# Approximation Algorithms for Dependency-Aware Rule-Caching in Software-Defined Networks

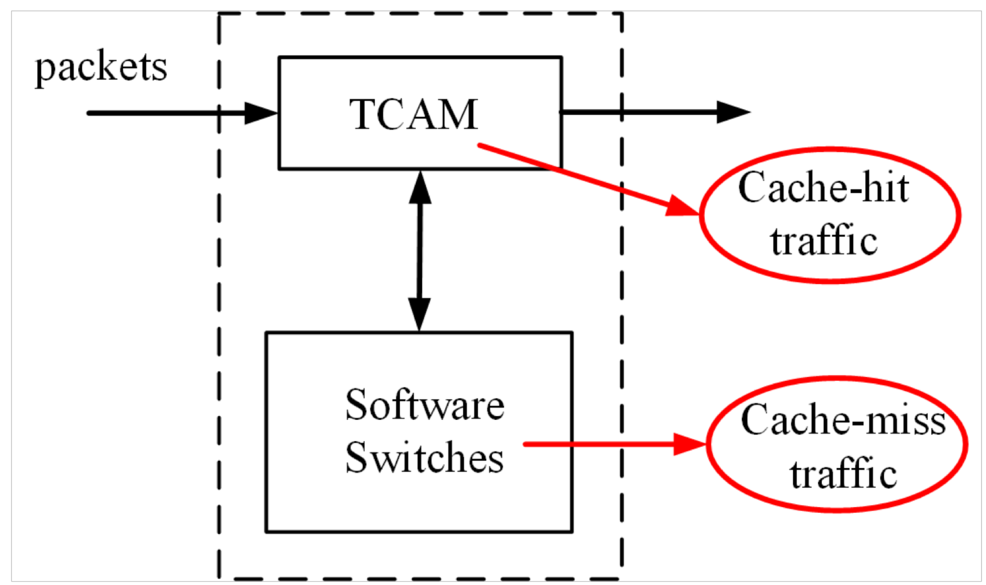
Jie Wu, Yang Chen and Huanyang Zheng
Center for Networked Computing
Temple University, USA

# 1. Introduction of Rule Caching

- Rule caching
  - Install packet-processing rules in switches
  - Switch types

| Switch types | Pros | Cons |
|---|---|---|
| Hardware | Fast (>400 Gbps) | Small in capacity (2K~10K) |
| Software | Large in capacity | Slow (40 Gbps) |

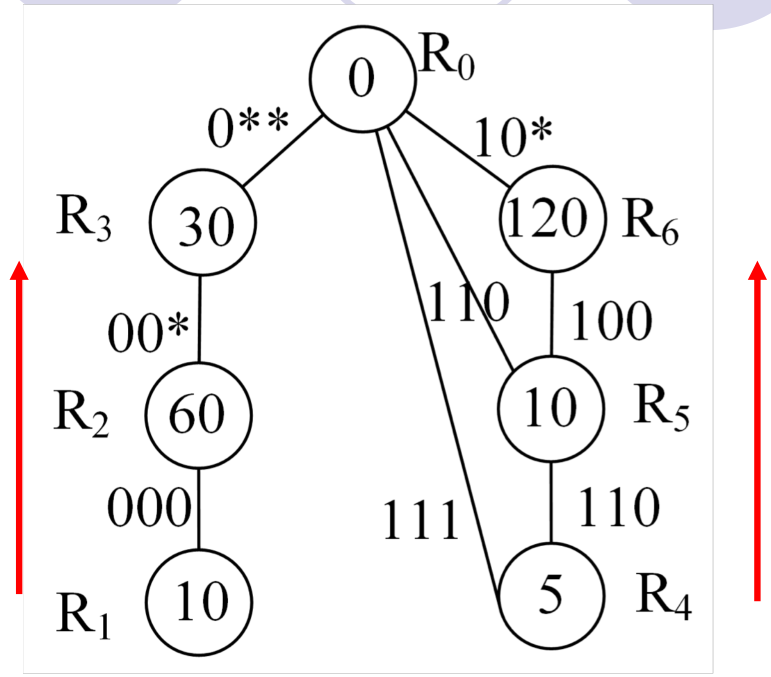
  - Hardware: Ternary Content Addressable Memory (TCAM)
  - Software: Software-based switches



# General Rule Matching Problem

Rule Table

| Rule | Code | Priority | Weight |
|---|---|---|---|
| $R_1$ | 000 | 6 | 10 |
| $R_2$ | 00* | 5 | 60 |
| $R_3$ | 0** | 4 | 30 |
| $R_4$ | 11* | 3 | 5 |
| $R_5$ | 1*0 | 2 | 10 |
| $R_6$ | 10* | 1 | 120 |

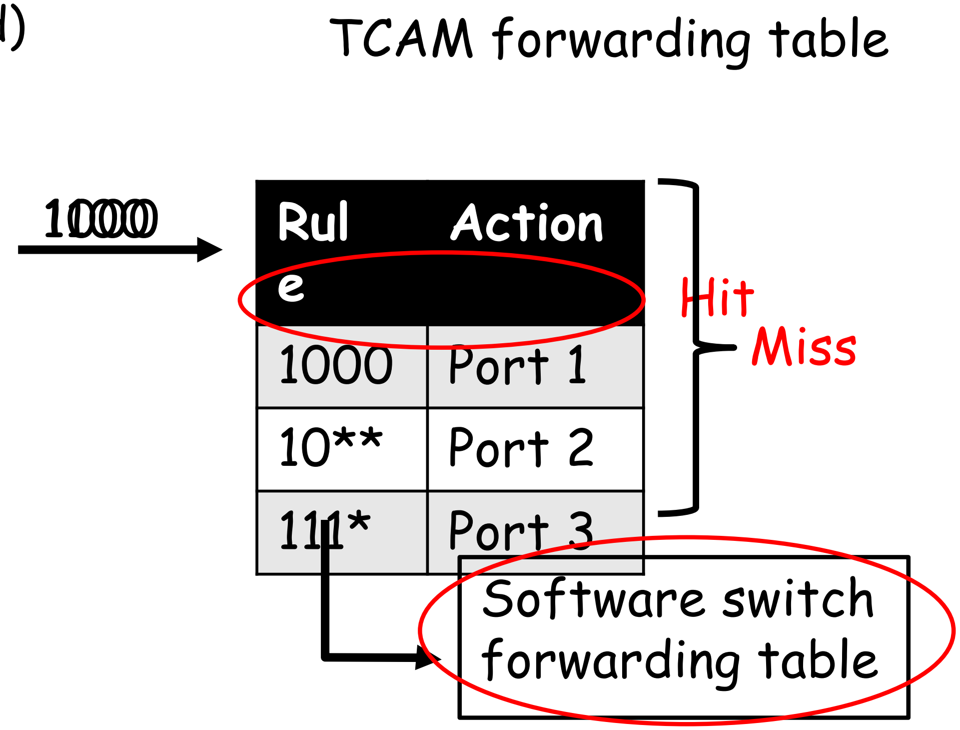



- Rule dependency graph
  - Use of wild card * to reduce rule number
  - Directed acyclic graph: rule and all its decedents (to be in cache)
- Maximum traffic-hit by placing no more than k rules
  - NP-hard [1]

[1] Cacheflow: Dependency-aware rule-caching for software-gdefined networks (SOSR'16)

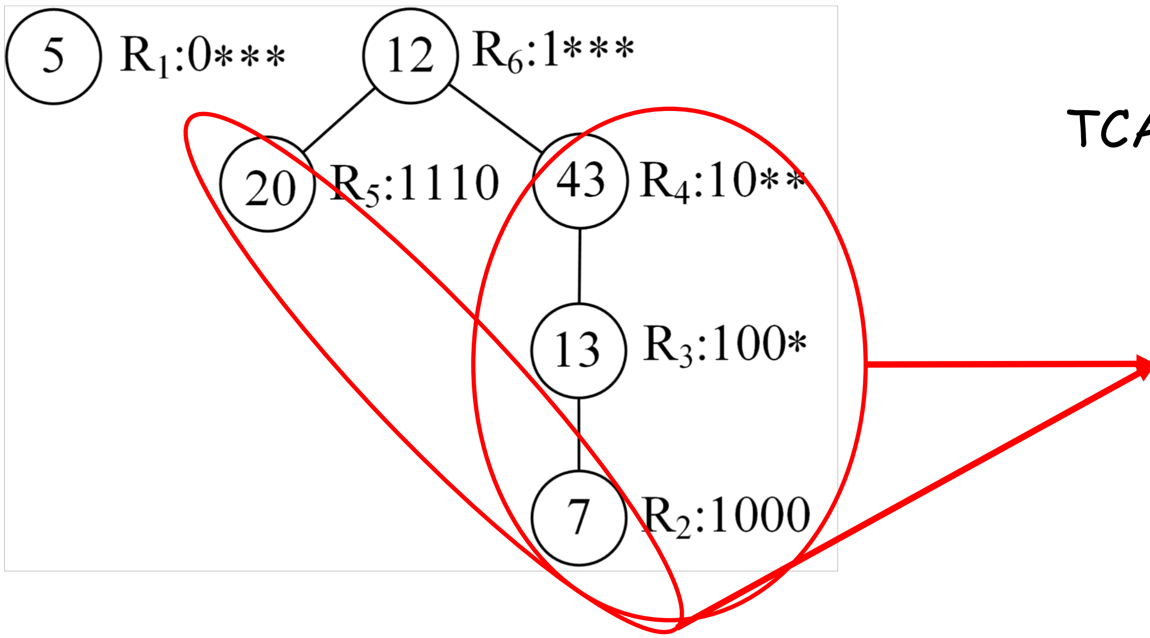
# Efficient Rule Caching

- Assumption
  - Prefix coding to reduce rule number (optimal coding [2] is hard)
  - All rules form a forest of trees
- Constraint
  - Descendant constraint
  - Limited number of cached rules k
- Objective
  - Maximize number of rules hits




[2] Explicit path control in commodity data centers: Design and applications (ToN'16)

# A Motivating Example




TCAM forwarding table

| Rule | Action |
|---|---|
| Rule | Action |
| | |
| | |
| | |
| | |
| | |

k=3

With maximum hit

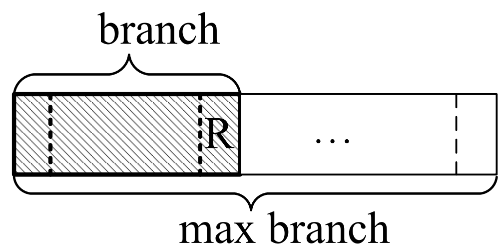
# 2. Solutions

## Greedy Solution One (Branch)

- Definition
  - Branch (which includes fork)
    - A rule and all its descendants
  - Max branch
    - If it meets either of the two conditions:

      (1) Branch size is k

      (2) If size < k, not a branch of another branch with a size of k or less
  - Maintaining max branches will include all cacheable branches



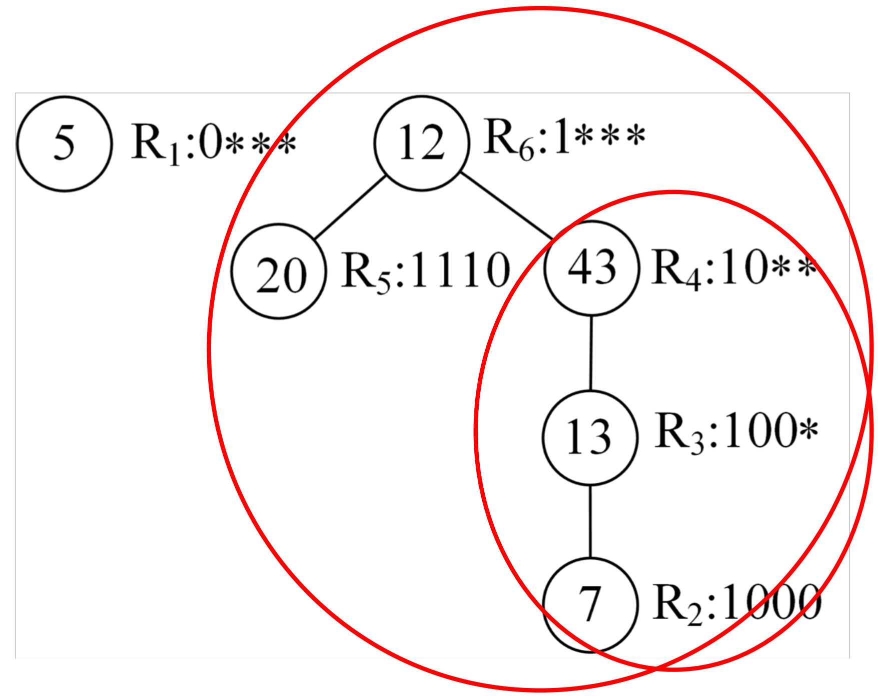

Branch in a max branch

| Definition | Explanation |
|---|---|
| Unit cost $C$ | Each rule has a unit cost |
| Weight W | Rule hits |
| Unit benefit $\Delta W/\Delta C$ | Ratio of rule weight to rule cost |

# Greedy Solution One

- Steps
  - Select the branch with the maximum unit benefit ($\Delta W / \Delta C$)
  - Update unit benefit values of other branches
  - Use a heap to maintain max unit benefit for each max branch
- Time complexity

  $O(n + k \log n + k^2)$
  - n: rule number
  - k: cache size
- Approximation ratio: 2
  - First i items *vs* i+1th item




k=3

k=5

Optimal unit benefit

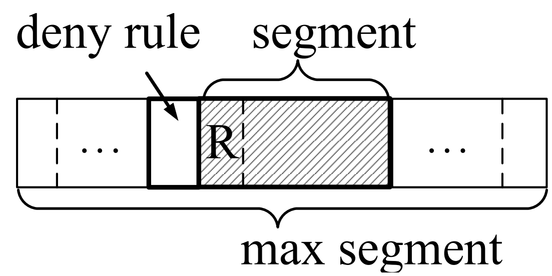
(43+13+7)/3=21

(43+13+7+12+20)/5=19

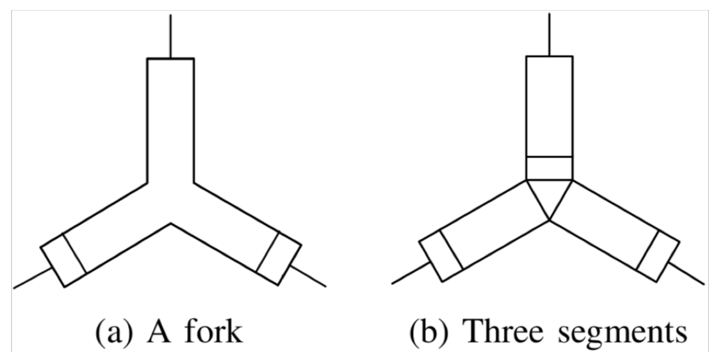
# 2. Solutions (cont'd)

## Greedy Solution Two (Segment)

- Definition
  - Segment
    - Cut off a branch
  - Deny rule
    - A dummy rule to forward to the software switch
    - Cut branches with low-weights
    - Unit benefit ($\Delta W / \Delta C$+1)
- We only consider segments without a fork
  - To avoid non-polynomial number of choices



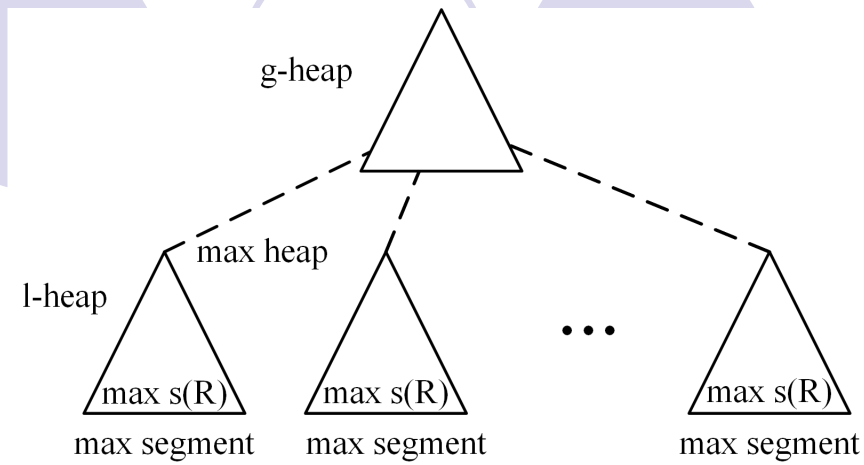
Segment in a max segment



Converting a folk into multiple segments

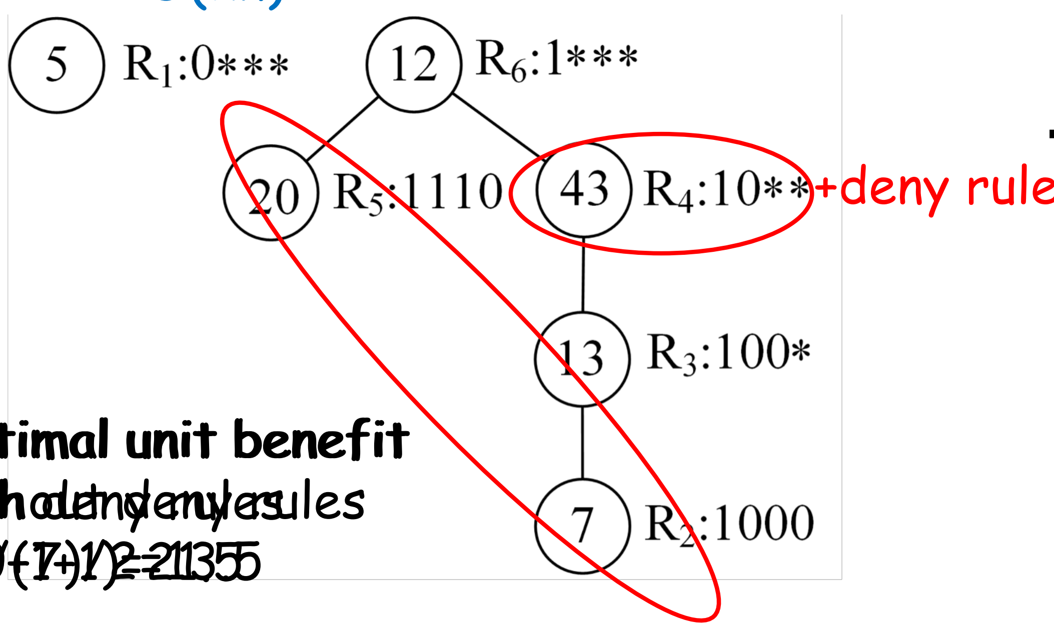
# Greedy Solution Two

- Steps
  - Select the max segment with the maximum unit benefit
  - Update unit benefit values of other segments
  - Use two heaps to maintain segments
- Time complexity:
  - O(kn)

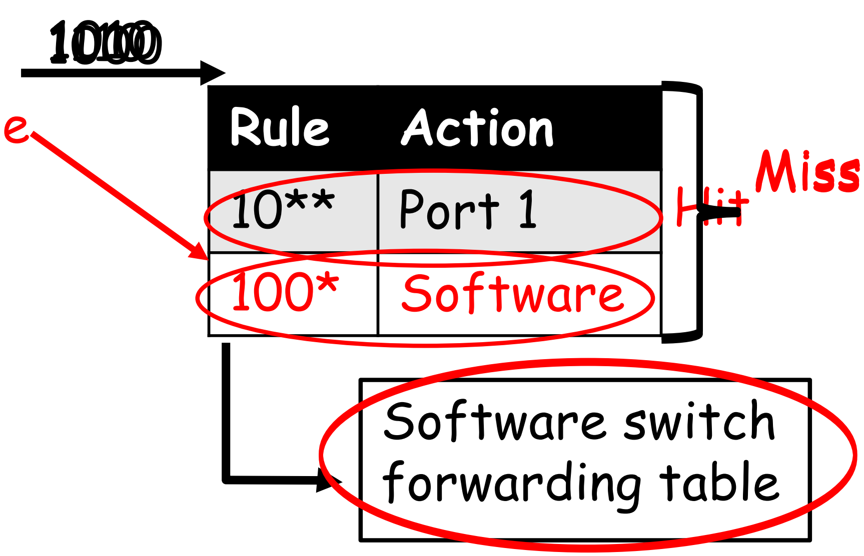



Constructing the global heap (g-heap) from the max heaps of local heaps (l-heaps)

k=2



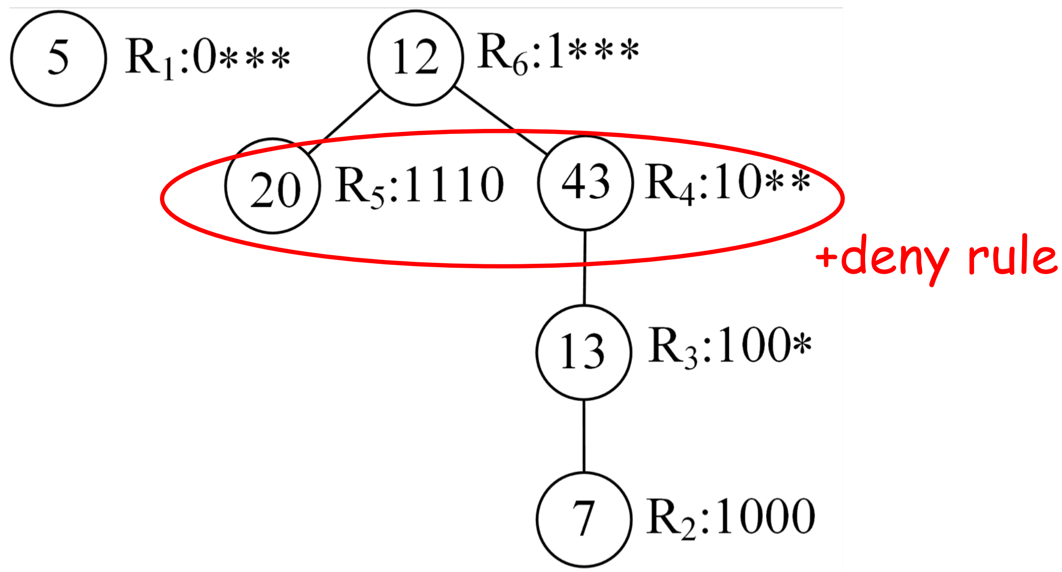

**Optimal unit benefit**
**with deny rules** / without deny rules
43/(1+1)=21.5 / (20+7)/2=13.5



# 2. Solutions (cont'd)

## Combined Greedy Solution

- Insight
  - Combine the two greedy solutions
  - Use branch and segments with the same criterion
    - Maximum unit benefit
    - Each maintains its own heap
- Time complexity
  - O(kn)
- Approximation ratio
  - 24/5

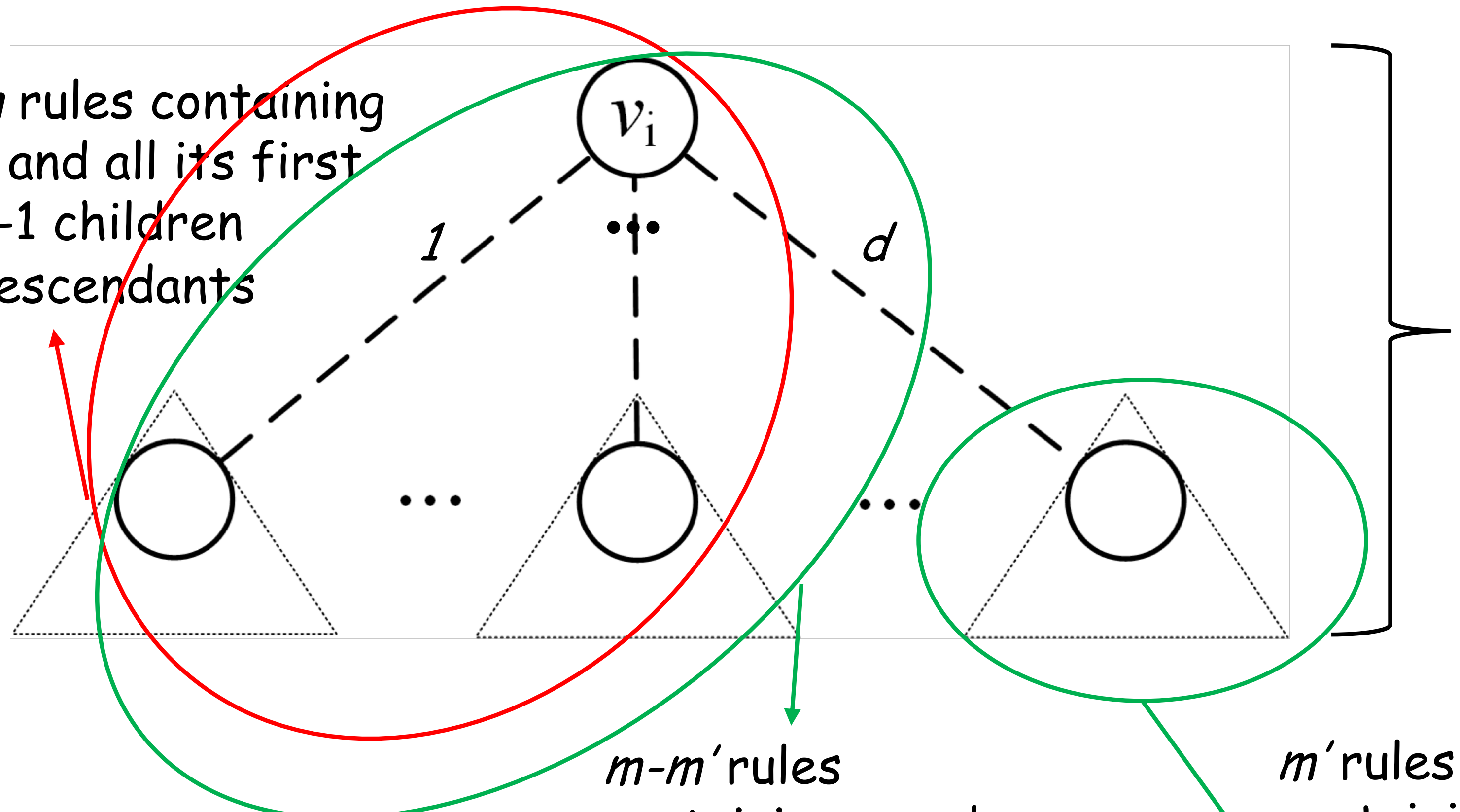



k=3

Optimal unit benefit with deny rules (43+20)/(2+1)=21

# 2. Solutions (cont'd)

## Dynamic Programming (DP) Solution




- Time complexity
  - $O(k^2n)$

# Dynamic Programming Solution (cont'd)

- T[R,d]
  - Subtree of rule R, and its first d children's subtrees
  - Depth-first-search
- O[R,d,m]
  - Optimal cache-hits by caching m rules in T[R,d]
  - $O[R_0, d(R_0), k]$

    Our objective
    - $R_0$: tree root
    - $d(R_0)$: all $R_0$'s children
- Initialization

$$O[R_i, 0, m] = \begin{cases} W_i & \text{if } m \geq 1 \text{ and } i \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

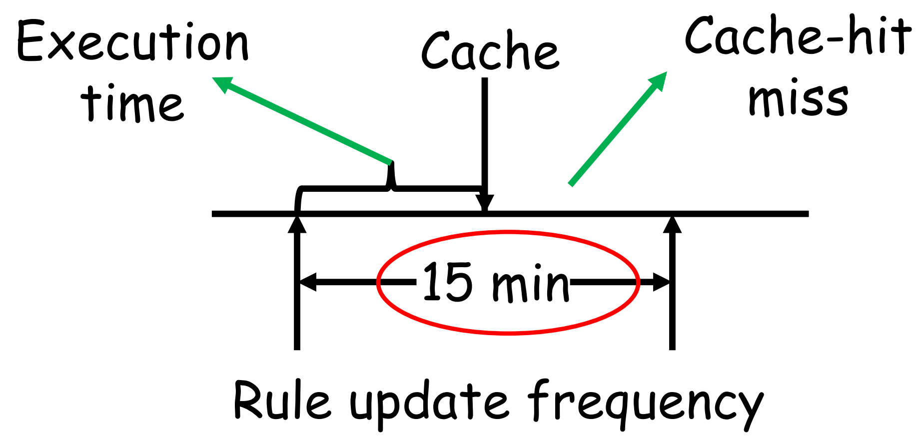
- Formulation

$$O[R_i, d, m] = \max \Big\{ O[R_i, d-1, m],$$
$$\max_{0 \leq m' \leq m} \Big[ O[R_{id}, d(R_{id}), m'] + O[R_i, d{-}1, m{-}m'] \Big] \Big\}$$

  - $R_{id}$: *d*-th child of $R_i$

# 7. Simulation

- Comparison algorithms [1]
  - Dependent
    - Branch without using heap
  - Cover
    - Segment without using heaps
- Our algorithms
  - Branch
  - Segment
  - Combined
  - DP (optimal)

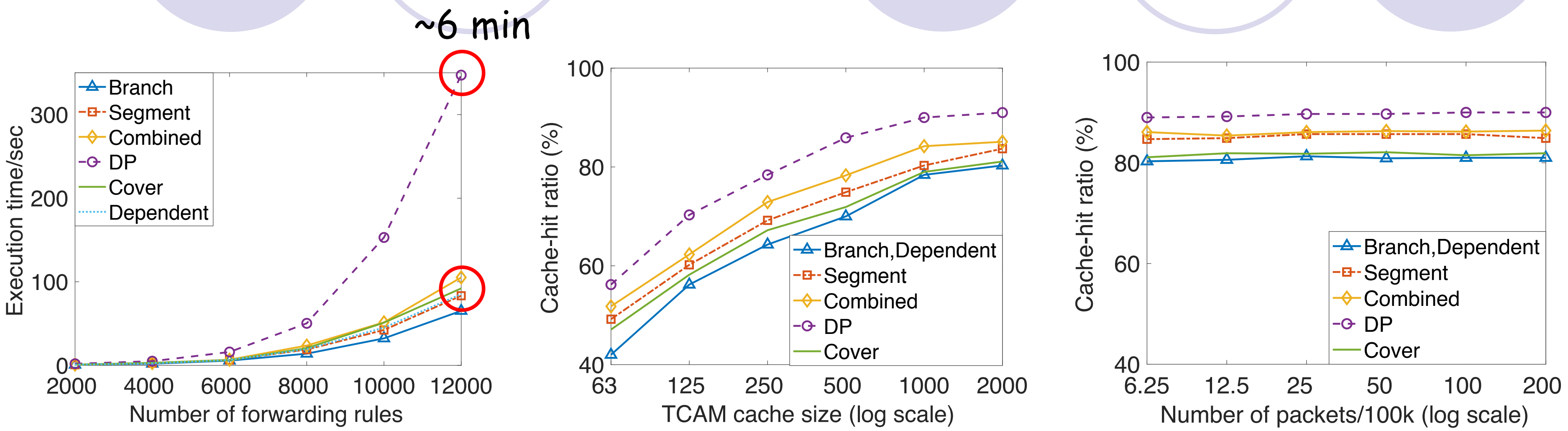



[1] Cacheflow: Dependency-aware rule-caching for software-defined networks (SOSR'16)

# Settings

- Data sets
  - CAIDA packet trace
    - 12,000 forwarding rules
  - Stanford Backbone packet trace
    - 180,000 forwarding rules
- Metrics
  - Execution time
  - Cache-hit ratio with TCAM size
  - Cache-hit ratio with number of packets
- Variables
  - TCAM cache size: k= 63~2000

# Simulation Results

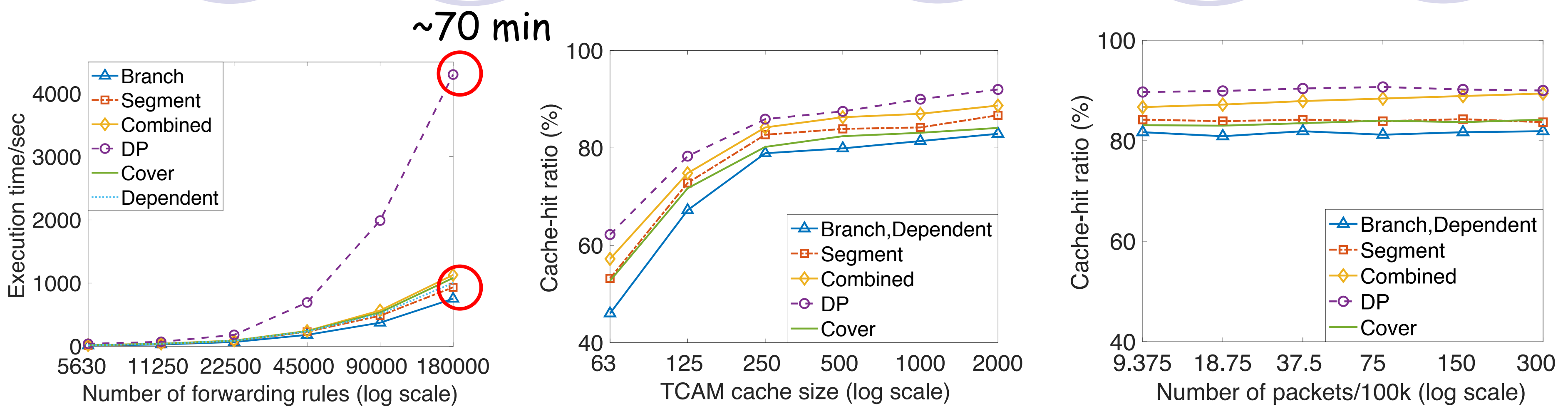



(a) Algorithm execution time.　(b) Cache hit traffic and TCAM size.　(c) Cache hit traffic and the number of packets.

CAIDA packet trace

- DP has a much larger execution time than others
- Branch is faster than Dependent because of using heaps
- Our four algorithms achieve at least a 79.8% hit ratio with 2,000 cache size, which is just 1.1% of the total rule table.
- DP achieves the best cache-hit ratio.

# Simulation Results (cont'd)




(a) Algorithm execution time. (b) Cache hit traffic and TCAM size. (c) Cache hit traffic and the number of packets.

Stanford backbone trace

- More rules result in a much larger execution time
- Our three greedy algorithms achieve better ratios than CAIDA one with the same TCAM size because of deeper dependencies
- For 30 million packets, DP's cache-hit ratio reaches 90.2%, Combined reaches 89.4%, Segment reaches 83.7% and Branch reaches 81.9% with 2,000 cache size.

# 8. Conclusion

- Hardware and software switches
- Caching technology
  - Wildcard (*) rule matching
  - Rule dependency constraints
  - Deny rule
  - limited number of rules in TCAM
- Objective
  - Maximize cache-hit ratio
- Solutions
  - Three greedy algorithms with approximation ratios
  - Optimal DP solution

Q & A