Many different approaches: software/hardware, space/time/power, update complexity

- data structures are large $\Rightarrow$ need to store in DRAM
- minimize number of DRAM accesses (added arithmetic, indexing, etc. is acceptable)

What's the difference between latches, SRAMs, DRAMs?

- Latches: very large
- RAMs: regular wiring, complex peripheral circuits, senseamps
  - SRAMs: actively driven wires
  - DRAMs: capacitor holds bit

Trie-based forwarding

Question: why can't we use a binary search tree, or a hash table?

Trie: data structure for fast lookups

- Example on page 493, TAOCP-Knuth vol 3
- Can reduce storage significantly (albeit with higher build time, runtime, complexity)

Software-based IP forwarding

Adnan Aziz

The University of Texas

adnan@ece.utexas.edu

Excellent survey in Radia Perlman's "Interconnections: Second Edition," Chapter 13, Addison-Wesley, 2000.

Review

Routing software come up with an optimized "forwarding table"

- $\{(*, 2), (1*, 0), (0*, 1), (00*, 3), (110*, 1), (000*, 1), \ldots\}$

Router selects which output link to forward incoming packet to based on destination IP address

- select based on longest prefix matched in forwarding table

Lulea

What if we wanted to query 16 bits at a time?

- 2^{16} nodes, each having 2^{16} children, etc.
- huge number of repeated nodes $\Rightarrow$ lots of memory

Lulea solution: keep bit vector of length 2^{16} at node "child array'

- set bit iff node corresponds to a 0-padded prefix, or an exact match, or follows a 1-padded prefix

Key idea: all entries with a 0 in the child array have same info as closest entry on left with bit set to 1

- keep separate "info-array" for all nodes in range

How to index into "info-array"?

- count number of bits set to 1 on your left
- keep extra space in child array, say every 64 bits
  - entry holds number of 1s to left, add count of number of 1s to 64 bit chunk

Reference: Degermark, *et al.*, "Small forwarding tables for fast routing lookups," ACM Sigcomm, 1998

Can use trie to perform longest prefix matching: store pointer to longest current match

Example forwarding table —
$\{*, 00*, 0001*, 11*, 101*, 0101*, 111*, 10100*\}$

- complexity — independent of the number of prefixes
- storage requirement?

Optimization 1: avoid one-way branching by keeping number of bits to skip over

- Patricia — don't store keys at nodes

Optimization 2: query multiple bits

- prefix capture problem — expand $*, 10*, 100*, 1000*$ to four bits?
- time-space tradeoff: compute optimum points using dynamic programming

References:

- Knuth, vol. 3, TAOCP, pages 492–512
- Srinivasan and Varghese, "Faster IP Lookups Using Controlled Prefix Expansion," ACM Sigmetrics, 1998

Optimizations:

- build custom made hash function for each prefix length
  - try out several seeds till one gives say no more than 3 collisions
  - store chains as array, retrieve all in one probe
- don't need to keep marker at all shorter prefixes for a prefix
  - Ex. if best match is 0011 1011 0000 1111 101, probe sequence is 16 (marker), 24 (empty), 20 (empty), 18 (marker), 19

More optimizations:

- exploit statistics, perform asymmetric search
- mutating binary search
- initial array lookup

Summary — perform lookup in 80ns

Reference: M. Waldvogel, *et al.*, "Scalable High Speed IP Routing Lookups," ACM Sigcomm, 1997

Binary search on prefix lengths

Can use 32 different hash tables (actually fewer, some prefix lengths don't exists)

- linear search (longest to shortest)

Why can't we do binary search on the tables?

- suppose we have a 19 bit prefix 0011 1011 0000 1111 101, but no proper prefix of it in table
  - start by searching for 16 bit prefix 0011 1011 0000 1111 $\Rightarrow$ problem

Need a "marker" — something to indicate that there exists a longer prefix with that 16 bit prefix

- store 0011 1011 0000 1111& in table 16

What happens if destination IP matches 0011 1011 0000 1111 but not 0011 1011 0000 1111 101?

- along with marker, keep longest prefix of marker prefix that is in forwarding table
- Ex. if 0011 1011 in database, but nothing else till 0011 1011 0000 1111 101, then keep (0011 1011 0000 1111&,8) in table 16

Search for 1100(0) — end just to right of $\overset{10_{111}}{)}$

- which interval does it match? walk left, match parens till reach corresponding left parens
- can precompute the interval

Complexity — N prefixes $\Rightarrow$ about $\lg N$ DRAM accesses

- $N = 60000$ need 16 DRAM accesses

Reference: B. Lampson, *et al.*, "IP Lookups using Multiway and Multicolum Search," IEEE Infocom, 1998

Binary search on array

Intuition: think of 32 bit IP address as binary encoding of a number in $[0, 1)$

- prefix corresponds to an interval
- proposition: given two prefixes p, q either their intervals are disjoint, or one contains other

Natural to think of binary search array of interval end points

- some technical issues have to be addressed

Keep simple — use 4 bit IP addresses

- How to deal with 4 bit prefixes? Add dummy 5-th bit, so all prefixes correspond to two distinct end points

Problem: interval endpoints may overlap

- $10*$ and $1000*$ both start at 10000

Solution: make 10_{000} less than 1000_0, 10_{111} greater than 1000_1

Example: for prefixes $= *, 1, 10, 100, 101, 1110$, address ranges are

$$\underset{(}{00000}\ \underset{(}{1_{0000}}\ \underset{(}{10_{000}}\ \underset{(}{100_{00}}\ \underset{)}{100_{11}}\ \underset{(}{101_{00}}\ \underset{)}{101_{11}}\ \underset{)}{10_{111}}\ \underset{(}{1110_0}\ \underset{)}{1110_1}\ \underset{)}{1_{1111}}\ \underset{)}{11111}$$

Search for 1001(0) — end just to right of $\overset{100_{00}}{(}$

- matches 100