T-cache: Dependency-free Ternary Rule Cache for Policy-based Forwarding

Ying Wan[1], Haoyu Song[2], Yang Xu[3], Yilun Wang[1], Tian Pan[4], Chuwen Zhang[1] and Bin Liu[1]

[1]Tsinghua University, China, [2]Futurewei Technologies, USA

[3]Fudan University, Beijing University of Posts and Telecommunications, China

Email: wany16@mails.Tsinghua.edu.cn

清華大學 Tsinghua University

Outline

- Background
- Rule-isolation
- T-cache architecture
- Evaluation
- Conclusion

Policy in different scenarios

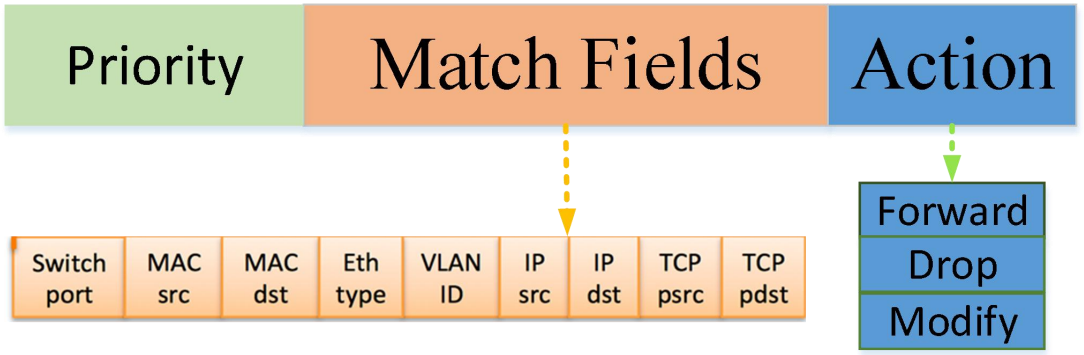
- FIB in Routing (prefix)

Priority	DIP	Next hop
24	192.168.1.0/24	127.0.0.1
32	192.168.1.1/32	10.0.0.1

- Packet classification (rule)

Priority	SIP	DIP	SP	DP	P	Action

- Flow table in SDN (policy)



Challenges in policy-based forwarding

- Lookup speed
  - Ever-increasing network throughput
  - Rule table scale blast
  - Policy dependency
- Uncoordinated Sleeping

Why Zero-Time Wakeup Matters?

- If rou
- Unc
- C

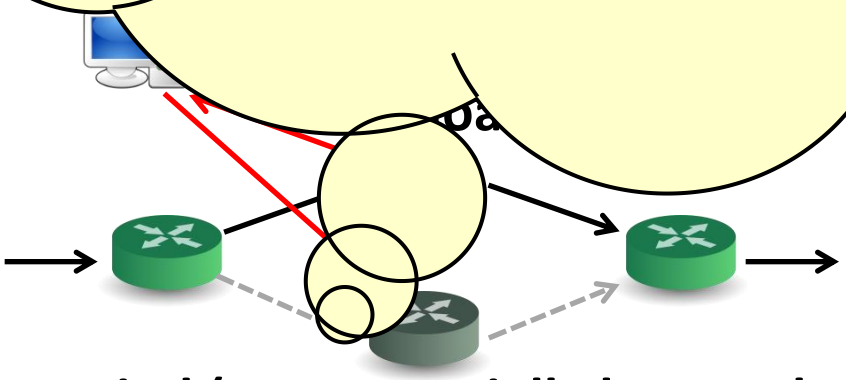
We presume links can be waken up in zero-time!

Hardware can finally support us!

We are working on high-level abstractions!

We leave this to hardware guys!

……



idle period (can potentially be very long)

low!

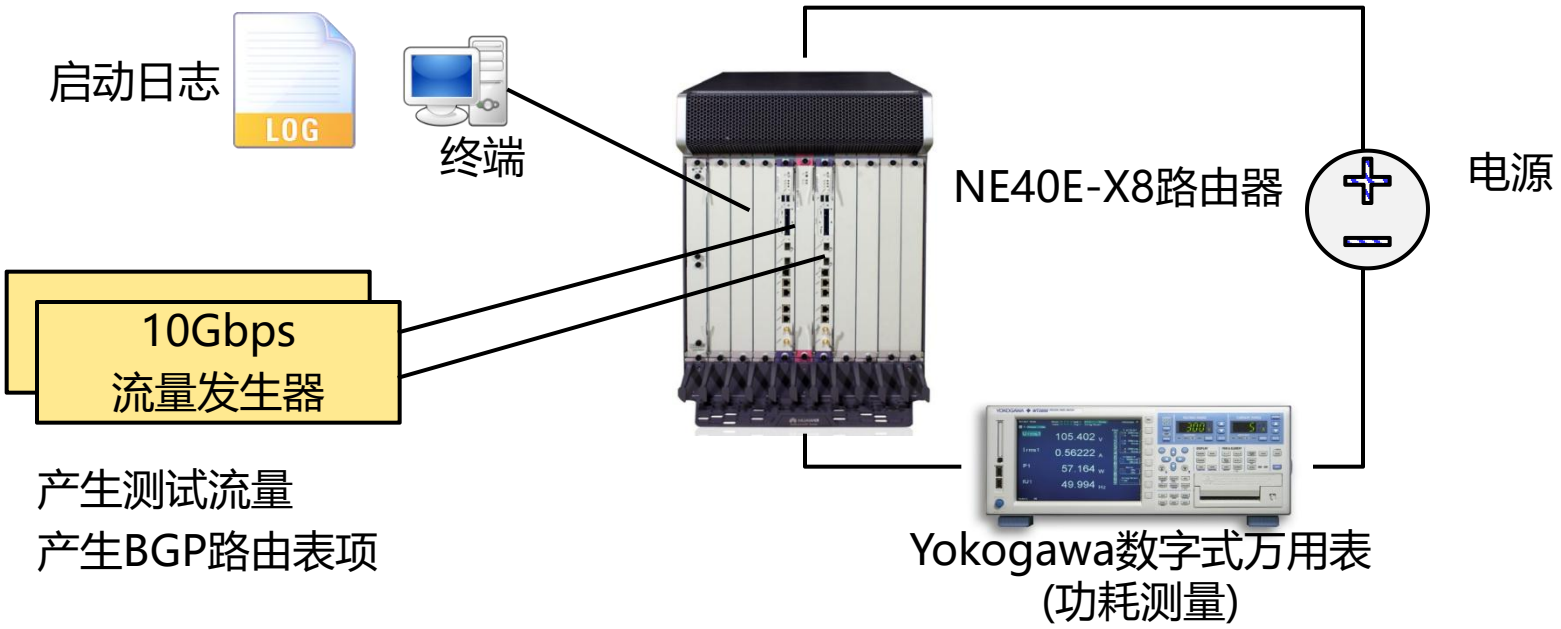
Pa oss

+ Coa grained

power saving!

Measurement with Real Devices

- **Huawei NE40E-X8 (under minimum configuration)**
  - **Time to market: 2010**
  - **Line Card * 2 (each can handle 20Gbps)**
  - **Route Processor Card * 2**
  - **Fans (speed set to 60%), Switch Fabric Card, Power Supply and Backplane**

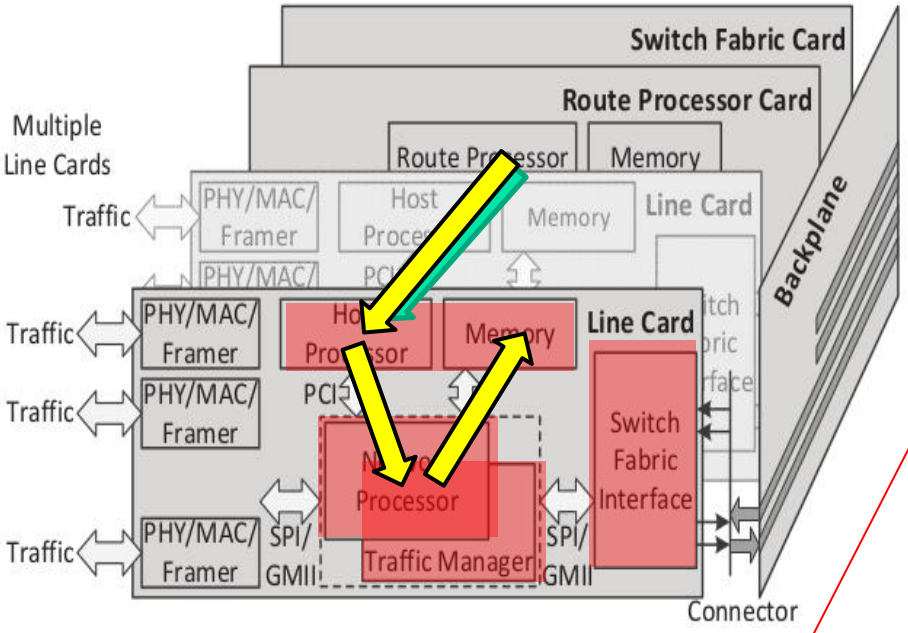


1. 产生测试流量
2. 产生BGP路由表项

Line Card Wakeup Process

- **Wakeup events interpreted from boot log**

Software (VxWorks OS) preparation on host processor (60s)



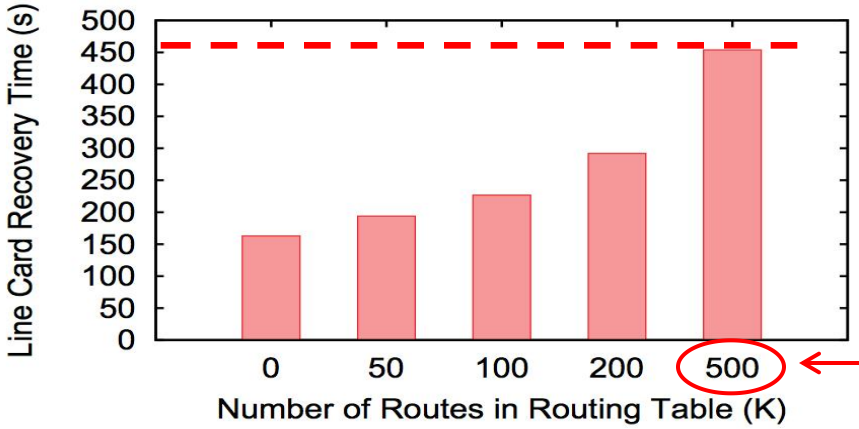
Memory diagnosis and hardware configuration on data plane (75s)

Routing table download into TCAM (157s)

Time	Event
0-12	Line card powered on
12-24	Bootrom used sdram tested
24-42	Software downloaded from route processor
42-60	Vxworks OS initialized in host processor
60-67	Hardware detected on line card
67-84	Traffic manager used sram tested
84-106	Network processor used buffer and bus tested
106-116	On-board ddr3 dram and tcam tested
116-120	Switch fabric interface initialized
120-135	Software tasks on data plane created
135-292	Routing table downloaded (200K routes)

Other Interesting Measurement Results

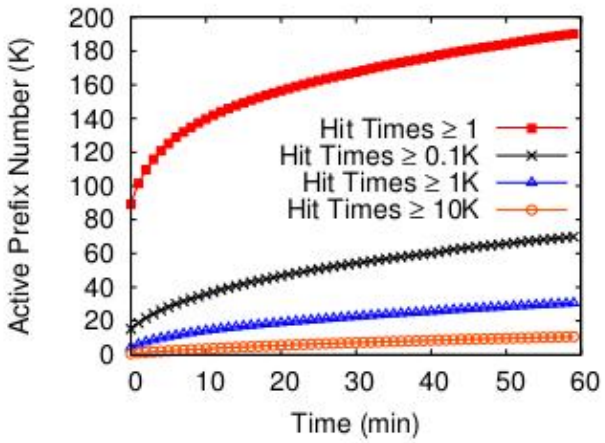
- **Line Card Wakeup Time vs Routing Table Size**

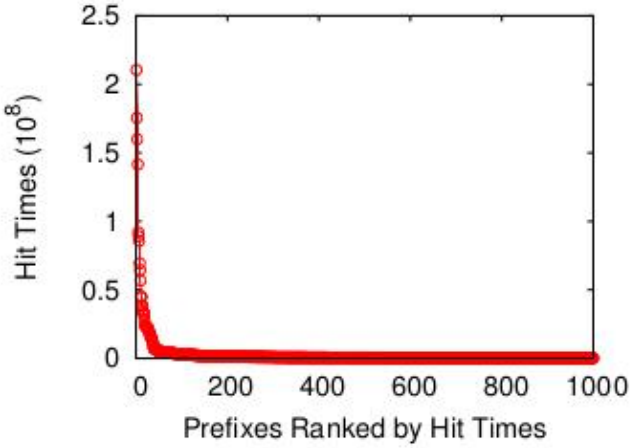


Routing table download dominates line card wakeup process

2014 BGP Table Size (CIDR Report)

- **Power-Law Behavior in Route Lookup**

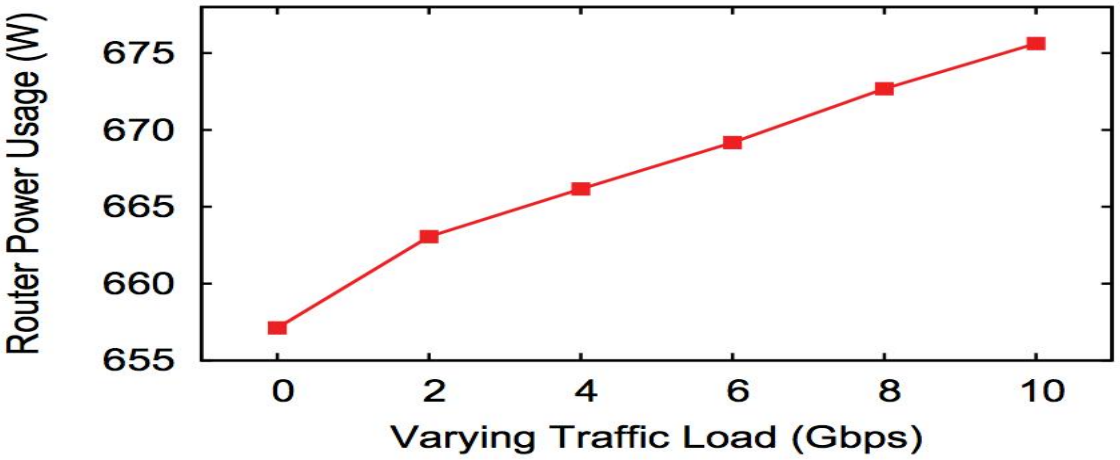




Prefixes with hit times more than 100K only occupy 0.6% of the total prefixes, but dominate 85.2% of the traffic (power-law)

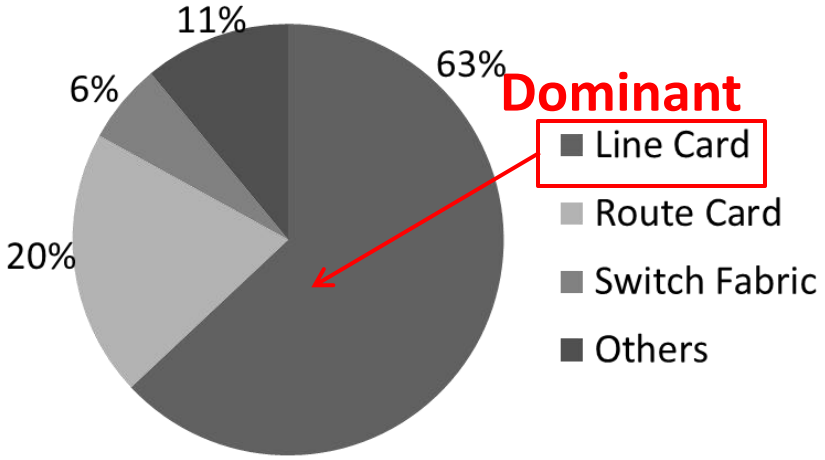
Other Interesting Measurement Results (cont)

- **Power Consumption Under Varying Traffic Loads**

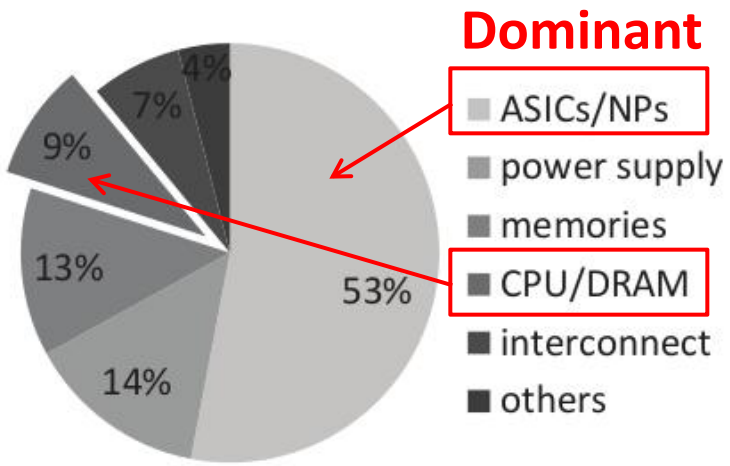


Power changes little under varying traffic loads => Insensitive to the traffic load change to take power saving actions adaptively

- **Power Breakdown of Router and Line Card**



Router Power Breakdown (NE40E-X8)



Line Card Power Breakdown (Cisco Report)

Design Principles

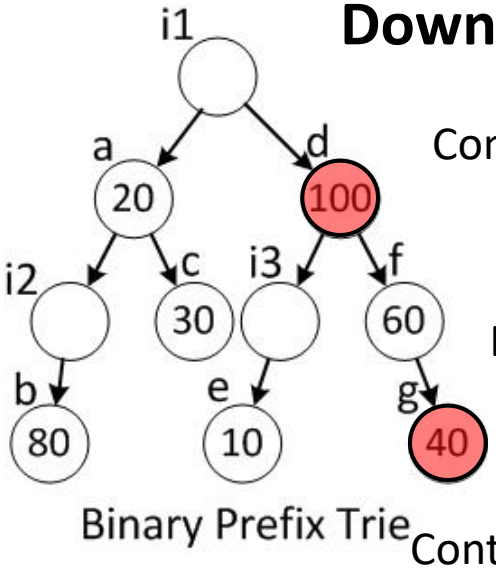
- Keep Host Processor Standby
  - It costs 60s wakeup time with only 9% power consumption
- Skip Memory Diagnosis
  - Memory diagnosis is very time-consuming, but should be a feature only when router is cold started
- Recover Data Plane Tasks Incrementally
  - Packet can be allowed in, once a minimized but essential tasks are done, even though there may be some initialization still going on (e.g., QoS in Traffic Manager)
- Download Popular Routes First, Then Let Traffic in
  - Route download dominates wakeup time (the bottleneck)
  - A slim slot of popular prefixes will cover majority of traffic

The Problem (LPM Violation)

- A straightforward strategy: Download by Popularity
- But this will violate "Longest Prefix Match"

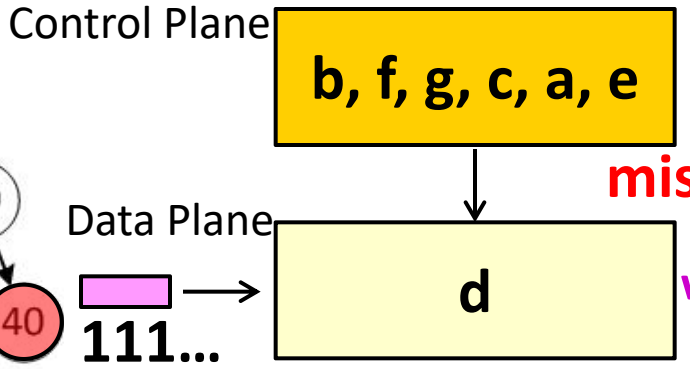
Routing Table:

ID	Prefix	Popularity
a	0*	20
b	000*	80
c	01*	30
d	1*	100
e	100*	10
f	11*	60
g	111*	40



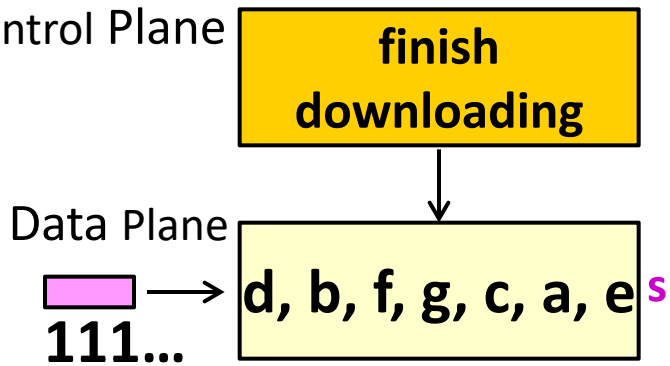
Binary Prefix Trie

Download by Popularity: d, b, f, g, c, a, e



Packet misforwarding!

will match d ✗



should match g

Reason: prefix g is more specific than prefix d, but prefix g has less popularity.

Problem Formalization

- Prefix download sequence: $p_0, p_1, p_2, \ldots, p_{n-1}$
- Given traffic will be allowed in after the download of the first m prefixes, we define cumulative popularity C(m) as $\sum_{i=0}^{m-1} p_i.pop$
- We expect C(m) can be as large as possible for a minimized routing table miss rate, under the constraint of longest prefix match, thus the optimal prefix download sequence must satisfy

  maximum (C(m))

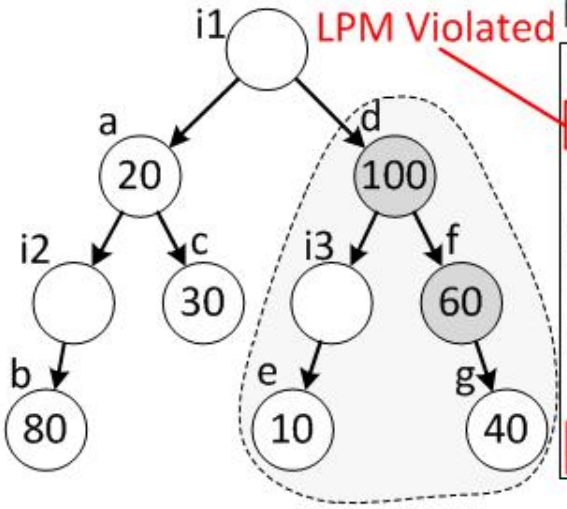
  s.t. $i < j$, if p_i is a more specific prefix of p_j, $\forall i, \forall j$

Heuristic Approach

- Basic Idea: sort prefixes by popularity first, adjust prefix sequence when LPM violation is detected
- When traversing the prefix trie, for each visited prefix v, we define v.sub_min as the prefix of the minimal popularity in the subtrie rooted at prefix v.
- LPM violation condition: v.sub_min.pop < v.pop
- If we detect LPM violation, we can move v behind v.sub_min to eliminate the violation
- We do the "check-and-adjust" procedure for each node in the routing prefix trie, v.sub_min can be precalculated

Heuristic Approach (cont)

```
1  Function lpm_adjust(TrieNode v)
2      if v == null then
3          return;
4      end
       /* get the least popular prefix in subtrie   */
5      v.sub_min ← get_sub_min(v);
6      if is_prefix(v) && v.pop > v.sub_min.pop then
           /* move v behind v.sub_min               */
7          move_behind(v, v.sub_min);
8      end
9      lpm_adjust(v.l);
10     lpm_adjust(v.r);
11 end
```



LPM Violated

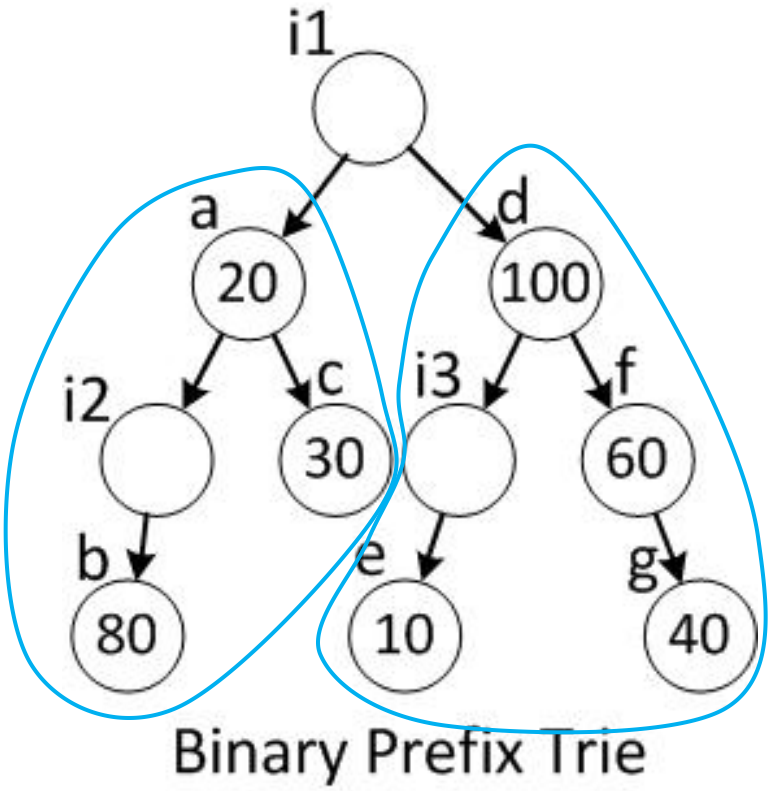
Preorder Traversal:

Ranking	Visited	Sub_Min
d b f g c a e	a(20)	a(20)
d b f g c a e	b(80)	b(80)
d b f g c a e	c(30)	c(30)
b f g c a e d	d(100)	e(10)
b f g c a e d	e(10)	e(10)
b g f c a e d	f(60)	g(40)
b g f c a e d	g(40)	g(40)

LPM Guaranteed

- ✓ O(nlgn)
- ✓ Suboptimal
- ✓ Effective on real trace

Optimal Approach by Dynamic Programming



Binary Prefix Trie

Assuming v.l and v.r have been already solved as c(v.l) and c(v.r), construct v's solution c(v) based on v.l's and v.r's solutions

c(v) = recursion(c(v.l), c(v.r))

We need to build the recursive relations

Recursive Relations

- For each prefix, we define v.c[i] as the cumulative popularity of the first i prefixes in the optimal prefix download sequence
- Hence, maximizing C(m) with LPM guaranteed is equal to solving root.c[m]
- Assuming v.l, v.r have already been solved, i.e., v.l.c[i] and v.r.c[i] have been calculated, we can derive v.c[i] according to the following recursive formulas
- If v is not included in the first i prefixes, we will have

  v.c[i] = max(v.l.c[i], …, v.l.c[i-j] + v.r.c[j], …, v.r.c[i]) 0 < j < i
- If v is included in the first i prefixes, we will have

  v.c[i] = sub_sum(v) // calculate the popularity summation of prefixes in the subtrie rooted at v
- Here, we leave out other boundary conditions for brevity

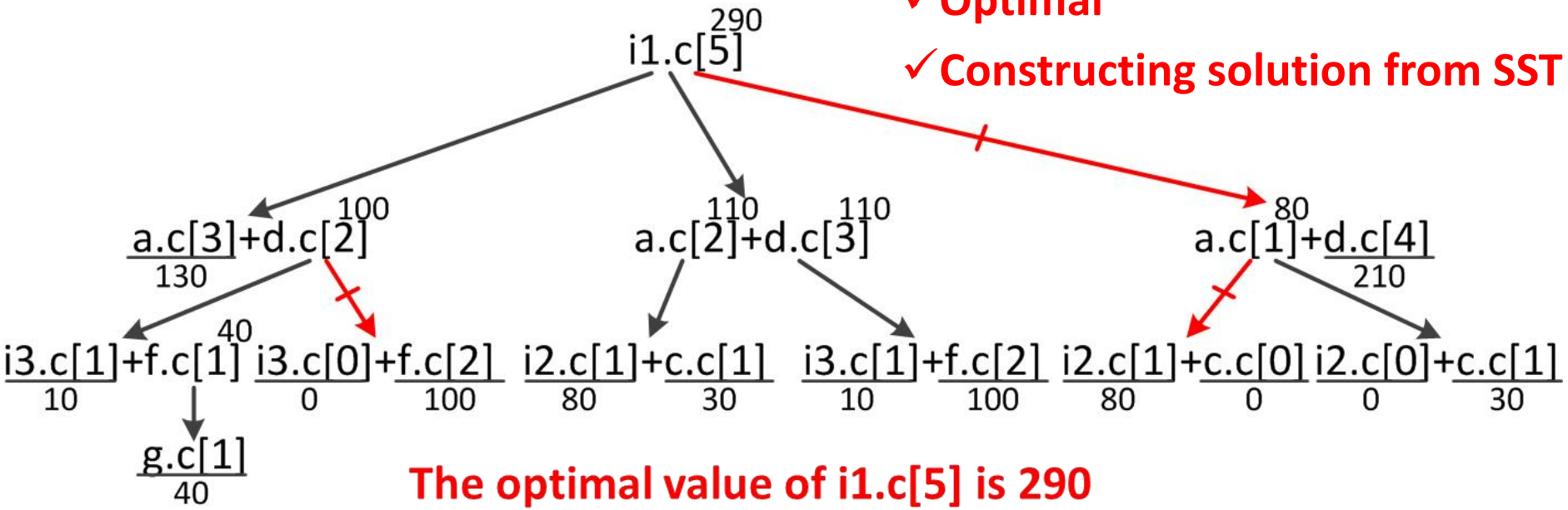
Solution Space Tree (SST)

- With memoization, we can generate the solution space tree (SST), from which we can construct the optimal solution

✓$O(n^3)$

✓Optimal

✓Constructing solution from SST



The optimal value of i1.c[5] is 290

The optimal solution can be constructed from SST as {{{b}, ∅}, {d, e, f, g}}

Batch Download of Routing Table

Traditional route download:

(1) Routes downloaded from route processor to host processor

(2) Host processor translates logic representation of routes into hardware-friendly instructions

(3) Hardware executes the translated instructions

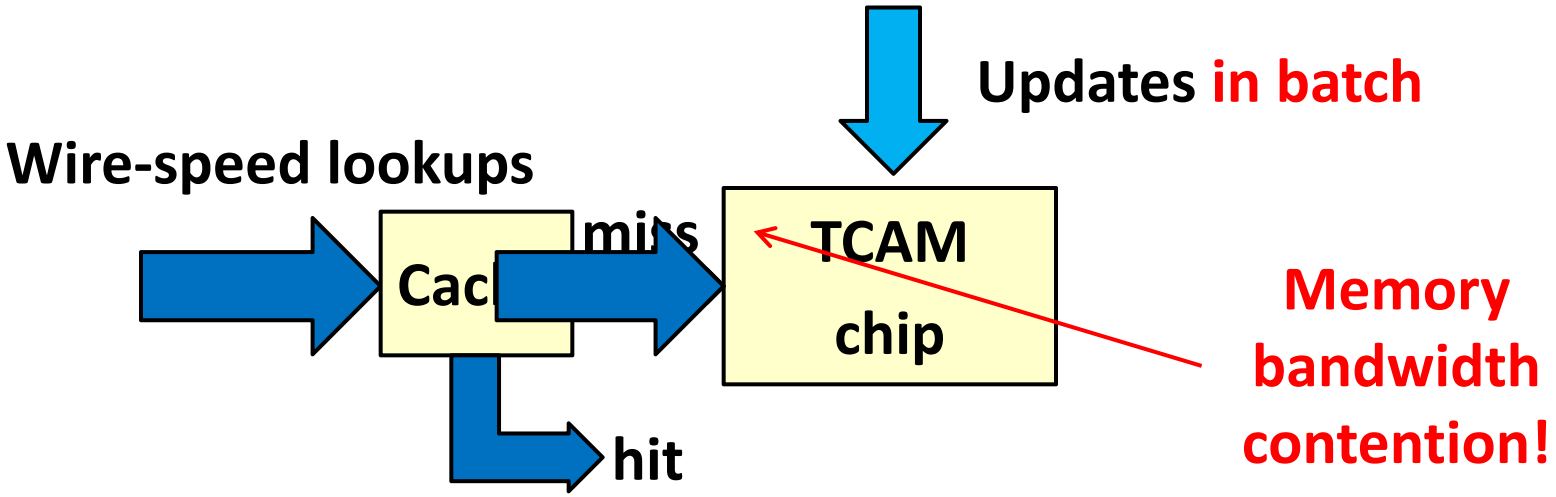
Fast route download (given host processor is standby):

~~**(1) Routes downloaded from route processor to host processor**~~

~~**(2) Host processor translates logic representation of routes into hardware-friendly instructions**~~

(3) Hardware executes the translated instructions *in batch*

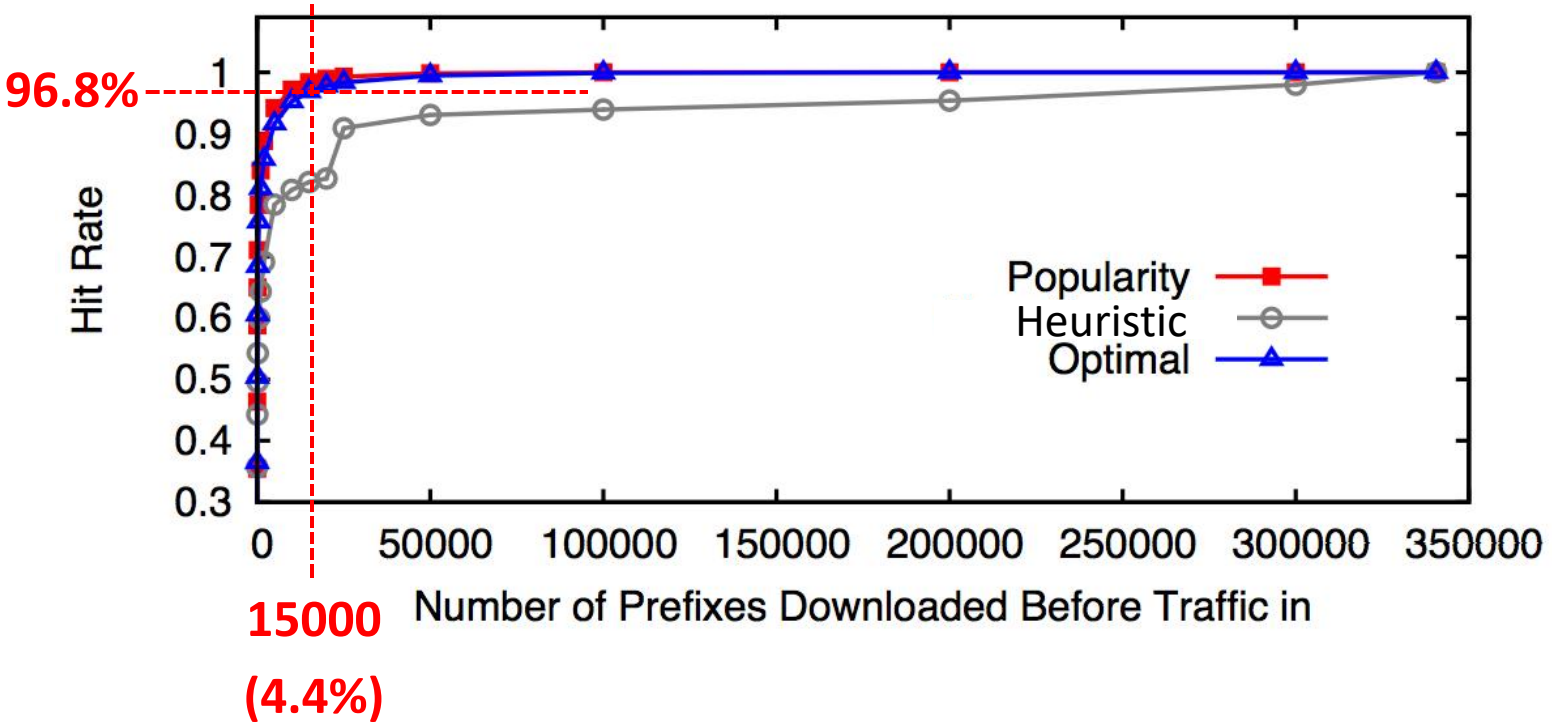
Reduce Lookup/Update Conflict by Caching



The cache can be de-allocated to on-chip memory pool when batch prefix update is completed

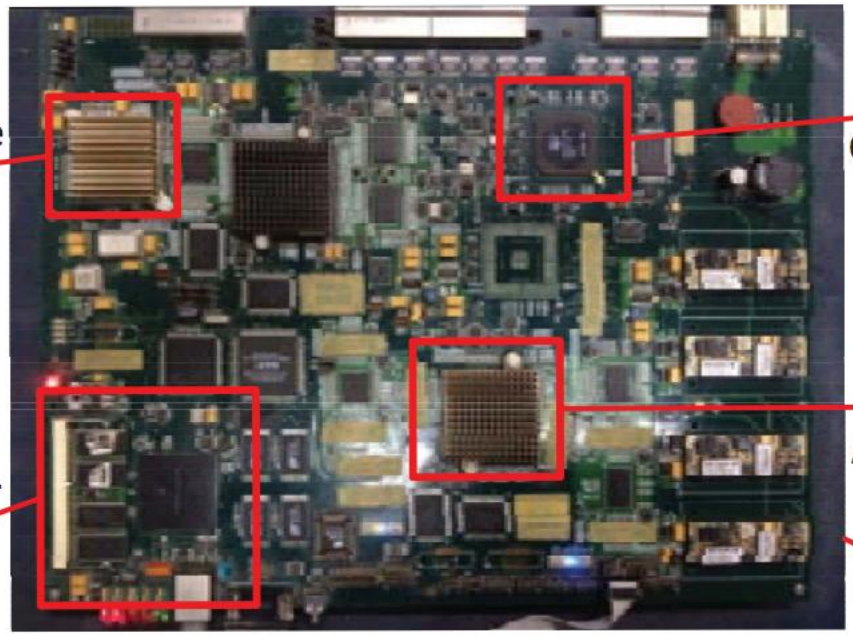
Evaluation of Prioritized Prefix Download

A routing table with 340584 routes, 60-minute traffic from 20Gbps gateway link



- **Optimal's result is extremely close to Popularity's**
- **Heuristic works well with O(nlgn) complexity**
- **With Optimal, first 4.4% prefixes will cover 96.8% traffic**

Wake up a Bare-Metal Router Prototype



Events	Time (ms)
~~software package downloaded via network~~	~~1000~~
~~VxWorks booted up~~	~~5000~~
~~routing table downloaded via network~~	~~4718.2~~
~~prefix parsing; trie construction~~	~~27539.6~~
prefix updated into TCAM	2545.3

0.3% of the original time!

× **0.05=127.27ms**

Conclusion

- Systematic measurement subverts the presumption of zero-time link wakeup
- New designs to reduce the line card wakeup time
- Algorithms to tackle the LPM violation issue
- Engineering efforts to make the speedup
- Radical reduction (to 0.3%) of wakeup time on a bare-metal prototype built for design verification
- Promising to build better power-efficient Internet

Thank You!

Q & A