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)

Institute of Computing Technology, Chinese Academy of Sciences
UNIVERSITE DE SAVOIE CHAMBERY ANNECY
Université de Liège

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

Source IP	Destination IP	Source Port	Destination Port	Protocol	Action
120.0.0.0/24	198.12.130.0/2	0:65535	11:17	0xFF/0xFF	Accept
138.42.83.1/0	174.3.18.0/8	50:10000	0:65535	0x06/0xFF	Deny

Packet classification is key for

- Security
- Traffic monitoring and analysis
- QoS

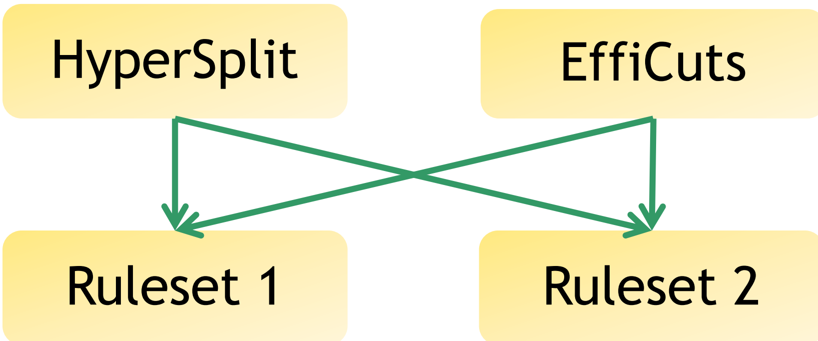
Packet classification prevalent in modern routers

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!

same algorithm, different ruleset,

Algorithm	Ruleset(size)	Memory size	Mem. accesses
HyperSplit	ACL1_100K	2.12MB	32
	ACL2_100K	83MB	43
EffiCuts	ACL1_100K	3.23MB	65
	ACL2_100K	4.81MB	136

TABLE I: Performance comparison on different rulesets

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 rule-sets. (INFOCOM 09')

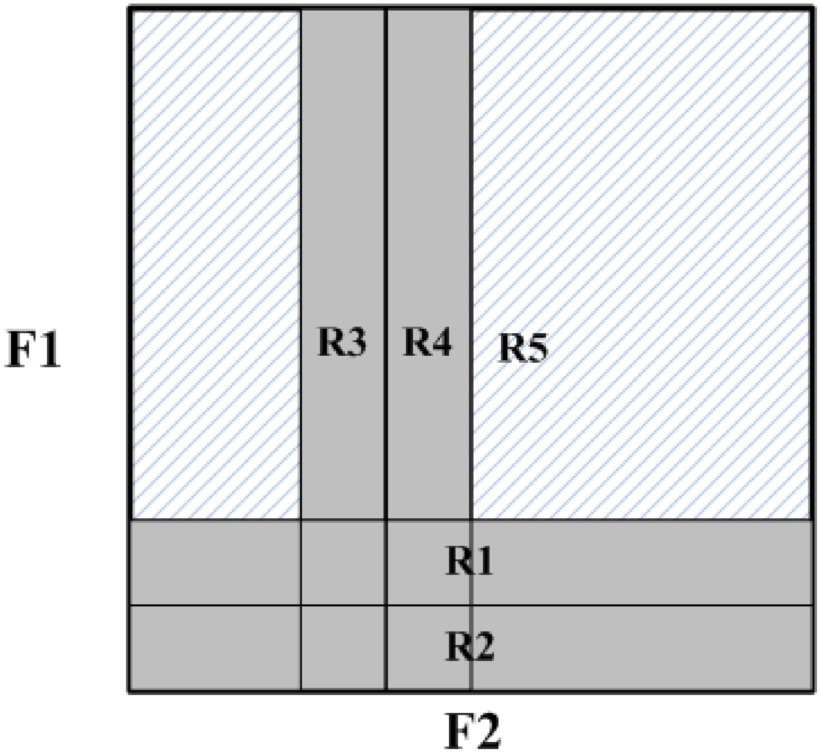
Agenda

- 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

Rule #	Field 1	Field 2	Action
R1	111*	*	DROP
R2	110*	*	PERMIT
R3	*	010*	DROP
R4	*	011*	PERMIT
R5	*	*	PERMIT

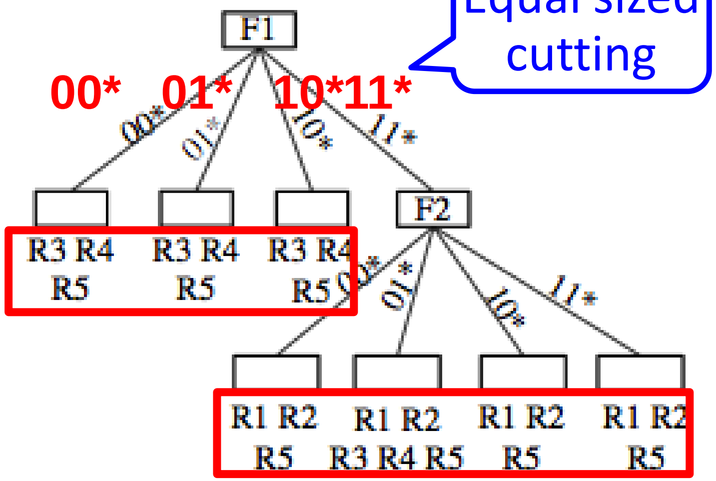
Table 2: An example ruleset



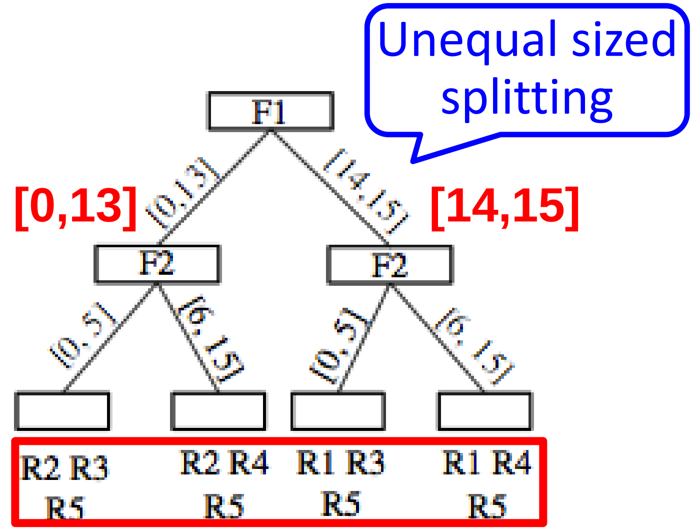
Each rule can be viewed as a ***hyperrectangle***

Algorithms performs "cuts" or "splits" on the space to reduce the search space.

An Example (HiCuts and HyperSplit)



(a) HiCuts



(b) HyperSplit

A lot of rule duplications, large memory size

Less rule duplications, smaller memory size

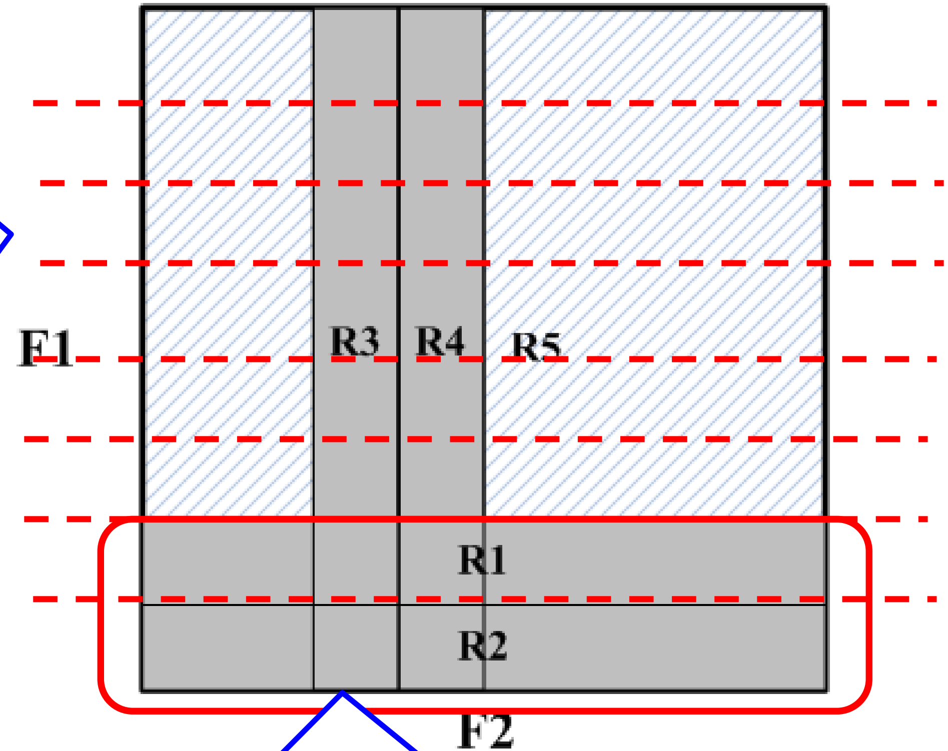
R...#	F...ld 1	Field 2	Action
	1*	*	DROP
	0*	*	PERMIT
	*	010*	DROP
	*	011*	PERMIT
R5	*	*	PERMIT

Table 2: An example ruleset

2 dimension rule-set
Each field has 4 bits

Non-uniformly distributed ranges

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

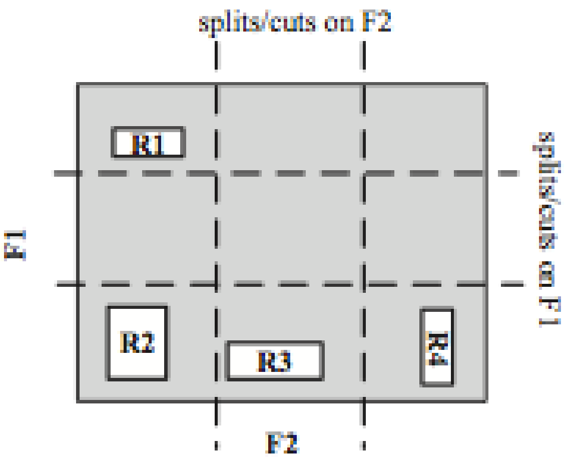


Rule #	Field 1	Field 2	Action
R1	111*	*	DROP
R2	110*	*	PERMIT
R3	*	010*	DROP
R4	*	011*	PERMIT
R5	*	*	PERMIT

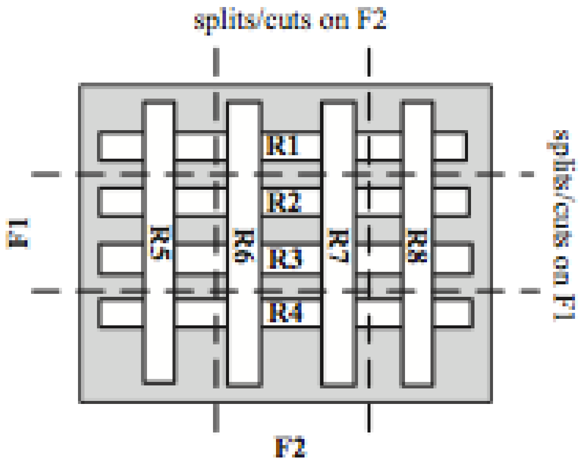
111* and 110* just concentrate in a small

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



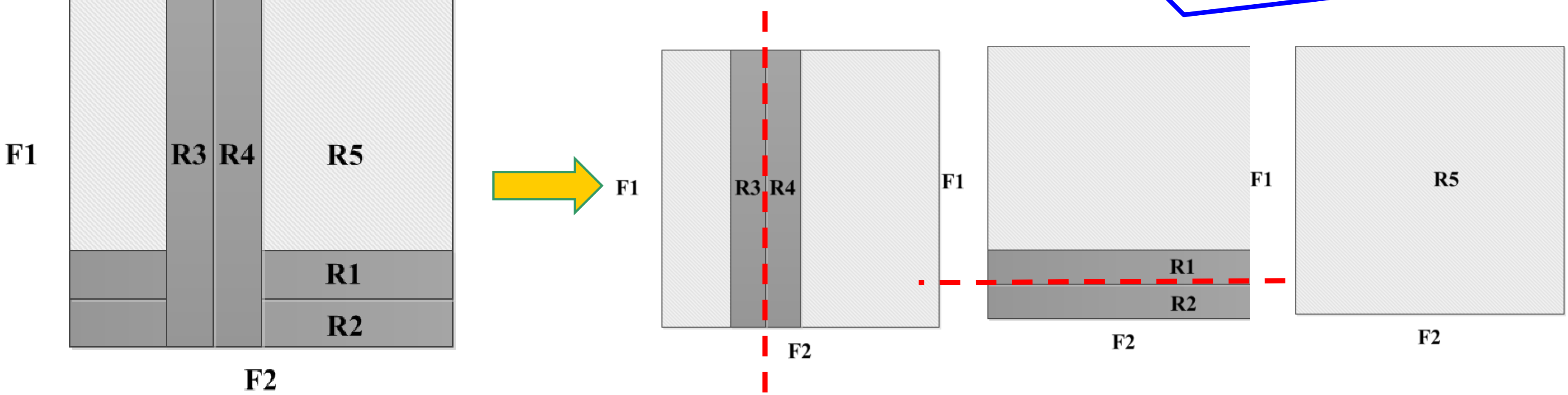
(b) orthogonal structure rules

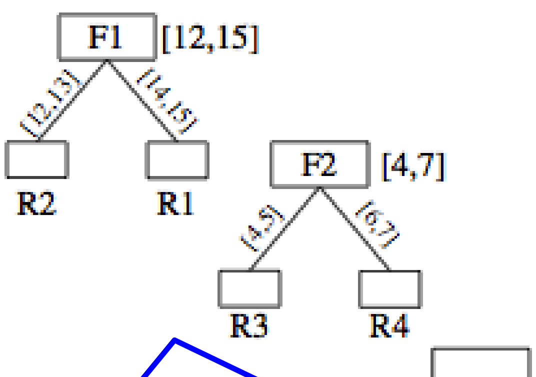
Rules can be easily sperated

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

EffiCuts

Untangle the orthogonal structures





Smallest memory size
No rules get duplicated

Rule #	Field 1	Field 2	Action
R1	111*	*	DROP
R2	110*	*	PERMIT
R3	*	010*	DROP
R4	*	011*	PERMIT
R5	*	*	PERMIT

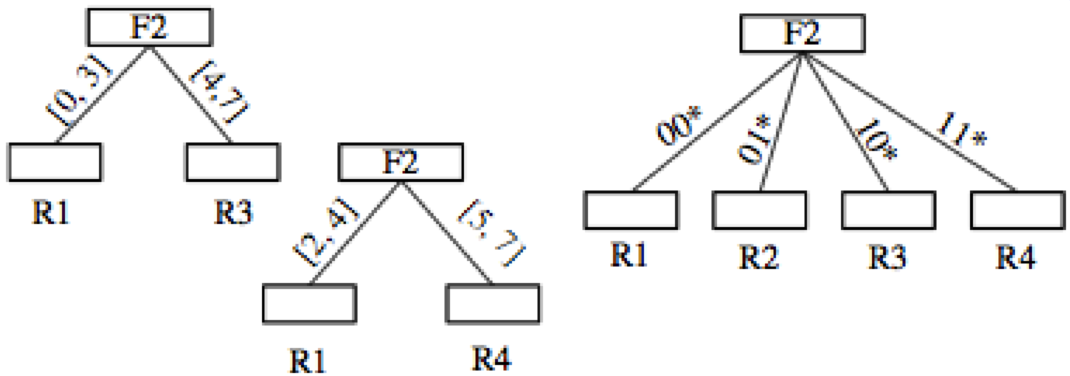
Table 2: An example ruleset

Orthogonal Structures can be invalid.

TABLE IV
UNIFORM DIMENSION HIDES THE "ORTHOGONAL STRUCTURE"

Field 1	Field 2	Field 3
00*	*	01
01*	01	*
10*	*	10
11*	10	*

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



(a) EffiCuts (b) HyperCuts

Fig. 3. Decision Trees of EffiCuts and HyperCuts on Table IV

2 memory accesses

1 memory access

Discussions

- “Orthogonal structures” should be considered, and rules should be eventually splitted 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

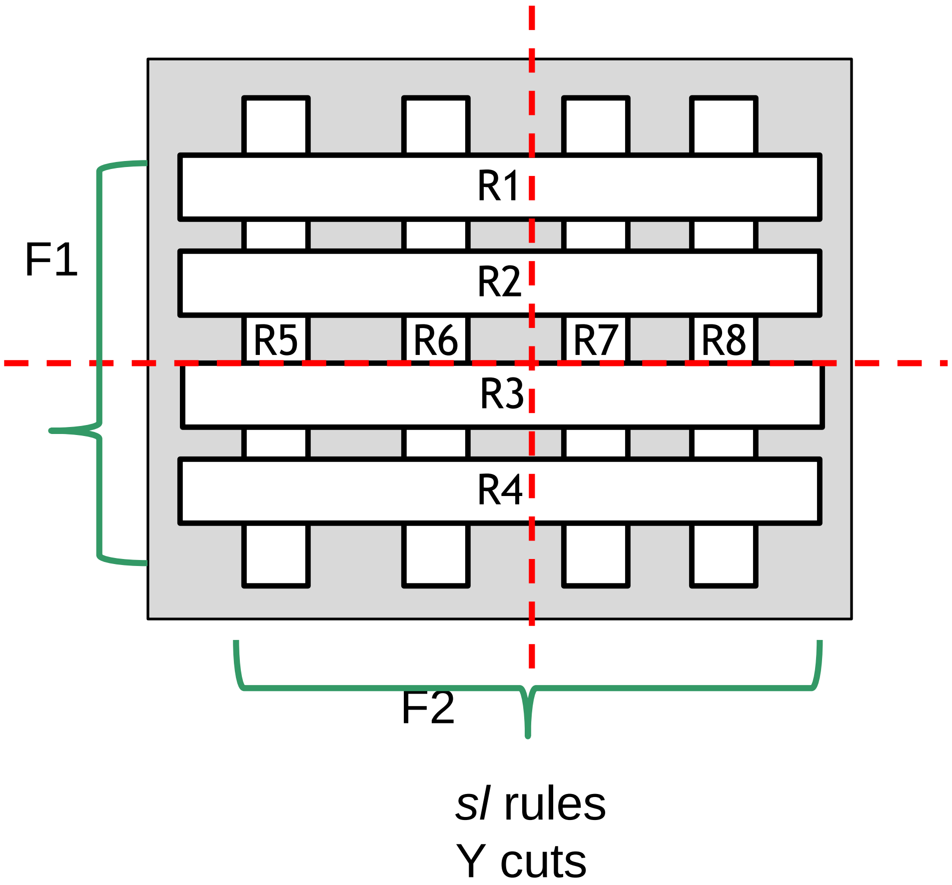
- Motivation
- Key Observations
- Two Meta Methods
  - Memory Consumption Model
  - Characterizing range distribution uniformity
- The AutoPC framework and SmartSplit algorithm
- Experiment Results.

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.



$ls \times Y + sl \times X$ rule duplications

Memory Consumption Model

Assume: in each divided space, there are *binth* rules. Half of rules are (large, small) rules and the other half are (small, large) rules.

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

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

$$M_{ss} = ss \times PTR$$

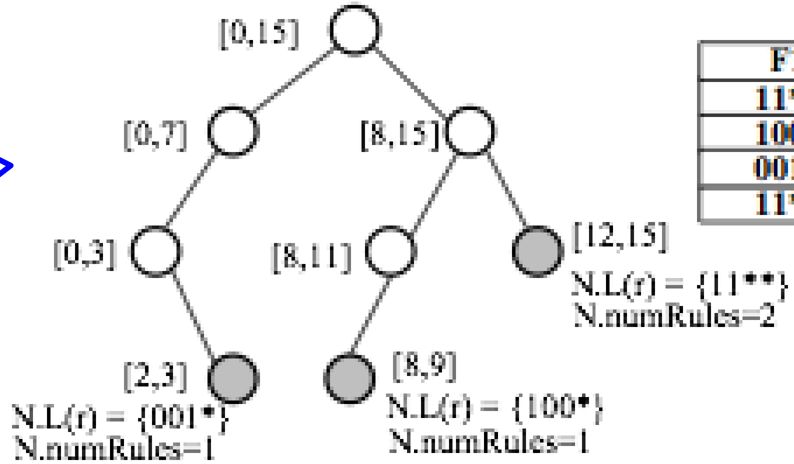
$$M_{ls} = ls \times \frac{sl + ss \times \alpha}{binth/2} \times PTR$$

$$M_{sl} = sl \times \frac{ls + ss \times (1 - \alpha)}{binth/2} \times PTR$$

$$M_{ll} = ll \times \frac{sl + ss \times \alpha}{binth/2} \times \frac{ls + ss \times (1 - \alpha)}{binth/2} \times PTR$$

Characterizing range distribution uniformity

Build a interval tree for the ranges on each field.



F1	F2
11**	111*
100*	11*
001*	001*
11**	0*

Characterizing range distribution by the *shape* of interval tree!

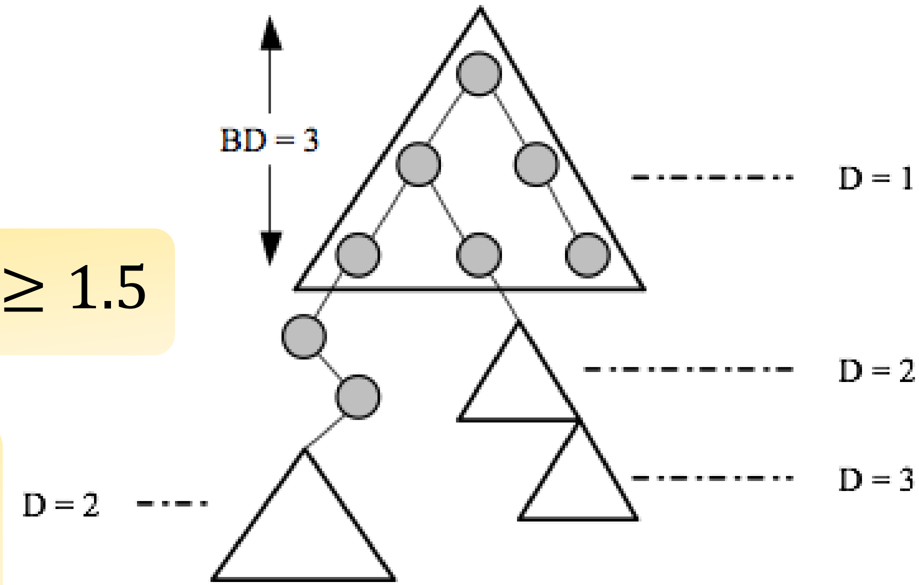
The shape of Interval trees

Quasi-Balanced subtrees

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

Balance tree distance *D*

Max Balance tree distance D_{max}



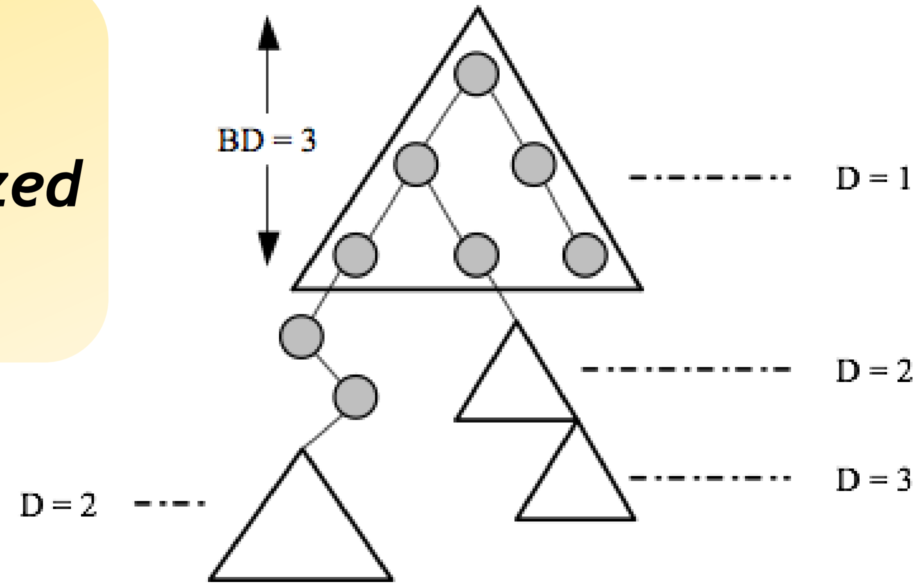
D_{max} = 3 *for this tree*

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.

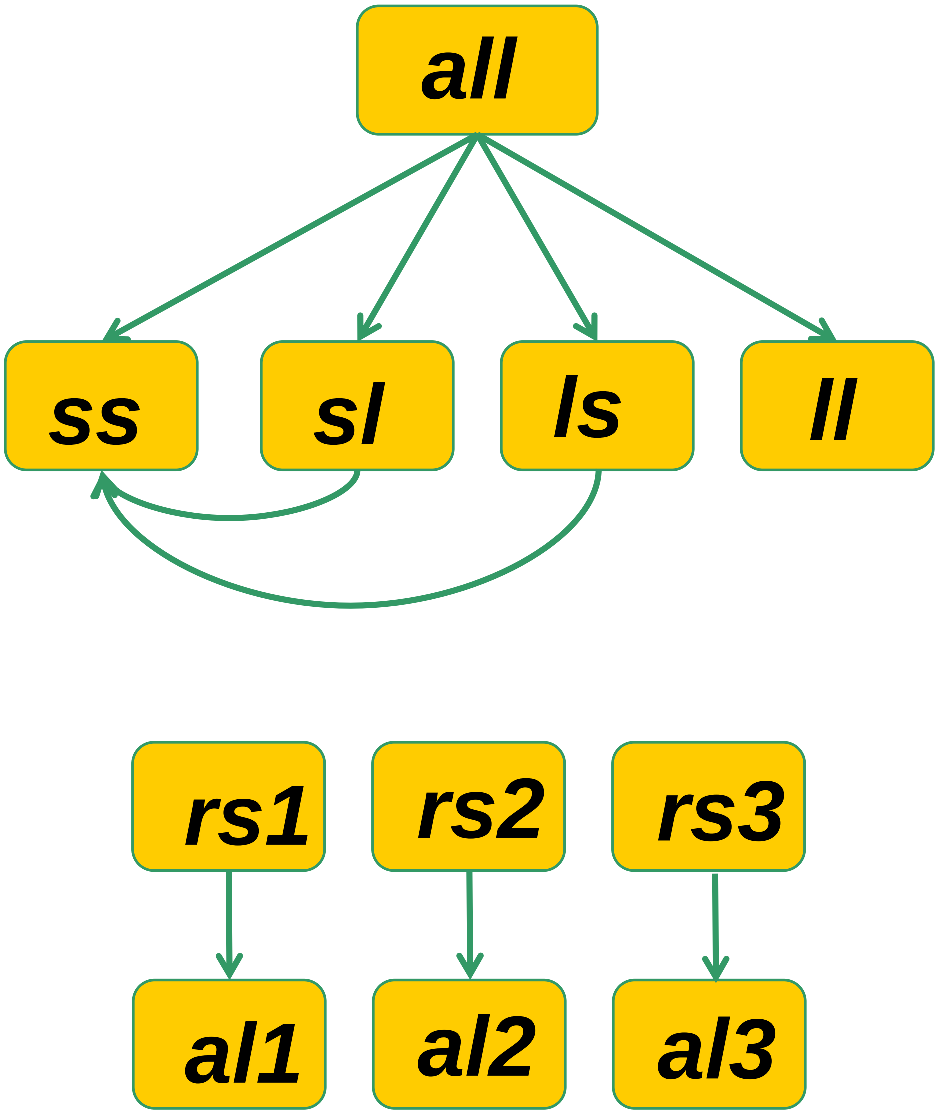


SmartSplit Algorithm

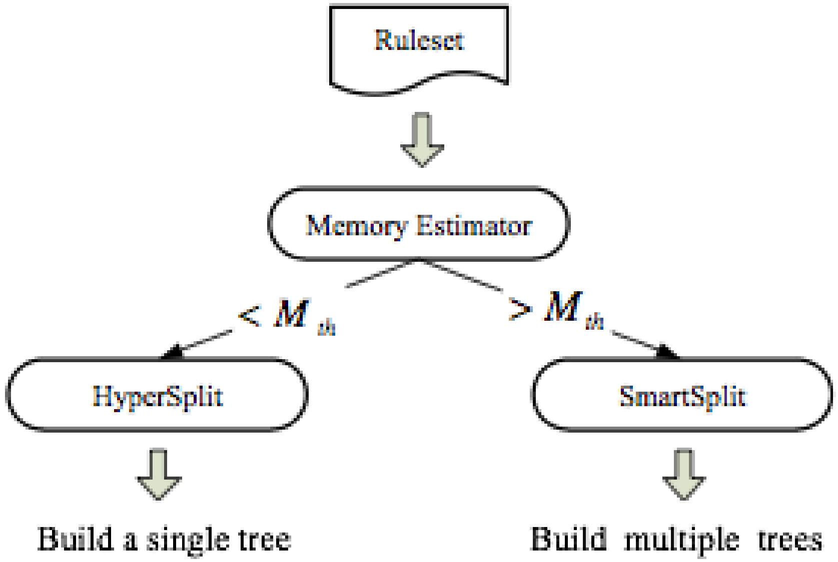
Split the ruleset according to large/small ranges on IP fields

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

Use different algorithms on three rulesets.



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

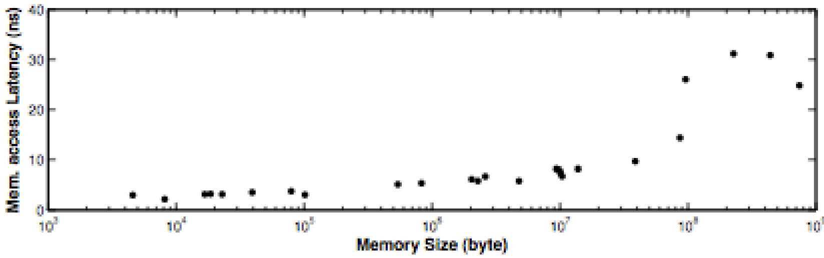


Fig. 7: Average Memory Access Latency and Memory Size

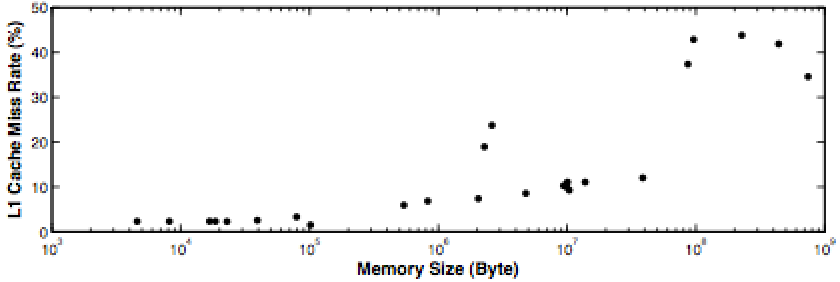


Fig. 8: Cache Misses Rate and Memory size

Experiments Results

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

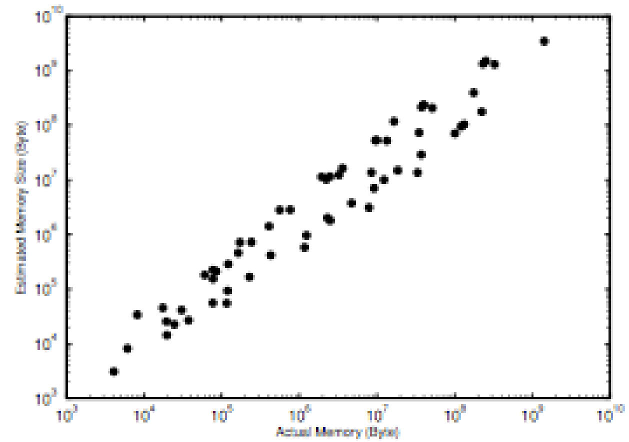


Fig. 9: Estimated and Actual memory size with $binth = 16$

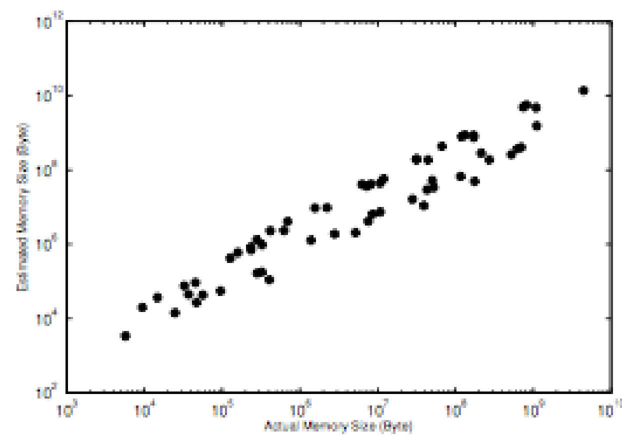


Fig. 10: Estimated and Actual memory size with $binth = 8$

Memory estimation is fast

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

TABLE VI: Estimated and Actual Memory size of Large rulesets

1000x faster then

really building a decision tree

Experiments Results

- Compare SmartSplit and EffiCuts

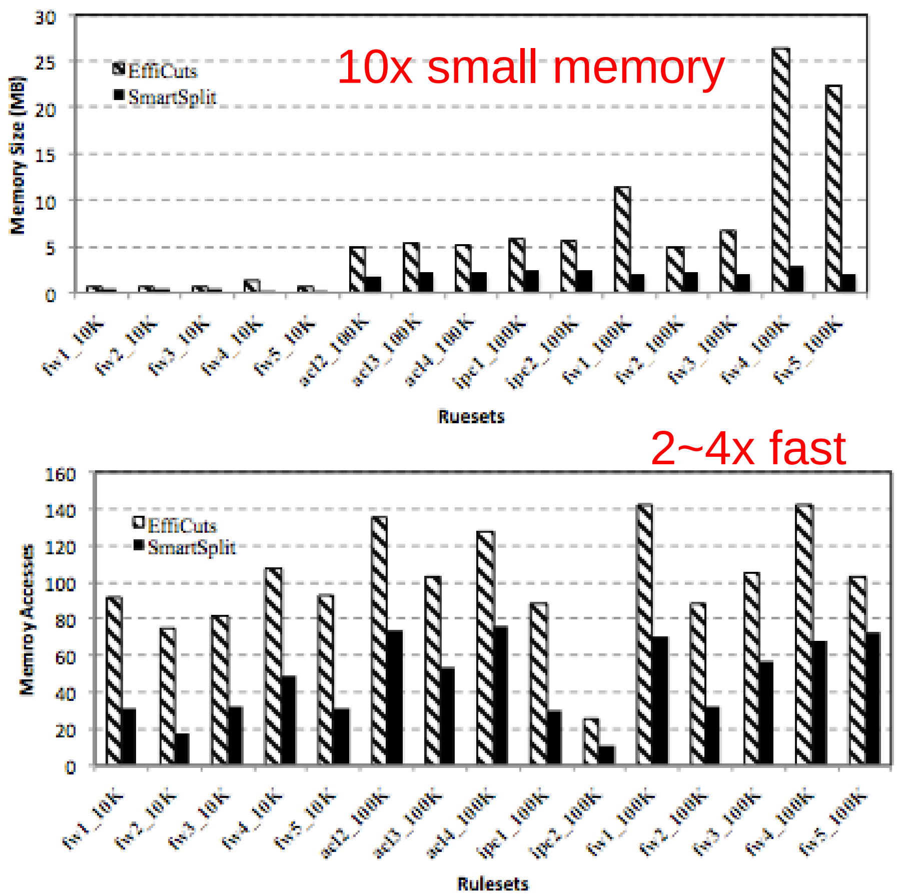


Fig. 12: Memory and Accesses for EffiCuts and SmartSplit

Real Evaluation

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

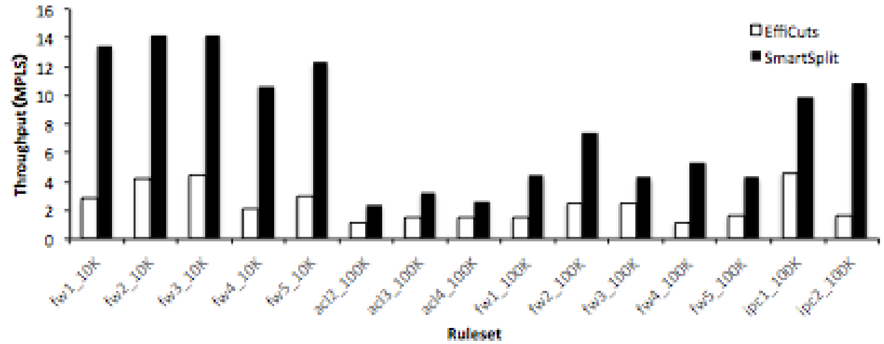


Fig. 13: Comparing the mesured performance of SmartSplit and EffiCuts

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

Type	Size	AutoPC (MLPS)	EffiCuts (MLPS)	speedup	HyperSplit (MLPS)	speedup
ACL	1K	11.3*	4.5	2.4	11.3	1
	10K	6.9*	3.1	2.2	6.9	1
	100K	8.6*	2.2	3.9	8.6	1
FW	1K	9.8*	2.4	4.1	9.8	1
	10K	10.7	2.1	5.1	2.6	4.1
	100K	7.4	2.5	3.0	-	-
IPC	1K	12.6*	3.0	4.25	12.6	1
	10K	5.3*	1.48	3.6	5.3	1
	100K	9.91	1.63	6.1	0.07	141
Average Speedup: 3.8(to EffiCuts) / 18.9 (to HyperSplit)						

TABLE VIII: Real Performance Evaluation of AutoPC, EffiCuts and HyperSplit

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