Clustered Linked List Forest For IPv6 Lookup

Oğuzhan ERDEM\textsuperscript{1} and Aydın CARUS\textsuperscript{2}

\textsuperscript{1}Department of Electrical and Electronics Engineering
\textsuperscript{2}Department of Computer Engineering
Trakya University, TURKEY
\{ogerdem, aydinc\}@trakya.edu.tr
Agenda

- Background
- Prior Work
- Our Solutions
- Implementation Results
- Conclusions & Future Work
Agenda

- Background
- Prior Work
- Our Solutions
- Implementation Results
- Conclusions & Future Work
Background

- **IP Lookup**
  - Core functions of Internet routers
  - Match the destination IP address to the prefixes in the routing table
  - Longest Prefix Matching (LPM)
    - Multiple matched prefixes possible
    - Only the longest matched prefix will be used

- **Example**
  - IP address “114.110.88.212”
  - Routing table:

<table>
<thead>
<tr>
<th>Routing Prefixes</th>
<th>Next-Hop Forwarding</th>
</tr>
</thead>
<tbody>
<tr>
<td>114.110.<em>.</em>.</td>
<td>Port 0</td>
</tr>
<tr>
<td>114.110.88.*.</td>
<td>Port 1</td>
</tr>
</tbody>
</table>

- Matches both, but the second match is used
Tree based IP Lookup

- **Binary Trie (BT)**
  - Simple data structure
  - Easy update process
  - Large number of memory accesses

- **Binary Search Tree (BST)**
  - Each node stores prefix values
  - Better memory performance
  - Complex pre-processing
    - Prefix expansion
    - Sorting requirement
  - Hard incremental updates
Performance Metrics

- **Throughput**
  - Support for higher link data rates
    - 100 Gbps network interfaces and links

- **Memory usage**
  - Rapid growth of routing table size
    - Estimated as ~500K entries in 2015
  - Transition from IPv4 (32-bit) to IPv6 (128-bit)
    - IPv4 address exhaustion
  - State-of-the-art designs cannot support large routing tables due to the available on-chip memory and the number of I/O pins of FPGAs

- **Update Capability**
  - Types of updates – Insert, delete and modify
  - How fast an update operation can be performed
Agenda

- Background
- Prior Work
- Our Solutions
- Implementation Results
- Conclusions & Future Work
Hardware based solutions

- **Linear pattern search in TCAM**
  - One lookup per cycle
  - Expensive, power-hungry, low density, large access time, need for a priority encoder

- **Hash based solutions**
  - Simple and fast
  - Large number of different hash tables and functions for each prefix lengths, hash collisions.

- **Binary bit traversal in pipelined tries**
  - Good throughput performance and support quick prefix updates
  - Inefficient memory usage

- **Binary value search in pipelined trees**
  - Good memory performance
  - Complex pre-processing and updates
Hierarchical Search Structures

- **Hierarchical trie for packet classification**
  - Hierarchically connected binary tries
  - One big SA trie from SA prefixes
  - Many small DA tries using DA prefixes
  - Hardware implementation unfeasible
    - Backtracking problem

- **Hierarchical structures for IP lookup**
  - Only DA is used
  - Split prefixes into two parts
  - Hierarchically connected data structures
  - Substantial memory saving
    - Bit strings shared by multiple prefixes are stored only once
    - No backtracking: Fixed size strings naturally pairwise disjoint
Agenda

- Background
- Prior Work
- Our Solutions
- Implementation Results
- Conclusions & Future Work
Our Solutions

- **Clustered Linked List Forest (CLLF) structure**
  - Hierarchical two stage data structure
  - Reduces the memory requirement
  - Simplifies the table updates

- **Linear pipelined SRAM-based architecture on FPGAs**
  - Supports up to 712K IPv6 prefixes
  - Sustained throughput of 432 Million lookups per second (138 Gbps for the minimum packet size of 40 Bytes)
**Clustered Linked List Forest (CLLF)**

- **Hierarchical two stage data structure**
  - A prefix $P$ can be expressed as the concatenation of two substrings $P_x$ and $P_y$ such that $P = P_x P_y$.
  - **Stage 1**: Clustering stage ($P_x$)  **Stage 2**: Linked List (LL) stage ($P_y$)

![Diagram of CLLF structure]

- $C_i^k$, $C_i^{10}$, $C_i^{11}$, $C_i^{12}$, $C_i^{13}$
- $N_{i,0}$, $N_{i,1}$, $N_{i,2}$
- $LL_0$, $LL_1$, $LL_2$
Clustering

- **Length-based**
  - 64 clusters in IPv6 the worst case
  - Each cluster is indexed starting from 1 to \( n \) (\( C_i \) represents \( i^{th} \) cluster)
  - Reduce the number of clusters by merging

- **Parameters**
  - **Block size (BS)**: The number of distinct lengths involved in a cluster
    - \( C_i^L \): Each distinct length (L) block of prefixes in a \( C_i \)
    - Ex: If a cluster \( C_i \) includes the prefixes of lengths from 8 to 11, then \( BS(C_i)=4 \) and \( C_i^9 \) represents a bunch of prefixes with length 9.
  - **Initial prefix stride (IPS)**: The length of substring \( P_x (|P_x|) \) in \( P = P_x P_y \)
    - Ex: If \( IPS(C_i)=4 \), then a prefix \( P = 110010111^* \) in \( C_i \) can be represented using two substrings \( P_x = 1100 \) and \( P_y = 10111^* \)
Data Structure

- **Stage 1: Clustering stage**
  - Each group has a *Binary Search Tree (BST)* using $P_x$ of prefixes.

- **Stage 2: Linked List (LL) stage**
  - Each node of BST connects to a *linked list data structure*.
  - Each node in a LL contains different $P_y$ substring with its length.
  - $BS(C_1)=2$, $BS(C_2)=2$, $BS(C_3)=3$, $IPS(C_1)=2$, $IPS(C_2)=3$, $IPS(C_3)=4$.

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Next Hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$</td>
<td>10*</td>
</tr>
<tr>
<td>$P_1$</td>
<td>00*</td>
</tr>
<tr>
<td>$P_2$</td>
<td>000*</td>
</tr>
<tr>
<td>$P_3$</td>
<td>101*</td>
</tr>
<tr>
<td>$P_4$</td>
<td>111*</td>
</tr>
<tr>
<td>$P_5$</td>
<td>1000*</td>
</tr>
<tr>
<td>$P_6$</td>
<td>0111*</td>
</tr>
<tr>
<td>$P_7$</td>
<td>10110*</td>
</tr>
<tr>
<td>$P_8$</td>
<td>11110*</td>
</tr>
<tr>
<td>$P_9$</td>
<td>101110*</td>
</tr>
<tr>
<td>$P_{10}$</td>
<td>011110*</td>
</tr>
<tr>
<td>$P_{11}$</td>
<td>101111*</td>
</tr>
<tr>
<td>$P_{12}$</td>
<td>0111011*</td>
</tr>
<tr>
<td>$P_{13}$</td>
<td>0110111*</td>
</tr>
<tr>
<td>$P_{14}$</td>
<td>01100000*</td>
</tr>
<tr>
<td>$P_{15}$</td>
<td>01101011*</td>
</tr>
</tbody>
</table>

$C_1=$ {$C_1^2, C_1^3$}  

$C_2=$ {$C_2^4, C_2^5$}  

$C_3=$ {$C_3^6, C_3^7, C_3^8$}
IP Lookup

Split IP address into two: $IP_x$ and $IP_y$
- Based on the IPS value of corresponding cluster
- Search $IP_x$ in Stage 1 (BST) and $IP_y$ in Stage 2 (Linear search).

Parallel searches in clusters
- If a match is found in BST, proceed to the connected LL.
- At each LL node, an $IP_y$ sub-address is compared with $P_y$ sub-prefix.
- Update search result if longer match than the previous match.
- Search terminates when all the nodes in a LL is traversed
- Outputs from all clusters are given to the priority encoder that chooses the longest prefix.
Optimization

- **Node size optimizations**
  - **BST**
    - The nodes of BST can be enhanced to store $P_y$ substrings.
    - $P_{\text{tree}}$: A limit for the number of $P_y$ substrings per tree node.
  - **LL**
    - LL nodes can be enhanced to store more than one $P_y$ substrings.
    - $P_{LL}$: Upper bound for the number of $P_y$ prefixes per LL node.

- **Clustering optimizations**
  - **BS**
    - Defines the number and size of clusters.
    - Larges BS results in less number of pipelines in hardware implementation.
  - **IPS**
    - Determines the number of nodes in BST and LL.
    - Smaller IPS reduces the number of BST nodes but increase the number of LL nodes and implicitly the delay.
Architecture

➢ Overall

Incoming Packet

Header extractor

BST

BST

BST

Cluster_1

Cluster_2

LL

Cluster_n

Priority Encoder

Result

➢ 1 stage

Clustering stage

LL stage
Agenda

- Background
- Prior Work
- Our Solutions
- Implementation Results
- Conclusions & Future Work
Results

- Experimental setup
  - Ten IPv4 core routing tables from Project - RIS on 06/03/2010.
  - Ten synthetic IPv6 routing tables generated using IPv4 core RTs

<table>
<thead>
<tr>
<th>IPv4</th>
<th>rrc00</th>
<th>rrc03</th>
<th>rrc05</th>
<th>rrc06</th>
<th>rrc07</th>
<th>rrc10</th>
<th>rrc11</th>
<th>rrc12</th>
<th>rrc15</th>
<th>rrc16</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPv6</td>
<td>rrc00_6</td>
<td>rrc03_6</td>
<td>rrc05_6</td>
<td>rrc06_6</td>
<td>rrc07_6</td>
<td>rrc10_6</td>
<td>rrc11_6</td>
<td>rrc12_6</td>
<td>rrc15_6</td>
<td>rrc16_6</td>
</tr>
<tr>
<td># prefixes</td>
<td>332118</td>
<td>321617</td>
<td>322997</td>
<td>321577</td>
<td>322557</td>
<td>319952</td>
<td>323668</td>
<td>320016</td>
<td>323987</td>
<td>328296</td>
</tr>
</tbody>
</table>

![Graph showing number of prefixes vs prefix length]

- The graph illustrates the distribution of prefixes across different prefix lengths for various routing tables.
Memory Requirement

- For various BS and IPS ($P_{LL}=P_{tree}=1$)
- For various $P_{tree}$ and $P_{LL}$ (BS=4, IPS=13)

<table>
<thead>
<tr>
<th></th>
<th>rrc00_6</th>
<th>rrc03_6</th>
<th>rrc05_6</th>
<th>rrc06_6</th>
<th>rrc07_6</th>
<th>rrc10_6</th>
<th>rrc11_6</th>
<th>rrc12_6</th>
<th>rrc15_6</th>
<th>rrc16_6</th>
</tr>
</thead>
<tbody>
<tr>
<td>BT</td>
<td>341</td>
<td>330</td>
<td>332</td>
<td>331</td>
<td>332</td>
<td>329</td>
<td>347</td>
<td>329</td>
<td>334</td>
<td>337</td>
</tr>
<tr>
<td>BTPC</td>
<td>72</td>
<td>69</td>
<td>70</td>
<td>69</td>
<td>70</td>
<td>69</td>
<td>72</td>
<td>70</td>
<td>70</td>
<td>71</td>
</tr>
<tr>
<td>DBPC</td>
<td>45</td>
<td>45</td>
<td>45</td>
<td>45</td>
<td>46</td>
<td>45</td>
<td>48</td>
<td>45</td>
<td>45</td>
<td>46</td>
</tr>
<tr>
<td>BST</td>
<td>35.5</td>
<td>34.4</td>
<td>34.6</td>
<td>34.4</td>
<td>34.5</td>
<td>34.6</td>
<td>34.6</td>
<td>34.7</td>
<td>35.2</td>
<td></td>
</tr>
<tr>
<td>CTF</td>
<td>27.9</td>
<td>27.3</td>
<td>27.0</td>
<td>29.2</td>
<td>27.2</td>
<td>27.1</td>
<td>27.1</td>
<td>26.9</td>
<td>28.0</td>
<td>27.2</td>
</tr>
<tr>
<td>CLLF</td>
<td>16.4</td>
<td>15.8</td>
<td>15.9</td>
<td>15.9</td>
<td>16.0</td>
<td>15.7</td>
<td>16.0</td>
<td>15.7</td>
<td>15.9</td>
<td>16.1</td>
</tr>
</tbody>
</table>
Delay

- The delay for various $P_{tree}$ and $P_{LL}$ ($BS=4$, $IPS=13$)
- The % of memory by BST and LL for various $IPS$ ($BS=4$, $P_{LL}=5$, $P_{tree}=1$)

<table>
<thead>
<tr>
<th>rrc00_6</th>
<th>rrc03_6</th>
<th>rrc05_6</th>
<th>rrc06_6</th>
<th>rrc07_6</th>
<th>rrc10_6</th>
<th>rrc11_6</th>
<th>rrc12_6</th>
<th>rrc15_6</th>
<th>rrc16_6</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLLF</td>
<td>45</td>
<td>44</td>
<td>45</td>
<td>44</td>
<td>45</td>
<td>45</td>
<td>44</td>
<td>45</td>
<td>46</td>
</tr>
</tbody>
</table>

- Depth (cycles) ($BS=4$, $IPS=13$, $P_{LL}=5$, $P_{tree}=1$)
Performance Comparison

FPGA implementation
- Implemented in Verilog using Xilinx ISE 12.4 and target device Xilinx XC6VSX475T
- 216 MHz (432 million packets per second with dual-ported BRAMs)
- Clock period is 4.63 ns, 138 Gbps for the minimum packet size of 40 bytes

Memory efficiency (in Bits/prefix)
Throughput efficiency (in Gbps/Bits)

<table>
<thead>
<tr>
<th>Architecture</th>
<th># prefix</th>
<th>Mem. Eff. Bits/prefix</th>
<th>Throughput Gbps</th>
<th>Throughput eff. Gbps/B</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLLF</td>
<td>324</td>
<td>51.8</td>
<td>138</td>
<td>2.66</td>
</tr>
<tr>
<td>CTF</td>
<td>324</td>
<td>88.1</td>
<td>135</td>
<td>1.53</td>
</tr>
<tr>
<td>BST</td>
<td>324</td>
<td>124</td>
<td>125</td>
<td>1.01</td>
</tr>
<tr>
<td>DBPC</td>
<td>324</td>
<td>146</td>
<td>120</td>
<td>0.82</td>
</tr>
<tr>
<td>2-3 tree</td>
<td>324</td>
<td>167</td>
<td>120</td>
<td>0.72</td>
</tr>
<tr>
<td>Flashtrie</td>
<td>310</td>
<td>171</td>
<td>64</td>
<td>0.37</td>
</tr>
<tr>
<td>TreeBitmap</td>
<td>310</td>
<td>405</td>
<td>6</td>
<td>0.02</td>
</tr>
</tbody>
</table>
Agenda

• Background
• Prior Work
• Our Solutions
• Implementation Results
• Conclusions & Future Work
Conclusions and Future Work

- **Clustered Linked List Forest**
  - Significant memory saving
  - Performance improvements with optimizations

- **Linear pipelined SRAM-based architecture on FPGAs**
  - Supports up to 712K IPv6 prefixes
  - Sustained throughput of 138 Gbps

- **Future work**
  - Extend the algorithm to virtual routers to improve their memory efficiency
THANK YOU!