Dynamic Forwarding Table Aggregation without Update Churn: The Case of Dependent Prefixes

Marcin Bienkowski (Uni Wroclaw)
Nadi Sarrar (TU Berlin)
Stefan Schmid (TU Berlin & T-Labs)
Steve Uhlig (Queen Mary, London)
Wow! Growth of Forwarding Tables

**Why?** Scale, virtualization, …

**Problem:**  - TCAM expensive and power-hungry!
  - IPv6 may not help!
Local FIB Compression: 1-Page Overview

Model
- FIB: Forwarding Information Base
- FIB consists of
  - set of <prefix, next-hop>
  - IP only: most specific IP prefix
- Control: (1) RIB or (2) SDN Controller (s. picture)

Basic Idea
- Dynamically aggregate FIB
  - “Adjacent” prefixes with same next-hop (= color): one rule only!
- But be aware that BGP updates (next-hop change, insert, delete) may change forwarding set, need to de-aggregate again

Benefits
- Only single router affected
- Aggregation = simple software update

Stefan Schmid (T-Labs)
Local FIB Compression: 1-Page Overview

Model
- FIB: Forwarding Information Base
  - FIB consists of
    - set of <prefix, next-hop>
    - IP only: most specific IP prefix
  - Control: (1) RIB or (2) SDN Controller (s. picture)

Basic Idea
- Dynamically aggregate FIB
  - “Adjacent” prefixes with same next-hop (= color): one rule only!
- But be aware that BGP updates (next-hop change, insert, delete) may change forwarding set, need to de-aggregate again

Benefits
- Only single router affected
- Aggregation = simple software update
Goal: keep FIB small but consistent!
Without sending too many additional updates.
Setting: A Memory-Efficient Switch/Router

Goal: keep FIB small but consistent!
Without sending too many additional updates.
Setting: A Memory-Efficient Switch/Router

Goal: keep FIB small but consistent!
Without sending too many additional updates.

Update Churn?
Data structure, networking, …
Motivation: FIB Compression and Update Churn

Benefits of FIB aggregation
- Routeviews snapshots indicate 40% memory gains
- More than under uniform distribution
- But depends on number of next hops

Churn
- Thousands of routing updates per second
- Goal: do not increase more (or improve!)
Online algorithms make decisions at time $t$ without any knowledge of inputs at times $t'>t$.

**Competitive Analysis**

An $r$-competitive online algorithm $ALG$ gives a worst-case performance guarantee: the performance is at most a factor $r$ worse than an optimal offline algorithm $OPT$!

**Competitive Ratio**

Competitive ratio $r$, 

$$r = \frac{\text{Cost}(ALG)}{\text{cost}(OPT)}$$

The price of not knowing the future!

No need for complex predictions but still good!
Model: Online Input Sequence

Route processor
(RIB or SDN controller)

full list of forwarded prefixes: (prefix, port)

BGP updates

Update: Color change

Update: Insert/Delete
Cost = \alpha \left( \# \text{ updates to FIB} \right) + \int_t^{memory}
Uncompressed FIB (UFIB): independent prefixes

size 5

Theorem:
BLOCK(A,B) is 3.603-competitive.

Theorem:
Any online algorithm is at least 1.636-competitive.
(Even ALG can use exceptions and OPT not.)
Model 1: Aggregation without Exceptions (SIROCCO 2013)

**BLOCK(A,B) operates on trie:**

- Two parameters A and B for amortization ($A \geq B$)
- Definition: internal node $v$ is $c$-mergeable if subtree $T(v)$ only contains color $c$ leaves
- Trie node $v$ monitors: how long was subtree $T(v)$ $c$-mergeable without interruption? Counter $C(v)$.
- If $C(v) \geq A \alpha$, then aggregate entire tree $T(u)$ where $u$ is furthest ancestor of $v$ with $C(u) \geq B \alpha$. (Maybe $v$ is $u$.)
- Split lazily: only when forced.

Nodes with square inside: mergeable. Nodes with bold border: suppressed for FIB1.
BLOCK(A,B) operates on trie:

- Two parameters A and B for amortization (A ≥ B)
- Definition: internal node v is c-mergeable if subtree $T(v)$ only contains color c leaves
- Trie node v monitors: how long was subtree $T(v)$ c-mergeable without interruption? Counter $C(v)$.

$\text{BLOCK}(\alpha)$:

1. balances memory and update costs
2. exploits possibility to merge multiple tree nodes simultaneously at lower price (threshold A and B)

Nodes with square inside: mergeable. Nodes with bold border: suppressed for FIB1.
Uncompressed FIB (UFIB): dependent prefixes

size 5

Theorem:
HIMS is $O(w)$-competitive, $w = \text{address length}$.

Theorem:
Asymptotically optimal for general class of online algorithms.
**Sticks**

Maximal subtrees of UFIB with colored leaves and blank internal nodes.

Idea: if all leaves in Stick have same color, they would become mergeable.
The HIMS Algorithm

- Hide Invisibles Merge Siblings (HIMS)
- Two counters in Sticks:

\[
C(u) = \text{time since Stick descendants are unicolor}
\]

\[
H(u) = \text{how long do nodes have same color as the least colored ancestor?}
\]

Note: \( C(u) \geq H(u) \), \( C(u) \geq C(p(u)) \), \( H(u) \geq H(p(u)) \), where \( p() \) is parent.
The HIMS Algorithm

Keep rule in FIB if and only if all three conditions hold:

1. $H(u) < \alpha$ (do not hide yet)
2. $C(u) \geq \alpha$ or $u$ is a stick leaf (do not aggregate yet if ancestor low)
3. $C(p(u)) < \alpha$ or $u$ is a stick root

Examples:

Ex 1. Trivial stick: node is both root and leaf (Conditions 2+3 fulfilled). So HIMS simply waits until invisible node can be hidden.

Ex 2. Stick without colored ancestors: $H(u)=0$ all the time (Condition 1 fulfilled). So everything depends on counters inside stick. If counters large, only root stays.
Theorem:
HIMS is $O(w)$-competitive.

Proof idea:
- In the absence of further BGP updates
  1. HIMS does not introduce any changes \textit{after time }$\alpha$
  2. After time $\alpha$, the memory cost is at most an factor $O(w)$ off
- In general: for any snapshot at time $t$, either HIMS already started aggregating or changes are quite new
- Concept of rainbow points and line coloring useful

- A rainbow point is a “witness” for a FIB rule
- Many different rainbow points over time give lower bound
Lower Bound

Theorem:
Any (online or offline) Stick-based algo is $\Omega(w)$-competitive.

Proof idea:
Stick-based:
1. never keep a node outside a stick
2. inside a stick, for any pair u,v in ancestor-descendant relation, only keep one

Consider single stick: prefixes representing lengths $2^{w-1}, 2^{w-2}, ..., 2^1, 2^0, 2^0$

Cannot aggregate stick!
But OPT could do that:

QED
LFA: A Simplified Implementation

- LFA: Locality-aware FIB aggregation

- Combines stick aggregation with offline optimal ORTC
  - Parameter $\alpha$: depth where aggregation starts
  - Parameter $\beta$: time until aggregation
For small alpha, Aggregated Table (AT) significantly smaller than Original Table (OT)
Conclusion

- Without exceptions in input and output: BLOCK is constant competitive
- With exceptions in input and output: HIMS is $O(w)$-competitive
- Note on offline variant: fixed parameter tractable, runtime of dynamic program in $f(\alpha) \cdot n^{O(1)}$

Thank you! Questions?