FastUp: Fast TCAM Update for SDN Switches in Datacenter Networks

Ying Wan[1], Haoyu Song[2], Hao Che[3], Yang Xu[4], Yi Wang[5], Chuwen Zhang[1], Zhijun Wang[3], Tian Pan[8], Hao Li[7], Hong Jiang[3], Chengchen Hu[6,7], Bin Liu[1]

[1]Tsinghua University, China, [2]Futurewei Technologies, USA
[3]University of Texas at Arlington, USA, [4]Fudan University, China
[5]Southern University of Science and Technology, China,
[6]Xilinx, Singapore, [7]Xi'an Jiaotong University, China
[8]Beijing University of Posts and Telecommunications, China
Email: wany16@mails.Tsinghua.edu.cn

Tsinghua University

Outline

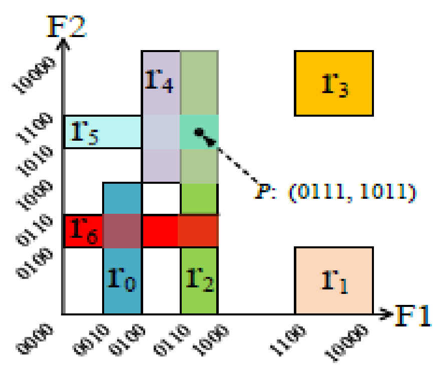
- Background and related work
- The principle of *FastUp*:
- Analysis of the optimal TCAM update
- Performance evaluation
- Conclusion

Background —— TCAM for rule tables

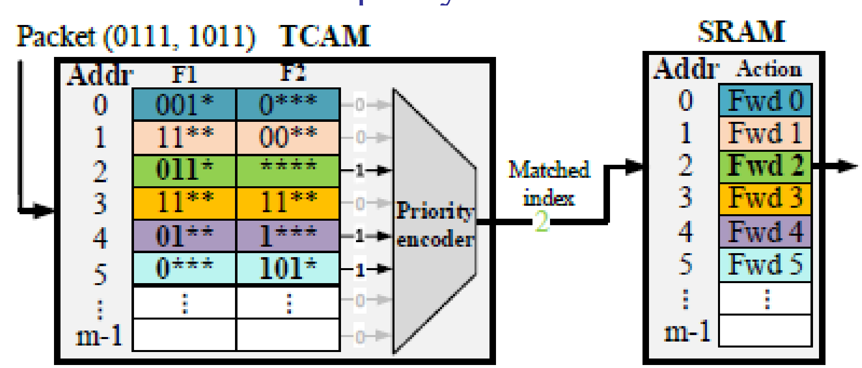
- **The *de facto* industry standard for rule tables**
  - line-speed lookup speed: compare a key with all rules in parallel
  - flexible matching pattern: LPM, EM, and RM
- **The placed rules are arranged in priority order**
  - TCAM always returns the first matched rule (i.e., with the lowest physical address)
  - It is the highest-priority matched rule that counts
  - E.g., packet p matches r2, r4, and r5, but the lowest one r2 is returned
- **A new rule insertion incurs the moves of existing rules**
  - R6 overlaps with r0 and r2. r6 has a higher priority than r2 and has a lower priority than r0
  - Up to 10 ms on P4 switch to insert a new rule when the TCAM capacity is set to 4K entries

Rule	Pri	Match		Action
		F1	F2	
r_0	9	001*	0***	Fwd 0
r_1	7	11**	00**	Fwd 1
r_2	6	011*	****	Fwd 2
r_3	5	11**	11**	Fwd 3
r_4	2	01**	1***	Fwd 4
r_5	1	0***	101*	Fwd 5

(a) Flow table.



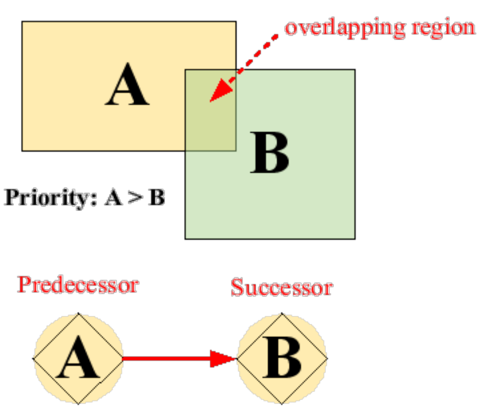
(b) Rule overlapping.

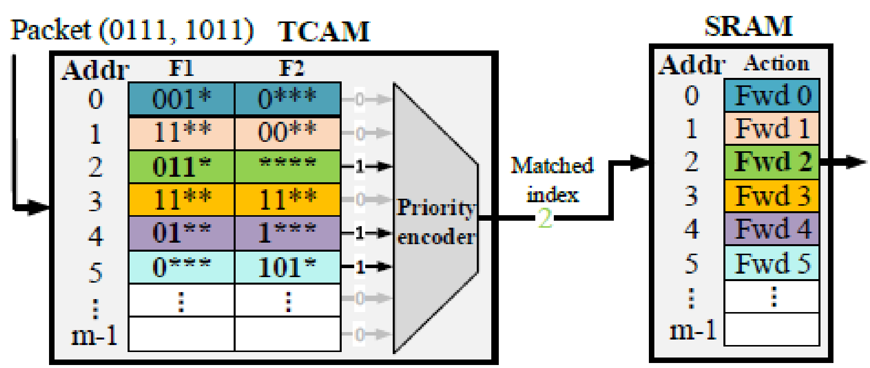


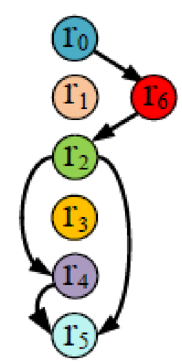
(c) Implementation of the flow table in TCAM and SRAM.

Background —— TCAM update problem

- **Rule relation graph**
  - each node represents a rule and each edge represents the overlapping relationship
  - each rule must be placed above all its descendants and below all its ascendants.
    - R’s predecessor: the ascendant of r that is the closest to r
    - R’s successor: the descendant of r that is the closest to r
- **Rule moving strategy**
  - A rule can be inserted between its predecessor and its successor
  - A rule r can be moved downward directly to its successor
  - A rule r can be moved downward to any entry between it and its successor.





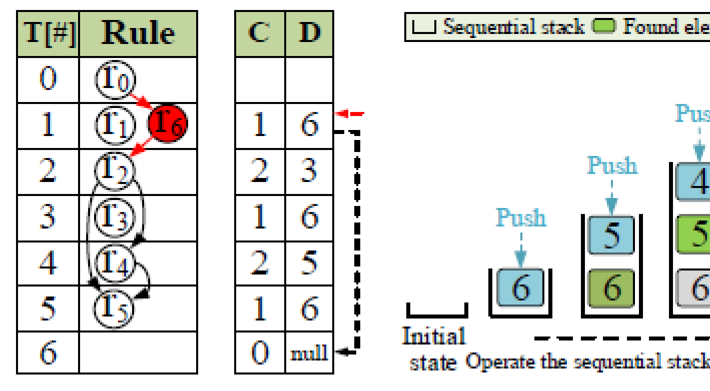


SSA——sequential-stack based algorithm

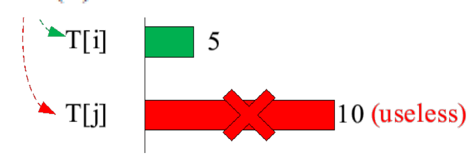
- Definition of moving cost
  - C[i]: the minimum rule moves to relocate the rule in the i-th TCAM entry T[i]
    - Succ(i).addr: the address of the entry of the successor of the rule in T[i]

$$C[i] = \min_{j\in(i,\ succ(i).addr]} \{C[j]\} + 1$$

    - D[i] records the destination entry to relocate the rule currently placed in T[i]
  - Calculation of C[:]/D[:] is typical dynamic programming (DP) process.
- Optimization opportunity
  - If j > i and C[j] > C[i], T[j] will never be the best-c
    - Dynamic identify and remove the useless entries
    - Only Find the best candidate among the entries potentially t
  - Sequential stack S
    - Time complexity: O(m^2)→O(m*lgh)
      - The elements of S is in strict decreasing order.
      - If T[i] is placed in S[p], then C[i] = p.
      - The length of S will never exceed the diameter of rule graph h.
    - Memory footprint: O(m)→O(h)
      - If T[i] is placed in S[p], then D[i]=S[p-1].
      - The best candidate of the update rule must stay in S after calculation.

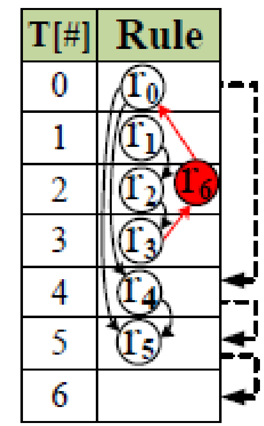


(a) DAG. (b) DPA. (c

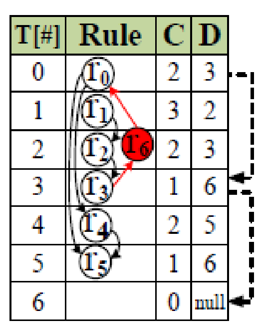


RCA—Rule chain based algorithm for reorder resolution

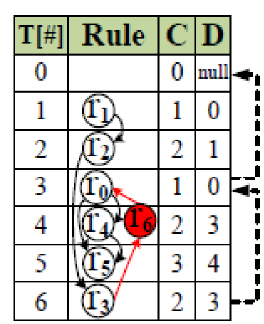
- Definition of reorder problem
  - SSA assumes that a new rule ru can always find one or more candidate locations.
  - If succ(ru).addr<pred(ru).addr, the reorder problem happens.
    - E.g., r3→r6→r0
  - Relocate the out-of-order rules until succ(ru).addr > pred(ru).addr
- Reorder resolution
  - Why not choose SSA:
    - Time complexity and infinite loop
  - RCA:
    - Move one of out-of-order two rules and keep the other one in place
      - R0→r0.succ→r0.succ.succ
    - Guarantee the reorder resolution
      - in one round, if the reorder is not resolved, the gap between the rules is reduced.



(c) *RCA*.



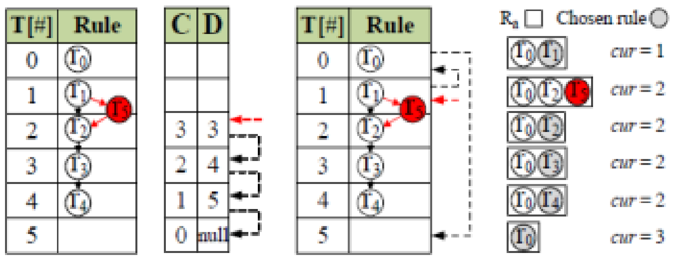
(a) *DPA*.



(b) *DPA*.

BBA—branch-and-bound algorithm for optimal TCAM update

- Definition of optimal TCAM update (OTU)
  - The solution with the theoretically smallest rule moves
- Why exist algorithms fail to find the optimal solution
  - The rules are moved in the fixed and same direction



(a) DAG. (b) *DPA*. (c) OTU. (d) *BBA*.

- The difficulty in achieving OTU
  - any topological order of the DAG is a feasible TCAM placement and vice versa.
    - N, m, N_TT, and N_L are the number of rules, TCAM entries, topological orders and feasible TCAM layouts

$$N_L = N_\pi * \binom{n}{m} = N_\pi * \frac{m!}{n!(m-n)!}$$

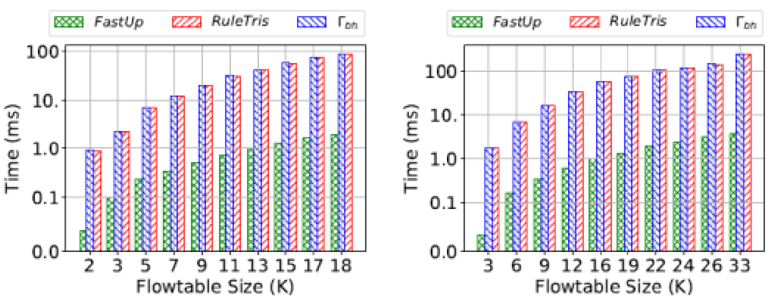
  - just finding the number of all topological orders N_TT has been proven to be NP-hard.
- Why need to find the OTU
  - The degree of optimality, j, for a Design Under Test (DUT) is defined as

$$\lambda_{\mathrm{DUT}} = \frac{N_{\mathrm{OTU}}}{N_{\mathrm{DUT}}} \times 100\%$$

  - can be used to guide further algorithm optimizations.
- How to find the OTU
  - BBA processes each entry from top to bottom
  - BBA tries to place each available rule in the current entry, or leave it empty.
  - BBA avoids searching the space when the cumulative update cost exceeds the found OTU

Performance evaluation

- Experiment setup
  - Compare objects: RuleTris, T_bh, T_down
  - Testbed: a programmable OpenFlow switch——ONetSwitch
  - Dataset：Access Control List (ACL) and Firewall (FW), generated by ClassBench [59].
- Experimental results：
  - Metric: Compute time, interrupt time, reorder efficiency, and optimality degree



(a) ACL (b) FW

$S_2(k)$	#Updates	Average time (ms)			Maximal time (ms)		
		FastUp	Γ_{bh}	Γ_{down}	*FastUp*	Γ_{bh}	Γ_{down}
3.4	340	0.62	0.65	2.31	1.2	1.8	6.6
7.4	740	0.64	0.78	3.16	2.4	3.6	10.8
11.4	1140	0.65	0.85	4.23	2.4	4.2	10.8
15.4	1540	0.70	0.86	4.57	3	4.8	11.4
18.6	1860	0.72	0.95	4.78	3.6	6	18

$S_2(k)$	#Updates	#Reorder	#Resolved		Average time(ms)	
			FastUp	Γ_{down}, Γ_{bh}	*FastUp*	Γ_{down}, Γ_{bh}
3.4	340	1	1	1	13.10	16.20
7.4	740	5	5	5	19.72	21.96
11.4	1140	9	9	8	20.52	23.79
15.4	1540	14	14	13	20.70	24.17
18.6	1860	18	18	16	21.34	25.47

Case	Probability	Interrupt time (ms)					
		FastUp		*BBA*		λ_{FastUp}	
		Avg	Max	Avg	Max	Avg	Max
Normal	97.23%	1.29	4.2	1.24	3.0	96%	71%
Reorder	2.77%	5.20	12.6	2.30	3.0	44%	23%
Mixed	100%	1.40	12.6	1.27	3.0	90%	23%

[1] CacheFlow: Dependency-Aware Rule-Caching for SDN, SOSR 2016, best paper

Conclusion

- **FastUp optimizes both the compute time and interrupt time**
- **FastUp solves the reorder problem efficiently**
- **FastUp is close to the optimal TCAM update**

Thank You!

Q & A