Heuristic Binary Search: Adaptive and Fast IPv6 Route Lookup with Incremental Updates

Donghong Jiang, Yanbiao Li, Yuxuan Chen, Jing Hu, Yi Huang, Gaogang Xie
SDN and NFV have revolutionized modern network architectures

Hardware-based Devices
- carrier-grade NAT
- transcoder
- session border controller
- firewall
- DDoS protection
- QoE monitor
- DPI
- load balancer
- WAN accelerator
- BRAS
- ad insertion

Software-based Devices

Offer greater:
- programmability
- flexibility
- cost-effectiveness
Route/IP Lookup is a crucial function of software-based devices

- Principle: Longest Prefix Matching (LPM)

**Forwarding Information Base (FIB)**

<table>
<thead>
<tr>
<th>Marks</th>
<th>IP Prefixes</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1*</td>
</tr>
<tr>
<td>B</td>
<td>01*</td>
</tr>
<tr>
<td>C</td>
<td>000*</td>
</tr>
<tr>
<td>D</td>
<td>001*</td>
</tr>
<tr>
<td>E</td>
<td>010*</td>
</tr>
<tr>
<td>F</td>
<td>1011*</td>
</tr>
<tr>
<td>G</td>
<td>01000*</td>
</tr>
<tr>
<td>H</td>
<td>101111*</td>
</tr>
<tr>
<td>I</td>
<td>1110000*</td>
</tr>
<tr>
<td>J</td>
<td>1111111*</td>
</tr>
</tbody>
</table>

dIP: 10111111  →  Result: H
IPv6 lookup faces two key challenges (1/2)

- Challenge 1: high-speed lookup
  - Reason: IPv6 has longer addresses and prefixes

![Graph showing lookup speed comparison between IPv4 and IPv6 across different sites.](image-url)
IPv6 lookup faces two key challenges (2/2)

- **Challenge 2: dynamic adaptability**
  - **Reason:** variations in prefix distributions across network scenarios

- Internet backbone: /32~/48
- ISP networks: /48~/64
- Cloud and Campus gateways: /64
Existing software-based IP lookup schemes

<table>
<thead>
<tr>
<th>Category</th>
<th>Trie-based schemes</th>
<th>Hash-based schemes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Radix Trie</td>
<td>BS</td>
</tr>
<tr>
<td></td>
<td>Tree Bitmap</td>
<td>ABS</td>
</tr>
<tr>
<td></td>
<td>SAIL</td>
<td>MBS</td>
</tr>
<tr>
<td></td>
<td>Poptrie</td>
<td></td>
</tr>
</tbody>
</table>

Prioritize more prevalent prefix lengths

Search from short to long prefixes
Binary search on prefix lengths

<table>
<thead>
<tr>
<th></th>
<th>Trie-based</th>
<th>Hash-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>High-speed lookup</td>
<td>×</td>
<td>🍯</td>
</tr>
<tr>
<td>Dynamic adaptability</td>
<td>×</td>
<td>🍯</td>
</tr>
</tbody>
</table>

Our method, HBS, is based on ABS, with enhanced lookup performance and dynamic adaptability
Step 1: group prefixes based on lengths

Each group is maintained by a hash table

<table>
<thead>
<tr>
<th>Marks</th>
<th>IP Prefixes</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1*</td>
</tr>
<tr>
<td>B</td>
<td>01*</td>
</tr>
<tr>
<td>C</td>
<td>000*</td>
</tr>
<tr>
<td>D</td>
<td>001*</td>
</tr>
<tr>
<td>E</td>
<td>010*</td>
</tr>
<tr>
<td>F</td>
<td>1011*</td>
</tr>
<tr>
<td>G</td>
<td>01000*</td>
</tr>
<tr>
<td>H</td>
<td>101111*</td>
</tr>
<tr>
<td>I</td>
<td>1110000*</td>
</tr>
<tr>
<td>J</td>
<td>1111111*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Len</th>
<th>IP Prefixes (Rows: Seven Groups)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A: 1*</td>
</tr>
<tr>
<td>2</td>
<td>B: 01*</td>
</tr>
<tr>
<td>3</td>
<td>C: 000* D: 001* E: 010*</td>
</tr>
<tr>
<td>4</td>
<td>F: 1011*</td>
</tr>
<tr>
<td>5</td>
<td>G: 01000*</td>
</tr>
<tr>
<td>6</td>
<td>H: 101111*</td>
</tr>
<tr>
<td>7</td>
<td>I: 1110000* J: 1111111*</td>
</tr>
</tbody>
</table>
Review Asymmetric Binary Search (2/5)

- Step 2: organize groups into an Asymmetric Binary Search Tree

  Principle: place groups with more prefixes higher

<table>
<thead>
<tr>
<th>Len</th>
<th>IP Prefixes (Rows: Seven Groups)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A: 1*</td>
</tr>
<tr>
<td>2</td>
<td>B: 01*</td>
</tr>
<tr>
<td>3</td>
<td>C: 000*</td>
</tr>
<tr>
<td></td>
<td>D: 001*</td>
</tr>
<tr>
<td></td>
<td>E: 010*</td>
</tr>
<tr>
<td>4</td>
<td>F: 1011*</td>
</tr>
<tr>
<td>5</td>
<td>G: 01000*</td>
</tr>
<tr>
<td>6</td>
<td>H: 101111*</td>
</tr>
<tr>
<td>7</td>
<td>I: 1110000*</td>
</tr>
<tr>
<td></td>
<td>J: 1111111*</td>
</tr>
</tbody>
</table>

Diagram:

```
        3  C: 000  D: 001  E: 010
        /     /     /    /
       2  B: 01  7  I: 1110000  J: 1111111
       /     
      1  A: 1
       /     
      4  F: 1011
       /     
      5    G: 01000
       /     
      6  H: 101111
```
Review Asymmetric Binary Search (3/5)

Step 3: Prefix Projection

- Prefixes managed by a node’s right sub-tree project markers onto that node.
Review Asymmetric Binary Search (4/5)

- Step 4: Pre-compute a Best Matching Prefix (BMP) for each marker
  - Reason: resolve the “backtracking” issue introduced by markers

- BMP is the longest sub-prefix

Without a match, the BMP of the most recent marker is used as the result.
Review Asymmetric Binary Search (5/5)

● Lookup example: Lookup(10111111)

  ➢ Principle: hit a marker, go right; miss, go left; hit pure prefix, end

```
F: 1011
H: 101111
A: 1
M: 10111(F)
3
CD
E: 010
M1: 111(A)
M2: 101(A)
2
B: 01
1
A: 1
4
F: 1011
5
G: 01000
M3: 10111(F)
7
I
J: 1111111
101
Miss
3
5
4
6
Hit
Hit
Hit
Temporary result
A F
Final result
H
```
Defects of ABS and our solutions

- **High-speed Lookup**
  - Matching prefixes located in **lower layers**
  - Numerous distinct prefix **lengths**

- **Dynamic Adaptability**

  Our solution: **Heuristic Binary Search**
Heuristic Binary Search (1/4)

- Observation: markers often originate from a **limited** number of nodes

  ➢ **Origin Nodes**

  Experiments on 13 FIBs:
  ➢ over 88%, only one
  ➢ over 99%, no more than four
Heuristic Binary Search (2/4)

- Narrow the search scope to the origin nodes
  - Example: Lookup(01000000)

```
Prefix marker

<table>
<thead>
<tr>
<th>Node</th>
<th>Value</th>
<th>Prefix</th>
<th>Marker</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>CD</td>
<td>E: 010</td>
<td>M₁: 111, M₂: 101</td>
</tr>
<tr>
<td>2</td>
<td>B: 01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>A: 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>F: 1011</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>H: 101111</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>I</td>
<td>J: 111111</td>
<td></td>
</tr>
</tbody>
</table>
```

Example:
```
lookup(01000000)
```

1. Narrow the search scope to the origin nodes.
2. Temporary result: E
3. Final result: G
4. Reduce one hash table lookup.
Multiple origin nodes: binary search on them

Example: Lookup(10110000)

Reduce one hash table lookup
Heuristic Binary Search (4/4)

- Accelerate lookup: pre-compute a Heuristic Search Path (HSP)

  Example: Lookup(10111111)

  \[ \text{HSP: } 5 \rightarrow 4 \]
  \[ \text{[4, 6]} \]

  Initial HSP: \[ \begin{align*} 3 \rightarrow 2 \rightarrow 1 \end{align*} \]

  Current HSP: \[ \begin{align*} 5 \rightarrow 4 \end{align*} \]

  Current HSP: \[ \begin{align*} 6 \end{align*} \]

  \[
  \begin{array}{|c|c|}
  \hline
  \text{prefix} & \text{marker} \\
  \hline
  \text{A: 1} & \text{F: 1011} \\
  \hline
  \end{array}
  \]

  \[
  \begin{array}{|c|c|c|}
  \hline
  \text{E: 010} & \text{M}_1: 111 & \text{M}_2: 101 \\
  \hline
  \end{array}
  \]

  \[
  \begin{array}{|c|c|c|}
  \hline
  \text{J: 111111} & \text{M}_3: 10111 \\
  \hline
  \end{array}
  \]
Defects of ABS and our solutions

- High-speed Lookup

- Dynamic Adaptability
  - Prefix distribution of the FIB is **unknown**
  - Prefix distribution **will change** over time

Our solution: **Tree Rotation**
Tree Rotation (1/6)

If a node $x$ has more prefixes than its parent $y$, we rotate $x$ up.

Left Rotation

Right Rotation
Delete markers that originates only from x and A

Traverse all elements of node y and delete eligible markers
Tree Rotation (3/6)

Right Rotation

Project node y’s all elements onto node x

Project markers
Other adjustments: calculation method of HSP

How to reduce the overhead of tree rotations?
- Add a parameter $\alpha$ to the rotation condition

How to determine the optimal value of $\alpha$?
- Address the problem both theoretically and experimentally
Tree Rotation (5/6)

In theory

Amortized time complexity of each rotation operation:

\[
\begin{align*}
O(W^2N), & \quad \text{if } \alpha = 1 \\
O(W^2 \log_\alpha N), & \quad \text{if } \alpha > 1
\end{align*}
\]

Note: \( w \) is the number of distinct prefix lengths, \( N \) is the number of inserted prefixes.

As long as \( \alpha \) exceeds 1, the complexity goes from linear to logarithmic
For experiment

\( \alpha = 1.1 \) provides a good trade-off between overhead and efficiency
Evaluation Setup

● Platform
  ➢ AMD EPYC 7742 CPU

● FIBs
  ➢ 13 FIBs from IXP, ISP, Campus, and Cloud

● Traffics
  ➢ Synthesized a trace for each prefix

● Compared with
  ➢ Tree Bitmap, SAIL, Poptrie, BS, and ABS

### FIB details

<table>
<thead>
<tr>
<th>Name</th>
<th>Source</th>
<th>IP prefix distribution</th>
<th># of prefixes</th>
</tr>
</thead>
<tbody>
<tr>
<td>IXP1</td>
<td>RRC01</td>
<td>32(14.2%), 40(7.3%), 44(8.3%), 48(46.6%), ...</td>
<td>163,769</td>
</tr>
<tr>
<td>IXP2</td>
<td>RRC11</td>
<td>32(14.3%), 40(8.0%), 44(8.1%), 48(47.4%), ...</td>
<td>167,114</td>
</tr>
<tr>
<td>IXP3</td>
<td>RRC15</td>
<td>32(13.8%), 40(7.9%), 44(7.9%), 48(46.9%), ...</td>
<td>168,898</td>
</tr>
<tr>
<td>IXP4</td>
<td>RRC19</td>
<td>32(13.6%), 40(7.1%), 44(8.1%), 48(46.4%), ...</td>
<td>164,474</td>
</tr>
<tr>
<td>IXP5</td>
<td>RRC23</td>
<td>32(14.3%), 40(7.2%), 44(8.2%), 48(46.3%), ...</td>
<td>165,586</td>
</tr>
<tr>
<td>ISP1</td>
<td>Netcologne</td>
<td>47(2%), 48(60%), 57(3%), 61(3%), 64(32%)</td>
<td>1,000,000</td>
</tr>
<tr>
<td>ISP2</td>
<td>Orange</td>
<td>54(3%), 55(16%), 56(80%), 64(1%)</td>
<td>1,000,000</td>
</tr>
<tr>
<td>ISP3</td>
<td>DTAG</td>
<td>56(50%), 62(1%), 63(3%), 64(46%)</td>
<td>1,000,000</td>
</tr>
<tr>
<td>ISP4</td>
<td>Sky UK</td>
<td>55(5%), 56(78%), 60(2%), 63(1%), 64(14%)</td>
<td>1,000,000</td>
</tr>
<tr>
<td>ISP5</td>
<td>Free SAS</td>
<td>58(4%), 59(18%), 60(70%), 62(1%), 64(7%)</td>
<td>1,000,000</td>
</tr>
<tr>
<td>ISP6</td>
<td>Kabel DE</td>
<td>59(1%), 60(1%), 62(55%), 63(8%), 64(35%)</td>
<td>1,000,000</td>
</tr>
<tr>
<td>Cloud</td>
<td>public cloud</td>
<td>48(0.24%), 56(0.17%), 64(99.5%), ...</td>
<td>7,812</td>
</tr>
<tr>
<td>Campus</td>
<td>university</td>
<td>54(0.2%), 56(3.8%), 60(0.9%), 64(95.1%)</td>
<td>471</td>
</tr>
</tbody>
</table>
HBS achieves $2.32X \sim 9.02X$, $1.33X \sim 5.32X$, and $1.05X \sim 5.56X$ speed-up, compared to Tree Bitmap, SAIL, and Poptrie.
HBS achieves up to 98.5% and 89.3% reductions, compared to SAIL and Poptrie, while the gap with Tree Bitmap ranges from 6.0MB to 31.8MB.
Update Performance

- Using IXP1 FIB

**Insert Overhead**

<table>
<thead>
<tr>
<th></th>
<th>Tree Bitmap</th>
<th>Poptrie</th>
<th>SAIL</th>
<th>HBS</th>
</tr>
</thead>
<tbody>
<tr>
<td>IXP1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ISP1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cloud</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Campus</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Delete Overhead**

<table>
<thead>
<tr>
<th></th>
<th>Tree Bitmap</th>
<th>Poptrie</th>
<th>SAIL</th>
<th>HBS</th>
</tr>
</thead>
<tbody>
<tr>
<td>IXP1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ISP1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cloud</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Campus</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Comprehensive View

- Using IXP1 FIB
Conclusion

- HBS
  - Heuristic Binary Search (High-speed Lookup)
  - Tree Rotation (Dynamic Adaptability)
  - Incremental Prefix Updates (See our paper)

- Future works
  - Evaluate with real traffic
  - Evaluate with more platforms
  - Integrated to the DPDK L3fwd application
Thanks!

Q&A