Advanced Computer Networking (ACN)

IN2097 – WiSe 2025–2026

Prof. Dr.-Ing. Georg Carle, Sebastian Gallenmüller

Christian Dietze, Marcel Kempf, Lorenz Lehle

Chair of Network Architectures and Services
School of Computation, Information and Technology
Technical University of Munich

Routing and Forwarding

Lookup Data Structures for Computer Networks
Using Data from Routing Protocols

What is the interface of a routing protocol?

- Routing protocols know all possible routes to a destination network
  - Routing Information Base (RIB)
- Routing protocols choose which route is the “best” route (FIB)
  - Forwarding Information Base (FIB), also known as routing table
- Routing protocols do not concern themselves with forwarding individual packets
  - Routing: Routing protocols are used to create a RIB, RIB is used to create a FIB (routing table)
  - Updates to RIB/FIB itself happen less frequently $\rightarrow$ **less time critical**
  - Forwarding: actually looking at the packets and relaying packets according to the routing table
  - Forwarding lookups need to be performed for *every* packet in *every* router $\rightarrow$ **highly time critical**

Lookup Data Structures for Computer Networks
Longest Prefix Matching (LPM)

Goal of LPM

- Find an IP address with identical network parts (prefix matching)
- Find the IP address with longest matching network parts (longest prefix matching)

Lookup Data Structures for Computer Networks
Naive LPM

Strict Longest Prefix Matching (LPM) Algorithm

1. Sort FIB: longest prefixes first
2. Scan for matching prefix (top to bottom)
3. First prefix that matches is the correct one

Evaluation:

- Very easy to implement
- Very slow (linear probing)

Lookup Data Structures for Computer Networks
Naive LPM - Example

Routing Table

Status	Prefix	Next Hop	Interface
Checking	131.159.2.0/26	131.159.0.12	eth0
-	131.159.2.64/26	131.159.0.13	eth0
-	131.159.6.128/26	131.159.1.2	eth1
-	131.159.20.0/24	131.159.1.67	eth2
-	131.159.42.0/24	131.159.1.67	eth2
-	131.159.0.0/16	131.159.1.132	eth3
-	0.0.0.0/0	131.159.1.133	eth3

Lookup Data Structures for Computer Networks
Naive LPM - Example

Routing Table

Status	Prefix	Next Hop	Interface
Checking	131.159.2.0/26	131.159.0.12	eth0
-	131.159.2.64/26	131.159.0.13	eth0
-	131.159.6.128/26	131.159.1.2	eth1
-	131.159.20.0/24	131.159.1.67	eth2
-	131.159.42.0/24	131.159.1.67	eth2
-	131.159.0.0/16	131.159.1.132	eth3
-	0.0.0.0/0	131.159.1.133	eth3

Example: Destination address: `131.159.42.5`

`131.159.42.5    10000011 . 10011111 . 00101010 . 00000101`

Naive LPM - Example

Routing Table

Status	Prefix	Next Hop	Interface
Checking	131.159.2.0/26	131.159.0.12	eth0
-	131.159.2.64/26	131.159.0.13	eth0
-	131.159.6.128/26	131.159.1.2	eth1
-	131.159.20.0/24	131.159.1.67	eth2
-	131.159.42.0/24	131.159.1.67	eth2
-	131.159.0.0/16	131.159.1.132	eth3
-	0.0.0.0/0	131.159.1.133	eth3

Example: Destination address: `131.159.42.5`

```
131.159.42.5    10000011 . 10011111 . 00101010 . 00000101
/26             11111111 . 11111111 . 11111111 . 11000000
```

Lookup Data Structures for Computer Networks
Naive LPM - Example

Routing Table

Status	Prefix	Next Hop	Interface
Checking	131.159.2.0/26	131.159.0.12	eth0
-	131.159.2.64/26	131.159.0.13	eth0
-	131.159.6.128/26	131.159.1.2	eth1
-	131.159.20.0/24	131.159.1.67	eth2
-	131.159.42.0/24	131.159.1.67	eth2
-	131.159.0.0/16	131.159.1.132	eth3
-	0.0.0.0/0	131.159.1.133	eth3

Example: Destination address: `131.159.42.5`

131.159.42.5	10000011 . 10011111 . 00101010 . 00000101
/26	11111111 . 11111111 . 11111111 . 11000000
AND	10000011 . 10011111 . 00101010 . 00000000

Lookup Data Structures for Computer Networks
Naive LPM - Example

Routing Table

Status	Prefix	Next Hop	Interface
Checking	131.159.2.0/26	131.159.0.12	eth0
-	131.159.2.64/26	131.159.0.13	eth0
-	131.159.6.128/26	131.159.1.2	eth1
-	131.159.20.0/24	131.159.1.67	eth2
-	131.159.42.0/24	131.159.1.67	eth2
-	131.159.0.0/16	131.159.1.132	eth3
-	0.0.0.0/0	131.159.1.133	eth3

Example: Destination address: `131.159.42.5`

131.159.42.5	10000011 . 10011111 . 00101010 . 00000101
/26	11111111 . 11111111 . 11111111 . 11000000
AND	10000011 . 10011111 . 00101010 . 00000000
131.159.2.0	10000011 . 10011111 . 00000010 . 00000000

Naive LPM - Example

Routing Table

Status	Prefix	Next Hop	Interface
Checking	131.159.2.0/26	131.159.0.12	eth0
-	131.159.2.64/26	131.159.0.13	eth0
-	131.159.6.128/26	131.159.1.2	eth1
-	131.159.20.0/24	131.159.1.67	eth2
-	131.159.42.0/24	131.159.1.67	eth2
-	131.159.0.0/16	131.159.1.132	eth3
-	0.0.0.0/0	131.159.1.133	eth3

Example: Destination address: `131.159.42.5`

```
131.159.42.5   10000011 . 10011111 . 00101010 . 00000101
/26            11111111 . 11111111 . 11111111 . 11000000
--------------------------------------------------------
AND            10000011 . 10011111 . 00101010 . 00000000
131.159.2.0    10000011 . 10011111 . 00000010 . 00000000
--------------------------------------------------------
XOR            00000000 . 00000000 . 00101000 . 00000000
```

Naive LPM - Example

Routing Table

Status	Prefix	Next Hop	Interface
Checking	131.159.2.0/26	131.159.0.12	eth0
-	131.159.2.64/26	131.159.0.13	eth0
-	131.159.6.128/26	131.159.1.2	eth1
-	131.159.20.0/24	131.159.1.67	eth2
-	131.159.42.0/24	131.159.1.67	eth2
-	131.159.0.0/16	131.159.1.132	eth3
-	0.0.0.0/0	131.159.1.133	eth3

Example: Destination address: `131.159.42.5`

131.159.42.5	10000011 . 10011111 . 00101010 . 00000101
/26	11111111 . 11111111 . 11111111 . 11000000
AND	10000011 . 10011111 . 00101010 . 00000000
131.159.2.0	10000011 . 10011111 . 00000010 . 00000000
XOR	00000000 . 00000000 . 00101000 . 00000000

Result is not 0: Route does not match

Lookup Data Structures for Computer Networks
Naive LPM - Example

Routing Table

Status	**Prefix**	**Next Hop**	**Interface**
Does not match	131.159.2.0/26	131.159.0.12	eth0
Checking	131.159.2.64/26	131.159.0.13	eth0
-	131.159.6.128/26	131.159.1.2	eth1
-	131.159.20.0/24	131.159.1.67	eth2
-	131.159.42.0/24	131.159.1.67	eth2
-	131.159.0.0/16	131.159.1.132	eth3
-	0.0.0.0/0	131.159.1.133	eth3

Example: Destination address: 131.159.42.5

131.159.42.5	10000011 . 10011111 . 00101010 . 00000101
/26	11111111 . 11111111 . 11111111 . 11000000
AND	10000011 . 10011111 . 00101010 . 00000000
131.159.2.64	10000011 . 10011111 . 00000010 . 01000000
XOR	00000000 . 00000000 . 00101000 . 01000000

Result is not 0: Route does not match

Lookup Data Structures for Computer Networks
Naive LPM - Example

Routing Table

Status	Prefix	Next Hop	Interface
Does not match	131.159.2.0/26	131.159.0.12	eth0
Does not match	131.159.2.64/26	131.159.0.13	eth0
Checking	131.159.6.128/26	131.159.1.2	eth1
-	131.159.20.0/24	131.159.1.67	eth2
-	131.159.42.0/24	131.159.1.67	eth2
-	131.159.0.0/16	131.159.1.132	eth3
-	0.0.0.0/0	131.159.1.133	eth3

Example: Destination address: `131.159.42.5`

131.159.42.5	10000011 . 10011111 . 00101010 . 00000101
/26	11111111 . 11111111 . 11111111 . 11000000
AND	10000011 . 10011111 . 00101010 . 00000000
131.159.6.128	10000011 . 10011111 . 00000110 . 10000000
XOR	00000000 . 00000000 . 00101100 . 10000000

Result is not 0: Route does not match

Lookup Data Structures for Computer Networks
Naive LPM - Example

Routing Table

Status	Prefix	Next Hop	Interface
Does not match	131.159.2.0/26	131.159.0.12	eth0
Does not match	131.159.2.64/26	131.159.0.13	eth0
Does not match	131.159.6.128/26	131.159.1.2	eth1
Checking	131.159.20.0/24	131.159.1.67	eth2
-	131.159.42.0/24	131.159.1.67	eth2
-	131.159.0.0/16	131.159.1.132	eth3
-	0.0.0.0/0	131.159.1.133	eth3

Example: Destination address: `131.159.42.5`

131.159.42.5	10000011 . 10011111 . 00101010 . 00000101
/24	11111111 . 11111111 . 11111111 . 00000000
AND	10000011 . 10011111 . 00101010 . 00000000
131.159.20.0	10000011 . 10011111 . 00010100 . 00000000
XOR	00000000 . 00000000 . 00111110 . 00000000

Result is not 0: Route does not match

Naive LPM - Example

Routing Table

Status	Prefix	Next Hop	Interface
Does not match	131.159.2.0/26	131.159.0.12	eth0
Does not match	131.159.2.64/26	131.159.0.13	eth0
Does not match	131.159.6.128/26	131.159.1.2	eth1
Does not match	131.159.20.0/24	131.159.1.67	eth2
Checking	131.159.42.0/24	131.159.1.67	eth2
-	131.159.0.0/16	131.159.1.132	eth3
-	0.0.0.0/0	131.159.1.133	eth3

Example: Destination address: `131.159.42.5`

131.159.42.5	10000011 . 10011111 . 00101010 . 00000101
/24	11111111 . 11111111 . 11111111 . 00000000
AND	10000011 . 10011111 . 00101010 . 00000000
131.159.42.0	10000011 . 10011111 . 00101010 . 00000000
XOR	00000000 . 00000000 . 00000000 . 00000000

Result is 0: Route does match, next Hop is `131.159.1.67`

Lookup Data Structures for Computer Networks
Naive LPM - Example

Routing Table

Status	Prefix	Next Hop	Interface
Does not match	131.159.2.0/26	131.159.0.12	eth0
Does not match	131.159.2.64/26	131.159.0.13	eth0
Does not match	131.159.6.128/26	131.159.1.2	eth1
Does not match	131.159.20.0/24	131.159.1.67	eth2
Checking	131.159.42.0/24	131.159.1.67	eth2
-	131.159.0.0/16	131.159.1.132	eth3
-	0.0.0.0/0	131.159.1.133	eth3

Example: Destination address: `131.159.42.5`

131.159.42.5	10000011 . 10011111 . 00101010 . 00000101
/24	11111111 . 11111111 . 11111111 . 00000000
AND	10000011 . 10011111 . 00101010 . 00000000
131.159.42.0	10000011 . 10011111 . 00101010 . 00000000
XOR	00000000 . 00000000 . 00000000 . 00000000

Result is 0: Route does match, next Hop is `131.159.1.67`

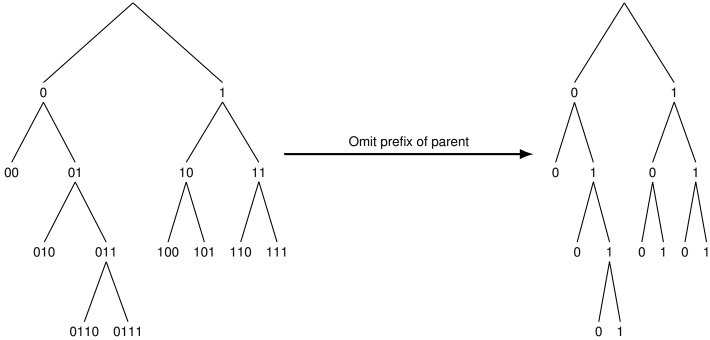
Note: Additional steps for MAC address resolution may be necessary

Trie structures

Basic idea of a trie:

- Tree with more invariants
- The content of a parent node is a prefix of the node's content
- Useful for a variety of index structures, can be used for LPM
- Idea: Model the entire IP space inside a trie

Simple (generic) example:



Tree-based Lookup
Basic Trie - Algorithm I

Build Algorithm

1. Write all next hops (including interface) into a table, assign each node a unique ID (NH-ID)
2. Write all FIB entry prefixes at the corresponding position in the trie, starting with the root (default gateway)

Tree-based Lookup
Basic Trie - Example

A

Corresponding routing table:

Prefix	binary	Next Hop	Interface
0.0.0.0/0	`b0000...`	192.0.0.1	eth0

NH-ID	Next Hop	Interface
A	192.0.0.1	eth0

Build-up process:

- Add default route
- Enter one next hop into NH-ID table
- Enter default route as root

Tree-based Lookup
Basic Trie - Example

Corresponding routing table:

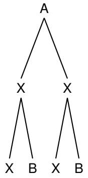
Prefix	`binary`	Next Hop	Interface
0.0.0.0/0	`b0000...`	192.0.0.1	eth0

NH-ID	Next Hop	Interface
A	192.0.0.1	eth0

Build-up process:

- No new routes with length /1
- Add null nodes at level 1

Tree-based Lookup
Basic Trie - Example



Corresponding routing table:

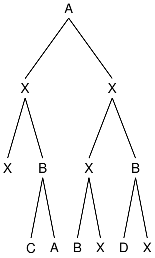
Prefix	binary	Next Hop	Interface
64.0.0.0/2	`b0100...`	192.0.0.2	eth1
192.0.0.0/2	`b1100...`	192.0.0.2	eth1
0.0.0.0/0	`b0000...`	192.0.0.1	eth0

NH-ID	Next Hop	Interface
A	192.0.0.1	eth0
B	192.0.0.2	eth1

Build-up process:

- Add two /2 routes
- Add one next hop entry to NH-ID table
- Enter nodes in trie at level 2

Tree-based Lookup
Basic Trie - Example



Corresponding routing table:

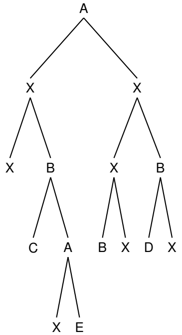
Prefix	binary	Next Hop	Interface
64.0.0.0/3	`b0100...`	192.0.0.3	eth2
96.0.0.0/3	`b0110...`	192.0.0.1	eth0
128.0.0.0/3	`b1000...`	192.0.0.2	eth1
192.0.0.0/3	`b1100...`	192.0.0.4	eth3
64.0.0.0/2	`b0100...`	192.0.0.2	eth1
192.0.0.0/2	`b1100...`	192.0.0.2	eth1
0.0.0.0/0	`b0000...`	192.0.0.1	eth0

NH-ID	Next Hop	Interface
A	192.0.0.1	eth0
B	192.0.0.2	eth1
C	192.0.0.3	eth2
D	192.0.0.4	eth3

Build-up process:

- Add four /3 routes
- Add two hop entries to NH-ID table
- Enter nodes in trie at level 3

Tree-based Lookup
Basic Trie - Example



NH-ID	Next Hop	Interface
A	192.0.0.1	eth0
B	192.0.0.2	eth1
C	192.0.0.3	eth2
D	192.0.0.4	eth3
E	192.0.0.5	eth4

Corresponding routing table:

Prefix	binary	Next Hop	Interface
112.0.0.0/4	b0111...	192.0.0.5	eth4
64.0.0.0/3	b0100...	192.0.0.3	eth2
96.0.0.0/3	b0110...	192.0.0.1	eth0
128.0.0.0/3	b1000...	192.0.0.2	eth1
192.0.0.0/3	b1100...	192.0.0.4	eth3
64.0.0.0/2	b0100...	192.0.0.2	eth1
192.0.0.0/2	b1100...	192.0.0.2	eth1
0.0.0.0/0	b0000...	192.0.0.1	eth0

Build-up process:

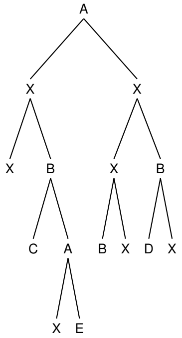
- Add one /4 route
- Add one hop entry to NH-ID table
- Enter nodes in trie at level 4

Basic Trie - Algorithm II

Lookup Algorithm

1. Set `curNode` to the root
2. Set `level` to `0`
3. Set `lastNH-ID` to `invalid`
4. If `curNode.NH-ID` exists: Set `lastNH-ID` to `curNode.NH-ID`
5. Check `level`-th bit of destination address
    - If 0: Set `curNode` to `curNode.left`
    - If 1: Set `curNode` to `curNode.right`
6. If `curNode` is `NULL`: Return `lastNH-ID`
7. `level++`
8. Goto step 4

Tree-based Lookup
Basic Trie - Example

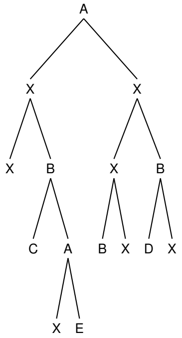


NH-ID	Next Hop	Interface
A	192.0.0.1	eth0
B	192.0.0.2	eth1
C	192.0.0.3	eth2
D	192.0.0.4	eth3
E	192.0.0.5	eth4

Corresponding routing table:

Prefix	Next Hop	Interface
112.0.0.0/4	192.0.0.5	eth4
64.0.0.0/3	192.0.0.3	eth2
96.0.0.0/3	192.0.0.1	eth0
128.0.0.0/3	192.0.0.2	eth1
192.0.0.0/3	192.0.0.4	eth3
64.0.0.0/2	192.0.0.2	eth1
192.0.0.0/2	192.0.0.2	eth1
0.0.0.0/0	192.0.0.1	eth0

Tree-based Lookup
Basic Trie - Example



NH-ID	Next Hop	Interface
A	192.0.0.1	eth0
B	192.0.0.2	eth1
C	192.0.0.3	eth2
D	192.0.0.4	eth3
E	192.0.0.5	eth4

Corresponding routing table:

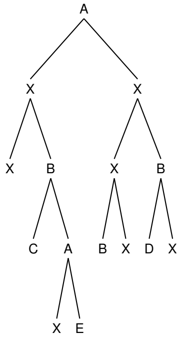
Prefix	Next Hop	Interface
112.0.0.0/4	192.0.0.5	eth4
64.0.0.0/3	192.0.0.3	eth2
96.0.0.0/3	192.0.0.1	eth0
128.0.0.0/3	192.0.0.2	eth1
192.0.0.0/3	192.0.0.4	eth3
64.0.0.0/2	192.0.0.2	eth1
192.0.0.0/2	192.0.0.2	eth1
0.0.0.0/0	192.0.0.1	eth0

Example lookup

Destination IP address `112.1.1.1`:

$\Rightarrow$ `0x70010101` $\Rightarrow$ `b01110000 ...`

Tree-based Lookup
Basic Trie - Example



NH-ID	Next Hop	Interface
A	192.0.0.1	eth0
B	192.0.0.2	eth1
C	192.0.0.3	eth2
D	192.0.0.4	eth3
E	192.0.0.5	eth4

Corresponding routing table:

Prefix	Next Hop	Interface
112.0.0.0/4	192.0.0.5	eth4
64.0.0.0/3	192.0.0.3	eth2
96.0.0.0/3	192.0.0.1	eth0
128.0.0.0/3	192.0.0.2	eth1
192.0.0.0/3	192.0.0.4	eth3
64.0.0.0/2	192.0.0.2	eth1
192.0.0.0/2	192.0.0.2	eth1
0.0.0.0/0	192.0.0.1	eth0

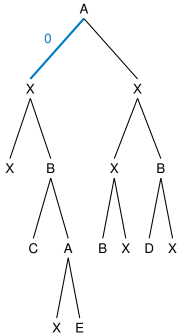
Example lookup

Destination IP address `112.1.1.1`:

$\Rightarrow$ `0x70010101` $\Rightarrow$ `b01110000 ...`

- Start at root $\rightarrow$ NH-ID: A

Tree-based Lookup
Basic Trie - Example



NH-ID	Next Hop	Interface
A	192.0.0.1	eth0
B	192.0.0.2	eth1
C	192.0.0.3	eth2
D	192.0.0.4	eth3
E	192.0.0.5	eth4

Corresponding routing table:

Prefix	Next Hop	Interface
112.0.0.0/4	192.0.0.5	eth4
64.0.0.0/3	192.0.0.3	eth2
96.0.0.0/3	192.0.0.1	eth0
128.0.0.0/3	192.0.0.2	eth1
192.0.0.0/3	192.0.0.4	eth3
64.0.0.0/2	192.0.0.2	eth1
192.0.0.0/2	192.0.0.2	eth1
0.0.0.0/0	192.0.0.1	eth0

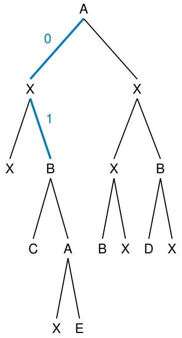
Example lookup

Destination IP address `112.1.1.1`:

$\Rightarrow$ `0x70010101` $\Rightarrow$ `b01110000` . . .

- Start at root $\rightarrow$ NH-ID: A
- 1st bit 0 $\rightarrow$ NH-ID: still A

Tree-based Lookup
Basic Trie - Example



NH-ID	Next Hop	Interface
A	192.0.0.1	eth0
B	192.0.0.2	eth1
C	192.0.0.3	eth2
D	192.0.0.4	eth3
E	192.0.0.5	eth4

Corresponding routing table:

Prefix	Next Hop	Interface
112.0.0.0/4	192.0.0.5	eth4
64.0.0.0/3	192.0.0.3	eth2
96.0.0.0/3	192.0.0.1	eth0
128.0.0.0/3	192.0.0.2	eth1
192.0.0.0/3	192.0.0.4	eth3
64.0.0.0/2	192.0.0.2	eth1
192.0.0.0/2	192.0.0.2	eth1
0.0.0.0/0	192.0.0.1	eth0

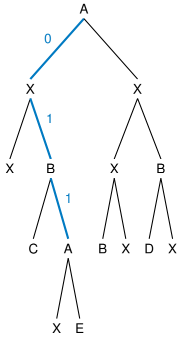
Example lookup

Destination IP address `112.1.1.1`:

$\Rightarrow$ `0x70010101` $\Rightarrow$ `b01110000 ...`

- Start at root $\rightarrow$ NH-ID: A
- 1st bit 0 $\rightarrow$ NH-ID: still A
- 2nd bit 1 $\rightarrow$ NH-ID: B

Tree-based Lookup
Basic Trie - Example



NH-ID	Next Hop	Interface
A	192.0.0.1	eth0
B	192.0.0.2	eth1
C	192.0.0.3	eth2
D	192.0.0.4	eth3
E	192.0.0.5	eth4

Corresponding routing table:

Prefix	Next Hop	Interface
112.0.0.0/4	192.0.0.5	eth4
64.0.0.0/3	192.0.0.3	eth2
96.0.0.0/3	192.0.0.1	eth0
128.0.0.0/3	192.0.0.2	eth1
192.0.0.0/3	192.0.0.4	eth3
64.0.0.0/2	192.0.0.2	eth1
192.0.0.0/2	192.0.0.2	eth1
0.0.0.0/0	192.0.0.1	eth0

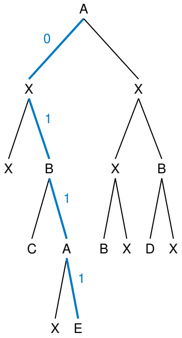
Example lookup

Destination IP address `112.1.1.1`:

$\Rightarrow$ `0x70010101` $\Rightarrow$ `b01110000` . . .

- Start at root $\rightarrow$ NH-ID: A
- 1st bit 0 $\rightarrow$ NH-ID: still A
- 2nd bit 1 $\rightarrow$ NH-ID: B
- 3rd bit 1 $\rightarrow$ NH-ID: A

Tree-based Lookup
Basic Trie - Example



NH-ID	Next Hop	Interface
A	192.0.0.1	eth0
B	192.0.0.2	eth1
C	192.0.0.3	eth2
D	192.0.0.4	eth3
E	192.0.0.5	eth4

Corresponding routing table:

Prefix	Next Hop	Interface
112.0.0.0/4	192.0.0.5	eth4
64.0.0.0/3	192.0.0.3	eth2
96.0.0.0/3	192.0.0.1	eth0
128.0.0.0/3	192.0.0.2	eth1
192.0.0.0/3	192.0.0.4	eth3
64.0.0.0/2	192.0.0.2	eth1
192.0.0.0/2	192.0.0.2	eth1
0.0.0.0/0	192.0.0.1	eth0

Example lookup

Destination IP address `112.1.1.1`:

$\Rightarrow$ `0x70010101` $\Rightarrow$ `b01110000 ...`

- Start at root $\rightarrow$ NH-ID: A
- 1st bit 0 $\rightarrow$ NH-ID: still A
- 2nd bit 1 $\rightarrow$ NH-ID: B
- 3rd bit 1 $\rightarrow$ NH-ID: A
- 4th bit 1 $\rightarrow$ NH-ID: E

Tree-based Lookup
Path Compressed Trie

Problem

- Basic Tries are very wasteful
- Most nodes would not be occupied by a route
- Most routes have long prefixes (16 or 24)

Solution

- Aggregate chains of “empty” nodes into one (or zero) nodes
- This compresses the path the algorithm has to traverse
- Fewer nodes to save $\longrightarrow$ less memory consumption
- Fewer nodes to visit $\longrightarrow$ less CPU consumption

Such a trie is used in the FreeBSD kernel

Tree-based Lookup
Level Compressed Trie

Problem

- Basic Tries are very wasteful
- Each node has only two children
- Very large (binary) trees are built

Solution

- Nodes have 2^n children
- Pull children's children into the parent
- Fewer (but bigger) nodes to save $\longrightarrow$ better cache utilization
- Fewer nodes to visit $\longrightarrow$ less CPU consumption (branch misses)

Tree-based Lookup
Level and Path Compressed Trie

Idea

- Path compression and level compression are orthogonal
- Both can be combined
- Result: LPC-Trie

The Linux kernel uses an LPC-Trie

Routing and Forwarding

More Advanced Lookup Structures
Dir 24-8 [1]

Idea

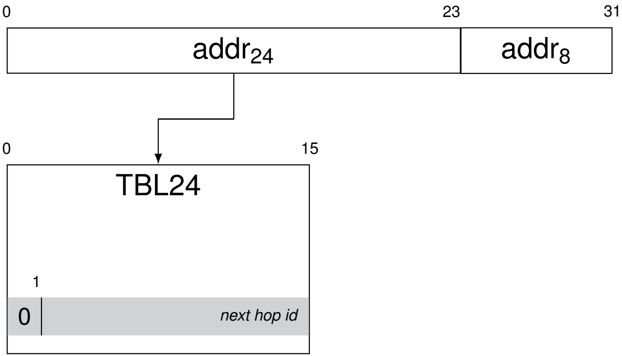
- Construct two large tables
- First 24 bit of the prefix are handled in the "TBL24"
- Last 8 bit in "TBLlong"
- Works because most prefixes are shorter than 24 bit (for IPv4)
- Dir 24-8 was designed for hardware implementation
  + simple control flow
  + only one memory lookup for $\leq$ /24 prefixes (2 accesses for $>$ /24 prefixes)
  - requires lots of memory ($>$ 32 MiB)

Construction

1. Project routes into flat, continuous address space that is associated with next hops
2. Collect all possible next hops, and store them in an array
3. If: there is one next hop for the entire /24 prefix p:
   - True: TBL24[p] = next-hop-ID
   - False: Enumerate all 256 addresses and next hops in TBLlong,
     TBL24[p] = start-of-TBLlong-entries $\oplus$ mark

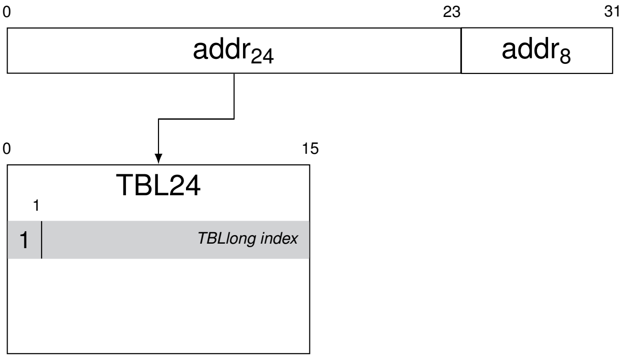
More Advanced Lookup Structures
Dir 24-8 Lookup [1]

Looking up $\leq$ /24 prefixes



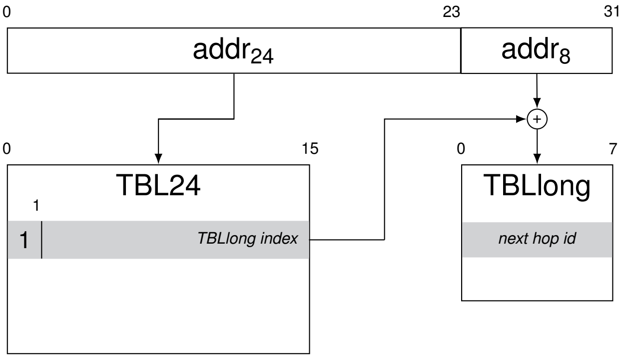
More Advanced Lookup Structures
Dir 24-8 Lookup [1]

Looking up $>$ /24 prefixes (part 1)



More Advanced Lookup Structures
Dir 24-8 Lookup [1]

Looking up $>$ /24 prefixes (part 2)



More Advanced Lookup Structures
DXR [2]

- DIR-24-8 uses fixed address separation (24/8bit)
- DXR offers a flexible address separation, e.g., 16/16bit
- Idea: create smaller routing tables to perform faster lookups

More Advanced Lookup Structures
DXR architecture

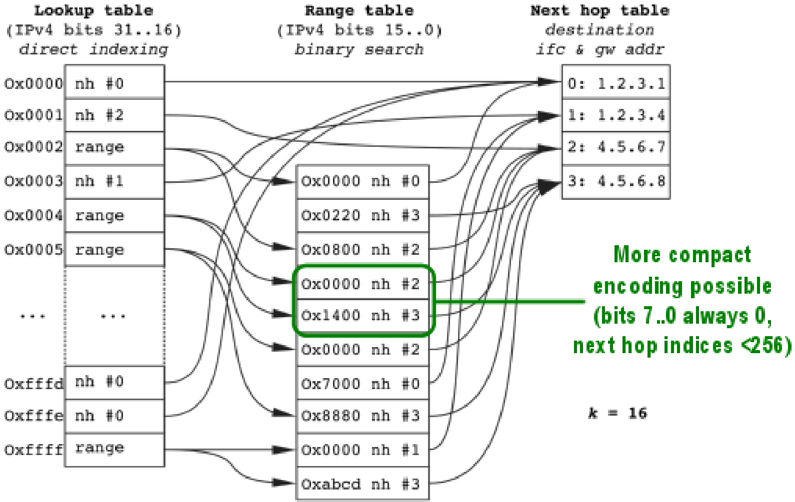


Figure 1: cf. Zec et al. [3]

More Advanced Lookup Structures
DXR table size

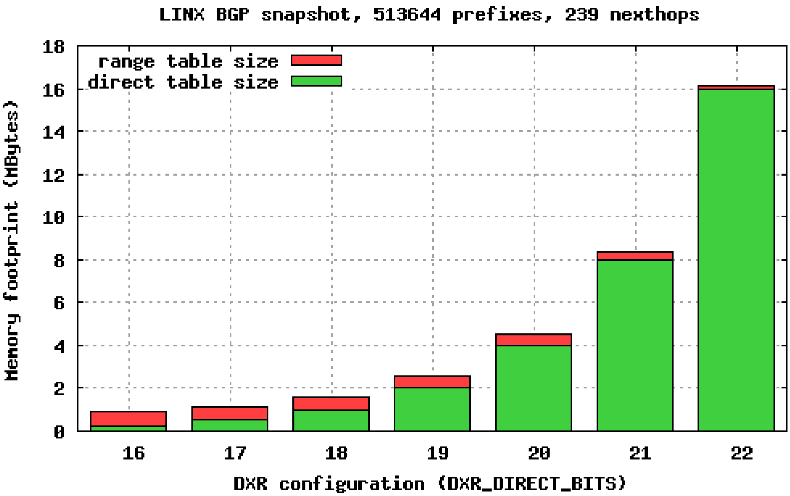


Figure 2: cf. Zec et al. [3]

More Advanced Lookup Structures
DXR performance test

- AMD FX-8350: 4 GHz, 8 cores, 16 KiB / 2 MiB / 8 MiB (L1/L2/L3 Cache)
- Xeon E5-1650: 3.2 GHz, 6 cores, 32 KiB / 256 KiB / 20 MiB (L1/L2/L3 Cache)

More Advanced Lookup Structures
DXR performance

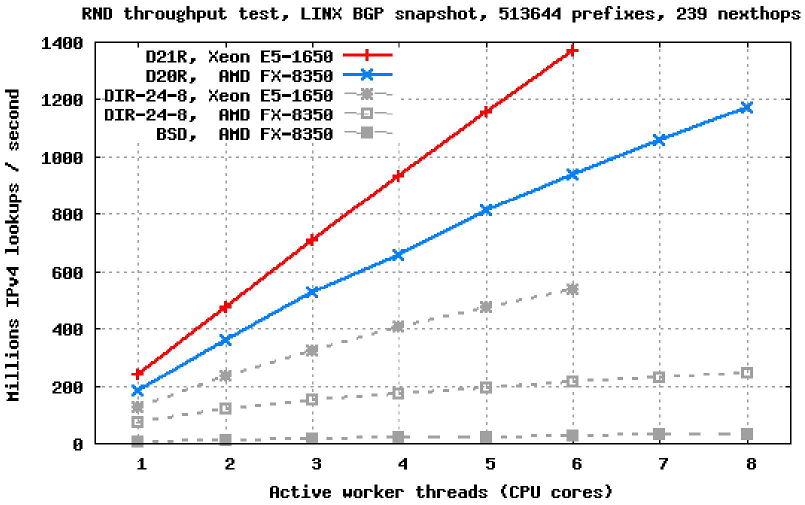


Figure 3: cf. Zec et al. [3]

More Advanced Lookup Structures
Conclusion

Costs of LPM

- LPM algorithms perform only cheap, simple operations (e.g., AND, XOR)
- Costs of LPM is mainly determined by the number of lookups performed:
  - Naive LPM worst case: #(number of routing table entries) lookups
  - Trie-based LPM worst case: #(number of bits in the IP address) lookups
  - DIR-24-8 LPM worst case: one or two lookups
- Faster lookups using smaller routing tables:
  - Smaller routing tables fit into smaller CPU caches
  - Trade-off between lookup optimization (DIR-24-8) and space optimization (Trie-based LPM)

Routing and Forwarding

[1] P. Gupta, S. Lin, and N. McKeown, “Routing lookups in hardware at memory access speeds,” in INFOCOM’98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, IEEE, vol. 3, 1998, pp. 1240–1247.

[2] M. Zec, L. Rizzo, and M. Mikuc, “Dxr: Towards a billion routing lookups per second in software,” ACM SIGCOMM Computer Communication Review, vol. 42, no. 5, pp. 29–36, 2012.

[3] M. Zec, L. Rizzo, and M. Mikuc, Dxr: Beyond two billion ipv4 routing lookups per second in software, last accessed: 2019-11-21, 2012. [Online]. Available: `http://www.nxlab.fer.hr/dxr/`.