Reducing Routing Table Size Using Ternary-CAM

Hot Interconnect 9

Aug. 22, 2001

Huan Liu

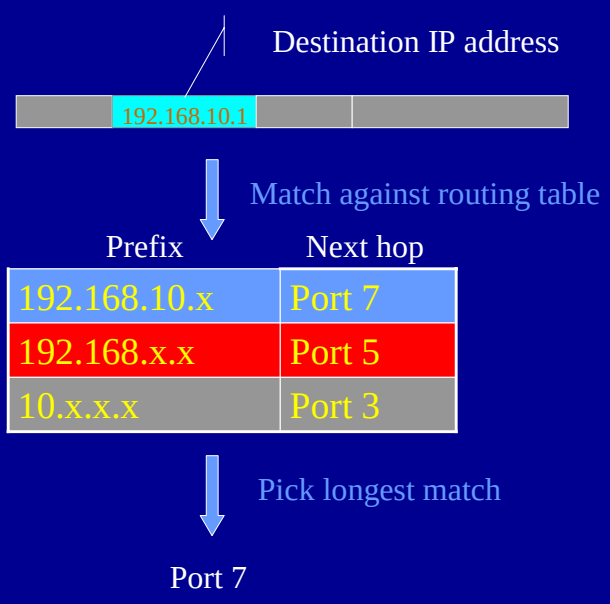
Department of Electrical Engineering

Stanford University

huanliu@stanford.edu

http://www.stanford.edu/~huanliu

Longest prefix matching (LPM)



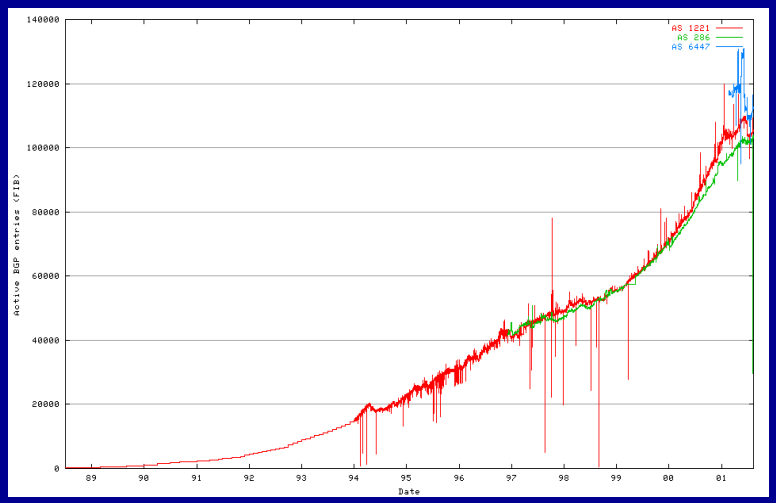
Existing LPM algorithms

- Software based algorithms (LC Trie, Hash..)
  - Cheap
  - But slow, at least 4 memory accesses
- Hardware based algorithm (CAM/TCAM)
  - Fast
  - Deterministic
  - Expensive
  - Power hungry

? How can we make it less expensive and less power hungry?

Other half of the story….

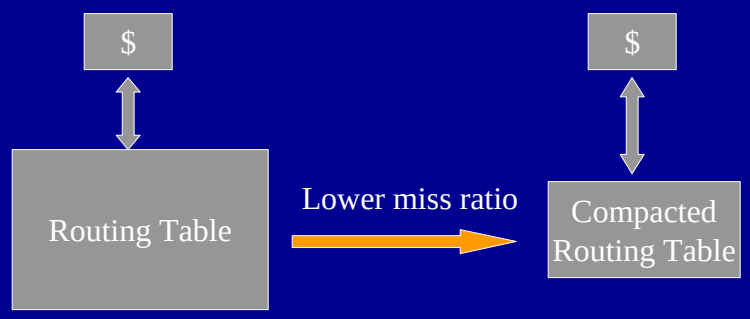
Need to worry about growth as well:



Source: http://www.telstra.net/ops/bgptable.html

More motivation

- It is beneficial for routing prefix caching
- For detail on how to cache prefix, see: http://www.stanford.edu/~huanliu/icccn01.pdf



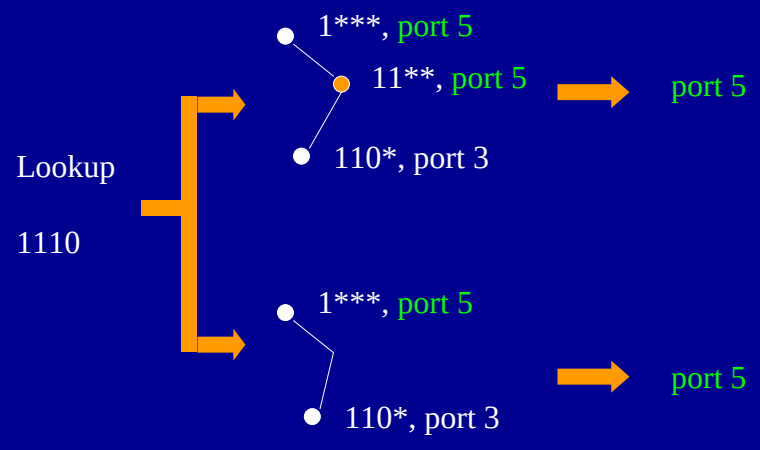
Key observation

- Number of prefixes is large
- Number of distinct next hop is small

	maeeast	maewest	pacbell	aads	paix
# of routes	36	40	19	37	26
# of prefixes	23554	32139	38791	15906	29195

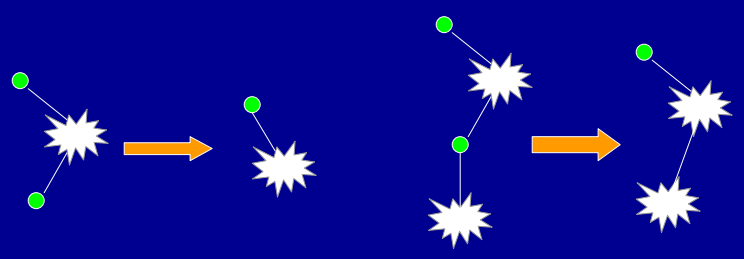
Source: http://www.merit.edu/ipma Mar. 7, 2001

Redundancy

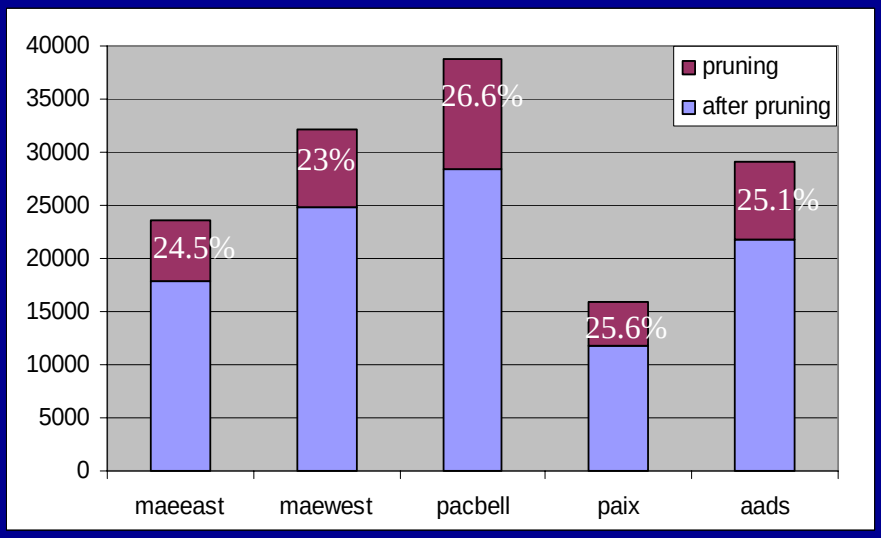


Pruning

- Remove redundant routing entries
- Redundant entries are very common
- Fairly easy to do: simple recursion

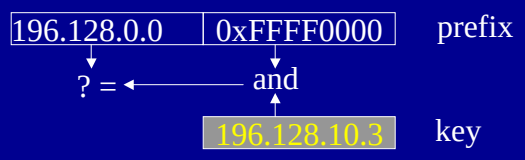


Pruning result



Content addressable memory

- Fully associative memory: search in 1 cycle
- Binary CAM: fixed length exact match
  - Need several chips, one for each distinct prefix length
- Ternary CAM:
  - Store 0, 1, *
  - Masked match
  - For routing prefix, mask is all 1's followed by all 0's



Mask extension

	Prefix	Next hop
prefix	1101010*****	
prefix	1100010*****	

minterm	110*010*****	

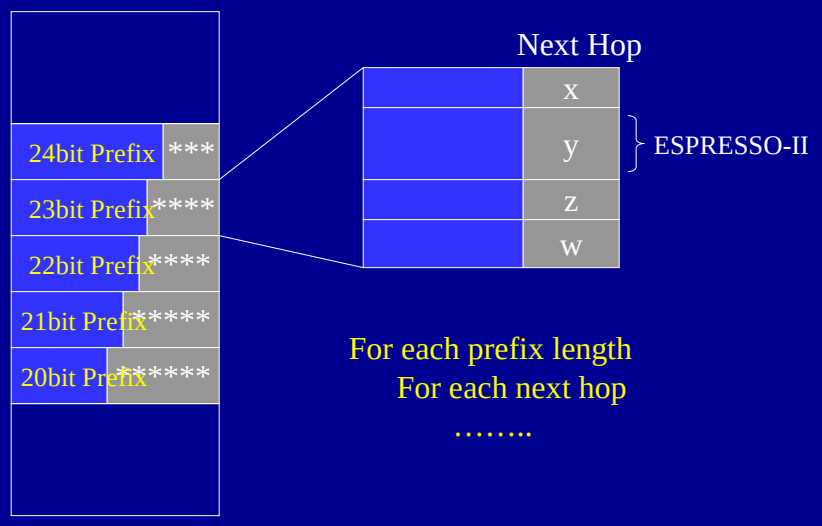
Logic minimization

- Mask extension becomes a logic minimization problem
- ESPRESSO-II algorithm is used

……	
1101010*****	
1100010*****	
……	

Minimize using ESPRESSO-II

Convert whole routing table



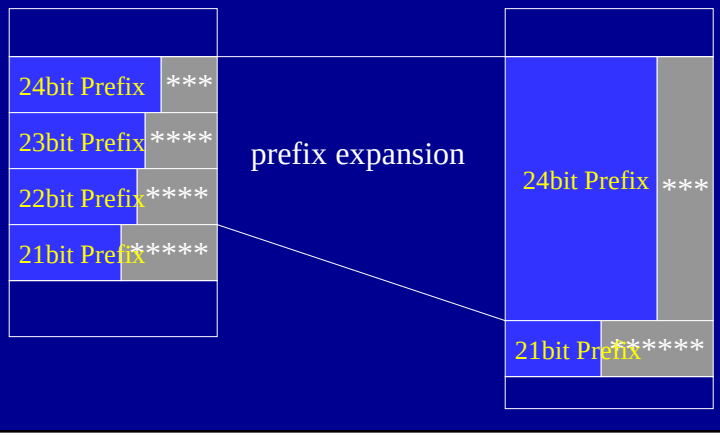
For each prefix length
For each next hop
……..

ESPRESSO-II options

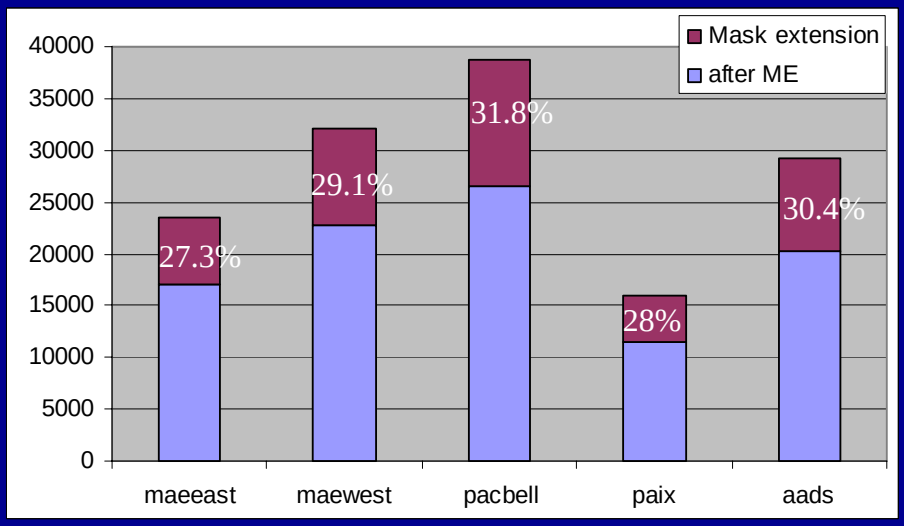
- Exact
  - Minimum number of minterms but not necessarily minimum number of literals
  - Optimal for us because minterms correspond to CAM entries
  - Runtime close to other options (fast, 1 pass) after pruning is applied first

Just say No

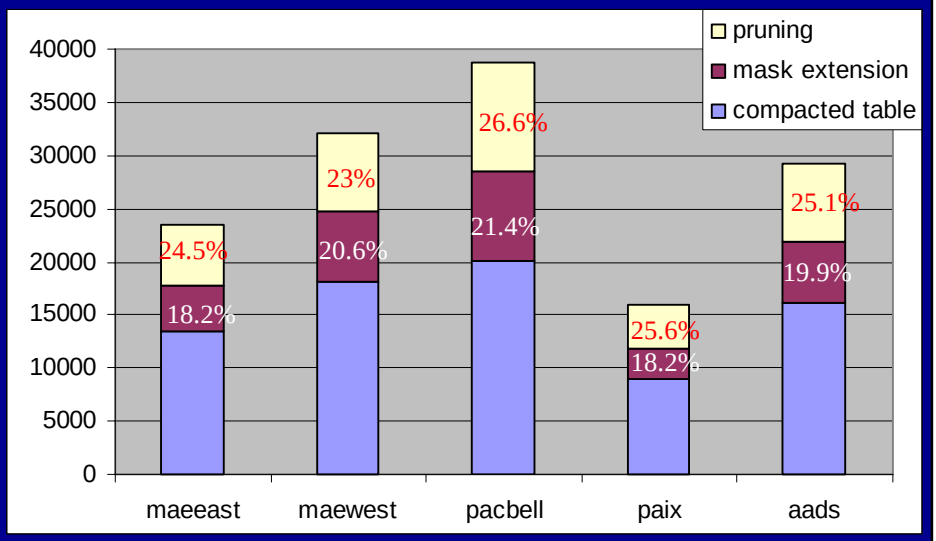
- Prefix expansion can increase optimization opportunity
- But, runtime is also substantially longer



Mask extension result



Pruning + Mask Extension result



Summary

- Two techniques to compact routing table
  - Pruning
    - Applicable to all LPM algorithms
    - Fast and easy to compute
  - Mask extension
    - Applicable only to TCAM
    - CPU intensive to compute
    - Need fast incremental update algorithm

Update: route insert/withdraw

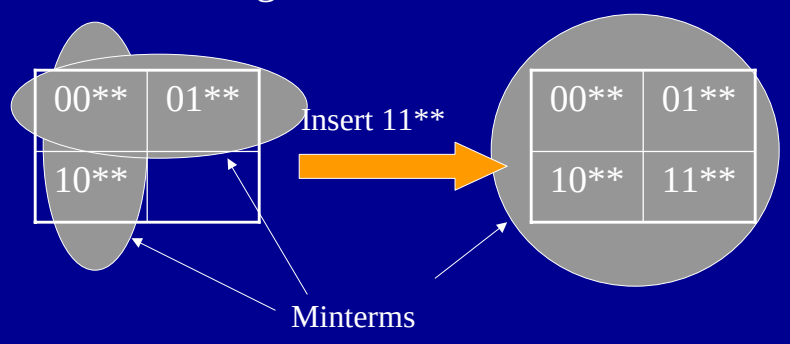
- Pruning
  - Incremental update is trivial
  - Fast
- Mask extension
  - Slow

	maeeast	maewest	pacbell	paix	aads
Runtime (s)	14.2	49.3	77	8.26	44.1

  - Can not afford to recompute after each update, need faster incremental update algorithm

Incremental insertion

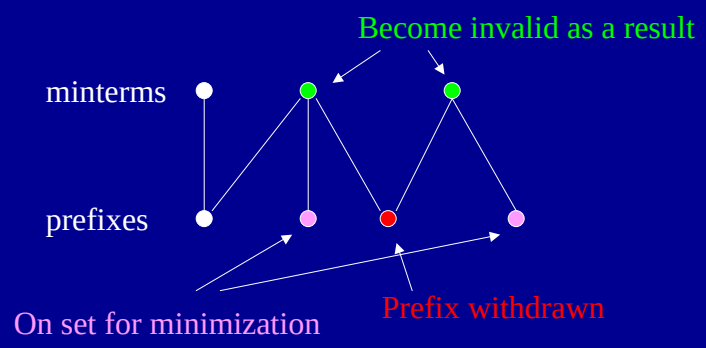
- Use existing minterms as Don't care set



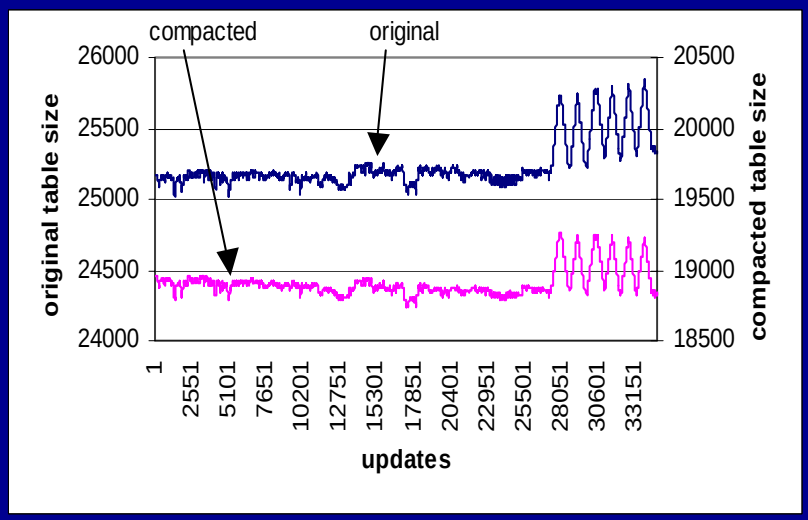
On-set: 11**
Don't care: *0**
0***

Incremental withdrawal

- Several minterms may need to be removed
- Several prefixes may need to be re-minimized
- Use remaining minterms as Don't care set



Incremental update result



Incremental update speed

- Our implementation: 22ms/per update on Pentium III 500Mhz
- Can support ~50 updates/second
- Most route updates are routing flap: same prefix inserted/removed repeatedly
- Can use buffer to store routing flap before committing update so that we can support much more updates per second