Wildcard Compression of Inter-Domain Routing Tables for OpenFlow-Based Software-Defined Networking

Wolfgang Braun, Michael Menth

kn.inf.uni-tuebingen.de
SDN-based Inter-Domain Routing

Peering domains with IP routers

Performs BGP & communicates with peers

IP routers are replaced with simple OpenFlow switches

BGP and table computation is performed in the control plane

Only forwards BGP messages

Flow tables are installed using OpenFlow

Flow table size is limited
Limiting Flow Table Sizes

Routing tables consist of over 500K entries
- Even forwarding tables are generally too big for TCAM found in OpenFlow switches
- Reactive approaches install only the important entries
  1. Forwarding ‘unknown packets’ to the control plane
     → Additional load on control plane
  2. Replacing old flow entries with required new flow entries
     → Flow setup delay; buffering necessary
- Software-defined Internet Exchange (SDX)
  - Virtual next-hops and MAC addresses can improve aggregation

Wildcard compression
- Reduces the effective size of the table
- Improves and is orthogonal to approaches discussed above
- Potential of wildcard compression on routing tables unknown
### Wildcard Compression

<table>
<thead>
<tr>
<th>FIB entry</th>
<th>Next-hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>***</td>
<td>1</td>
</tr>
<tr>
<td>101</td>
<td>1</td>
</tr>
<tr>
<td>1**</td>
<td>2</td>
</tr>
<tr>
<td>01*</td>
<td>3</td>
</tr>
<tr>
<td>11*</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FIB entry</th>
<th>Next-hop</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>***</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>101</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1**</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>01*</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>11*</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

- A prefix has only ending *
  - Longest prefix match (LPM)
- A wildcard (match) can have * at any position
  - Priorities replace LPM
- Prefix length is an appropriate priority

- Compression of 01* and 11* results in *1*
- Compression of *** and 101 results in ***
  - Addresses starting with 101 would conflict with 1**
- Aggregate only equal priorities
Logic minimization (LM) can perform wildcard compression

- Prefixes and wildcards are interpreted as logical terms
- Number of terms is reduced
  ➔ Fewer wildcards

LM is very computation-intensive

- Espresso logic minimization heuristic
  - Used in very-large-scale integration (VLSI)
- Long runtimes for routing tables compression
  - Increases significantly with table size but fast for small input

Solution: threshold parameter $T$

- Split a prefix set into smaller sets $S_i$ with $|S_i| < T$
  - Aggregate and combine smaller sets
- Logical equivalence ensures correctness
- $T$ sacrifices compression efficiency for runtime
Evaluation Setup

- Optimal Routing Table Constructor (ORTC)
  - Prefix-based table compression
  - Results in minimum-sized tables

- Wildcard compression using Espresso
  - Compression efficiency compared to ORTC
  - Impact of threshold parameter
    - Compression ratio
    - Compression time
Evaluation Setup

► Routing tables from the Route Views project
  ▪ Evaluated on various routing tables from different years
    – Only present results for a RIB from 2013
    – RIB consists of 500,495 entries

► Converting the RIB into a FIB
  ▪ We specify a maximum number of next-hops \( n \)
  ▪ We assign a random next-hop to each RIB entry
Compression: Espresso vs. ORTC

- Similar results for 1 next-hop for ORTC and Espresso (~ 50K entries)
- For 2 and more next-hops, Espresso compresses significantly better
- Espresso has 180 – 310K flow entries; ORTC 220 – 345K respectively
- Approximately 40K FIB entry reduction for $n = 2$
Impact of the Threshold Parameter

Lower thresholds result in less compression
- Reduction by approximately 1.4 – 2%
Hardware & Setup
- Intel CPU i5-2500K (4 cores), 3.3 GHz, 6M Cache
- 8GB DDR3-RAM
- Single threaded C++ program

High or no threshold can lead to 341 – 436s compression time
T = 100 or T = 250 results in less than 1 – 2s
We have applied wildcard compression to routing tables

- Espresso can further compress optimal prefix-based tables
  - Up to 40K entries; ~ 17% smaller tables
- Found a way to enable the compression of routing tables
  - Logic minimization is very computation-intensive
  - Threshold parameter sacrifices compression efficiency for runtime

Wildcard compression does not solve the BGP state problem

- Problem is only slightly mitigated
- However, other approaches benefit from this method
  - Reactive flow management
  - SDX approach
BACKUP SLIDES