Leveraging Parallelism for Multi-dimensional Packet Classification on Software Routers

THE UNIVERSITY of WISCONSIN MADISON

Yadi Ma Suman Banerjee
Shan Lu Cristian Estan*

Presenter: Chang Chen
Mar 19, 2014

Question

- *If we were to implement classification using state-of-art desktops, given that it may find applications in router design, then what classification speeds can be achieved on it, by designing a fully software-based classification system that can exploit the degree of parallelism provided by this particular platform?*

Outline

- Introduction
- Overview of Storm
- Storm Design
    - Thread Assignment
    - Rule Cache Updater
    - Micro Benchmarks
- Experimental Results

Introduction

- **Storm:** a software-based solution to the multi-dimensional packet classification problem.

- A new software system which
  - takes existing classification algorithms;
  - utilizes a common idea of caching;
  - partitions critical tasks into multiple threads that effectively leverage desktop platforms to meet various computation and memory access needs.

Prior Work

Q. Dong, et al. Wire Speed Packet Classification Without TCAMs: A Few More Registers (And A Bit of Logic) Are Enough, in ACM SIGMETRICS, 2007.

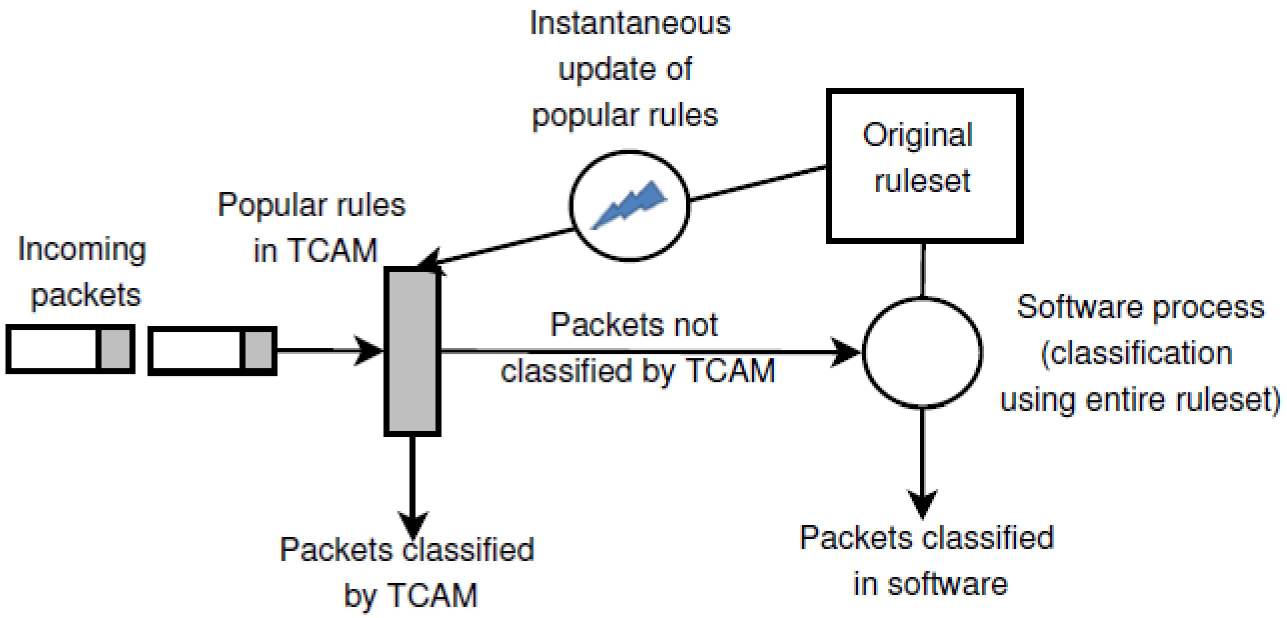


Figure 1: Framework of SRC, its use of TCAMs, and simplifying assumptions.

Overview of Storm

- A parallelized software system using multi-core desktop platforms.
- Uses a combination of task, data and pipeline parallelism.

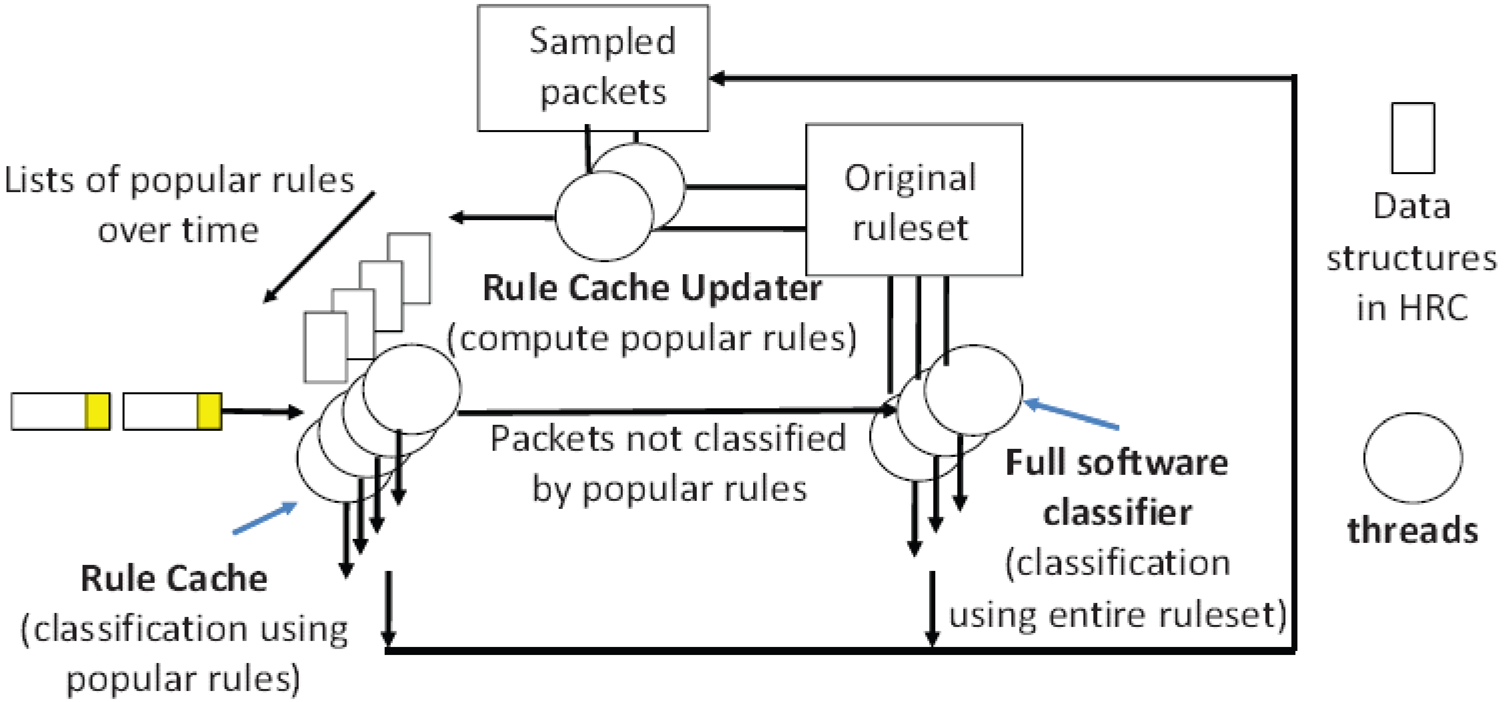


Figure 2: Architecture of Storm.

Overview of Storm

- Task Parallelism
  - Rule cache threads
    - ✓ Match incoming packets against cached rules.
  - Full software classifier threads
    - ✓ Use the entire rule set to carry out classification (with HyperCuts algorithm).
  - Rule cache updater threads
    - ✓ Continuously sample incoming traffic, identify the evolving set of popular rules and update the rule caches.

Overview of Storm

- Data Parallelism
  - By creating multiple instances of each thread and allowing them to operate on different packets.

- Pipeline Parallelism
  - Multiple tasks executed in a specific pre-defined order for each incoming packet.
  - Following a producer-consumer pattern between the two classification stages.

Overview of Storm

- Main contributions of Storm:
  - A practical, multi-threaded, software-only packet classification system.
  - Design of dynamic balancing of computation resources to tasks.

Outline

- Introduction
- Overview of Storm
- Storm Design
  - Thread Assignment
  - Rule Cache Updater
  - Micro Benchmarks
- Experimental Results

Thread Assignment

- Three different tasks threads assignment
  - Rule cache lookup
  - Full software classification
  - Rule cache updating
- Theoretical model

$$\frac{1}{T} = \frac{d_1}{t_1} \times r + \frac{d_2}{t_2} \times (1 - r) + C$$

T: throughput
d_1: average delay of rule cache lookup
d_2: average delay of full software classification
t_1: number of rule cache threads
t_2: number of full software classifier threads
r: average rule cache hit ratio
C: const (queuing delay, synchronization overhead, etc.)

Thread Assignment

- Theoretical model

$$\frac{\partial \frac{1}{T}}{\partial t_1} = -\frac{d_1 r}{t_1^2} + \frac{d_2(1-r)}{(N-t_1)^2} = 0$$

$$t_1 = N\frac{\sqrt{d_1 r}}{\sqrt{d_2(1-r)} + \sqrt{d_1 r}}$$

d_1: average delay of rule cache lookup
d_2: average delay of full software classification
t_1: number of rule cache threads
t_2: number of full software classifier threads
N: $t_1 + t_2$
r: average rule cache hit ratio

- Use only one thread to carry out the sampling and rule cache updating task.

Thread Assignment

- Static thread assignment (on 8-core machines)

$$t_1 = N\frac{\sqrt{d_1 r}}{\sqrt{d_2(1-r)} + \sqrt{d_1 r}}$$

- Control total number of thread around 8
- d_1: about 200 ns (rule cache lookup)
- d_2: about 2000 ns (full classification using HyperCuts)
- r: around 0.95
- minimize synchronization overhead

- Potential thread partitions: 3-3-1, 4-2-1, 2-4-1

Thread Assignment

- Dynamic thread assignment (on 8-core machines)
  - Initially create 4 rule cache threads and 4 full software classification threads.
  - Define the total buffer size in between to be B; define two thresholds, r1 and r2, where $0 < r1 < r2 < 1$.
  - Depending on the available buffer size, switch between 4-2-1 (if $< r1 \times B$) and 2-4-1 ($> r2 \times B$).

Rule Cache Updater

- A rule cache updater thread is responsible for periodically sampling the incoming packets and updating the rules in the rule cache.

- Each entry in the rule cache stores an **evolving rule**.

Rule Cache Updater

Table 3: A simple ruleset on 2 fields.

Rule	*Field*$_1$	*Field*$_2$	Action
$R0$	1-9	4-10	$action_0$
$R1$	7-14	3-8	$action_1$
$R2$	3-11	1-6	$action_2$
$R3$	0-15	0-11	$action_3$

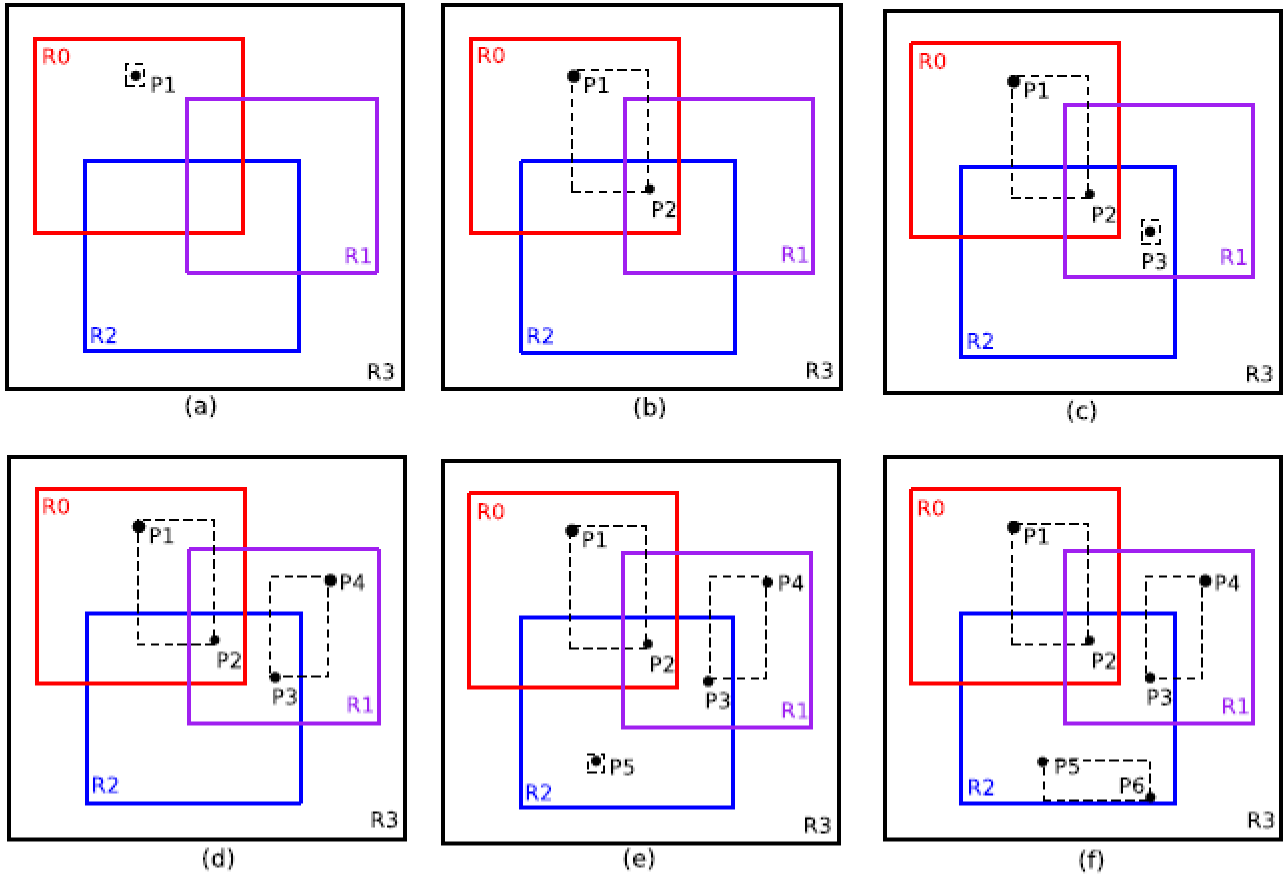


Figure 4: Constructing evolving rules, rule priority: $R0 > R1 > R2 > R3$, $action_0 \neq action_1 \neq action_2 \neq action_3$.

Rule Cache Updater

- Sliding window
  - A FIFO queue that stores recently sampled packets.

- Evolving rule list
  - A list of proposed evolving rules waiting to be checked for conflicts and to be transferred into rule cache;
  - Each evolving rule includes a weight field;

Rule Cache Updater

- Five properties of evolving rules:
  - Represents a d-dimensional hypercube.
  - Be associated with a single action consistent with the original ruleset.
  - Each sample packet in the sliding window is assigned to one evolving rule that matches it.
  - Evolving rules either have the same action or are non-overlapping.
  - Lies entirely inside one of the rules in the original ruleset.

Rule Cache Updater

- Check conflicts
  - Start with root
  - Runs recursively on overlapping child node until a leaf node is identified
  - Check each rule in the leaf node is a match or a conflict

Algorithm 1 CheckConflict(Cnode, ExpandedRule, MatchID, ConflictID)

```
if Cnode is a leaf node then
    for each rule r in the rule list of Cnode do
        if r.ID > min(MatchID, ConflictID) then
            return
        end if
        if r overlaps with ExpandedRule then
            if r.action ≠ ExpandedRule.action then
                ConflictID = r.ID
                return
            end if
            if ExpandedRule lies entirely inside r then
                MatchID = r.ID
                return
            end if
        end if
    end for
end if
if Cnode is not a leaf node then
    for each child c of Cnode that overlaps with ExpandedRule do
        CheckConflict(c, ExpandedRule, MatchID, ConflictID);
    end for
end if
```

Rule Cache Updater

- Check conflicts

Table 1: A simple example with 8 rules on 5 fields

Rule	F_1	F_2	F_3	F_4	F_5	Action
R0	000*	111*	10	*	UDP	$action_0$
R1	000*	10*	01	10	TCP	$action_1$
R2	000*	01*	*	11	TCP	$action_0$
R3	0*	1*	*	01	UDP	$action_2$
R4	0*	0*	10	*	UDP	$action_1$
R5	000*	0*	*	01	UDP	$action_1$
R6	*	*	*	*	UDP	$action_3$
R7	*	*	*	*	TCP	$action_4$

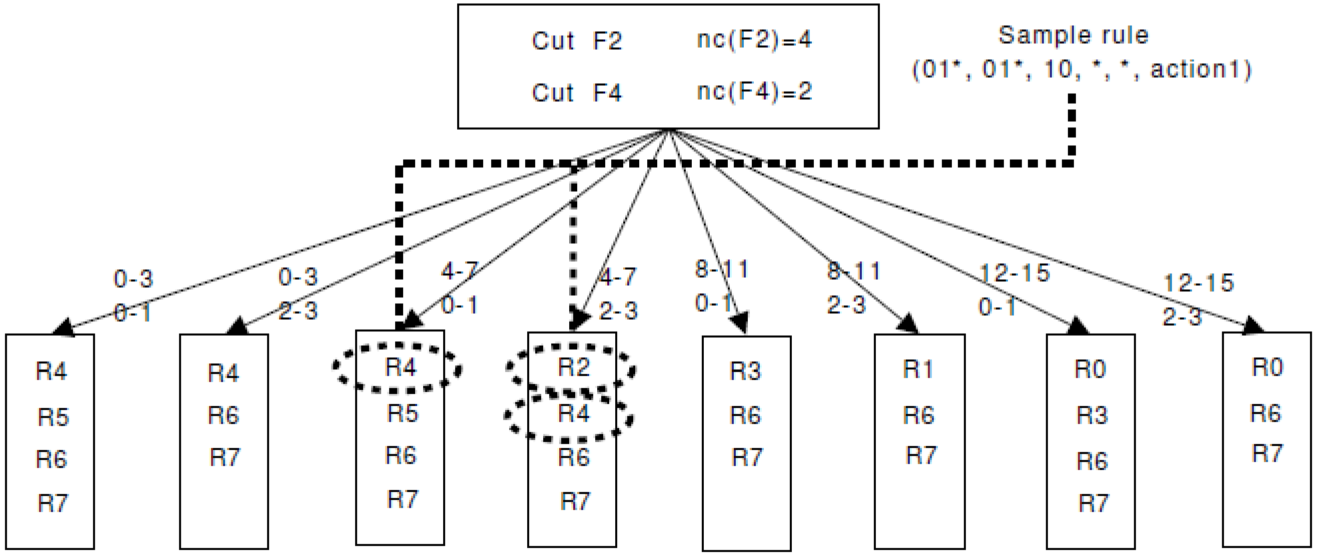


Figure 5: Check an expanded rule for conflicts using HyperCuts decision tree.

Rule Cache Lookup

- How to lookup rules in rule cache?
  - Use linear search to search rules in cache to find a match or cache miss.

- What the rule cache size should be?

Table 5: Performance with different cache sizes on ruleset R3

Cache size	Delay(ns/p)	Hit ratio(%)
10	95.21	93.92
15	93.27	95.31
20	82.01	96.55
25	83.28	95.92
30	96.52	96.52

Experimental Results

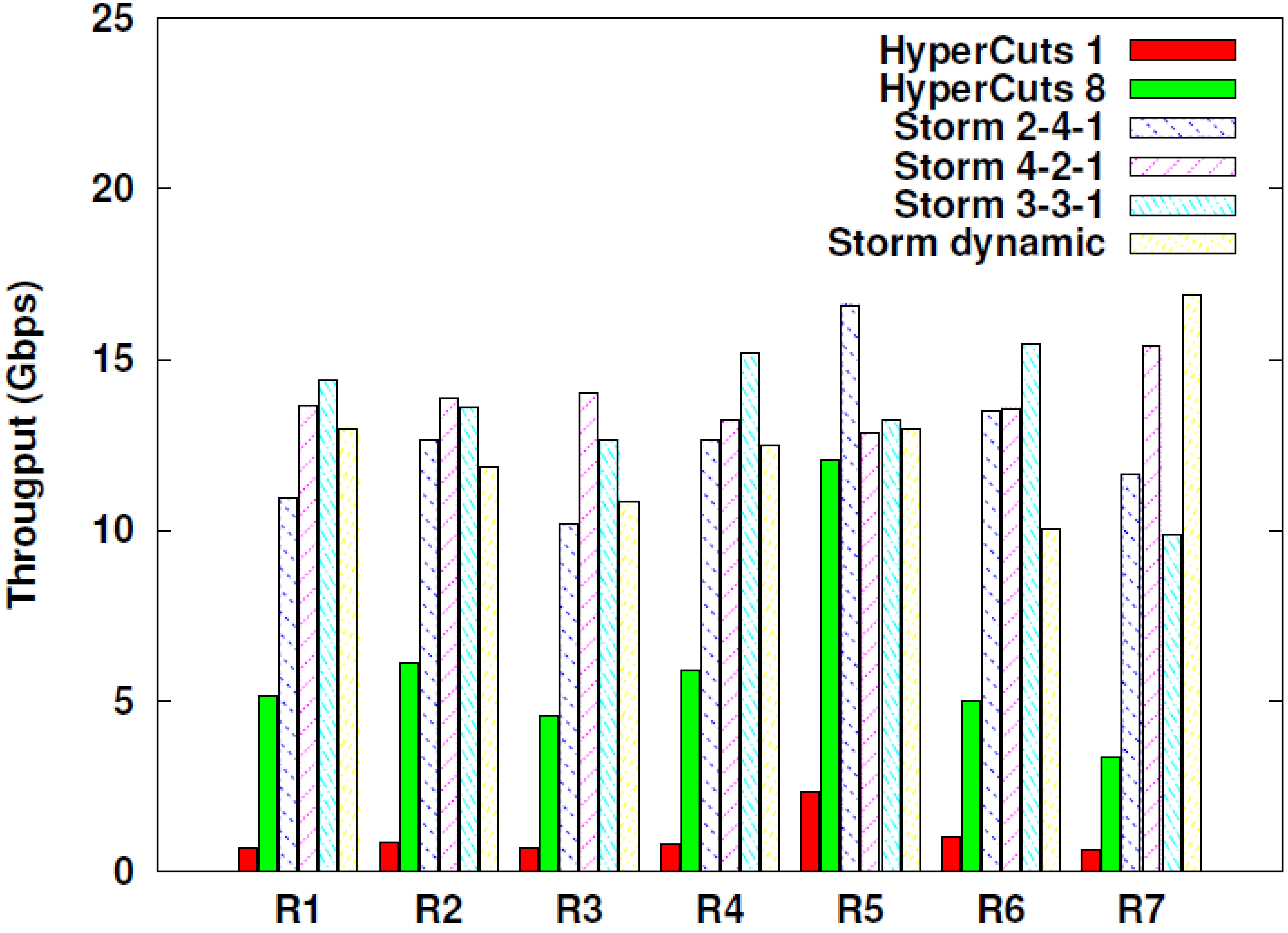
- Rulesets

Rule set	Type	Size
R1	real	460
R2	real	711
R3	real	852
R4	real	1036
R5	real	1802
R6	synthetic	4415
R7	synthetic	9603

  - 5 tuples; action is either permit or deny.
- Traces
  - Five traces each contain about 7M packets (packet size: 128 bytes).
  - Different incoming traffic rates.

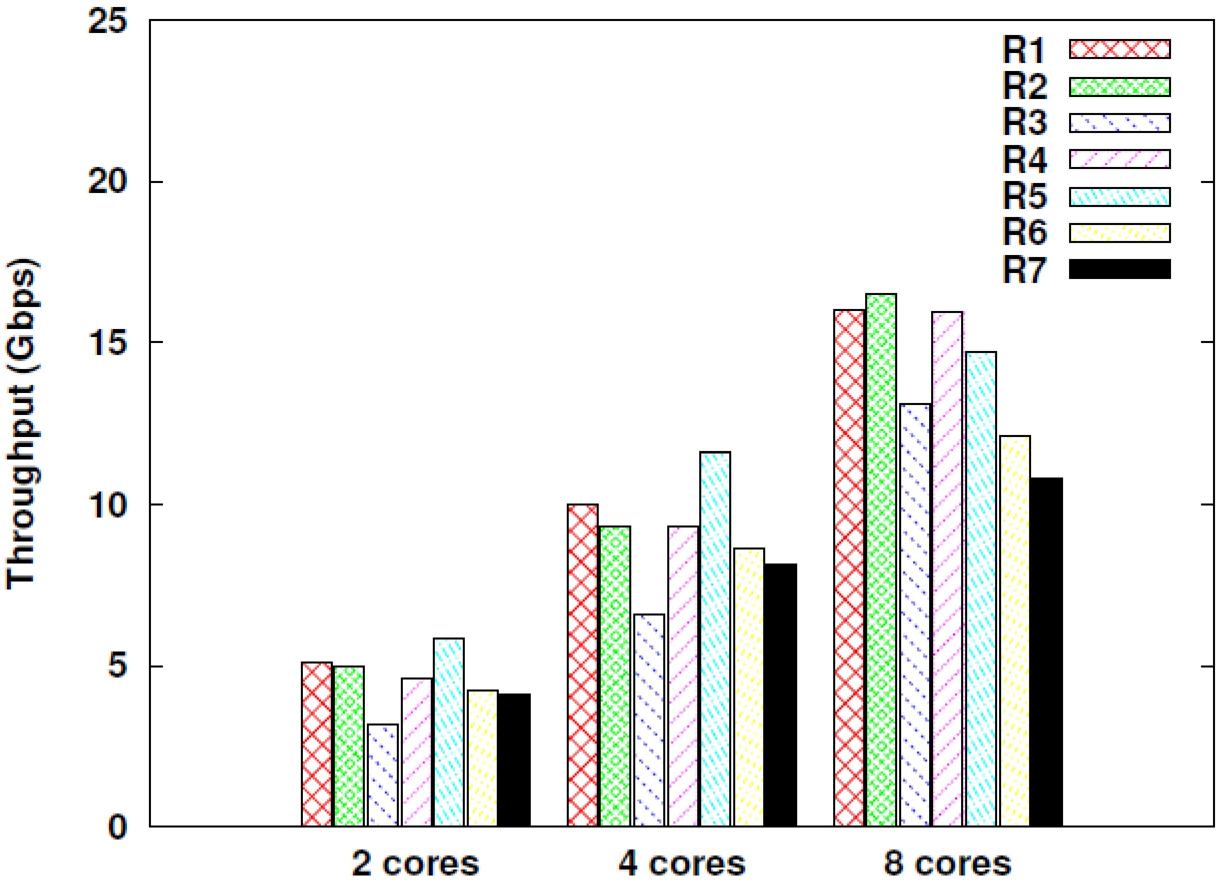
Experimental Results

- Maximum achievable throughput



Experimental Results

- Scalability with number of cores



Experimental Results

- Cache hit ratio

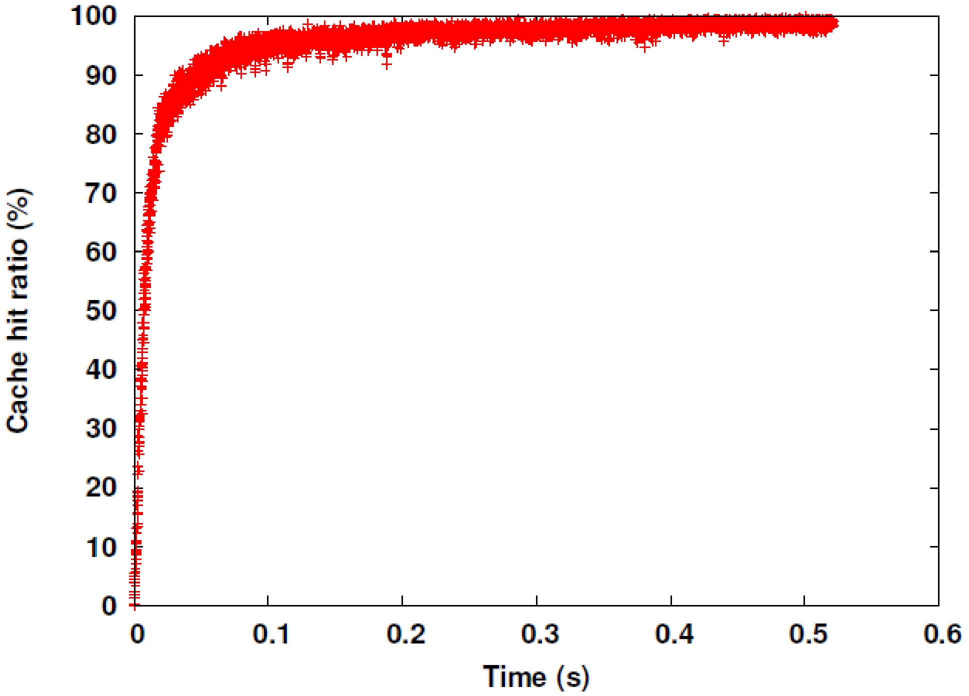


Figure 8: Storm's rule cache hit ratio ramps up quickly (example uses ruleset R3).

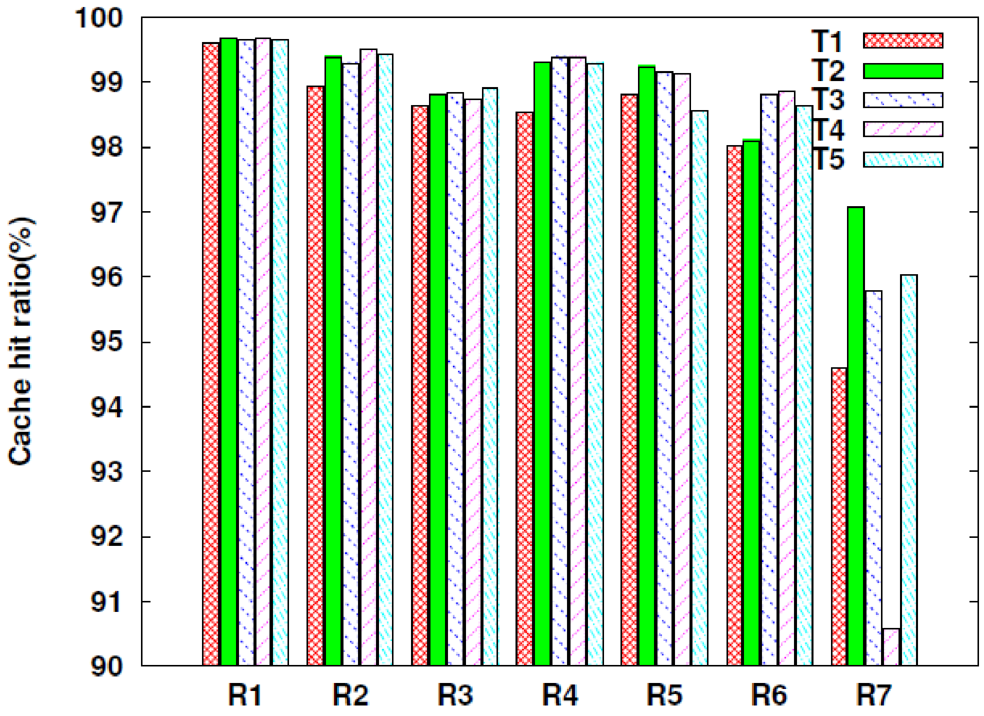


Figure 9: Cumulative rule cache hit ratios of rulesets R1 through R7 with traces T1 through T5 (Y-axis starts at 90%).

Experimental Results

- Micro benchmarks

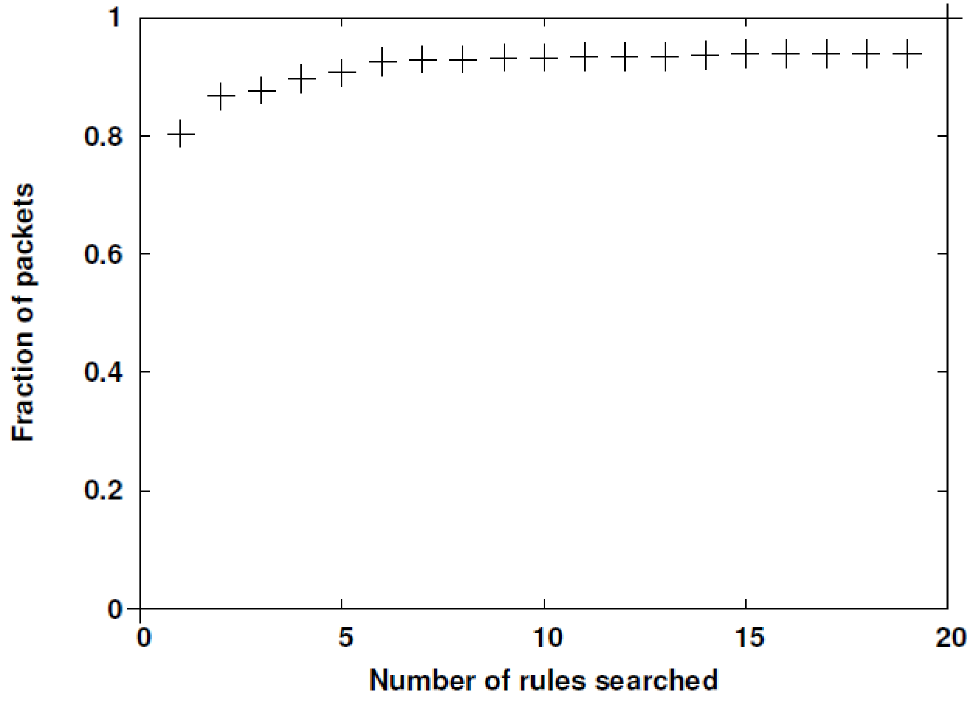


Figure 10: Cumulative distribution of number of entries searched in Storm's rule cache for 1000 randomly sampled packets (using ruleset R2).

Experimental Results

- Micro benchmarks

Table 6: Warmup time (in ms) of the seven rulesets using thread partition 2-4-1 and 4-2-1 for Storm.

Rule set	Storm 2-4-1	Storm 4-2-1
R1	3 ms	5 ms
R2	2 ms	4 ms
R3	10 ms	13 ms
R4	6 ms	12 ms
R5	1 ms	1 ms
R6	8 ms	26 ms
R7	36 ms	79 ms

Conclusions

- **Storm:** a software-based solution to the multi-dimensional packet classification problem.

- A new software system which
  - takes existing classification algorithms;
  - utilizes a common idea of caching;
  - partitions critical tasks into multiple threads that effectively leverage desktop platforms to meet various computation and memory access needs.

Thank You!