Network Algorithms, Lecture 4:
Longest Matching Prefix Lookups

George Varghese
Plan for Rest of Lecture

- Defining Problem, why it’s important
- Trie Based Algorithms
- Multibit trie algorithms
- Compressed Tries
- Binary Search
- Binary Search on Hash Tables
Longest Matching Prefix

• Given N prefixes $K_i$ of up to W bits, find the longest match with input $K$ of W bits.
• 3 prefix notations: slash, mask, and wildcard. 192.255.255.255 /31 or $1^*$
• N =1M (ISPs) or as small as 5000 (Enterprise). W can be 32 (IPv4), 64 (multicast), 128 (IPv6).
• For IPv4, CIDR makes all prefix lengths from 8 to 28 common, density at 16 and 24
Why Longest Match

- Much harder than exact match. Why is thus dumped on routers.
- Form of compression: instead of a billion routes, around 500K prefixes.
- Core routers need only a few routes for all Stanford stations.
- Really accelerated by the running out of Class B addresses and CIDR
Sample Database

P1 = 10 *
P2 = 111 *
P3 = 11001*
P4 = 1*
P5 = 0*
P6 = 1000 *
P7 = 100000*
P8 = 1000000*
Compressed Trie using Sample Database
Skip versus Path Compression

• Removing 1-way branches ensures that tries nodes is at most twice number of prefixes.
• Skip count (Berkeley code, Juniper patent) requires exact match and backtracking: bad!
Multibit Tries

Diagram of a multibit trie with nodes labeled P5, P6, P7, P8, P3, P2, P1, and P4. The trie branches based on binary prefixes: 00, 01, 10, and 11.
Optimal Expanded Tries

- Pick stride $s$ for root and solve recursively
Leaf Pushing: entries that have pointers plus prefix have prefixes pushed down to leaves.
Why Lulea Speed Loss is not too bad

\[
\begin{array}{c|c|c|c}
0 & 3 & 5 & \ldots & \text{NumSet}[J] & \ldots \\
\hline
10001001 & 10000001 & \ldots & 11111000 & \ldots \\
\end{array}
\]

\[\text{NumSet}[J] + 3\]

\[\text{Uncompressed Index} \quad \rightarrow \quad \text{Compressed Index}\]
Why Compression is Effective

• Breakpoints in function (non-zero elements) is at most twice the number of prefixes
**Eatherton-Dittia-Varghese**

**Eatherton Scheme: Compression alternative to Lulea**

\[ P1 = 0^*, \quad P2 = 11^*, \quad P3 = 10000^* \]

<table>
<thead>
<tr>
<th>000</th>
<th>P1</th>
<th></th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>P1</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>010</td>
<td>P1</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>011</td>
<td>P1</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>def</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>101</td>
<td>def</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>110</td>
<td>P2</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>111</td>
<td>P2</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

Lulea uses large arrays: TreeBitMap uses small arrays, counts bits in hardware. No leaf pushing, 2 bit maps per node. CRS-1
Binary Search

- **Natural idea:** reduce prefix matching to exact match by padding prefixes with 0’s.
- **Problem:** addresses that map to diff prefixes can end up in same range of table.

```
A  0  0  0  
A  B  0  0  
A  B  C  0  
ABCFA  
AFFFF  
```
Modified Binary Search

- **Solution**: Encode a prefix as a range by inserting two keys A000 and AFFF
- Now each range maps to a unique prefix that can be precomputed.

```
A   0   0   0
A   B   0   0
A   B   C   0
A   B   C   F
A   B   F   F
A   F   F   F
```

```
A B C D
A B D D
A D D D
```
Why this works

- Any range corresponds to earliest L not followed by H. Precompute with a stack.
**Modified Search Table**

- Need to handle equality (=) separate from case where key falls within region (>).

```
1)   A   0   0   0   >   1   1
2)   A   B   0   0   >   2   2
3)   A   B   C   0   >   3   3
4)   A   B   C   D   >   3   4
     A   B   C   F   >   2   3
     A   B   F   F   >   1   2
     A   F   F   F   >   -   1
```
Transition to IPv6

• So far: schemes with either log N or W/C memory references. IPv6?
• We describe a scheme that takes $O(\log W)$ references or $\log 128 = 7$ references
• Waldvogel-Varghese-Turner. Uses binary search on \textit{prefix lengths} not on \textit{keys}.  
Why Markers are Needed
Why backtracking can occur

- Markers announce “Possibly better information to right”. Can lead to wild goose chase.
Avoid backtracking by . . .

- Precomputing longest match of each *marker*
2011 Conclusions

• Fast lookups require fast memory such as SRAM $\rightarrow$ compression $\rightarrow$ Eatherton scheme.
• Can also cheat by using several DRAM banks in parallel with replication. EDRAM $\rightarrow$ binary search with high radix as in B-trees.
• IPv6 still a headache: possibly binary search on hash tables.
• For enterprises and reasonable size databases, ternary CAMs are way to go. Simpler too.
Principles Used

- **P1**: Relax Specification (*fast lookup, slow insert*)
- **P2**: Utilize degrees of freedom (*strides in tries*)
- **P3**: Shift Computation in Time (*expansion*)
- **P4**: Avoid Waste seen (*variable stride*)
- **P5**: Add State for efficiency (*add markers*)
- **P6**: Hardware parallelism (*Pipeline tries, CAM*)
- **P8**: Finite universe methods (*Lulea bitmaps*)
- **P9**: Use algorithmic thinking (*binary search*)