Longest Prefix Matching

Jon Turner
Computer Science & Engineering
Washington University

www.arl.wustl.edu/~jst
IP Address Lookup

- Routing tables contain \( (\text{prefix, next hop}) \) pairs
- Address in packet is compared to stored prefixes, starting at left
- Prefix that matches largest number of address bits is desired match
- Packet is forwarded to the specified next hop.
- Problem - large router may have 100,000 prefixes
Address Lookup Using Tries

- Prefixes stored in trie
- Prefixes “spelled” out by following path from top
  - dark nodes mark prefix ends
  - to find best prefix, spell out address in tree
  - last dark node marks longest matching prefix
- Straightforward version can be too slow
- Can speed up with alternate data representation
Multibit Trie

- Match several bits in one step instead of single bit
  - turns sub-trees of binary trie into single nodes
- Store next hop information in per-node arrays
- For stride of s, reduces tree depth by factor of s
- Gives nodes that use $O(2^s)$ space
Tree Bitmap Version of Multibit Trie

- External bit vector identifies outgoing branches from node
- Internal bit vector defines prefixes for node
- Siblings placed in consecutive memory locations
  » so, single child pointer – complicates memory mgmt.
- Pointer to next hop array completes per node data
Shape Shifting Tries

- Tree bitmap can use memory inefficiently
  - sparse nodes with lots of 0 bits; IPv6 exacerbates this
- Replace sparse nodes with shape-shifting nodes
  - requires extra bit vector to encode subtree shape
  - group bits into subtrees to limit data structure depth
Search by Prefix Length

- Build hash table for prefixes of each distinct length
  - key: length+(padded prefix)

- To find longest match for a given IP address
  - form all prefixes and search
  - start with longest prefix and stop on first match
    - so search for x4.1101 probes tables 8 then 7

- Refinements
  - reduce number of distinct lengths by expanding prefixes
  - add on chip Bloom filter for each hash table
    - search hash table only if match in Bloom filter
Binary Search on Prefix Length

- Define recursive search order
  - search top table first
  - if match, search longer tables
  - if no match, search shorter

- **Markers** added to guide search
  - marker $xB.0/4$ in length 5 table leads to longer prefixes
    - $xB.001, xB.010, xB.0011$
  - markers include length of longest matching sub-prefix
    - used to find match after failed search for longer prefixes

- At most $\log_2 W$ hash probes

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>10 01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>110 101/2 000/2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1011 0001</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>x5.1 x1.0 x3.0/0 xB.0/4 x4.1/2 x9.1/2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>x3.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>xB.001 xB.010 x4.110 x9.100/2 x5.100/5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>x4.1100 xB.0011 x9.1000 x5.1001</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example: $xB.0001$
- search 5, match $xB.0/4$
- search 7, no match
- search 6, no match
- so, best match has length 4

Example: $xA.0000$
- search 5, no match
- search 3, match 101/2
- search 4, no match
- so, best match has length 2
Comparison

- Tree bitmap hard to beat for IPv4
  - for stride 4, 10 memory accesses suffice
    - child pointer and external bit vectors in same word
    - use prefix bits to avoid examining most internal bit vectors
  - can reduce to 7 or 8 using large direct lookup root node
  - simple to implement, straightforward to update
  - shape-shifting nodes useful for IPv6, but more complex and harder to update

- Binary search on prefix lengths
  - best asymptotic search time – most important for IPv6
    - but hash probes may require multiple memory accesses
  - space for markers, expensive to update

- Many other approaches – most usable, not better