td>
<td>8.8.7.*</td>
<td>0-65535</td>
<td>443</td>
<td>Port 2</td>
<td>2</td>
</tr>
<tr>
<td>10.0.<em>.</em></td>
<td>8.8.7.1</td>
<td>0-65535</td>
<td>0-65535</td>
<td>Drop</td>
<td>1</td>
</tr>
</tbody>
</table>

### Outgoing Packets

```
0100 1001 10...
0100 1001 10...
0100 1001 10...
```

```
0100 1001 10...
0100 1001 10...
0100 1001 10...
```

```
0100 1001 10...
0100 1001 10...
0100 1001 10...
```

```
0100 1001 10...
0100 1001 10...
0100 1001 10...
```

```
0100 1001 10...
0100 1001 10...
0100 1001 10...
```

```
0100 1001 10...
0100 1001 10...
0100 1001 10...
```

```
0100 1001 10...
0100 1001 10...
0100 1001 10...
```

```
0100 1001 10...
0100 1001 10...
0100 1001 10...
```

```
0100 1001 10...
0100 1001 10...
0100 1001 10...
```

```
0100 1001 10...
0100 1001 10...
0100 1001 10...
```

```
0100 1001 10...
0100 1001 10...
0100 1001 10...
```

```
0100 1001 10...
0100 1001 10...
0100 1001 10...
```

```
0100 1001 10...
0100 1001 10...
0100 1001 10...
```

```
0100 1001 10...
0100 1001 10...
0100 1001 10...
```

```
0100 1001 10...
0100 1001 10...
0100 1001 10...
```

```
0100 1001 10...
0100 1001 10...
0100 1001 10...
```

```
0100 1001 10...
0100 1001 10...
0100 1001 10...
```

```
0100 1001 10...
0100 1001 10...
0100 1001 10...
```

```
0100 1001 10...
0100 1001 10...
0100 1001 10...
```
Hardware vs. Software

Ternary Content Addressable Memory (TCAM)

Various Algorithms

High Throughput Dedicated ASICs

Low Throughput Commodity Hardware
Hardware vs. Software

Hardware
- High Throughput
- Dedicated ASICs
- Ternary Content Addressable Memory (TCAM)

Software
- Low Throughput
- Commodity Hardware
- Various Algorithms
As we *add more rules* to virtual switches...
The Problem (2/2)

Let’s add rules... Bottleneck!

Virtual Switch

...we lower their throughput!
The Problem (2/2)

Let’s add rules... Bottleneck!

Large Rule Sets Spill Out of CPU Core Cache!

Virtual Switch

...we lower their throughput!
Large Rule Sets Spill Out of CPU Cache (1/2)

Large Rule Sets Spill Out of CPU Cache (2/2)

L3 cache is 10x slower than L1 cache

TupleMerge:
State-of-the-art

Observations

1. The rule space is sparse

2. L3 Cache and DRAM access time is the major bottleneck

3. Hardware trends towards commodity accelerators
Observations

1. The rule space is sparse
2. L3 Cache and DRAM accesses are the major bottleneck
3. Hardware trends towards commodity accelerators

Let’s trade memory accesses for computations
Recursive Model Index (RMI) (1/5)

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>56</td>
<td>value0</td>
</tr>
<tr>
<td>60</td>
<td>value1</td>
</tr>
<tr>
<td>68</td>
<td>value2</td>
</tr>
<tr>
<td>71</td>
<td>value3</td>
</tr>
<tr>
<td>80</td>
<td>value4</td>
</tr>
</tbody>
</table>

(*) Kraska et al., "The Case for Learned Index Structures". SIGMOD 2018.
Recursive Model Index (RMI) (1/5)

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>56</td>
<td>value0</td>
</tr>
<tr>
<td>60</td>
<td>value1</td>
</tr>
<tr>
<td>68</td>
<td>value2</td>
</tr>
<tr>
<td>71</td>
<td>value3</td>
</tr>
<tr>
<td>80</td>
<td>value4</td>
</tr>
</tbody>
</table>

0.15 MB
Neural Network Inference
98ns per lookup

13 MB
Tree Traversal
256ns per lookup

2.7x lookup performance in databases

(*) Kraska et al., "The Case for Learned Index Structures". SIGMOD 2018.
## Recursive Model Index (RMI) (2/5)

<table>
<thead>
<tr>
<th>Mem Offset</th>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>56</td>
<td>value0</td>
</tr>
<tr>
<td>101</td>
<td>60</td>
<td>value1</td>
</tr>
<tr>
<td>102</td>
<td>68</td>
<td>value2</td>
</tr>
<tr>
<td>103</td>
<td>71</td>
<td>value3</td>
</tr>
<tr>
<td>104</td>
<td>80</td>
<td>value4</td>
</tr>
<tr>
<td>105</td>
<td>87</td>
<td>value5</td>
</tr>
<tr>
<td>106</td>
<td>93</td>
<td>value6</td>
</tr>
<tr>
<td>107</td>
<td>100</td>
<td>value7</td>
</tr>
<tr>
<td>108</td>
<td>101</td>
<td>value8</td>
</tr>
<tr>
<td>109</td>
<td>117</td>
<td>value9</td>
</tr>
</tbody>
</table>

\[ IndexedKeys = \{ \text{value0, value1, value2, value3, value4, value5, value6, value7, value8, value9} \} \]

(*) Kraska et al., "The Case for Learned Index Structures". SIGMOD 2018.
Recursive Model Index (RMI) (3/5)

\[ x \rightarrow \text{Model} \rightarrow \hat{f}(x) \]

Max Approximation Error

\[ |f(x) - \hat{f}(x)| < \epsilon \]

\( x \in \text{Keys} \)

Exhaustive Scan Over All Keys

(*) Kraska et al., "The Case for Learned Index Structures". SIGMOD 2018.
Recursive Model Index (RMI) (4/5)

\[ x \xrightarrow{\text{Model}} \hat{f}(x) \]

Stage 0 \hspace{2cm} Stage 1 \hspace{2cm} Stage 2

- **Stage 0**: 
  - \( x \) to submodel
  - \( \hat{f}(x) \)

- **Stage 1**: 
  - \( x \) to submodel
  - \( x \) to submodel
  - \( \hat{f}(x) \)

- **Stage 2**: 
  - \( x \) to submodel
  - \( x \) to submodel
  - \( \hat{f}(x) \)

**Configurable:**
1. Neural Network
2. B-Tree

\(*)\) Kraska et al., "The Case for Learned Index Structures". SIGMOD 2018.
Recursive Model Index (RMI) (5/5)

 Memor Offset | Key | Value  
-------------|-----|--------
     101      |  60 | value1 
     102      |  68 | value2 
     103      |  71 | value3 
     104      |  80 | value4 
     105      |  87 | value5 
     106      |  93 | value6 
     107      | 100 | value7 

(*) Kraska et al., "The Case for Learned Index Structures". SIGMOD 2018.
Recursive Model Index (RMI) (5/5)

\[ f(x) = 102 \]

\[ x = 68 \rightarrow \text{Model} \]

\[ \hat{f}(x) = 104 \]

Error \[ \epsilon = 2 \]

<table>
<thead>
<tr>
<th>Mem Offset</th>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>60</td>
<td>value1</td>
</tr>
<tr>
<td>102</td>
<td>68</td>
<td>value2</td>
</tr>
<tr>
<td>103</td>
<td>71</td>
<td>value3</td>
</tr>
<tr>
<td>104</td>
<td>80</td>
<td>value4</td>
</tr>
<tr>
<td>105</td>
<td>87</td>
<td>value5</td>
</tr>
<tr>
<td>106</td>
<td>93</td>
<td>value6</td>
</tr>
<tr>
<td>107</td>
<td>100</td>
<td>value7</td>
</tr>
</tbody>
</table>

(*) Kraska et al., "The Case for Learned Index Structures". SIGMOD 2018.
Recursive Model Index (RMI) (5/5)

\[ f(x) = 102 \]

\[ x = 68 \rightarrow \text{Model} \]

\[ \epsilon = 2 \]

\[ f(x) = 104 \]

<table>
<thead>
<tr>
<th>Mem Offset</th>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>60</td>
<td>value1</td>
</tr>
<tr>
<td>102</td>
<td>68</td>
<td>value2</td>
</tr>
<tr>
<td>103</td>
<td>71</td>
<td>value3</td>
</tr>
<tr>
<td>104</td>
<td>80</td>
<td>value4</td>
</tr>
<tr>
<td>105</td>
<td>87</td>
<td>value5</td>
</tr>
<tr>
<td>106</td>
<td>93</td>
<td>value6</td>
</tr>
<tr>
<td>107</td>
<td>100</td>
<td>value7</td>
</tr>
</tbody>
</table>

(*) Kraska et al., "The Case for Learned Index Structures". SIGMOD 2018.
RMI is Not Enough for Packet Classification!

<table>
<thead>
<tr>
<th>Mem Offset</th>
<th>dst-ip</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>127.0.0.1</td>
<td>value0</td>
</tr>
<tr>
<td>1</td>
<td>127.0.0.*</td>
<td>value1</td>
</tr>
<tr>
<td>2</td>
<td>8.8.<em>.</em></td>
<td>value2</td>
</tr>
<tr>
<td>100</td>
<td>3.<em>.</em>.*</td>
<td>value3</td>
</tr>
</tbody>
</table>

RMI does not support wildcards
RMI is Not Enough for Packet Classification!

<table>
<thead>
<tr>
<th>Mem Offset</th>
<th>dst-ip</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>127.0.0.1</td>
<td>value0</td>
</tr>
<tr>
<td>1</td>
<td>127.0.0.*</td>
<td>value1</td>
</tr>
<tr>
<td>2</td>
<td>8.8.<em>.</em></td>
<td>value2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>3.<em>.</em>.*</td>
<td>value3</td>
</tr>
</tbody>
</table>

RMI does not support wildcards

\[ f(x) = \text{rep.} \]
RMI is Not Enough for Packet Classification!

<table>
<thead>
<tr>
<th>Mem Offset</th>
<th>dst-ip</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>127.0.0.1</td>
<td>value0</td>
</tr>
<tr>
<td>1</td>
<td>127.0.0.*</td>
<td>value1</td>
</tr>
<tr>
<td>2</td>
<td>8.8.<em>.</em></td>
<td>value2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>100</td>
<td>3.<em>.</em>.*</td>
<td>value3</td>
</tr>
</tbody>
</table>

RMI does not support wildcards
RMI is Not Enough for Packet Classification!

<table>
<thead>
<tr>
<th>Mem Offset</th>
<th>dst-ip</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>127.0.0.1</td>
<td>value0</td>
</tr>
<tr>
<td>1</td>
<td>127.0.0.*</td>
<td>value1</td>
</tr>
<tr>
<td>2</td>
<td>8.8.<em>.</em></td>
<td>value2</td>
</tr>
<tr>
<td>100</td>
<td>3.<em>.</em>.*</td>
<td>value3</td>
</tr>
</tbody>
</table>

256 * 256 * 256 = 16 million options!

RMI does not support wildcards
RMI is Not Enough for Packet Classification!

The Model Becomes too Large! Exponential Blowup!

RMI does not support wildcards

<table>
<thead>
<tr>
<th>Mem Offset</th>
<th>dst-ip</th>
<th>Action</th>
<th>Offset</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>127.0.0.1</td>
<td>0</td>
<td>value0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>127.0.0.*</td>
<td>1</td>
<td>value1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>8.8.<em>.</em></td>
<td>2</td>
<td>value2</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>3.<em>.</em>.*</td>
<td>3</td>
<td>value3</td>
<td></td>
</tr>
</tbody>
</table>

$256 \times 256 \times 256 = 16$ million options!
Our Solution – ...

1. Fields contain *wildcards*

2. Rules may *overlap* in one or more fields

3. Rules have *high dimensionality*
Our Solution - Range-Query RMI Models

1. Fields contain wildcards
   - Approximate ranges instead of keys

2. Rules may overlap in one or more fields

3. Rules have high dimensionality
Our Solution - Range-Query RMI Models

1. Fields contain *wildcards*  
   Approximate *ranges* instead of keys

2. Rules may *overlap* in one or more fields

3. Rules have *high-dimensionality*

Efficient rule-partitioning
Our Solution – **Range-Query RMI Models**

1. Fields contain **wildcards**  
   Approximate **ranges** instead of keys

2. Rules may **overlap** in one or more fields

3. Rules have **high-dimensionality**

   **Efficient rule-partitioning**

Training and lookup are **efficient and fast**
Handling Wildcards: Range-Query RMI (1/4)

<table>
<thead>
<tr>
<th>Range</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-70</td>
<td>600</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
</tr>
<tr>
<td>80-104</td>
<td>125</td>
</tr>
</tbody>
</table>
Handling Wildcards: **Range-Query RMI (2/4)**

\[ x \rightarrow \text{RQ-RMI model} \rightarrow \hat{f}(x) \]

\[ M_{i,j}(x) \]

Regression using a shallow neural network
Handling Wildcards: **Range-Query RMI (3/4)**

A bounded approximation error is guaranteed for the entire input domain!

\[ |f(x) - \hat{f}(x)| < \epsilon \]

Neural networks with ReLU activations are piecewise linear functions.
Handling Wildcards: Range-Query RMI (4/4)

1. Enables **wildcard matching**!

2. **Effective training** technique!

3. **Correctness** guarantees!

See paper for details

See paper for proofs
Handling Overlaps + Dimensionality: iSets (1/5)
Handling Overlaps + Dimensionality: iSets (1/5)

Ruleset geometrical representation

See paper for details
Handling Overlaps + Dimensionality: iSets (1/5)

Ruleset geometrical representation

See paper for details
Handling Overlaps + Dimensionality: iSets (1/5)

iSet 1 - rules do no overlap on dst-ip
{4 5 6 3}

iSet 2 - rules do no overlap on src-ip
{1 2 7}

See paper for details
Handling Overlaps + Dimensionality: iSets (1/5)

Ruleset geometrical representation

- iSet 1 - rules do no overlap on dst-ip
  \{ 4 \, 5 \, 6 \, 3 \}  
- iSet 2 - rules do no overlap on src-ip
  \{ 1 \, 2 \, 7 \}  
- Remainder - rules that do not fit in any iSet
  \{ 8 \}  

See paper for details
Handling Overlaps + Dimensionality: iSets (2/5)

RQ-RMI model

<table>
<thead>
<tr>
<th>dst-IP</th>
<th>Index within iSet</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-60</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>80-670</td>
<td>4</td>
</tr>
</tbody>
</table>

Each iSet can be approximated using an RQ-RMI model

iSet 1 - rules do no overlap on dst-ip

\{ 4, 5, 6, 3 \}

iSet 2 - rules do no overlap on src-ip

\{ 1, 2, 7 \}

Remainder - rules that do not fit in any iSet

\{ 8 \}
Handling Overlaps + Dimensionality: iSets (3/5)

The remainder set is handled by any classification method.

iSet 1 - rules do no overlap on dst-ip
{4 5 6 3}

iSet 2 - rules do no overlap on src-ip
{1 2 7}

Remainder - rules that do not fit in any iSet
{8}
RQ-RMI solves the problem for a single field. What about all other fields?

**iSet 1** - rules do no overlap on **dst-ip**

\[ \{ 4 \ 5 \ 6 \ 3 \} \]

**iSet 2** - rules do no overlap on **src-ip**

\[ \{ 1 \ 2 \ 7 \} \]

**Remainder** - rules that do not fit in any iSet

\[ \{ 8 \} \]
Handling Overlaps + Dimensionality: iSets (4/5)

0100
1001
10...

RQ-RMI

Secondary Search

Validation

Match / No Match

Neural Network Inference on field f

Resolve approximation errors for field f

Validate all other fields

Matching Candidate
Handling Overlaps + Dimensionality: iSets (5/5)
Handling Overlaps + Dimensionality: iSets (5/5)
The iSet Tradeoff (1/2)

iSet coverage is high for large rulesets!

<table>
<thead>
<tr>
<th>Ruleset Size</th>
<th>1 iSet</th>
<th>2 iSets</th>
<th>3 iSets</th>
<th>4 iSets</th>
</tr>
</thead>
<tbody>
<tr>
<td>1K</td>
<td>20.2 ± 18.6</td>
<td>28.9 ± 22.3</td>
<td>34.6 ± 25.6</td>
<td>38.7 ± 27.2</td>
</tr>
<tr>
<td>10K</td>
<td>45.1 ± 31.6</td>
<td>59.6 ± 38.9</td>
<td>62.6 ± 37.1</td>
<td>65.1 ± 35.7</td>
</tr>
<tr>
<td>100K</td>
<td>80.0 ± 14.5</td>
<td>96.5 ± 8.3</td>
<td>98.1 ± 4.8</td>
<td>98.8 ± 2.7</td>
</tr>
<tr>
<td>500K</td>
<td>84.2 ± 10.5</td>
<td>98.8 ± 1.5</td>
<td>99.4 ± 0.6</td>
<td>99.7 ± 0.2</td>
</tr>
<tr>
<td>183,376</td>
<td>57.8</td>
<td>91.6</td>
<td>96.5</td>
<td>98.2</td>
</tr>
</tbody>
</table>
# The iSet Tradeoff (1/2)

iSet coverage is high for large rulesets!

<table>
<thead>
<tr>
<th>Ruleset Size</th>
<th>1 iSet</th>
<th>2 iSets</th>
<th>3 iSets</th>
<th>4 iSets</th>
</tr>
</thead>
<tbody>
<tr>
<td>1K</td>
<td>20.2 ± 18.6</td>
<td>28.9 ± 22.3</td>
<td>34.6 ± 25.6</td>
<td>38.7 ± 27.2</td>
</tr>
<tr>
<td>10K</td>
<td>45.1 ± 31.6</td>
<td>59.6 ± 38.9</td>
<td>62.6 ± 37.1</td>
<td>65.1 ± 35.7</td>
</tr>
<tr>
<td>100K</td>
<td>80.0 ± 14.5</td>
<td><strong>96.5 ± 8.3</strong></td>
<td>98.1 ± 4.8</td>
<td>98.8 ± 2.7</td>
</tr>
<tr>
<td>500K</td>
<td>84.2 ± 10.5</td>
<td>98.8 ± 1.5</td>
<td>99.4 ± 0.6</td>
<td>99.7 ± 0.2</td>
</tr>
<tr>
<td>183,376</td>
<td>57.8</td>
<td>91.6</td>
<td>96.5</td>
<td>98.2</td>
</tr>
</tbody>
</table>
The iSet Tradeoff (2/2)
NuevoMatch: Putting it All Together

<table>
<thead>
<tr>
<th>Src IP</th>
<th>Dst IP</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.10.*</td>
<td>8.8.**</td>
<td>0-80</td>
</tr>
<tr>
<td>10.0.20.*</td>
<td>8.8.7.*</td>
<td>443</td>
</tr>
<tr>
<td>10.0.*</td>
<td>8.8.7.1</td>
<td>*</td>
</tr>
<tr>
<td>55.0.*</td>
<td>3.3.**</td>
<td>22</td>
</tr>
</tbody>
</table>

**iSet partitioning algorithm**

- iSet 1
- iSet 2
- iSet N
- Remainder
- RQ-RMI 1
- RQ-RMI 2
- RQ-RMI N
- External

**Selector**

All Fit in Core Cache
Can We Fit in L2 Cache? (1/2)

Li et al., "CutSplit: A Decision Tree Combining Cutting and Splitting for Scalable Packet Classification". INFOCOM 2018.
Liang et al., "Neural Packet Classification". SIGCOMM 2019.
Can We Fit in L2 Cache? (2/2)

Geomean compression factor of 4.9x, 8x, 82x

State of the art techniques

Li et al., "CutSplit: A Decision Tree Combining Cutting and Splitting for Scalable Packet Classification". INFOCOM 2018.
Liang et al., "Neural Packet Classification". SIGCOMM 2019.
Isn’t Neural Network Inference Slow?

Not if you...

Use **Small and Shallow** Neural Networks

Use **Wide Vector** Instructions

40-50 ns on a CPU!
What About Updates? (1/2)

We easily can remove rules from the models. New rules get into the remainder. We retrain the models after a certain threshold in the number of new rules.
Training with TensorFlow takes between 10-40 min on a CPU
TensorFlow has many overheads

In our ongoing work we currently reach 1-20 seconds on a CPU
using native implementation
Evaluation

End-to-end performance
- Single-core vs. multi-core settings
- Small vs. large rule sets
- Traffic with uniform / skewed temporal locality
- Memory footprint comparison
- Performance under L3 cache contention
- Real-world forwarding rules

Performance analysis
- iSet Coverage
- Impact of the number of iSets
- Partitioning effectiveness
- Training time and search bounds
- Performance with many fields
Evaluation

End-to-end performance

- **Single-core** vs. multi-core settings
- Small vs. large rule sets
- Traffic with uniform / skewed temporal locality
- Memory footprint comparison
- Performance under L3 cache contention
- Real-world forwarding rules

Performance analysis

- iSet Coverage
- Impact of the number of iSets
- Partitioning effectiveness
- Training time and search bounds
- Performance with many fields
Evaluation – Skewed Traces

![Graph showing throughput speedup for different Zipf distributions with varying skewness. The graph compares NuevoMatch w/ CutSplit and NuevoMatch w/ TupleMerge.]

(*) The CAIDA UCSD Anonymized Internet Traces 2019 (www.caida.org/data/passive/passive_dataset.xml).
Evaluation - Skewed Traces

Li et al., "CutSplit: A Decision Tree Combining Cutting and Splitting for Scalable Packet Classification". INFOCOM 2018.
The CAIDA UCSD Anonymized Internet Traces 2019 (www.caida.org/data/passive/passive_dataset.xml).

Real-world temporal locality

Throughput Speedup

2.5
2
1.5
1
0.5
0
0.5
1
1.05
1.79
1.62
0.89
0.99
1.06
1.95
2.06

Zipf 80% (α=1.05) Zipf 85% (α=1.10) Zipf 90% (α=1.15) Zipf 95% (α=1.25) CAIDA

NuevoMatch w/ CutSplit
NuevoMatch w/ TupleMerge
Evaluation - Skewed Traces

Li et al., "CutSplit: A Decision Tree Combining Cutting and Splitting for Scalable Packet Classification". INFOCOM 2018.
The CAIDA UCSD Anonymized Internet Traces 2019 (www.caida.org/data/passive/passive_dataset.xml).
Evaluation - Uniform Access

Geomean throughput speedup of 2.4x, 2.6x, 1.6x

500K Classifiers

Throughput Speedup

NuevoMatch w/ CutSplit  NuevoMatch w/ NeuroCuts  NuevoMatch w/ TupleMerge

---

(*) Li et al., "CutSplit: A Decision Tree Combining Cutting and Splitting for Scalable Packet Classification". INFOCOM 2018.
(*) Liang et al., "Neural Packet Classification". SIGCOMM 2019.
Evaluation – Real-World Rule Sets


Geomean throughput speedup of 3.5x

Throughput (Mpps)

<table>
<thead>
<tr>
<th>Ruleset</th>
<th>TupleMerge</th>
<th>NuevoMatch w/ TupleMerge</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.51x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3.49x</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3.40x</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3.56x</td>
<td></td>
</tr>
</tbody>
</table>
Conclusions

1. **NuevoMatch**: a new point in the design space of packet classification
2. **RQ-RMI** for range-value queries
3. Error bound guarantees using piecewise-linear functions

Thank You

See More On https://acsl.group/publications/

Ori Rottenstreich or@technion.ac.il
Mark Silberstein mark@ee.technion.ac.il

Alon Rashelbach alonrs@campus.technion.ac.il
Extra Slides
Decision Tree Approaches (1/3)

<table>
<thead>
<tr>
<th>Src IP</th>
<th>Dst IP</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.10.*</td>
<td>8.8.<em>.</em></td>
<td>Port 1</td>
</tr>
<tr>
<td>10.0.20.*</td>
<td>8.8.7.*</td>
<td>Port 2</td>
</tr>
<tr>
<td>10.0.<em>.</em></td>
<td>8.8.7.1</td>
<td>Drop</td>
</tr>
<tr>
<td>55.0.0.*</td>
<td>3.3.<em>.</em></td>
<td>Port 1</td>
</tr>
</tbody>
</table>

Ruleset geometrical representation

## Decision Tree Approaches (2/3)

<table>
<thead>
<tr>
<th>Src IP</th>
<th>Dst IP</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.10.*</td>
<td>8.8.<em>.</em></td>
<td>Port 1</td>
</tr>
<tr>
<td>10.0.20.*</td>
<td>8.8.7.*</td>
<td>Port 2</td>
</tr>
<tr>
<td>10.0.<em>.</em></td>
<td>8.8.7.1</td>
<td>Drop</td>
</tr>
<tr>
<td>55.0.0.*</td>
<td>3.3.<em>.</em></td>
<td>Port 1</td>
</tr>
</tbody>
</table>

### Ruleset geometrical representation

Decision Tree Approaches (3/3)

<table>
<thead>
<tr>
<th>Src IP</th>
<th>Dst IP</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.10.*</td>
<td>8.8.<em>.</em></td>
<td>Port 1</td>
</tr>
<tr>
<td>10.0.20.*</td>
<td>8.8.7.*</td>
<td>Port 2</td>
</tr>
<tr>
<td>10.0.<em>.</em></td>
<td>8.8.7.1</td>
<td>Drop</td>
</tr>
<tr>
<td>55.0.0.*</td>
<td>3.3.<em>.</em></td>
<td>Port 1</td>
</tr>
</tbody>
</table>

Ruleset geometrical representation

Decision Tree Approaches (3/3)

Rule replication causes memory growth!
### Hash-Table Approaches (1/3)

<table>
<thead>
<tr>
<th>Src IP</th>
<th>Dst IP</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.10.*</td>
<td>8.8.<em>.</em></td>
<td>Port 1</td>
</tr>
<tr>
<td>10.0.20.*</td>
<td>8.8.7.*</td>
<td>Port 2</td>
</tr>
<tr>
<td>10.0.<em>.</em></td>
<td>8.8.7.1</td>
<td>Drop</td>
</tr>
<tr>
<td>55.0.0.*</td>
<td>3.3.<em>.</em></td>
<td>Port 1</td>
</tr>
</tbody>
</table>

Hash-Table Approaches (2/3)

<table>
<thead>
<tr>
<th>Src IP</th>
<th>Dst IP</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.10.*</td>
<td>8.8.**</td>
<td>Port 1</td>
</tr>
<tr>
<td>10.0.20.*</td>
<td>8.8.7.*</td>
<td>Port 2</td>
</tr>
<tr>
<td>10.0.*</td>
<td>8.8.7.1</td>
<td>Drop</td>
</tr>
<tr>
<td>55.0.0.*</td>
<td>3.3.*</td>
<td>Port 1</td>
</tr>
</tbody>
</table>

Hash Table (3,2)

<table>
<thead>
<tr>
<th>Hash Table (3,2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.10–8.8</td>
</tr>
<tr>
<td>10.0.20–8.8</td>
</tr>
<tr>
<td>55.0.0–3.3</td>
</tr>
</tbody>
</table>

Validation

<table>
<thead>
<tr>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>–</em>*</td>
</tr>
<tr>
<td><em>–7.</em></td>
</tr>
<tr>
<td><em>–</em>*</td>
</tr>
</tbody>
</table>

Hash-Table Approaches (3/3)

\[ H_0(x) \]

\[ H_1(x) \]

src-ip 10.0.20.1

dst-ip 8.8.7.2

<table>
<thead>
<tr>
<th>Hash Table (3,2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

1. 10.0.10-8.8
2. 10.0.20-8.8
3. 10.0-8.8.7.1
4. 55.0.0-3.3

<table>
<thead>
<tr>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>_-**.*</td>
</tr>
</tbody>
</table>

**Found**

<table>
<thead>
<tr>
<th>Hash Table (2,4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

1. 10.0-8.8.7.1

**Not Found!**

<table>
<thead>
<tr>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>_-7.*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>_-**.*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>*<em>.</em></td>
</tr>
</tbody>
</table>

---

References:

Hash-Table Approaches (3/3)

Several hash tables; Spill out of Core cache!

Validation

\*\*\*
\*7\*
\*\*\*

Validation

\*\*

\[ H_0(x) \]

\[ H_1(x) \]

Not Found!

\begin{align*}
\text{src-ip} & \quad \text{10.0.20.1} \\
\text{dst-ip} & \quad \text{8.8.7.2}
\end{align*}

\begin{align*}
\text{Hash Table (2,4)} & \quad \begin{array}{c}
10.0-8.8.7.1 \\
55.0.0-3.3
\end{array}
\end{align*}
Recursive Model Index (RMI)

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>56</td>
<td>600</td>
</tr>
<tr>
<td>104</td>
<td>125</td>
</tr>
</tbody>
</table>

(*) Kraska et al., "The Case for Learned Index Structures". SIGMOD 2018.
Handling Wildcards: **Range-Query RMI (4/4)**

- **Case:**
  - $f(x) = 102$
  - $x = 807$
  - Error $\epsilon = 1$
  - Approximation $\hat{f}(x) = 104$

<table>
<thead>
<tr>
<th>Mem Offset</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>98</td>
<td>0-100</td>
</tr>
<tr>
<td>100</td>
<td>130-700</td>
</tr>
<tr>
<td>102</td>
<td>800-801</td>
</tr>
<tr>
<td>104</td>
<td>807-890</td>
</tr>
<tr>
<td>106</td>
<td>970-990</td>
</tr>
<tr>
<td>108</td>
<td>991-1000</td>
</tr>
</tbody>
</table>
Introducing **NuevoMatch** (1/4)

### Rule Set

<table>
<thead>
<tr>
<th>Src IP</th>
<th>Dst IP</th>
<th>Src Port</th>
<th>Dst Port</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.10.*</td>
<td>8.8.**</td>
<td>0–65535</td>
<td>80</td>
<td>Port 1</td>
</tr>
<tr>
<td>10.0.**</td>
<td>8.8.7.1</td>
<td>0–65535</td>
<td>0–6</td>
<td>Drop</td>
</tr>
</tbody>
</table>

**Approximate Ruleset Using Machine Learning Techniques**

**Always Fits in Cache**

**RQ-RMI Models**
Introducing **NuevoMatch** (2/4)

### Rule Set

<table>
<thead>
<tr>
<th>Src IP</th>
<th>Dst IP</th>
<th>Src Port</th>
<th>Dst Port</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.10.*</td>
<td>8.8.**</td>
<td>0-65535</td>
<td>80</td>
<td>Port 1</td>
</tr>
<tr>
<td>10.0.**</td>
<td>8.8.7.1</td>
<td>0-65535</td>
<td>0-6</td>
<td>Drop</td>
</tr>
</tbody>
</table>

### Approximate Ruleset

Using Machine Learning Techniques

Always Fits in Cache

### RQ-RMI Models

![Diagram showing RQ-RMI model with neural networks](image.png)
Introducing NuevoMatch (3/4)

Rule Set

<table>
<thead>
<tr>
<th>Src IP</th>
<th>Dst IP</th>
<th>Src Port</th>
<th>Dst Port</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.10.*</td>
<td>8.8.**</td>
<td>0-65535</td>
<td>80</td>
<td>Port 1</td>
</tr>
<tr>
<td>10.0.**</td>
<td>8.8.7.1</td>
<td>0-65535</td>
<td>0-6</td>
<td>Drop</td>
</tr>
</tbody>
</table>

Some part cannot be approximated, so we keep it as a remainder

Always Fits in Cache

RQ-RMI Models

Remainder
Introducing **NuevoMatch** (4/4)

Always Fits in Cache

Perform classification on CPU using Neural Network Inference

RQ-RMI Models

Remainder
RQ-RMI

Neural Network

\( \hat{f}(x) \)