**Trie Trees**

- Create a simple binary tree where each vertex represents a prefix and each arc represents a binary choice of 0 or 1.
- Interim prefixes contain a flag indicating if there is a routing entry corresponding to the prefix. Leaves always have this flag.
- Given an IP address, start traversing the tree, making arc choices based on the bit location in the IP address, starting MSB to LSB.
- Trie trees can be collapsed if interim vertices in the tree have no alternate arcs that be followed.
  - Intuitively, you can collapse a non-branching path into 1 entry corresponding to the leaf.

**Binary Search on Hash Tables**

- Maintain hash tables for each distinct prefix length.
- Maintain an index array of all unique prefix lengths
- Search all hash tables pointed to by the index array backwards, starting from longest prefix to shortest.
- Computational complexity: $O(W_{\text{max}})$. Worst case: $O(W)$.

**Binary Search on Hash Tables (2)**

- Optimization: Use binary search instead of linear search of hash index array.
- Problem: If you don’t find a match at a lower prefix hash table, you may still find a match at a higher prefix hash table.
- Solution: Maintain markers at lower prefix hash tables to indicate presence of entries at higher prefix hash tables.

**Binary Search on Hash Tables (3)**

- Optimizations:
  - Store markers only at levels visited by binary search. Reduces marker storage to $O(\log W)$ for each real prefix.
  - Algorithm can backtrack causing computation time to reach $O(W)$. Take the case where there are valid prefixes with all 0’s, a 1-bit prefix set to 1 and a 32-bit prefix with the MSB and LSB set to 1. What happens if you search for an address $1(0^*)$?
  - Can use precomputation to avoid backtracking
    - Store the best matching prefix of the marker along with the marker (M.bmp). This can be precomputed when the marker is inserted.
    - No need to backtrack any more. If search on a marker fails, the current value of M.bmp is the best match.
- Computational Complexity: $O(\log W)$.

**Controlled Prefix Expansion**

- Goal: Improve trie performance by using multiple bits on each arc.
- Need to expand prefixes to a fixed set of target prefix lengths (strides).
- Problem: Prefix capture. An expanded prefix may collide with an actual prefix.
  - Solution is simple. Choose the routing entry corresponding to the actual prefix.
Controlled Prefix Expansion

- Choice of strides length is based on a dynamic programming solution that minimizes the total memory taken up by the expanded prefixes. (Refer to reading list, paper 1)
- Stride length need not be constant. Each level of the trie may use a different stride length. This can be used to reduce memory consumption of expanded prefixes.

Reading List

- Recommended Reading
  - Faster IP Lookups using Controlled Prefix Expansion, Srinivasan V., Varghese G., ACM SIGMETRICS 98
  - Small Forwarding Tables for Fast Routing Lookups, Degermark et al. SIGCOMM 97
  - Scalable High Speed IP Routing Lookups, Waldvogel M, et. Al, SIGCOMM 97