Dynamic FIB Aggregation without Update Churn
Dynamic FIB Aggregation without Update Churn

Marcin Bienkowski
Nadi Sarrar
Stefan Schmid
Steve Uhlig
Poor Routers!
Key Functionality: Forwarding
Poor Routers!
Key Functionality: Forwarding

Forwarding Information Base (FIB)

00* to

01* to

1* to

Ports
Poor Routers!

Key Functionality: Forwarding

Forwarding Information Base (FIB)

00* to
01* to
1* to

most specific prefix fast?

TCAM memory expensive and power-hungry...

Ports
Poor Routers!

Key Functionality: Forwarding

Forwarding Information Base (FIB)

00* to
01* to
1* to

most specific prefix fast?

TCAM memory expensive and power-hungry...

... and requirements grow quickly (e.g., virtualization). IPv6 does not help.
Poor Routers!
Key Functionality: Forwarding

Forwarding Information Base (FIB)

00* to
01* to
1* to

most specific prefix fast?

TCAM memory expensive and power-hungry...

Ports

... and requirements grow quickly (e.g., virtualization). IPv6 does not help.

Idea to prolong the router lifetime: Compress the FIB!
Idea: Compress the FIB

- **00*** to
- **01*** to
- **1*** to

represent as trie...
Idea: Compress the FIB

represent as trie...

... and compress it!
Idea: Compress the FIB

represent as trie...

Good Potential:

Down to 40% (RouteView), depending on # ports.

... and compress it!
Idea: Compress the FIB

00* to
01* to
1* to

represent as trie...

But: May introduce churn!
Deaggregate upon route change.
Idea: Compress the FIB

Represent as trie...

But: May introduce churn!
Deaggregate upon route change.

Update cost 2: remove + add subtree

Update cost 3: remove + add subtree
Idea: Compress the FIB

But: May introduce churn!
Deaggregate upon route change.

Already without churn: 1000s/sec updates in BGP!
An optimization problem:

1. Forwarding must always be correct (equivalent)
2. Minimize update cost and memory size
An online problem:
1. Forwarding must always be correct (equivalent)
2. Minimize update cost and memory size
An online problem:
1. Forwarding must always be correct (equivalent)
2. Minimize update cost and memory size
An online problem:

1. Forwarding must always be correct (equivalent)
2. Minimize update cost and memory size

Joint Optimization:

$$\text{Cost} = \alpha \cdot \text{(\# updates)} + \int_0^t \text{FIB size}$$
An Optimization Problem

Joint Optimization:

Cost = \alpha \ (\# \text{ updates}) + \int_t FIB \text{ size}

Online Input

Update Cost

Memory Cost

Realm of Online Algorithms and Competitive Analysis!

Competitive Ratio = \max \frac{\text{Cost(ON)}}{\text{Cost(OFF)}}

1. Forwarding must always be correct (equivalent)

2. Minimize update cost and memory size
What Was Known So Far?

Static Compression

E.g., ORTC (INFOCOM’99): dynamic programming algorithm to compute optimally compressed FIB

Dynamic Compression

No free lunch! (But can also reduce churn!)
Lots of work over the last 15 years (since Lulea) to find better tradeoffs between memory and updates. Often implicitly though. Heuristics: SMALTA (CoNEXT’11), FIFA (INFOCOM’13)

Online algorithm BLOCK for «independent prefixes»
a.k.a. without exceptions (SIROCCO 2013)

size 5

size 3

BLOCK is 3.603-competitive.

Any online algorithm is at least 1.636-competitive.
(Even if ON can use exceptions and OFF not.)
Idea of BLOCK

BLOCK(A,B) operates on trie:
1. balances memory and update costs
2. exploits possibility to merge multiple tree nodes simultaneously at lower price (thresholds A and B)

- Timers/counters for each trie node
- Wait before aggregating to save update costs
- Thresholds A and B for amortization (A ≥ B) of update costs
- 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) ≥ A α, then aggregate entire tree T(u) where u is furthest ancestor of v with C(u) ≥ B α.
- Split lazily: only when forced.

Nodes with square inside: mergeable.
Our Contribution:
Competitive Compression for Dependent Prefixes
Our Contribution:
Competitive Compression for Dependent Prefixes

UFIB: Uncompressed trie with exceptions
FIB: Compressed equivalent trie

Always stored in controller!
Our Contribution: Competitive Compression for Dependent Prefixes

Invisible Node

UFIB: Uncompressed trie with exceptions

FIB: Compressed equivalent trie

Always stored in controller!
Our Contribution:
Competitive Compression for Dependent Prefixes

UFIB: Uncompressed trie with exceptions
FIB: Compressed equivalent trie

Always stored in controller!
Our Contribution:

Competitive Compression for Dependent Prefixes

Invisible Node

Sibling Nodes

Theorem 1: HIMS

HIMS («Hide Invisible Merge Siblings») is O(w)-competitive, where w is prefix length. HIMS is deterministic.
Our Contribution:
Competitive Compression for Dependent Prefixes

Invisible Node

Sibling Nodes

Theorem 1: HIMS
HIMS («Hide Invisible Merge Siblings») is O(w)-competitive, where w is prefix length. HIMS is deterministic.

Theorem 2: Lower Bound
This is optimal for a large class of online and offline algorithms.
Concept of Sticks: (on UFIB!)
Maximal subtrees of UFIB with colored leaves and blank internal nodes.

Idea: if all leaves in stick have same color, they would become mergeable.
Ideas of HIMS.

Two counters at nodes:

Merge Sibling Counter $C(u)$

$C(u) = \text{time since stick descendants are unicolor}$

Hide Invisible Counter $H(u)$

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

Concept of Sticks: (on UFIB!)

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

Idea: if all leaves in stick have same color, they would become mergeable.
Ideas of HIMS.

Two counters at nodes:

Merge Sibling Counter $C(u)$

$C(u) = \text{time since stick descendants are unicolor}$

Hide Invisible Counter $H(u)$

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

Concept of Sticks: (on UFIB!)

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

Idea: if all leaves in stick have same color, they would become mergeable.

Monotonic properties: E.g., color change of stick leaf resets all counters to root. So $C(u) \geq C(p(u))$ and $H(u) \geq H(p(u))$, and also $C(u) \geq H(u)$, as I need ancestor. Where $p()$ is parent in trie.
Ideas of HIMS.

Two counters at nodes:

- **Merge Sibling Counter** $C(u)$
  - $C(u) = \text{time since stick descendants are unicolor}$
  - Diagram: A tree with a counter symbol and text explaining the concept.

- **Hide Invisible Counter** $H(u)$
  - $H(u) = \text{how long do nodes have same color as the least colored ancestor in UFIB?}$
  - Diagram: A tree with a counter symbol and text explaining the concept.

Concept of Sticks: (on UFIB!)

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

**U-FIB:**

- Idea: if all leaves in stick have same color, they would become mergeable.

**Monotonic properties:** E.g., color change of stick leaf resets all counters to root. So $C(u) \geq C(p(u))$ and $H(u) \geq H(p(u))$, and also $C(u) \geq H(u)$, as I need ancestor. Where $p()$ is parent in trie.

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

1. $H(u) < \alpha$ (remove if hidden for long)
2. $C(u) \geq \alpha$ or $u$ is a stick leaf (always true for UFIB rule)
3. $C(p(u)) < \alpha$ or $u$ is a stick root (remove if amortized parent)
Ideas of HIMS.

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

Idea: if all leaves in stick have same color, they would become mergeable.

Two counters at nodes:
- Merge Sibling Counter $C(u)$
- Hide Invisible Counter $H(u)$

Properties:
- $C(u) \geq H(u)$
- $C(u) \geq C(p(u))$
- $H(u) \geq H(p(u))$, where $p()$ is parent in trie.

HIMS Algo:
- Keep rule in FIB if and only if all three conditions hold:
  1. $H(u) < \alpha$ (remove if hidden for long)
  2. $C(u) \geq \alpha$ or $u$ is a stick leaf (always true for UFIB rule)
  3. $C(p(u)) < \alpha$ or $u$ is a stick root (remove if amortized parent)

Example.

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 sibling counters inside stick. E.g., if no change for time $\alpha$ and unicolor leaves, only root stays (all three conditions fulfilled).

Ex 3. Inner nodes of the stick are never longer than $\alpha$ in the FIB. (Otherwise second condition violated.)

$H(u)$ = how long do nodes have same color as the least colored ancestor in UFIB?

(1) $H(u) < \alpha$ (remove if hidden for long)
(2) $C(u) \geq \alpha$ or $u$ is a stick leaf (always true for UFIB rule)
(3) $C(p(u)) < \alpha$ or $u$ is a stick root (remove if amortized parent)
Theorem:

HIMS is $O(w)$-competitive.

Proof idea:
- In the absence of further BGP updates
  1. HIMS does not introduce any changes 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
- Lower bound for OFF: 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.
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 $2^{w-1}, 2^{w-2}, \ldots, 2^1, 2^0, 2^0$

Cannot aggregate stick!
But OPT could do that:

QED
Simulations: The Simplified Version LFA

- 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 $f(\alpha) n^{O(1)}$

Thank you! Questions?
Why Aggregation in Worst Case?

With fixed switch size and sum over time?

- Start to make errors late!
- Or offload rest...
Analysis

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

**Proof idea (a bit technical):**
- Time events when ALG merges k nodes of T(u) at u
- **Upper bound ON cost:**
  - k+1 counters between B $\alpha$ and A $\alpha$
  - Merging cost at most (k+3) $\alpha$: remove k+2 leaves, insert one root
  - Splitting cost at most (k+1) 3$\alpha$: in worst case, remove-insert-remove individually
- **Lower bound OFF cost:**
  - Time period from t- $\alpha$ to t
  - If OPT does not merge anything in T(u) or higher: high memory costs
  - If OPT merges ancestor of u: counter there must be smaller than B $\alpha$, memory and update costs
  - If OPT merges subtree of T(u): update cost and memory cost for in- and out-subtree
- Optimal choice: $A = \sqrt{13} - 1$, $B = (2\sqrt{13})/3 - 2/3$
- Add event costs (inserts/deletes) later!

QED
Lower Bound

Theorem:
Any online algorithm is at least 1.636-competitive.

Proof idea:

- Simple example:

```plaintext
(1) If ALG does never changes to single entry, competitive ratio is at least 2 (size 2 vs 1).
(2) If ALG changes before time $\alpha$, adversary immediately forces split back! Yields costly inserts...
(3) If ALG changes after time $\alpha$, the adversary resets color as soon as ALG for the first time has a single node. Waiting costs too high.
```
Note on Adding Insertions and Deletions

- Algorithm can be extended to insertions/deletions

**Insert:**

- **u** becomes mergeable!

**Delete:**

- **u** no longer mergeable!