Scalable Packet Classification

Florin Baboescu
George Varghese

SIGCOMM'01

Motivation

- *Rule intersection* is very rare.
  - It is very rare to have a packet that matches multiple rules

Idea

- Enhancing scalability of the bit vector scheme by providing two new ideas
  - *Rule aggregation*
  - *Rule rearrangement*.

Outline

- Introduction
- Problem statement
- Bit vector scheme
- Aggregated bit vector algorithm
- Evaluation
- Conclusion

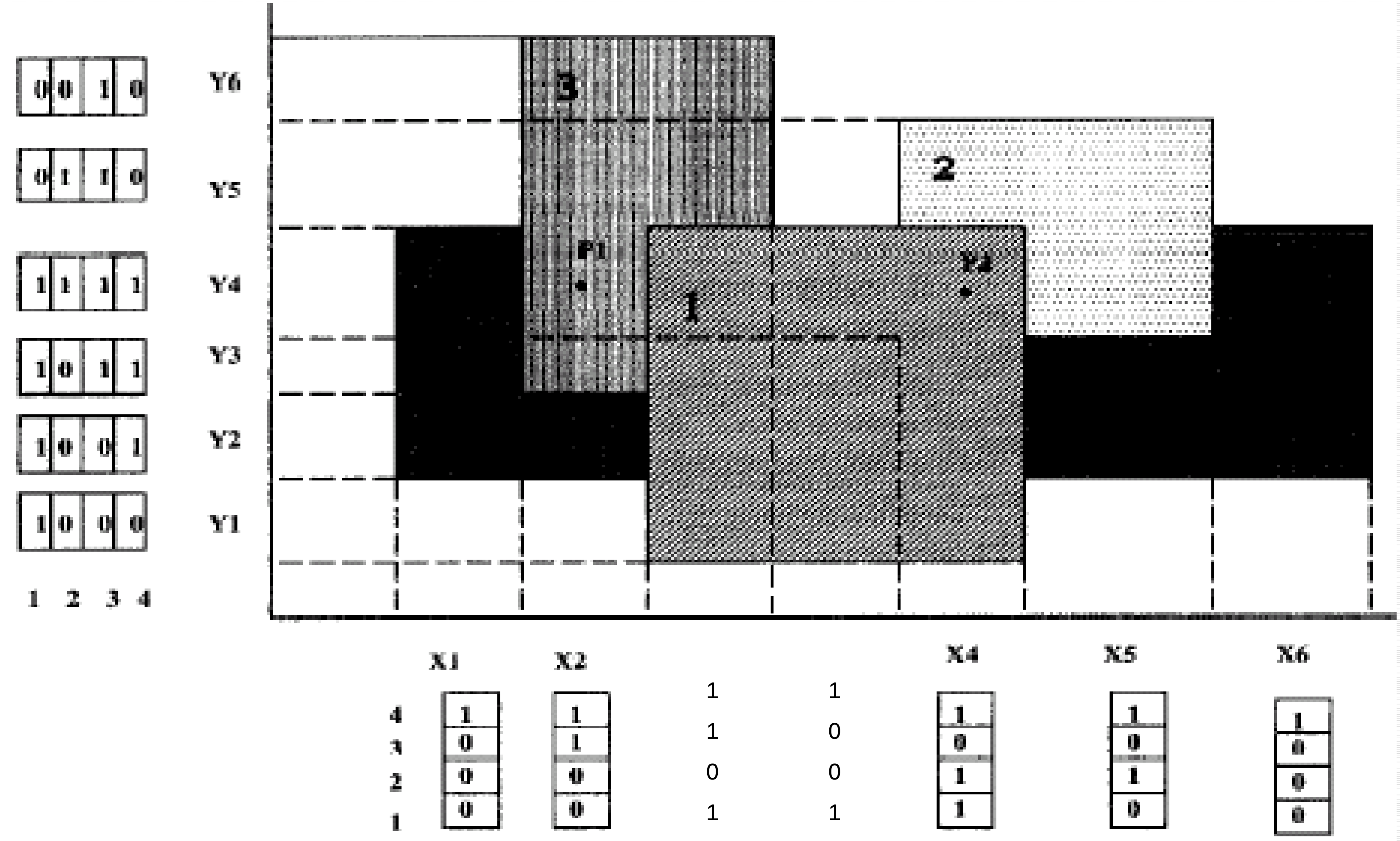
Introduction

- Packet classification is a process for routers to classify packets based on packet headers into equivalence classes called flows
- This paper performs scalable packet classification at wire speeds even as rule databases increase in size

Problem Statement

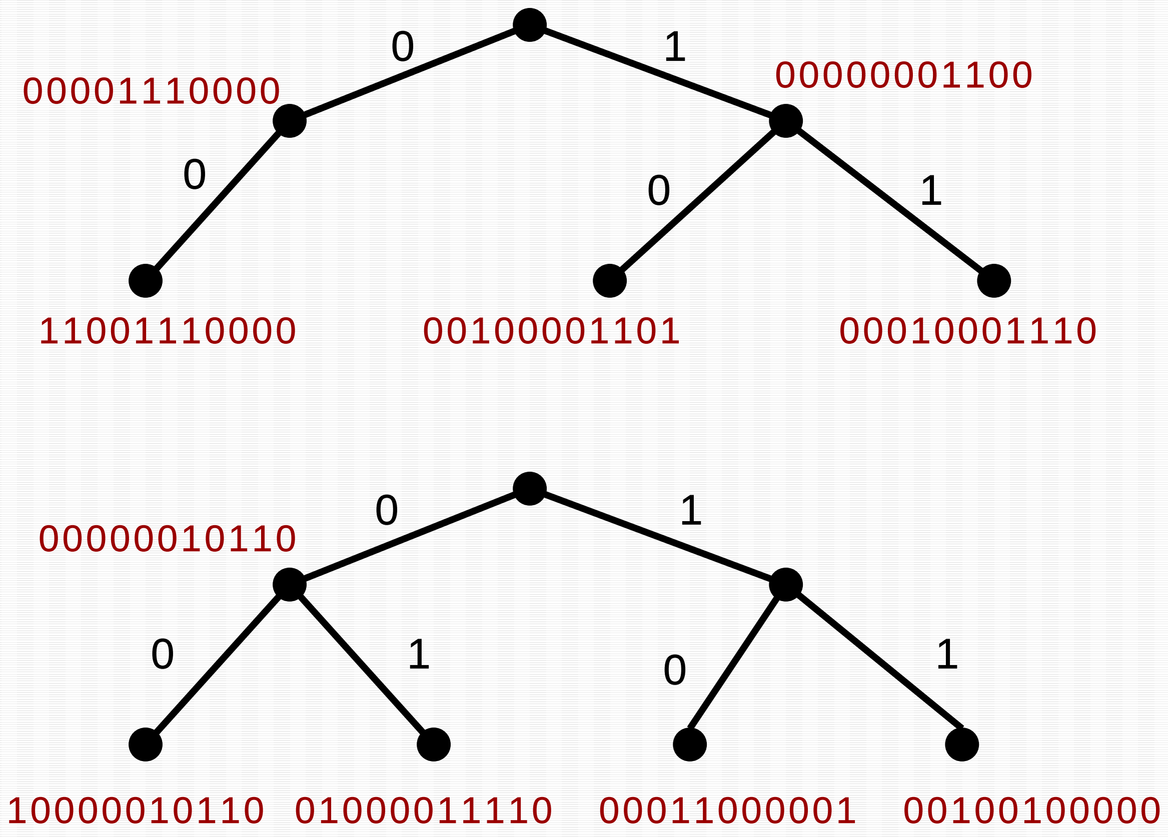
- A packet P matches a rule R if each field of P matches the corresponding field of R
  - Let R = (1010*, *, TCP, 1024-1080, *), then a packet with header (10101...1, 11110...0, TCP, 1050, 3) matches R
- Since a packet might match multiple rules, we define the matching rule to be the *earliest* one

Bit Vector (1/3)



Bit Vector (2/3)

$Rule$	$Field_1$	$Field_2$
F_0	00∗	00∗
F_1	00∗	01∗
F_2	10∗	11∗
F_3	11∗	10∗
F_4	0∗	10∗
F_5	0∗	11∗
F_6	0∗	0∗
F_7	1∗	01∗
F_8	1∗	0∗
F_9	11∗	0∗
F_{10}	10∗	10∗



Bit Vector (3/3)

- Handicap
  - These vectors have *N* bits in length; Computing the intersection requires *O(N)* operations
    - If *W* is the size of a word of memory, then these bit operations are responsible for *n*k/w* memory accesses in the worst case

Aggregated Bit Vector

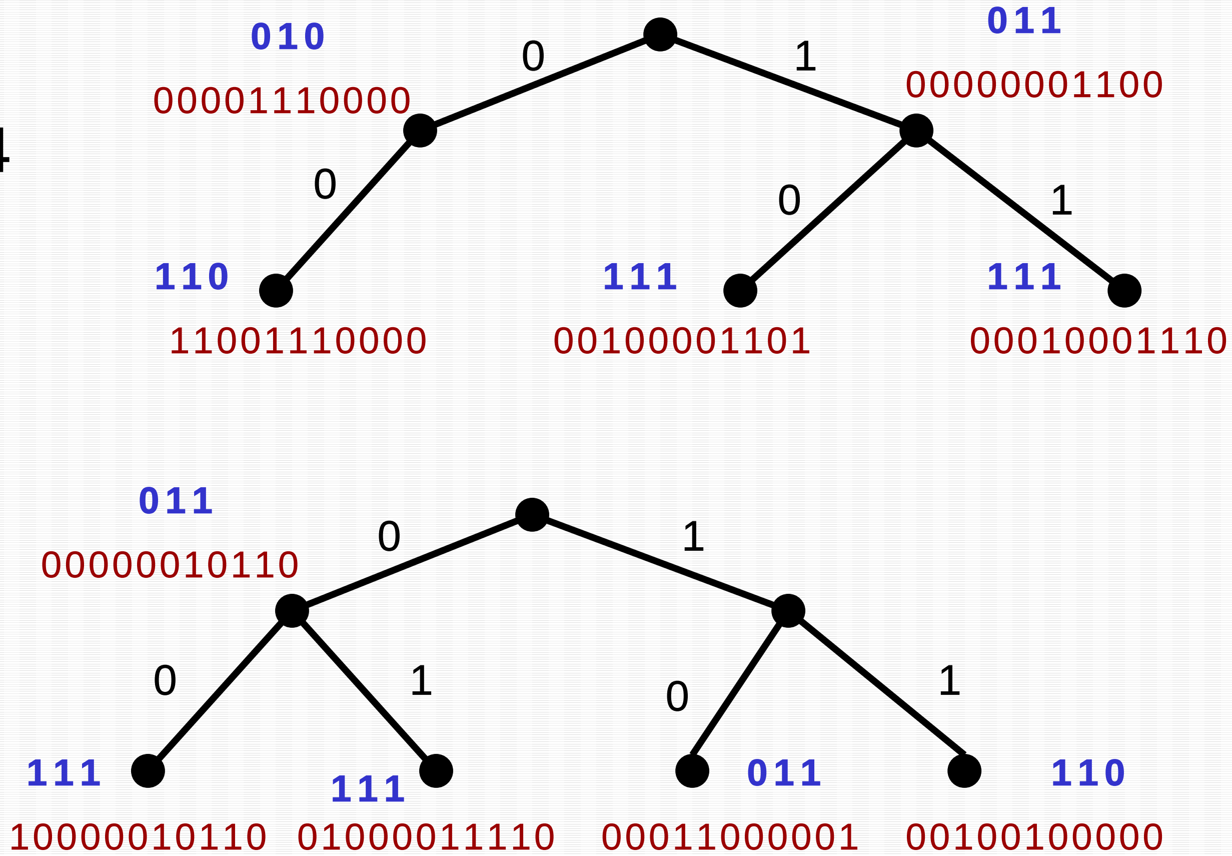
- Rule aggregation
- Rule arrangement

Rule Aggregation (1/3)

1. Fix an aggregate bit A
2. A bit i is set in the aggregate vector if there is at least one bit k set, $k \in [i * A, (i + 1) * A]$
3. Repeat the aggregation process at multiple levels

Rule Aggregation (2/3)

A = 4



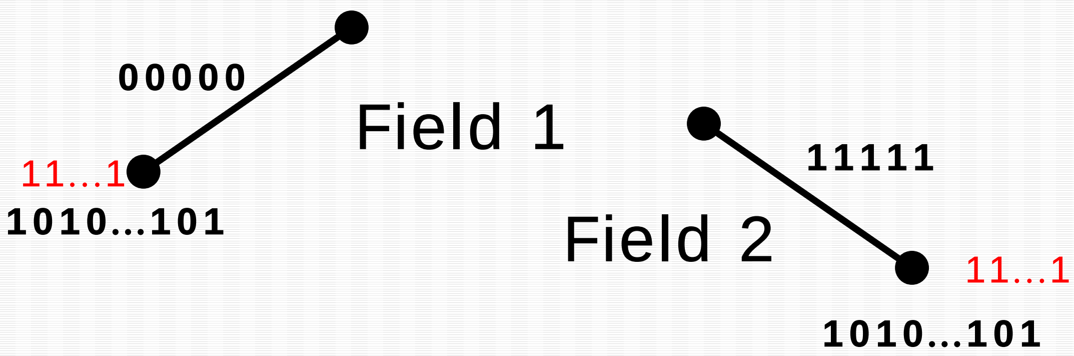
Rule Aggregation (3/3)

```
Get Packet P(H_1, ..., H_k);
for i ← 1 to k do
    N_i ← longestPrefixMatchNode(Trie_i, H_i);
Aggregate ← 11...1;
for i ← 1 to k do
    Aggregate ← Aggregate ⋂ N_i.aggregate;
BestRule ← Null;
for i ← 0 to sizeof(Aggregate) − 1 do
    if Aggregate[i] == 1 then
        for j ← 0 to A − 1 do
            if ⋂_{l=1}^{k} N_l.bitVect[i × A + j] == 1 then
                if R_{i×A+j}.cost < BestRule.cost then
                    BestRule = R_{i×A+j};
return BestRule;
```

Rule Arrangement (1/3)

- Assume $(X, A_1, \ldots, A_{30}, Y) = (00000^*, 00001^*, \ldots, 11110^*, 11111^*)$

A = 2

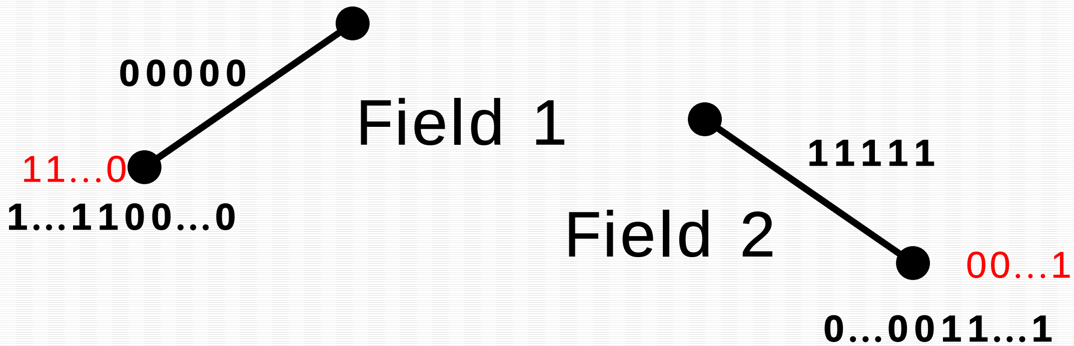


$Rule$	$Field_1$	$Field_2$
F_1	X	A_1
F_2	A_1	Y
F_3	X	A_2
F_4	A_2	Y
F_5	X	A_3
F_6	A_3	Y
F_7	X	A_3
...	...	...
...	...	...
F_{60}	A_{30}	Y
F_{61}	X	Y

This is called a "*false match*", resulted by invalid match in the group of rules identified by the aggregate

Rule Arrangement (2/3)

After arranging rules



A = 2

$Rule$	$Field_1$	$Field_2$
F_1	X	A_1
F_2	X	A_2
F_3	X	A_3
$\ldots$	$\ldots$	$\ldots$
F_{30}	X	A_{30}
F_{31}	X	Y
F_{32}	A_1	Y
F_{33}	A_2	Y
$\ldots$	$\ldots$	$\ldots$
F_{60}	A_{29}	Y
F_{61}	A_{30}	Y

What this does is to localize as many matches as possible for the sorted field to lie within a few aggregation groups instead of having matches dispersed across many groups

Rule Arrangement (3/3)

```
ARRANGE-ENTRIES(first, last, col)
if(there are no more fields) or (first == last)
then return;
for (each valid (length) size of prefixes) then
    Group together all the elements
    with the same size;
    Sort the previously created groups.
    Create subgroups made up of elements
    having the same prefixes on the field col
    for (each subgroup S with more
    than two elements) then
        Arrange-Entries(S.first, S.last, col + 1);
```

Evaluation

- Experimental platform
- Performance evaluation on industrial firewall databases
- Experimental evaluation on synthetic two-dimensional databases
- Performance evaluation using synthetic five-dimensional databases

Experimental Platform

- Two different types of databases
  1. A set of four industrial firewall databases
  2. Randomly synthesized databases based on publicly available routing tables

Filter	*Number of rules specified by:*	
	Range	*Prefix*
DB_1	266	1640
DB_2	279	949
DB_3	183	531
DB_4	158	418

Routing Table	*Prefix Lengths:*					
	8	*9 to 15*	*16*	*17 to 23*	*24*	*25 to 32*
Mae – East	10	133	1813	9235	11405	58
Mae – West	15	227	2489	11612	16290	39
AADS	12	133	2204	10144	14704	55
PacBell	12	172	2665	12808	19560	54
Paix	22	560	6584	28592	49636	60

Performance evaluation on industrial firewall databases

Filter	No.of Nodes	No. of Valid Prefixes
DB_1	980	188
DB_2	1242	199
DB_3	805	127
DB_4	873	143

Filter	BV	ABV		
		unsorted	One Field Sorted	Two Fields Sorted
DB_1	260	120	75	65
DB_2	150	110	50	50
DB_3	85	60	50	50
DB_4	75	55	45	45

No. of memory accesses

Experimental evaluation on synthetic 2d databases (1/3)

DB Size	*BV*	*Percentage of prefixes of length zero; sorted(s)/usorted(u)*												
		0	1u	1s	2u	2s	5u	5s	10u	10s	20u	20s	50u	50s
1K	64	8	12	10	26	10	54	10	66	12	66	12	66	10
2K	126	10	28	14	58	12	84	14	126	14	130	14	130	14
5K	314	16	50	18	76	18	216	20	298	20	324	22	324	18
10K	626	26	78	30	196	28	426	34	588	34	644	32	646	30
20K	1250	48	148	48	346	50	860	52	1212	54	1288	52	1292	52

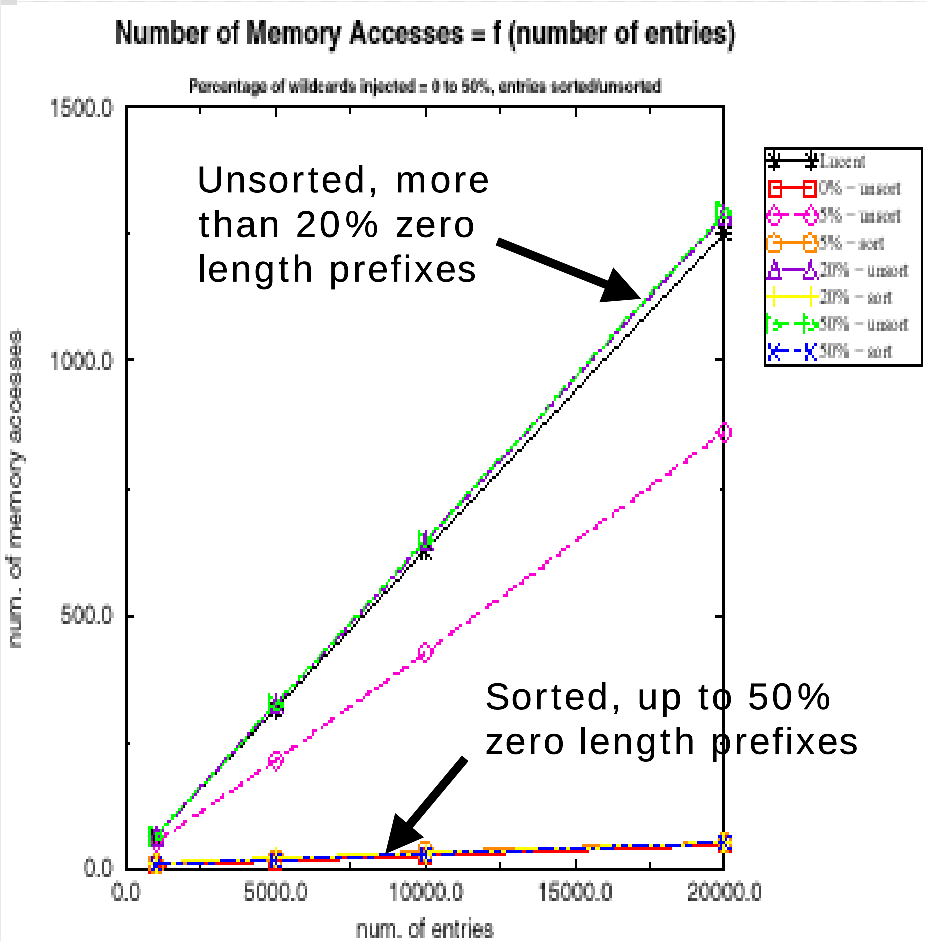
DB Size	*BV*	*W = 4*					*W = 6*					*W = 8*				
		1	10	20	50	90	1	10	20	50	90	1	10	20	50	90
1K	64	8	10	20	40	52	8	12	26	38	56	8	12	20	36	52
5K	314	16	28	56	124	144	16	32	56	126	148	16	30	50	120	162
10K	626	28	54	96	228	214	26	50	96	244	234	26	50	94	194	226
20K	1250	48	88	168	308	254	48	90	154	274	292	48	92	176	304	326

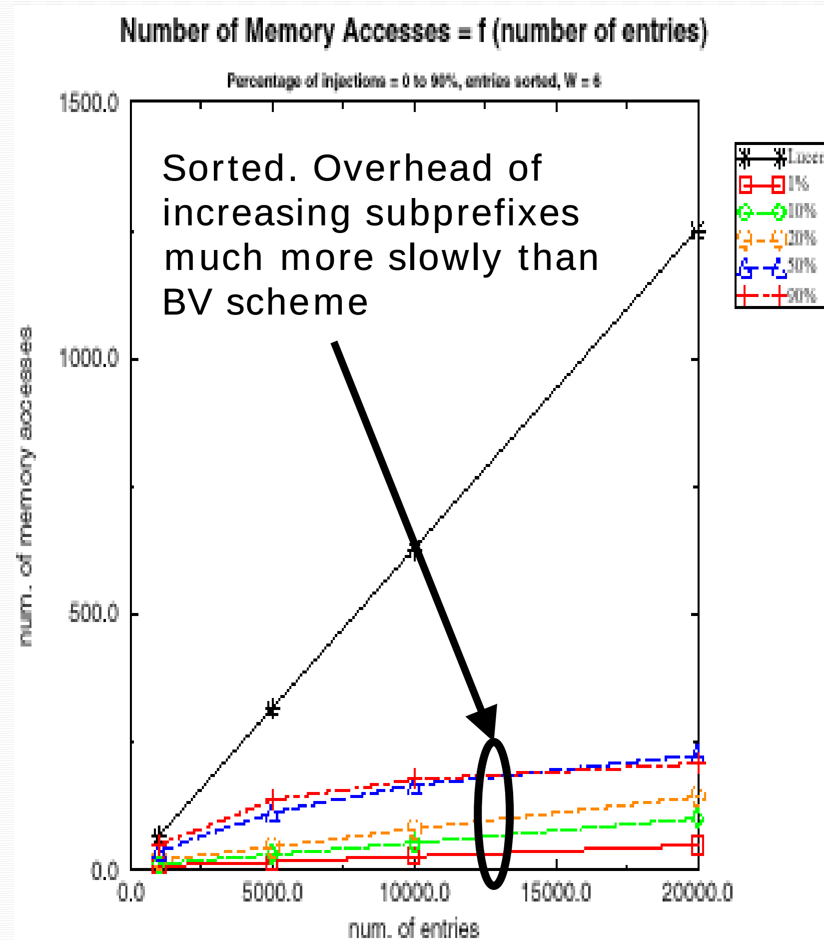
Unsorted, percentage of subprefixes. W is the depth of the subtrie

DB Size	*W = 4*					*W = 6*					*W = 8*				
	1	10	20	50	90	1	10	20	50	90	1	10	20	50	90
1K	6	12	16	34	54	8	12	18	36	48	8	12	16	36	48
5K	16	26	48	106	136	16	30	44	112	136	16	30	46	116	138
10K	26	46	82	176	154	26	52	80	166	176	26	48	84	198	178
20K	48	78	146	212	138	48	100	142	224	208	48	88	136	232	170

sorted

Experimental evaluation on synthetic 2d databases (2/3)





Experimental evaluation on synthetic 2d databases (3/3)

Word Size	BV	ABV
128	314	34
256	158	28
512	80	26
1024	40	20

A = 32, for 20000 rules database

Experiment	No. Of Entries = 5000		No. Of Entries = 10000		No. Of Entries = 20000	
	One Level	Two Levels	One Level	Two Levels	One Level	Two Levels
0% stars	16	14	26	14	46	18
1% stars	18	14	30	20	52	22
5% stars	20	14	30	18	52	26
10% stars	22	20	32	22	50	22
50% stars	20	18	30	18	50	20

sorted

Performance evaluation using synthetic 5-d databases

A = 32, no wildcard injections

Size	*BV*	*ABV - 32*
3722	585	40
7799	1220	65
21226	3320	140

Conclusion

- The paper introduces the notions of aggregation and rule arrangement to make the BV scheme more scalable, creating the ABV scheme
- The ABV scheme is at least an order of magnitude faster than the BV scheme on all performed tests