A *Concise* Forwarding Information Base for Scalable and Fast Name Switching

**Chen Qian**

University of Kentucky

University of California Santa Cruz

with Ye Yu, Djamal Belazzougui, Qin Zhang
Forwarding Information Base (FIB)

A data structure (typically a table) in a network device to determine forwarding actions.

Input: destination

Output: FWD action

To dest
Names vs. IP addresses

- (most) Names are flat, permanent, and location-independent
- Flexible network services for mobile devices and VMs: routing, firewall, VPN, etc.
- Flow IDs (Packet headers) can also be considered as names.
- Biggest problem: FIB explosion
Examples of network names

Enterprise and data center networks:

- SEATTLE [SIGCOMM’08], VIRO [Infocom’11], ROME [ICNP’12]

Future Internet Architecture (FIA)

- Layered Naming Architecture [SIGCOMM’04], AIP [SIGCOMM’08]
- NSF FIA projects: NDN, XIA, MobilityFirst

LTE access network [SIGCOMM’15]
New FIB design: **Concise**

1. Use the least memory ever
   - Fast memory is expensive and power-hungry.
   - Only **10% - 30%** of Cuckoo [CoNEXT’13, SIGCOMM’15]

2. Fast query speed ever
   - **2x to 5x** advantage

3. Update speed slower than some FIBs
   - Still support **millions** of updates per second.
Idea of **Concise**

SDN Controller

- Update via existing SDN API
- Construct
- Update
- Query

Optimize memory and query cost
Concise functions

- Classify $n$ names into $d$ different sets.
- Each set is a forwarding action

- Relying on a data structure Othello
A new data structure Othello

Classifies names to two sets $X$ and $Y$
- Based on MWHC perfect hashing, which is static

Query result
- $\tau(k) = 0 \iff k \in X$
- $\tau(k) = 1 \iff k \in Y$
Othello Query Structure

Two bitmaps $a, b$ with size $m$ ($m$ in $(1.42n, 2.86n)$)

Query is easy. Then how to construct it?
Othello Control Structure: Construct

$G$: acyclic bipartite graph

$h_a$ and $h_b$: functions that map nodes to states.

$k$, $h_a(k)$, $h_b(k)$:

<table>
<thead>
<tr>
<th>$k$</th>
<th>$h_a(k)$</th>
<th>$h_b(k)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td></td>
<td>5</td>
</tr>
</tbody>
</table>
If finding a cycle, use another pair $<h_a, h_b>$ until an acyclic graph is built.

For $n$ names, the time to find $G$ is $O(n)$. 
Compute Bitmap

\[ u_0, u_1, u_2, u_3, u_4, u_5, u_6, u_7 \]

\[ v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7 \]

<table>
<thead>
<tr>
<th>( k )</th>
<th>( h_a(k) )</th>
<th>( h_b(k) )</th>
<th>set</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5</td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
If $G$ is acyclic, easy to find a coloring plan
Name Addition – color flip

If $G$ is acyclic, flipping is trivial
Concise functionality

Classifies names into $2^l$ sets:
$Z_0, Z_1, \ldots, Z_{2^l-1}$

$l$ Othellos can classify names to $2^l$ sets

$l < 8$ for network devices
Same $G, h_a, h_b$. Different coloring plan and bitmaps.

Do we need $2l$ memory reads to query $l$ Othellos?

Othello 1  Same $X\ Y$  Othello 2
\( h_a^{A[0]} A[1] \)

\( \tau(k) = 01 \oplus 10 = (11)_2 \)

\( k \) is in set \( Z_3 \)

CPUs can read \( l \) bits at one time
Implementation of three prototypes

1. Memory mode
   - Query and control structures running on different threads.

2. CLICK modular router

3. Intel Data Plane Development Kit (DPDK)
Comparison: best solutions in the literature

Buffalo
in CoNEXT’09

Cuckoo hashing
in CoNEXT’13 and SIGCOMM’15
## Comparison: Memory size

<table>
<thead>
<tr>
<th>FIB Example</th>
<th>Memory Size</th>
<th>Name Type</th>
<th># Names</th>
<th># Actions</th>
<th>Concise</th>
<th>Cuckoo</th>
<th>Buffalo</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC (48 bits)</td>
<td></td>
<td></td>
<td>7*10^5</td>
<td>16</td>
<td>1M</td>
<td>5.62M</td>
<td>2.64M</td>
</tr>
<tr>
<td>MAC (48 bits)</td>
<td></td>
<td></td>
<td>5*10^6</td>
<td>256</td>
<td>16M</td>
<td>40.15M</td>
<td>27.70M</td>
</tr>
<tr>
<td>MAC (48 bits)</td>
<td></td>
<td></td>
<td>3*10^7</td>
<td>256</td>
<td>128M</td>
<td>321.23M</td>
<td>166.23M</td>
</tr>
<tr>
<td>IPv4 (32 bits)</td>
<td></td>
<td></td>
<td>1*10^6</td>
<td>16</td>
<td>2M</td>
<td>4.27M</td>
<td>3.77M</td>
</tr>
<tr>
<td>IPv6 (128 bits)</td>
<td></td>
<td></td>
<td>2*10^6</td>
<td>256</td>
<td>8M</td>
<td>34.13M</td>
<td>11.08M</td>
</tr>
<tr>
<td>OpenFlow (356 bits)</td>
<td></td>
<td></td>
<td>3*10^5</td>
<td>256</td>
<td>1M</td>
<td>14.46M</td>
<td>1.67M</td>
</tr>
<tr>
<td>OpenFlow (356 bits)</td>
<td></td>
<td></td>
<td>1.4*10^6</td>
<td>65536</td>
<td>8M</td>
<td>67.46M</td>
<td>18.21M</td>
</tr>
<tr>
<td>File name (varied)</td>
<td></td>
<td></td>
<td>359194</td>
<td>16</td>
<td>512K</td>
<td>19.32M</td>
<td>1.35M</td>
</tr>
</tbody>
</table>
Query speed

2x to 4x speed advantage
Update

Each update is a network-wide update
More possible applications of Concise

- Essentially a key-value mapping
- 1. Memory cache
- 2. Support query to distributed content storage
- 3. Sparse vector data processing
Thank You

Questions?