CATE: An Open and Highly Configurable Framework for Performance Evaluation of Packet Classification Algorithms

Wladislaw Gusew, Sven Hager, Björn Scheuermann

Computer Engineering Group, Humboldt University of Berlin
Network packet classification is a crucial task performed in several network applications, such as

- Firewalls
- Policy routing
- Traffic measurement and accounting
- Traffic rate limiting
- Intrusion Detection Systems
INTRODUCTION

Network packet classification is a crucial task performed in several network applications, such as

- **Firewalls**
- Policy routing
- Traffic measurement and accounting
- Traffic rate limiting
- Intrusion Detection Systems
PACKET FILTERING IN FIREWALLS

LAN

Secure

Firewall

Internet

Not Trusted
Packet Filtering in Firewalls
 PACKET FILTERING IN FIREWALLS

Rule Set:
- Rule 1: from 12.0.0.2 to 7.7.0.42 Allow
- Rule 2: from 3.0.0.4 to 7.7.0.1 Drop
- Rule 3: from 12.0.0.2 to 7.7.0.10 Allow

LAN
- IP: 7.7.0.10
- IP: 7.7.0.42

Secure
- IP: 7.7.0.1

Firewall

Internet
- IP: 3.0.0.4
- IP: 12.0.0.2

Not Trusted
Packet Filtering in Firewalls

Rule Set:

Rule 1: from 12.0.0.2 to 7.7.0.42 Allow
Rule 2: from 3.0.0.4 to 7.7.0.1 Drop
Rule 3: from 12.0.0.2 to 7.7.0.10 Allow

LAN
- IP: 7.7.0.10
- IP: 7.7.0.42
- IP: 7.7.0.1
Secure

Firewall

Internet
- IP: 3.0.0.4
- IP: 12.0.0.2

Packet
- IP-src: 12.0.0.2
- IP-dst: 7.7.0.10

Not Trusted
**Packet Filtering in Firewalls**

**LAN**
- IP: 7.7.0.1
- Secure

**Firewall**
- Rule Set
  - Rule 1: from 12.0.0.2 to 7.7.0.42 Allow
  - Rule 2: from 3.0.0.4 to 7.7.0.1 Drop
  - Rule 3: from 12.0.0.2 to 7.7.0.10 Allow

**Internet**
- IP: 3.0.0.4
- IP: 12.0.0.2

**Not Trusted**
- IP: 7.7.0.42

**Secure**
- IP: 7.7.0.10
 PACKET FILTERING IN FIREWALLS

Rule Set:
- Rule 1: from 12.0.0.2 to 7.7.0.42 Allow
- Rule 2: from 3.0.0.4 to 7.7.0.1 Drop
- Rule 3: from 12.0.0.2 to 7.7.0.10 Allow

LAN:
- IP: 7.7.0.1
- IP: 7.7.0.42
- IP: 7.7.0.10

Secure:
- IP: 7.7.0.10

Firewall

Internet:
- IP: 3.0.0.4
- IP: 12.0.0.2

Packet:
- IP-src: 3.0.0.4
- IP-dst: 7.7.0.1

Not Trusted:
Packet Filtering in Firewalls

Rule Set
- Rule 1: from 12.0.0.2 to 7.7.0.42 Allow
- Rule 2: from 3.0.0.4 to 7.7.0.1 Drop
- Rule 3: from 12.0.0.2 to 7.7.0.10 Allow

LAN
- IP: 7.7.0.10
- IP: 7.7.0.42
- IP: 7.7.0.2

Secure
- IP: 7.7.0.1

Internet
- IP: 3.0.0.4
- IP: 12.0.0.2

Not Trusted

Firewall

Drop
Packet Classification

For each incoming packet find the highest priority matching rule in a rule set and apply its associated action.
CHALLENGING REQUIREMENTS

► Classifying at line speed:
  ▶ For 40 Gbit/s: 125 millions IP-packets per second (with 40 Bytes each)
  ▶ Time to classify and forward each packet is only 8 ns
  ▶ Main memory access time is 5 to 60 ns
CHALLENGING REQUIREMENTS

▶ Classifying at line speed:
  ▶ For 40 Gbit/s: 125 millions IP-packets per second (with 40 Bytes each)
  ▶ Time to classify and forward each packet is only 8 ns
  ▶ Main memory access time is 5 to 60 ns
▶ Operate at the leading edge of today’s hardware technology
▶ Efficient packet classification is essential.
CLASSIFICATION ALGORITHMS

CLASSIFICATION ALGORITHMS

CLASSIFICATION ALGORITHMS

CLASSIFICATION ALGORITHMS

CLASSIFICATION ALGORITHMS

PROBLEM STATEMENT

- Plenty of classification algorithms with different properties
- Differences in classification performance and memory footprint
- Comparisons problematic, as no integrative framework exists
PROBLEM STATEMENT

▸ Plenty of classification algorithms with different properties
▸ Differences in classification performance and memory footprint
▸ Comparisons problematic, as no integrative framework exists

Main Questions

▸ For developers: Which classification algorithm is best suited for a certain application?
PROBLEM STATEMENT

- Plenty of classification algorithms with different properties
- Differences in classification performance and memory footprint
- Comparisons problematic, as no integrative framework exists

Main Questions

- **For developers**: Which classification algorithm is best suited for a certain application?
- **For researchers**: How to compare a new algorithm to already existing classification algorithms?
Classification
Algorithm
Testing
Environment
CLASSIFICATION ALGORITHM TESTING ENVIRONMENT

- Framework for benchmarking algorithms’ software implementations
- Measured values: time durations and memory usage
- Implemented in C++, runnable on many targets
- Highly customizable with Lua as configuration language
- Uses white box classification algorithms
SOFTWARE ARCHITECTURE
MEASURING PERFORMANCE

- Take time duration(s)
- Compare benchmarks: equal workload – different time durations
- Important aspects: control and granularity
MEASURING PERFORMANCE

:BenchmarkExecutor

:SomeAlgorithm

:ChronoManager

setRules(classifier)

start('convert')

stop('convert')

void

convert
MEASURING PERFORMANCE

:BenchmarkExecutor

:SomeAlgorithm

:ChronoManager

setRules(classifier)

start('convert')

start('classify')

stop('classify')

stop('convert')

classify(headers)

void

return indices

convert

classify
MEASURING PERFORMANCE

:BenchmarkExecutor

void
classify(headers)

setRules(classifier)

loop

search(header)

start('search')

stop('search')

return indices

:SomeAlgorithm

start('convert')

stop('convert')

:ChronoManager

start('classify')

stop('classify')

Conclusions
MEASURING MEMORY USAGE

- Goal: Precisely trace memory allocations and memory accesses (byte granularity)
MEASURING MEMORY USAGE

- Goal: Precisely trace memory allocations and memory accesses (byte granularity)
- High-level programming languages provide an abstraction of memory management
- In C++, memory information can be acquired implicitly by the `sizeof` operator
MEASURING MEMORY USAGE

▶ Realized by code-instrumentation
▶ Utilization of class inheritance and C++ template techniques
MEASURING MEMORY USAGE

- Realized by code-instrumentation
- Utilization of class inheritance and C++ template techniques
- Arbitrary data types can be regarded, e. g.
  - `int val;`  $\Rightarrow$  `MemTrace<int> val;`
  - `Stack st;`  $\Rightarrow$  `MemTrace<Stack> st;`
MEASURING MEMORY USAGE

- Realized by code-instrumentation
- Utilization of class inheritance and C++ template techniques
- Arbitrary data types can be regarded, e.g.
  - int val; ⇒ MemTrace<int> val;
  - Stack st; ⇒ MemTrace<Stack> st;
- Tracing memory usage imposes overhead on measured time durations
Evaluation
EVALUATION HIGHLIGHTS

1. Analyze probe effect (overhead by memory tracing)
2. Compare algorithms’ performance for varying rules
3. Effect of different rule set structures on classification performance
4. Vary field structure of packet headers
SYSTEM SETUP

- 2.27 GHz Intel i5 M430, single core, Linux 3.13
- Virtual Machine (32 bit) with 1024 MB RAM
- Synthetic ClassBench rules and 100K headers
- Measurements of time durations were repeated 8 times and arithmetic mean was taken.
ANALYZING PROBE EFFECT

- Measure time durations in three operation modes:
  - enabled: memory allocations and accesses are traced during time measurements
  - disabled: memory usage tracing is disabled by preprocessor directive
  - cleaned: code-instrumentations are removed manually from code
ANALYZING PROBE EFFECT

![Bar Chart]

- Lin. Search
- Bit Vector
- HiCuts
- Tuple Space

Classification Algorithm

Mean Classification Duration/Header [µs] vs. Classification Algorithm

- On
- Off
- Clean

Error Bars Represent Variability in Data
ANALYZING PROBE EFFECT

![Chart showing classification duration relative to linear search for different memory metering activation states.]

- **Linear Search:** 100% on, 100% off, 100% clean
- **Bit Vector:** 28% on, 27% off, 29% clean
- **HiCuts:** 6% on, 6% off, 7% clean
- **Tuple Space:** 45% on, 60% off, 59% clean

Classification Duration relative to Lin. Search [%]

<table>
<thead>
<tr>
<th>Memory Metering Activation State</th>
<th>Linear Search</th>
<th>Bit Vector</th>
<th>HiCuts</th>
<th>Tuple Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>on</td>
<td>100%</td>
<td>28%</td>
<td>6%</td>
<td>45%</td>
</tr>
<tr>
<td>off</td>
<td>100%</td>
<td>27%</td>
<td>6%</td>
<td>60%</td>
</tr>
<tr>
<td>clean</td>
<td>100%</td>
<td>29%</td>
<td>7%</td>
<td>59%</td>
</tr>
</tbody>
</table>
VARIATION OF RULES

- Vary number of rules (200, 400, 600, ..., 5000)
- Use common 5-tuple: (32 bit, 32 bit, 8 bit, 16 bit, 16 bit)
VARIATION OF RULES

- Vary number of rules (200, 400, 600, ..., 5000)
- Use common 5-tuple: (32 bit, 32 bit, 8 bit, 16 bit, 16 bit)
- Observe performance during preprocessing and classification stages
- Compare performance for rule sets with different structure
VARIATION OF RULES - PREPROCESSING

![Graph showing variation of rules - preprocessing](image-url)

- Linear Search
- Bit Vector
- HiCuts
- Tuple Space

Mean Preproc. Duration [µs]
Rule Set Size

Allocated Mem. [KB]
Rule Set Size
VARIATION OF RULES - CLASSIFICATION

The graphs show the mean duration per header in microseconds (µs) against rule set size for different methods:
- Linear Search
- Bit Vector
- HiCuts
- Tuple Space

The rule set size is plotted on the x-axis, ranging from 0 to 5000, and the mean duration per header is plotted on the y-axis, ranging from 0 to 40 µs.

For the fw1 and acl1 datasets, the graphs display a comparison of mean duration per header at various rule set sizes, with the methods mentioned above. The results indicate a trend where the mean duration increases as the rule set size grows, with some methods showing more significant increases than others.
VARIATION OF RULES - CLASSIFICATION

e.g., fw1_4000: 449 wildcard checks, acl1_4000: 22 wildcard checks
VARYING FIELD STRUCTURE

- Vary tuple size (2, 4, 10 fields), each field is 32 bit
- Expectations:
  - Increasing tuple size should not affect HiCuts’ classification performance
  - Allocated memory should increase with increasing tuple size for all algorithms
## Varying Field Structure

<table>
<thead>
<tr>
<th>Method</th>
<th>2-tuple</th>
<th>4-tuple</th>
<th>10-tuple</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lin. Search</td>
<td><img src="image1" alt="graph" /></td>
<td><img src="image2" alt="graph" /></td>
<td><img src="image3" alt="graph" /></td>
</tr>
<tr>
<td>Bit Vector</td>
<td><img src="image4" alt="graph" /></td>
<td><img src="image5" alt="graph" /></td>
<td><img src="image6" alt="graph" /></td>
</tr>
<tr>
<td>HiCuts</td>
<td><img src="image7" alt="graph" /></td>
<td><img src="image8" alt="graph" /></td>
<td><img src="image9" alt="graph" /></td>
</tr>
<tr>
<td>Tuple Space</td>
<td><img src="image10" alt="graph" /></td>
<td><img src="image11" alt="graph" /></td>
<td><img src="image12" alt="graph" /></td>
</tr>
</tbody>
</table>

### Classification Duration [µs]

<table>
<thead>
<tr>
<th>Method</th>
<th>2-tuple</th>
<th>4-tuple</th>
<th>10-tuple</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lin. Search</td>
<td><img src="image1" alt="graph" /></td>
<td><img src="image2" alt="graph" /></td>
<td><img src="image3" alt="graph" /></td>
</tr>
<tr>
<td>Bit Vector</td>
<td><img src="image4" alt="graph" /></td>
<td><img src="image5" alt="graph" /></td>
<td><img src="image6" alt="graph" /></td>
</tr>
<tr>
<td>HiCuts</td>
<td><img src="image7" alt="graph" /></td>
<td><img src="image8" alt="graph" /></td>
<td><img src="image9" alt="graph" /></td>
</tr>
<tr>
<td>Tuple Space</td>
<td><img src="image10" alt="graph" /></td>
<td><img src="image11" alt="graph" /></td>
<td><img src="image12" alt="graph" /></td>
</tr>
</tbody>
</table>

### Allocated Memory (log.) [KB]

<table>
<thead>
<tr>
<th>Method</th>
<th>Lin. Search</th>
<th>Bit Vector</th>
<th>HiCuts</th>
<th>Tuple Space</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><img src="image13" alt="graph" /></td>
<td><img src="image14" alt="graph" /></td>
<td><img src="image15" alt="graph" /></td>
<td><img src="image16" alt="graph" /></td>
</tr>
</tbody>
</table>
Conclusions
RESULTS AND CONCLUSIONS

- CATE provides an integrative measurement environment for practical implementations.
RESULTS AND CONCLUSIONS

- CATE provides an integrative measurement environment for practical implementations.
- Probe effect does not change time duration relations between algorithms.
- Classification algorithms can show unexpected characteristics for specific applications.
RESULTS AND CONCLUSIONS

- CATE provides an integrative measurement environment for practical implementations.
- Probe effect does not change time duration relations between algorithms.
- Classification algorithms can show unexpected characteristics for specific applications.
  - Memory allocation for HiCuts’ data structure is highly fluctuating.
  - Tuple Space Search performance depends on rule structure.
  - Increasing header fields can result in less memory allocation by HiCuts.
RESULTS AND CONCLUSIONS

- CATE provides an integrative measurement environment for practical implementations.

Open Access to CATE

Checkout CATE@github:

http://gusew.github.io/cate

- Increasing header fields can result in less memory allocation by HiCuts.
LUA CODE COMPOSITION

Helper Functions

User Configuration

Validity Checks

Composed Script

Lua Interpreter
MEMORY MEASURING REALIZATION

MemTraceRegistry

+ register(obj : RegistryItem)
  + deregister(obj : RegistryItem)

RegistryItem

+ getData() : RegistryData

register(*this)

deregister(*this)

MemTraceData

+ allocated
+ totalBytesAccessed
+ totalCountAccesses
+ writeBytesAccessed
+ writeCountAccesses

+ allocated(bytes : size_t)
+ read(bytes : size_t)
+ write(bytes : size_t)

T

This is a user defined type.
CONFIGURATION

Algorithm

Dynamic Loading (Shared Library) Parameters

Field Structure

IPv4 5-tuple (32bit, 32bit, 16bit, 16bit, 8bit) IPv6 2-tuple (128bit, 128bit) OpenFlow 9-tuple
CONFIGURATION

Algorithm
- Dynamic Loading (Shared Library)
- Parameters

Field Structure
- IPv4 5-tuple
  - (32bit, 32bit, 16bit, 16bit, 8bit)
- IPv6 2-tuple
  - (128bit, 128bit)
- OpenFlow 9-tuple
- ... (other structures)

Classifier
- Explicit Definition
CONFIGURATION

Algorithm
- Dynamic Loading (Shared Library)
- Parameters

Field Structure
- IPv4 5-tuple (32bit, 32bit, 16bit, 16bit, 8bit)
- IPv6 2-tuple (128bit, 128bit)
- OpenFlow 9-tuple

Classifier
- Explicit Definition

Header Set
- Explicit Definition
- Random Generation (Uniform, Pareto, Normal, ...
CONFIGURATION

- **Problem**: The framework has to provide batch processing capability.
- **Solution**: All information is specified on startup in a configuration file.
Problem: The framework has to provide batch processing capability.

Solution: All information is specified on startup in a configuration file.

Problem: There is a high number of configuration entities and some are inter-dependent.

Solution: A domain specific language (DSL) is provided to support the user.
CONFIGURATION

- **Problem**: The framework has to provide batch processing capability.

- **Solution**: All information is specified on startup in a configuration file.

- **Problem**: There is a high number of configuration entities and some are inter-dependent.

- **Solution**: A domain specific language (DSL) is provided to support the user.

- **Problem**: DSLs are limited to pre-defined use patterns.

- **Solution**: The DSL is embedded in a powerful Lua environment.
MINIMAL LUA CONFIGURATION (1)

```
alg = createAlgorithm("HiCuts5tpl.so", {
  100, -- checkpoint intervals
  16, -- binth value
  3.0 -- space factor value
})

strc = {32, 32, 16, 16, 8}

classifier = createRuleset()
addRuleToRuleset(classifier, {
  ruleAtomExact(ipv4Toi("10.0.0.20")),
  ruleAtomPrefix(ipv4Toi("10.10.0.1"), maskToi(16)),
  ruleAtomPrefix(0, maskToi(0)),
  ruleAtomRange(2000, 4000),
  ruleAtomExact(6)
})
```
**MINIMAL LUA CONFIGURATION (2)**

```
hdrs = createHeaderset()
addHeaderToHeaderset(hdrs, { ipv4Toi("10.0.0.19"),
    ipv4Toi("10.10.100.17"), 5994, 80, 6 })
addHeaderToHeaderset(hdrs, { ipv4Toi("10.0.0.21"),
    ipv4Toi("10.10.0.1"), 12000, 8080, 6 })
addHeaderToHeaderset(hdrs, { ipv4Toi("10.0.0.20"),
    ipv4Toi("10.10.99.12"), 443, 5000, 6 })

-- Define benchmarks for benchmark suite
registerBenchmark("HiCuts, 10 runs", alg, strc, classifier, hdrs, 10)
registerBenchmark("HiCuts, 50 runs", alg, strc, classifier, hdrs, 50)
```
VARIATION OF REPETITIONS

Mean Values with Std. Dev.
Min. Values
Max. Values
VARIATION OF REPETITIONS

Mean Values with Std. Dev.
Min. Values
Max. Values
### Bit Vector

Packet: **192.168.0.1** | **192.168.1.42** | **443** | **80** | TCP

<table>
<thead>
<tr>
<th>Source IP</th>
<th>Destination IP</th>
<th>Source Port</th>
<th>Destination Port</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.*</td>
<td>*</td>
<td>[1024:65535]</td>
<td>80</td>
<td>TCP</td>
</tr>
<tr>
<td>192.168.*</td>
<td>72.7.25.*</td>
<td>7777</td>
<td>*</td>
<td>TCP, UDP</td>
</tr>
<tr>
<td>*</td>
<td>192.168.1.42</td>
<td>*</td>
<td>80</td>
<td>TCP</td>
</tr>
<tr>
<td>*</td>
<td>192.168.*</td>
<td>*</td>
<td>666</td>
<td>*</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

Bit Vectors:
**Bit Vector**

Packet: 192.168.0.1 192.168.1.42 443 80 TCP

<table>
<thead>
<tr>
<th>Source IP</th>
<th>Destination IP</th>
<th>Source Port</th>
<th>Destination Port</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.*</td>
<td>*</td>
<td>[1024:65535]</td>
<td>80</td>
<td>TCP</td>
</tr>
<tr>
<td>192.168.*</td>
<td>72.7.25.*</td>
<td>7777</td>
<td>*</td>
<td>TCP, UDP</td>
</tr>
<tr>
<td>*</td>
<td>192.168.1.42</td>
<td>*</td>
<td>80</td>
<td>TCP</td>
</tr>
<tr>
<td>*</td>
<td>192.168.*</td>
<td>*</td>
<td>666</td>
<td>*</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

Bit Vectors: $\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$
# Bit Vector

Packet: **192.168.0.1** | **192.168.1.42** | **443** | **80** | **TCP**

<table>
<thead>
<tr>
<th>Source IP</th>
<th>Destination IP</th>
<th>Source Port</th>
<th>Destination Port</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.*</td>
<td>*</td>
<td>[1024:65535]</td>
<td>80</td>
<td>TCP</td>
</tr>
<tr>
<td>192.168.*</td>
<td>72.7.25.*</td>
<td>7777</td>
<td>*</td>
<td>TCP, UDP</td>
</tr>
<tr>
<td>*</td>
<td>192.168.1.42</td>
<td>*</td>
<td>80</td>
<td>TCP</td>
</tr>
<tr>
<td>*</td>
<td>192.168.*</td>
<td>*</td>
<td>666</td>
<td>*</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

Bit Vectors:

\[
\begin{pmatrix}
1 \\
1 \\
1 \\
1
\end{pmatrix}
\quad\begin{pmatrix}
1 \\
0 \\
1 \\
1
\end{pmatrix}
\]
**Bit Vector**

<table>
<thead>
<tr>
<th>Source IP</th>
<th>Destination IP</th>
<th>Source Port</th>
<th>Destination Port</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.*</td>
<td>*</td>
<td>[1024:65535]</td>
<td>80</td>
<td>TCP</td>
</tr>
<tr>
<td>192.168.*</td>
<td>72.7.25.*</td>
<td>7777</td>
<td>*</td>
<td>TCP, UDP</td>
</tr>
<tr>
<td>*</td>
<td>192.168.1.42</td>
<td>*</td>
<td>80</td>
<td>TCP</td>
</tr>
<tr>
<td>*</td>
<td>192.168.*</td>
<td>*</td>
<td>666</td>
<td>*</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

Packet: 192.168.0.1, 192.168.1.42, 443, 80, TCP

Bit Vectors:
\[
\begin{bmatrix}
1 \\
1 \\
1 \\
1
\end{bmatrix}
\begin{bmatrix}
1 \\
0 \\
1 \\
1
\end{bmatrix}
\begin{bmatrix}
0 \\
0 \\
1 \\
1
\end{bmatrix}
\begin{bmatrix}
1 \\
1 \\
1 \\
0
\end{bmatrix}
\begin{bmatrix}
1 \\
1 \\
1 \\
1
\end{bmatrix}
\begin{bmatrix}
1 \\
1 \\
1 \\
1
\end{bmatrix}
\]
**Bit Vector**

Packet: 192.168.0.1  192.168.1.42  443  80  TCP

<table>
<thead>
<tr>
<th>Source IP</th>
<th>Destination IP</th>
<th>Source Port</th>
<th>Destination Port</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.*</td>
<td>*</td>
<td>[1024:65535]</td>
<td>80</td>
<td>TCP</td>
</tr>
<tr>
<td>192.168.*</td>
<td>72.7.25.*</td>
<td>7777</td>
<td>*</td>
<td>TCP, UDP</td>
</tr>
<tr>
<td>*</td>
<td>192.168.1.42</td>
<td>*</td>
<td>80</td>
<td>TCP</td>
</tr>
<tr>
<td>*</td>
<td>192.168.*</td>
<td>*</td>
<td>666</td>
<td>*</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

Bit Vectors:

\[
\begin{pmatrix}
1 \\
1 \\
1
\end{pmatrix}
\&
\begin{pmatrix}
1 \\
1 \\
1
\end{pmatrix}
\&
\begin{pmatrix}
0 \\
0 \\
1
\end{pmatrix}
\&
\begin{pmatrix}
1 \\
1 \\
0
\end{pmatrix}
\&
\begin{pmatrix}
1 \\
1 \\
1
\end{pmatrix}
= 
\begin{pmatrix}
0 \\
0 \\
1
\end{pmatrix}
\text{Match!}
HIERARCHICAL INTELLIGENT CUTTINGS (HiCuts)

Geometric View:

Decision Tree:

[0:7]x[0:7], check 2, 2

geometric search space
cut-dimension
amount of children
HIERARCHICAL INTELLIGENT CUTTINGS (HiCuts)

Geometric View:

Decision Tree:

- [0:7]x[0:7], check 2, 2
- [0:7]x[0:3], check 1, 2
- [0:7]x[4:7], \{b, e\}

geometric search space

amount of children
Hierarchical Intelligent Cuttings (HiCuts)