Lecture 4: Router design - Forwarding & lookup

[illegible]:

Last lecture: - overview of router design
- s/w stack
- trade-offs in speed vs. fanout
- components
- backplane.

Today: More on router's data plane.

Things that happen before switching across the backplane.

Lookup and forwarding, classification.

Context: addressing

CIDR: previously - classes.

lead to inefficient use of address space.

multiple class C n/ws → routing table explosion

CIDR → allows aggregation

No rigid n/w number.

implications: longest prefix matching.

How to do LPM really fast:

① Trie: Worst-case O(W) lookups

② CAMs: large, power-consuming expensive, not very dense

③ Protocol-based approaches: tags, VCs.

ingress router needs to do full lookup.

Also, need global tag agreement and reuse protocols.

④ Caching: not very effective for backbones where there is insufficient locality

⑤ Binary search over entries: log(W) but efficient storage.

Requirements: ① few memory accesses per packet

② small amount of storage.

Ideas: A hash table per prefix length.

linear search. → ~~log~~ O(W) (start for largest prefix length)

~~Binary search~~

length	Hash	Tables
5	0 →	01010
7	0 →	0101011 0110110
12	0 →	011011010101

O(Wdist)

Binary search. → non-trivial to get it to work as markers are needed.

length	Tables		
1	0	Search for 111 would fail	add a marker in tables of lower prefix lengths
2	Add marker! → 11, 00		
3	111		

Marker Match → check lower half.

No Match → check upper half.

~~But~~ ~~match~~ How many markers: only in ~~as~~ lengths that are likely to be traversed in binary search

~ O(log W)

0000   000   1 / 00 / → 111 / → 0001   W = 4

Marker problems → backtracking.

1	1
2	⟨11⟩ 00
3	111

Suppose 110 arrives → should match ⟨1⟩ but search proceeds to level 3 and fails.
Need to backtrack to level 2 and try out top half.
→ worst-case backtracking complexity → O(W).

How to solve backtracking.
for each marker, remember the longest matching prefix → value of LMP of marker M.

⟨11⟩ ⟶ ⟨11⟩.1

Before going down the path suggested by a marker remember M.LMP.
No need to backtrack as the result of backtracking is stored in M.LMP.

How to compute these M.LMP → precomputation when lookup table is constructed.
Final algorithm: (N log W) space and log (W) speed
Algo on next page.

Some optimizations: (1) precompute search paths to maximize likelihood of finding a match
pick prefix length with most entries as the "root" of the binary search → shown to work well in practice

(2) cross router caching / hints

Final algorithm:

```
Binarysearch (D)          /* search for address D */
Initialize search range R over the whole array of lengths L
Initialize LMP found so far to NULL.

While R is not empty
    Let i correspond to the middle level in R
    Extract the first (i) [L(i).length] bits of D into D'
    M :=  Search (D', L(i).hash)
    if M is null then R := upper half of R
    elsif M is a prefix and not marker
    then LMP = M.LMP ; break
    else /* M is just a marker or marker + prefix */
        LMP = M.LMP
        R := lower half of R
    Endif
End while
```

In practice $W_{dest} \approx 20$ $\therefore$ $\log(W_{dist}) \approx 4$ or 4.5

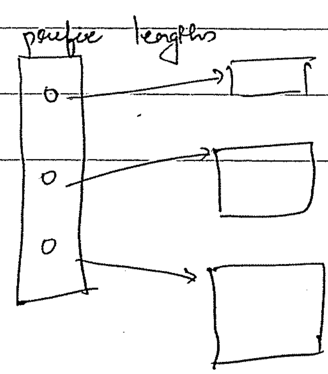
asymm. tree optizations bring expected case lookup times to 2

Can we do it in ~~one clock~~ round ~~cycle~~: faster:

Issues: (1) memory lookup per round → lookups are dependent and have to be done serially

(2)

Idea: bloom filters.



BF1 summarizes L1

BF2 L2

BF3. L3

Summary: k hash functions : $H_1 \cdots H_k$.

entry (e) $H_1(e) \cdots H_k(e)$ } set these bits in BF to 1 (some may already be set to 1 by others)

lookup (e) : compute $H_1(e) \cdots H_k(e)$; return yes if all are 1.

Likelihood for false positives : diminishes as ~~number of~~ ~~entries~~ as m/n falls ; $m \rightarrow$ number of entries
$n \rightarrow$ size of BF

$k \rightarrow$ too small is too bad
$\rightarrow$ too big is bad as well as many bits gets taken away.

It is possible to tune a BF to achieve a target FP rate.
$\rightarrow$ pointer on course Web page for

$$f = \left(1 - \left(1 - \frac{1}{n}\right)^{mk}\right)^k$$

$$\approx \left(1 - e^{-mk/n}\right)^k$$

optimal: $k = \frac{m}{n} \ln 2$
$fp = \left(\frac{1}{2}\right)^k$.

Back to the algo:
check all BFs. | — no time consumed here
└ start with longest prefix length and see if hash table has entry
FPs impact if entry exists or not

∴ use BFs $\rightarrow$ if bad $\rightarrow$ use older idea