Reducing Routing Table Size Using Ternary-CAM

Huan Liu

Department of Electrical Engineering
Stanford University
huanliu@stanford.edu
http://www.stanford.edu/~huanliu

Hot Interconnect 9
Aug. 22, 2001

Longest prefix matching (LPM)

Destination IP address

Prefix Next hop

Match against routing table

192.168.10.x Port 7
192.168.x.x Port 5
10.x.x.x Port 3

Pick longest match

Port 7
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

Routing Table

Lower miss ratio

Compacted Routing Table

Key observation

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

<table>
<thead>
<tr>
<th></th>
<th>maecast</th>
<th>maewest</th>
<th>pacbell</th>
<th>aads</th>
<th>paix</th>
</tr>
</thead>
<tbody>
<tr>
<td># of routes</td>
<td>36</td>
<td>40</td>
<td>19</td>
<td>37</td>
<td>26</td>
</tr>
<tr>
<td># of prefixes</td>
<td>23554</td>
<td>32139</td>
<td>38791</td>
<td>15906</td>
<td>29195</td>
</tr>
</tbody>
</table>

Redundancy

- 1***, port 5
- 11**, port 5
- 11*, port 3
- 1***, port 5
- 11*, port 3

Lookup

1110

1110

Pruning

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

![Pruning result chart]

**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

```
196.128.0.0 | 0xFFFF0000
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>prefix</td>
<td>and</td>
</tr>
<tr>
<td>? =</td>
<td>196.128.10.3</td>
</tr>
<tr>
<td>key</td>
<td></td>
</tr>
</tbody>
</table>
```
Mask extension

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Next hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>prefix</td>
<td>1101010****</td>
</tr>
<tr>
<td>prefix</td>
<td>1100010****</td>
</tr>
</tbody>
</table>

Logic minimization

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

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

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Prefix Expansion</th>
<th>Mask Extension after ME</th>
</tr>
</thead>
<tbody>
<tr>
<td>24bit</td>
<td>24bit Prefix</td>
<td>24bit Prefix</td>
</tr>
<tr>
<td>23bit</td>
<td>23bit Prefix</td>
<td>23bit Prefix</td>
</tr>
<tr>
<td>22bit</td>
<td>22bit Prefix</td>
<td>22bit Prefix</td>
</tr>
<tr>
<td>21bit</td>
<td>21bit Prefix</td>
<td>21bit Prefix</td>
</tr>
</tbody>
</table>

Maeeast: 27.3% 29.1% 31.8%
Maewest: 28%
Pacbell: 29.1% 30.4%
Paix: 28%
Aadds: 30.4%
**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

<table>
<thead>
<tr>
<th></th>
<th>maeast</th>
<th>maewest</th>
<th>pacbell</th>
<th>paix</th>
<th>aads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime (s)</td>
<td>14.2</td>
<td>49.3</td>
<td>77</td>
<td>8.26</td>
<td>44.1</td>
</tr>
</tbody>
</table>

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

Incremental insertion

- Use existing minterms as Don’t care set

Insert 11**

Minterms

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