FIB Aggregation at Scale: A Longitudinal Comparison of Research and Industry Algorithms

William Sun
Newark Academy
wsun29@newarka.edu

Peyton Mo
Ridgewood HS
peytonmo001@gmail.com

Timothy Liu
Bergen County Academies
liu.tim21@gmail.com

Tyler Wang
Livingston Heritage MS
tylerttt88@gmail.com

Yaoqing Liu
FDU
yliu@fdu.edu

***Abstract*—The global Internet routing table passed one million entries in 2025, making the cost of storing and updating forwarding state in Ternary Content-Addressable Memory (TCAM) a first-order constraint for network operators. Networking research has produced near-optimal, incrementally updatable Forwarding Information Base (FIB) aggregation algorithms, while vendors ship compression tuned for tightly packed hardware tables, yet the two are rarely compared. This paper presents a longitudinal, head-to-head comparison of a representative research algorithm, FIB Aggregation with Quick Selections (FAQS), and a Juniper-inspired industry algorithm. Replaying fifteen years (2010–2025) of BGP updates from AS 3356 through both algorithms for IPv4 and IPv6, we measure aggregated FIB size, per-update latency, and FIB churn (bursts). The algorithms reach similar compression but diverge sharply in update behavior. FAQS holds a near-flat, lower per-update latency (about 3.7–3.8 μs for IPv4; 2.3–3.6 μs for IPv6), while the Juniper-inspired algorithm is slower and grows faster (about 4.3–4.6 μs for IPv4; 3.8–4.8 μs for IPv6). FAQS also produces smaller, rarer bursts (2025 IPv4 maximum of 2,380 versus 4,437 entries). Compression parity thus hides a structural efficiency gap: by preserving locality and exiting early, the research algorithm delivers the same table size with more predictable control-plane cost, especially as the IPv6 table scales. These results show that research advances can reduce update overhead without sacrificing the compact forwarding state commercial hardware requires.**

***Index Terms*—FIB aggregation, forwarding information base, BGP, longest prefix match, TCAM, routing scalability, IPv6, incremental update, FAQS**

I. Introduction

By 2025 the global Internet routing table exceeded one million entries under the Border Gateway Protocol (BGP) [1]. In 2010 it held roughly three hundred thousand routes; the table has since grown more than threefold. Several forces drive this growth. Operators de-aggregate address blocks into more specific prefixes for traffic engineering; multihoming multiplies the number of advertised paths; and policy changes inject bursts of new state [2]. The practical consequence is pressure on fast forwarding memory. TCAM, which performs the parallel longest-prefix-match (LPM) lookups that line-rate forwarding depends on, is power-hungry, expensive, and physically bounded [3]. When the table outgrows the TCAM budget, a router must either be upgraded or begin to degrade, so keeping the installed FIB small is directly tied to hardware cost and forwarding stability.

FIB aggregation attacks this problem by merging routes that share forwarding behavior into fewer, more general prefixes while preserving the forwarding decision for every destination. Two communities have pursued it from different angles. The networking *research* community has produced a progression of algorithms, including ORTC, Level1-4, SMALTA, FIFA, HiMs, LFA, and FAQS, which target provable or near-optimal compression and, increasingly, cheap incremental updates [4]–[7]. Equipment vendors, meanwhile, ship compression whose first priority is a compact, tightly packed hardware table that fits commercial line cards [8], [9]. The two lines optimize for different things, theoretical update efficiency on one side and packed hardware footprint on the other, and they are seldom evaluated against each other on the same data.

Prior evaluations also tend to be short. Most measure a single algorithm on a fixed snapshot or a few months of updates, which cannot show how an algorithm behaves as the table grows by an order of magnitude or how a research design and a deployed design diverge over time. To the best of our knowledge, this paper closes that gap with the first longitudinal, multi-protocol, head-to-head comparison of a research algorithm and an industry-style algorithm on fifteen years of real BGP traces. We re-implement FAQS and a Juniper-inspired compression algorithm in a single common framework, replay historical one-month BGP updates from AS3356 (Lumen Technologies) for both IPv4 and IPv6 from 2010 to 2025, and verify forwarding equivalence after every replay with VeriTables [10]. Note that because the production Junos implementation is proprietary, we implement an industry-style compression algorithm based on publicly documented Juniper FIB compression principles. We therefore refer to it as the Juniper-inspired algorithm rather than a direct implementation of Junos. For each algorithm and year we report:

- Final aggregated FIB size (compression effectiveness);
- Average per-update processing latency; and
- FIB burst size, the number of installed entries changed per update.

Our central finding is that compression parity conceals a large efficiency gap. Both algorithms compress the table to nearly the same size, but the research algorithm processes updates faster, with a much flatter growth curve and far smaller bursts, because it confines work to the affected subtree and stops as soon as the aggregation state stops changing. The industry-style algorithm, by recompressing the full ancestor path on every change to keep the table packed, pays a per-

TABLE I. FIB AGGREGATION EXAMPLE. CONTIGUOUS PREFIXES SHARING A NEXT HOP AND PATH COLLAPSE INTO A SHORTER PREFIX; OTHERS ARE LEFT UNCHANGED

Before aggregation			
Prefix	Next hop	AS path	Len
203.0.112.0/24	192.0.2.1	3356	2
203.0.113.0/24	192.0.2.1	3356	2
203.0.114.0/24	192.0.2.2	3356	2
203.0.115.0/24	192.0.2.2	3356	2
198.51.100.0/24	192.0.2.3	3356	3
After aggregation			
203.0.112.0/23	192.0.2.1	3356	2
203.0.114.0/23	192.0.2.2	3356	2
198.51.100.0/24	192.0.2.3	3356	3

update cost that grows with table depth and diversity, which are most visibly in IPv6. Our results suggest that the observed update-cost tradeoff may be avoidable: a locality-preserving research design delivers the same table size with substantially more predictable control-plane behavior.

The paper has a structure as follows. Section II reviews longest-prefix match and the RIB/FIB pipeline and explains why IPv6 stresses the update path. Section III contrasts the research and industry aggregation trajectories and positions FAQS and the Juniper-inspired algorithm. Section IV details the two algorithms, the AS3356 dataset, worked examples on a common trie, and VeriTables-based correctness checking. Section V reports the fifteen-year measurements of FIB size, update latency, and burst size for both protocols. Section VI interprets the compression-versus-efficiency gap and its implications for IPv6, and Section VII concludes.

II. BACKGROUND

A. Longest Prefix Match, RIB, and FIB

Internet forwarding uses *longest prefix match*: a router compares a packet's destination address against all known prefixes and selects the most specific (longest-mask) match [11], [12]. The Routing Information Base (RIB) holds the routes a router learns from BGP; the Forwarding Information Base (FIB) is the compiled, hardware-resident table the data plane consults for each packet. Our RIB and update streams come from the University of Oregon RouteViews project, which archives full RIB snapshots and BGP updates from 2003 onward [13].

Aggregation reduces FIB size by combining route entries that share forwarding attributes into fewer, broader prefixes, lowering memory use and speeding lookups without changing where packets go [4], [14]. Table I shows the idea: contiguous prefixes with the same next hop and path collapse into a shorter prefix, while entries with no compatible sibling remain.

B. IPv4, IPv6, and why the protocol matters

IPv4 uses 32-bit addresses and thus about 4.3×10^9 (2^{32}) addresses, a ceiling that drove the transition to IPv6 [15]. IPv6 uses 128-bit addresses (2^{128}), a vastly larger and sparser space, and its deployment norms (for example /64 subnetting and more aggressive de-aggregation for policy and multihoming) produce prefix sets that grow both in size and in structural diversity [16]. Table II summarizes the differences that matter for aggregation. The key point for this study is that IPv6 stresses an aggregation algorithm's *update* path far more than IPv4: a deeper trie (up to 128 levels versus 32) and faster-growing diversity magnify any per-update cost that scales with table depth.

TABLE II. IPV4 VS. IPV6 CHARACTERISTICS RELEVANT TO FIB AGGREGATION

Feature	IPv4	IPv6
Address length	32 bits	128 bits
Format	dotted decimal, 4 octets	colon hexadecimal, 8 groups
Address count	$\sim$4.3 billion (2^{32})	$\sim 3.4 \times 10^{38}$ (2^{128})
Header	20 bytes +options	40 bytes (fixed, routing-oriented)
Trie depth	up to 32	up to 128

C. Data and update handling pipeline

Public BGP collectors such as RouteViews export binary Multi-Threaded Routing (MRT) files containing RIB snapshots and update streams. We convert these to text with BGPDump [17], extracting the prefix, next hop, AS path, and timestamp for each record. We then filter for consistency: drop duplicate prefixes and withdrawn/incomplete records, separate IPv4 from IPv6, and restrict to updates involving AS3356. General update handling then detects each announcement or withdrawal and applies it to the FIB in order, so that the forwarding table tracks path availability in real time [18]. Stale or misapplied updates cause loops or blackholes, so both the aggregation algorithm and the verification step below must preserve forwarding equivalence on every update.

III. RELATED WORK: RESEARCH VS. INDUSTRY AGGREGATION

FIB aggregation has been approached along two distinct trajectories, and framing them side by side clarifies what this paper measures.

A. The research trajectory

Research on FIB aggregation has steadily traded raw optimality for practical, incremental behavior. **ORTC** constructs a provably minimal aggregated FIB while preserving forwarding, but it is computationally expensive and offers no incremental update path—any change means recomputation [4]. Liu et al. made ORTC-style aggregation *incremental*, proposing update-handling schemes (an optimal-size and a minimal-time variant) that avoid re-aggregating the whole trie on every change [19]. The simple **Level-1/Level-2** heuristics instead remove or merge prefixes that share a next hop with an immediate ancestor or sibling; they are fast and predictable but leave compression on the table relative to ORTC [20]. **SMALTA** makes near-optimal static aggregation deployable as a software-only upgrade on top of a Tree Bitmap FIB, but its

update handling reverts affected subtrees to unaggregated form and re-aggregates, producing systematic bursts of overhead under churn [5].

The **FIFA** family reframed the problem around incremental updates, confining recomputation to the affected portion of a Patricia trie and traversing it at most twice per update to keep near-optimal aggregation after every announcement or withdrawal [6]. A parallel line studies churn directly: an empirical analysis shows that FIB updates exhibit strong spatial and temporal locality [21], which motivates two designs by Bienkowski, Sarrar, Schmid, and Uhlig. For *dependent* prefixes they give an online algorithm that delays re-aggregation to bound the number of FIB writes, achieving an $O(w)$-competitive churn bound with w the maximum address length, at some cost in compression [22]; **LFA** (locality-aware FIB aggregation) aggregates subtrees only once they have been stable, trading a little compression for markedly fewer FIB updates [23]. A complementary line attacks the *representation* rather than the prefix set: building on classic compact lookup structures [24], entropy-bounded and trie-folding encodings approach the information-theoretic limit for a prefix set while still supporting LPM and near-optimal-time updates [25], [26], and population-count tries such as Poptrie push software-lookup throughput on the compressed structure [27]—all at the price of implementation complexity and reduced hardware portability.

FAQS sits at the practical end of this trajectory. It performs both static aggregation and incremental updates with a single trie traversal, using a bottom-up selection rule and an early-exit test that halts propagation as soon as a node's selected next hop stops changing [7]. Its design goal is low, predictable per-update cost with near-optimal compression—exactly the properties an operator facing high churn cares about.

B. The industry trajectory

Vendor FIB compression optimizes for the realities of forwarding hardware. Juniper's implementation on PTX and ACX7000 routers running Junos EVO (enabled by default since release 21.2R1) is our industry reference and is publicly documented [8], [9]. It applies two rules on a radix tree of the FIB: *shadowing*—a more-specific prefix is not installed when a covering superset already forwards to the same next hop—and *compression*—contiguous prefixes sharing a next hop are summarized into a single superset, applied iteratively bottom-up so aggregates merge into larger aggregates level by level. The algorithm never fabricates `null`-route entries to fill gaps: if a contributing prefix is missing, or a more specific route carries a different next hop, the affected subtree simply compresses less. In production this roughly halves the installed FIB (a 55–60% reduction, and up to 82% for a single-next-hop Internet table); to stay correct under churn it rebuilds affected tree blocks *make-before-break*, programming new entries before removing old ones so no packet is dropped [9]. The design goal is a compact, tightly packed hardware table. The tradeoff is on the update path: preserving that packing means re-running compression over the affected ancestor chain on each change, with no early-exit, so a single next-hop change can re-shadow and reinstall many entries and reshuffle a large aggregate. The industry algorithm we evaluate is modeled directly on these two published rules and their full-recompression update behavior.

C. The gap this paper fills

Table III places these algorithms side by side. Prior work evaluates them individually or over short traces; none directly compares a research algorithm and a deployed-style algorithm across both IPv4 and IPv6 over a decade-plus of real BGP growth. We therefore pair FAQS (research) against a Juniper-inspired algorithm (industry) and measure how each scales as the table grows.

IV. Methodology

A. Dataset

All experiments use BGP data from AS3356, operated by Lumen Technologies (historically Level 3). AS3356 is one of the oldest and most densely connected transit ASes, so its routing data is a rich, representative sample of global behavior for both IPv4 and IPv6. We draw one RIB snapshot at beginning of July and all-month long update streams from RouteViews across 2010-2025, and process everything locally to keep the dataset self-contained. Measurement years are reported at 2010, 2015, 2020, and 2025.

B. FAQS: the research algorithm

FAQS operates on a Patricia trie in which each node is either REAL (an actual FIB prefix) or FAKE (an ancillary structural node) [7]. Each node n stores: its type $T(n)$; an original next hop $O(n)$, taken from the FIB for REAL nodes and inherited from the nearest REAL ancestor for FAKE nodes; a selected next hop $S(n)$ chosen during aggregation (which may differ from $O(n)$ but preserves forwarding); and a FIB status indicating whether the node is installed.

Static aggregation (Algorithm 1) is a single post-order traversal. Each node computes $O(n)$, then $S(n)$ via FAQS_SELECT, which adopts a child's next hop only when (i) both children exist, (ii) they agree on S, and (iii) that agreed value differs from the node's own $O(n)$; otherwise the node keeps $O(n)$. Algorithm 1 states the rule for two agreeing children; more generally FAQS lifts the next hop that covers the larger share of a node's subtree and installs the minority as exceptions, which is why in Fig. 1 the $128.0.0.0/1$ node adopts the dominant next hop 3. A node is installed only when its selected next hop differs from that of its nearest installed ancestor, which minimizes installed rows while preserving LPM semantics (Fig. 1).

Incremental updates (Algorithm 2) insert or locate the affected node, propagate the new next hop to FAKE descendants, then recompute selected next hops *up* the ancestor path—stopping early the moment a node's S is unchanged. Because real BGP traces exhibit strong locality (most prefixes are announced at long lengths, so high-level subtrees are rarely

TABLE III. FIB AGGREGATION ALGORITHMS ACROSS THE RESEARCH–INDUSTRY SPECTRUM

Algorithm	Contribution	Advantage	Limitation
ORTC [4]	Provably minimal aggregated FIB with forwarding correctness	Optimal table size	Costly; no incremental updates
Incremental ORTC [19]	Update-handling for ORTC-style aggregation	Avoids full re-aggregation per change	Size/time trade-off; pre-Patricia-trie
Level-1/Level-2 [20]	Fast, predictable heuristic aggregation	Simple and fast	Lower compression than optimal methods
SMALTA [5]	Near-optimal static aggregation, software-only	Deployable without hardware/protocol changes	Subtree reversion + re-aggregation causes high churn overhead
FIFA [6]	Incremental aggregation on a Patricia trie	Near-optimal ratio with fast, localized updates	Scaling pressure in very large IPv6 spaces
Dependent-prefix & LFA [22], [23]	Churn-bounded, locality-aware aggregation	Predictable, bounded update cost	Trades compression for stability
FAQS (research) [7]	Single-traversal aggregation with early-exit updates	Fast, near-constant per-update latency; near-optimal ratio	Higher implementation complexity than simple heuristics
Juniper (industry) [8], [9]	Shadowing + iterative radix-tree compression on commercial hardware (Junos EVO)	Strong ratio (∼50–60% in production); make-before-break, no packet loss	Full ancestor-path recompression per update; larger bursts as IPv6 grows

Algorithm 1 FAQS static aggregation

```
1:  procedure FAQS_STATICAGG(root)
2:      for each node v in post-order traversal of root do
3:          if v.isReal then v.O ← v.origNextHop
4:          else v.O ← nearest REAL ancestor next hop
5:          end if
6:          v.S ← FAQS_SELECT(v)
7:          if v.parent = NULL then v.install ← true
8:          else v.install ← (v.S ≠ v.parent.S)
9:          end if
10:     end for
11: end procedure
12: function FAQS_SELECT(v)
13:     if v.isReal then return v.O
14:     end if
15:     if v.left ≠ NULL and v.right ≠ NULL then
16:         if v.left.S = v.right.S and v.left.S ≠ v.O then
    return v.left.S
17:         end if
18:     end if
19:     return v.O
20: end function
```

Algorithm 2 FAQS incremental update (early exit)

```
1:  procedure FAQS_HANDLEUPDATE(prefix, nexthop)
2:      v ← insert or locate node for prefix
3:      v.isReal ← true; v.O ← nexthop
4:      UPDATEDESCENDANTS(v)
5:      u ← v
6:      while u ≠ NULL do
7:          oldS ← u.S; u.S ← FAQS_SELECT(u)
8:          if u.parent = NULL then u.install ← true
9:          else u.install ← (u.S ≠ u.parent.S)
10:         end if
11:         if u.S = oldS then break                ▷ early exit
12:         end if
13:         u ← u.parent
14:     end while
15: end procedure
16: procedure UPDATEDESCENDANTS(v)
17:     for each descendant d of v do
18:         if ¬d.isReal then d.O ← v.O
19:         end if
20:     end for
21: end procedure
```

touched), the early-exit test keeps the average update confined to a small region of the trie.

Complexity. Static aggregation is $O(N)$ in the number of trie nodes with $O(H)$ recursion memory. An update walks the ancestor path in $O(H)$ worst case ($H \leq 32$ for IPv4, ≤ 128 for IPv6) plus $O(M)$ for M FAKE descendants; locality makes the average cost substantially sub-linear in table size.

Worked example (Fig. 1). We reuse the exact trie of the original study. Eight prefixes are present: $0.0.0.0/4$, $16.0.0.0/4$, and $32.0.0.0/3$ with next hop 1; $128.0.0.0/4$ and $144.0.0.0/4$ with next hop 3; $160.0.0.0/3$ with next hop 2; $192.0.0.0/2$ with next hop 3; and $192.0.0.0/4$ with next hop 1. Each box in Fig. 1 shows a node's original next hop O and selected next hop S; shaded boxes are installed in the FIB. In panel (a) (FAQS), the right subtree ($128.0.0.0/1$) is dominated by next hop 3, so FAQS lifts $S = 3$ to the $/1$ node and installs it once, keeping only the two exceptions $160.0.0.0/3 \rightarrow 2$ and $192.0.0.0/4 \rightarrow 1$; the entire left subtree folds into the root's next hop 1. FAQS installs four entries $\{\text{root} \rightarrow 1,\ 128.0.0.0/1 \rightarrow 3,\ 160.0.0.0/3 \rightarrow 2,\ 192.0.0.0/4 \rightarrow 1\}$. Panel (b) is the Juniper-inspired rule on the same trie: it summarises a parent only when both children already carry the parent's next hop, so it cannot lift next hop 3 to the $/1$ node and instead installs $192.0.0.0/2$ and $128.0.0.0/3$ separately—five entries. This one-entry edge

Algorithm 3 Juniper-inspired static compression

```
1: procedure JUNIPER_STATICCOMPRESS(root)
2:     for each node v in post-order traversal of root do
3:         if v.isReal then v.O ← v.origNextHop
4:         else v.O ← v.parent.O
5:         end if
6:         APPLYCOMPRESSION(v)
7:     end for
8: end procedure
9: procedure APPLYCOMPRESSION(v)
10:    if v.left ≠ NULL and v.right ≠ NULL then
11:        if v.left.O = v.right.O = v.O then
12:            v.install ← true
13:            v.left.install ← false; v.right.install ← false
14:            return
15:        end if
16:    end if
17:    v.install ← v.isReal
18: end procedure
```

Algorithm 4 Juniper-inspired incremental update (no early exit)

```
1: procedure JUNIPER_HANDLEUPDATE(prefix, nexthop)
2:     v ← insert or locate node for prefix
3:     v.isReal ← true; v.O ← nexthop
4:     PROPAGATEDESCENDANTS(v)
5:     u ← v
6:     while u ≠ NULL do        ▷ full ancestor path always traversed
7:         APPLYCOMPRESSION(u)
8:         u ← u.parent
9:     end while
10: end procedure
```

is the "FAQS slightly smaller" effect that recurs at scale in Section V.

C. Juniper: the industry algorithm

Our Juniper-inspired algorithm prioritizes a small hardware table over update speed [8]. Static compression (Algorithm 3) is also a single post-order traversal, but it assigns FAKE nodes their *parent's* next hop and, whenever both children share the parent's next hop, installs only the parent and removes the children. This packs prefixes tightly but tightens the structural dependency between them (Fig. 1 contrasts the two selection rules).

The decisive difference is the update path (Algorithm 4): after propagating the change to descendants, it recompresses along the *entire* ancestor path to the root, with **no** early exit. This keeps the table maximally packed and correct, but every update can trigger re-shadowing and reinstallation of many entries—an effect that grows with table depth and diversity and is most pronounced in IPv6.

Complexity. Static compression is $O(N)$. An update is $O(H)$ along the ancestor path, but with no early exit each level can remove or install many hardware entries, so the realized per-update cost tracks table depth and diversity rather than the locality of the change.

D. Worked examples: update and withdrawal, early exit vs. full path

The algorithms' decisive difference is not the table they produce but the work they do to maintain it. Consider the exact update of the original study: prefix $192.0.0.0/3$ is announced with next hop 1 (Fig. 2). It sits on the path root $\rightarrow 128.0.0.0/1 \rightarrow 192.0.0.0/2 \rightarrow 192.0.0.0/3$.

FAQS (panel a) sets the new node's selected next hop and recomputes upward. It re-selects at $192.0.0.0/2$, then tests $128.0.0.0/1$: that node's selected next hop is still 3 (its subtree is still dominated by next hop 3), so FAQS *halts and never visits the root*. Only two nodes are touched; installation changes only locally, as $192.0.0.0/3 \rightarrow 1$ now covers $192.0.0.0/4$, which drops out.

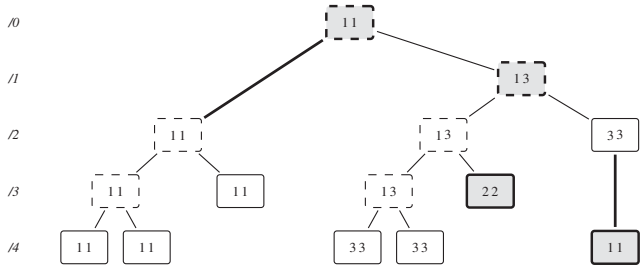
Juniper (panel b) applies the same announcement but recompresses the *entire* ancestor path $192.0.0.0/3 \rightarrow 192.0.0.0/2 \rightarrow 128.0.0.0/1 \rightarrow$ root (four nodes), re-evaluating compression at every level even though nothing above $128.0.0.0/1$ changes. Both algorithms preserve forwarding for every destination; only the cost differs. In this depth-4 example the saving is two levels; in a depth-128 IPv6 trie the same mechanism has FAQS stop within a few levels of the change while Juniper always climbs to the root—the direct cause of the latency and burst gaps reported in Section V.

A withdrawal behaves the same way. When $32.0.0.0/3$ (next hop 1) is withdrawn, its next hop already matched the root's default 1, so FAQS marks the node FAKE, finds the selection unchanged at its parent, and halts—dropping the now-redundant node without visiting the root. Juniper again recompresses the full path to the root before settling. Both remain forwarding-correct; FAQS touches the change-local region while Juniper traverses to the root, the same early-exit-versus-full-path split as the update.

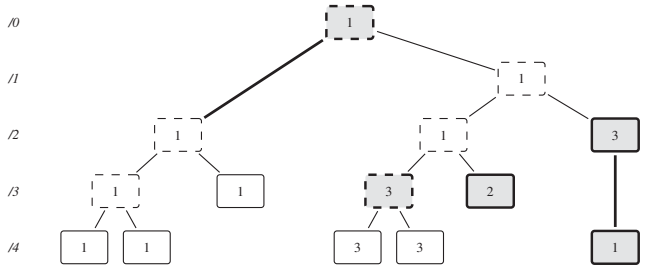
Table IV distils the three walkthroughs into asymptotic terms. The static build is a single traversal for both algorithms, so their table sizes are comparable (differing only by the relabelling edge of Fig. 1). The divergence is entirely in the incremental operations: FAQS's cost is $O(d)$, where d is the depth from the change to the first ancestor whose selected next hop is unaffected, whereas Juniper's is $O(H)$ in the trie height on *every* update and withdrawal. Because d is bounded by the locality of the change and H is bounded only by the address length (32 or 128), the gap is negligible for shallow IPv4 tries and pronounced for deep IPv6 ones—exactly the trend Section V measures.

E. Correctness verification with VeriTables

Because a router keeps several FIB copies that must forward identically, an aggregation error can silently loop or drop traffic. We verify every aggregated and updated table with

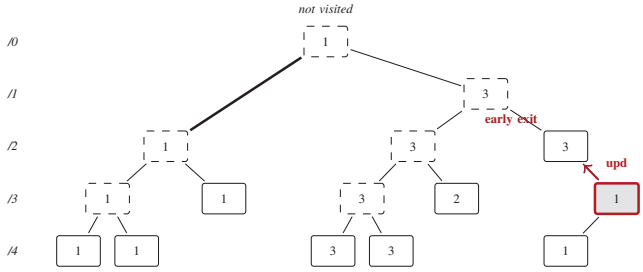


(a) FAQS: boxes show $O\,S$; dominant next hop 3 lifts to the /1 node. Installed (shaded): four entries.

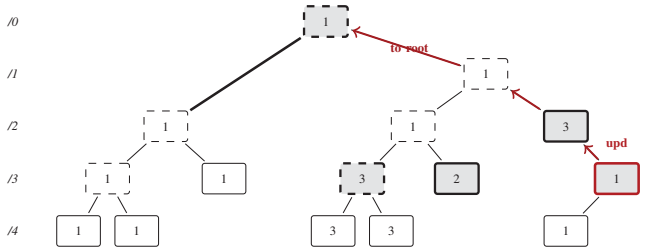


(b) Juniper: boxes show S; 192.0.0.0/2 and 128.0.0.0/3 install separately. Installed (shaded): five entries.

Fig. 1. Static aggregation on the exact eight-prefix trie of the original study (solid = REAL, dashed = FAKE; left axis gives prefix length). FAQS lifts the dominant next hop 3 to the 128.0.0.0/1 node and installs four entries; the Juniper-inspired rule cannot relabel that FAKE node and installs five—the one-entry edge FAQS holds at scale.



(a) FAQS: recompute 192.0.0.0/3 → 192.0.0.0/2, then halt—the /1 selection is unchanged (two nodes).



(b) Juniper: recompress 192.0.0.0/3 → 192.0.0.0/2 → 128.0.0.0/1 → root (four nodes).

Fig. 2. The exact update of the original study—announce 192.0.0.0/3 with next hop 1 (marked **upd**). FAQS recomputes up the ancestor path and *halts* once the selected next hop is unchanged at the 128.0.0.0/1 node, touching two nodes; the Juniper-inspired rule recompresses the full path to the root, touching four. Both keep forwarding identical.

TABLE IV. PER-OPERATION COST. N : PREFIXES; H: TRIE HEIGHT (≤ 32 IPv4, ≤ 128 IPv6); d: DEPTH FROM THE CHANGE TO THE FIRST STABLE ANCESTOR ($d \ll H$ UNDER LOCALITY).

Operation	FAQS (research)	Juniper (industry)
Static build	$O(N)$	$O(N)$
Insert / update	$O(d)$, early exit	$O(H)$, full path
Withdrawal	$O(d)$, early exit	$O(H)$, full path
Nodes touched / update	change-local ($\leq d$)	path-to-root (H)
Installed-row churn	bounded, local	grows with depth

VeriTables [10], a two-stage checker (Algorithm 5). First it merges all tables under comparison into a single Joint Patricia Trie, mapping specific and covering prefixes (e.g., 10.1.1.0/24 and 10.1.0.0/16) into one hierarchy. Then a single traversal performs top-down inheritance—any table missing a prefix adopts its nearest real ancestor's next hop—and a bottom-up check flags "leaks" (uncovered address space). Comparing the resulting next hop at every node and gap gives a mathematical guarantee that aggregation never changed the forwarding decision for any destination.

V. RESULTS

Both algorithms are implemented in the same Python framework and validated with VeriTables in every measurement year on a regular computer. We report (i) final aggregated FIB size, (ii) average per-update processing latency, and (iii) FIB burst—the number of installed entries changed per update.

A. Final FIB size: compression parity

Figures 3 and 4 show final aggregated FIB size against the unaggregated baseline. The headline result is parity: despite fundamentally different strategies, FAQS and Juniper compress to nearly the same size in every year, for both protocols. For IPv4 in 2025 both converge near 360k–375k entries against an unaggregated ∼730k—roughly a 50% reduction. For IPv6 in 2025 both land near 75k against an unaggregated ∼165k. FAQS is consistently a hair smaller, but the gap in table size is never the interesting story.

A visible anomaly appears at the IPv4 2020 point, where both algorithms dip slightly below 2015 before resuming growth in 2025. This is not measurement noise: a large BGP prefix withdrawal in the AS3356 stream near the 2020 window reduced active leaf prefixes, letting both algorithms collapse more subtrees. The dip is a reminder that FIB size responds to short-term update behavior (de-aggregation reversals, selective filtering), not only long-term growth. In IPv6, where the table grew more than 100-fold from 2010 to 2025, both algorithms scaled their compression proportionally with no meaningful divergence in size.

Algorithm 5 VeriTables forwarding-equivalence check

```
1:  function VERIFY(v, anc)
2:      if v.isReal then anc ← v
3:      end if
4:      leak ← false
5:      for child ∈ {v.left, v.right} do
6:          if child ≠ NULL then
7:              if child.isReal then INHERIT(anc, child)
8:              end if
9:              L ← VERIFY(child, anc)
10:             if L or child.len − v.len > 1 then leak ← true
11:             end if
12:         else leak ← true
13:         end if
14:     end for
15:     if (leaf) or (leak ∧ v.isReal) then
16:         if ¬ALLEQUAL(v.H) then return ERROR
17:         end if
18:         return false
19:     end if
20:     return leak
21: end function
```

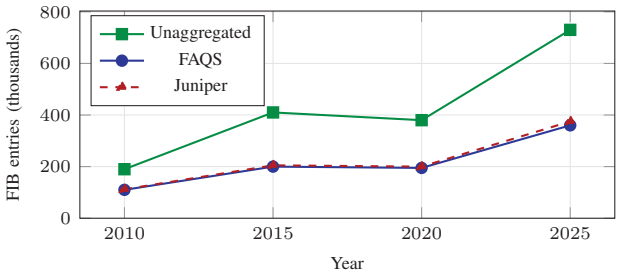


Fig. 3. Final aggregated FIB size, IPv4 (2010–2025), with the unaggregated baseline. FAQS and Juniper track each other closely.

B. Update latency: the structural gap

Figures 5 and 6 show where the algorithms part ways. FAQS achieves consistently lower per-update latency in every year for both protocols, and the gap widens as the table grows. For IPv4, FAQS is nearly flat, rising only from 3.67 to 3.79 μs over fifteen years-0.12 μs total, while Juniper climbs from 4.29 to 4.55 μs on a visibly steeper slope, where there is approximately a 16.7% lower latency for FAQS. IPv6 makes the effect stark: FAQS grows from 2.34 to 3.59 μs and Juniper from 3.78 to 4.77 μs, with a persistent $\sim$1.2–1.4 μs gap that reflects a structural advantage, not chance.

The reason is the update rule. FAQS recomputes only along the ancestor path of the affected node and exits as soon as a selected next hop is unchanged, so most updates stay local and per-update cost is largely decoupled from global table size—hence the flat IPv4 curve. Juniper, lacking an early exit, recompresses the full ancestor path on every change to keep the table packed; as the IPv6 table deepens and

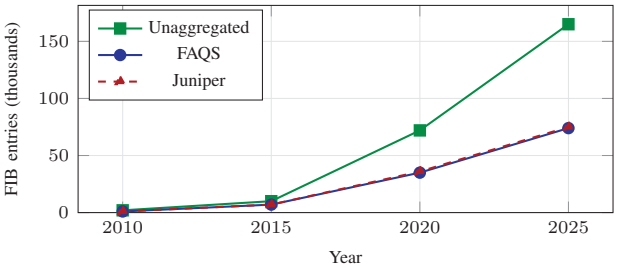


Fig. 4. Final aggregated FIB size, IPv6 (2010–2025). The table grows more than 100-fold; both algorithms scale their compression proportionally.

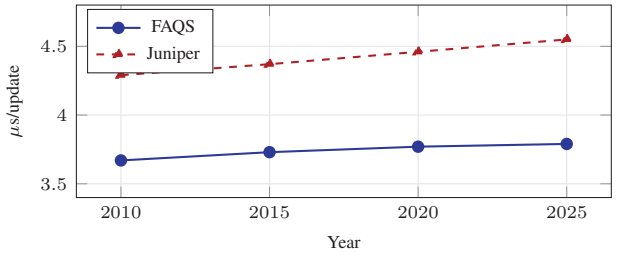


Fig. 5. Average per-update processing time, IPv4 (2010–2025). FAQS is nearly flat; Juniper is higher and rises faster.

diversifies, more ancestors are recompressed per update and each recompression is likelier to move an installed entry. Both reach the same compressed state, but FAQS gets there with far less work per update.

C. FIB burst: fewer and smaller

Tables V and VI report the distribution of FIB bursts (entries changed per update). FAQS places more updates in the zero-change and ≤ 1 bins than Juniper in every year. In 2025 for IPv4, FAQS keeps 79.35% of updates within a single change versus 70.27% for Juniper, a 9-point gap that mirrors Juniper's re-shadowing model, and its maximum single update burst is 2,380 entries versus Juniper's 4,437, which is a 46% reduction for FAQS.

IPv6 shows the same pattern with a sharper onset. The two look alike in 2010, when the table is small, but diverge by 2015: Juniper's maximum burst reaches 3,107 against just 48 for FAQS. This 2015 spike coincides with the fastest IPv6 growth in the trace, evidence that as the IPv6 trie deepens, Juniper's recompression occasionally triggers extreme installed-entry churn. By 2025 both show larger maxima (1,484 vs. 1,429) but FAQS still keeps more updates within ≤ 1 change (91.11% vs. 89.45%). Large FIB update bursts may increase hardware programming pressure, and many simultaneous TCAM writes can stall the forwarding pipeline, so smaller, rarer bursts are an operational advantage, not just a statistical one.

VI. Discussion

A. Compression is a tie; efficiency is not

The clearest lesson of the fifteen-year comparison is that the research and industry algorithms are indistinguishable on the

TABLE V. IPv4 FIB BURST STATISTICS. MIN/MAX ENTRIES CHANGED PER UPDATE AND THE SHARE OF UPDATES WITH EXACTLY 0, ≤ 1, AND ≤ 10 CHANGES.

Algorithm	Min	Max	FIB Δ=0	FIB $\Delta\leq$1	FIB $\Delta\leq$10	All updates
2010 (total: 3,465,726)						
FAQS	0	799	1,427,596 (41.19%)	2,795,617 (80.66%)	3,460,301 (99.84%)	3,465,725 (100.00%)
Juniper	0	1,071	1,401,734 (40.45%)	2,563,636 (73.97%)	3,457,820 (99.77%)	3,465,726 (100.00%)
2015 (total: 5,428,872)						
FAQS	0	522	2,135,391 (39.33%)	4,308,147 (79.36%)	5,420,993 (99.85%)	5,428,871 (100.00%)
Juniper	0	949	2,088,240 (38.47%)	3,914,402 (72.10%)	5,417,378 (99.79%)	5,428,872 (100.00%)
2020 (total: 7,575,754)						
FAQS	0	964	2,782,803 (36.73%)	5,833,656 (77.00%)	7,563,312 (99.84%)	7,575,750 (100.00%)
Juniper	0	1,571	2,685,201 (35.44%)	5,086,074 (67.14%)	7,558,852 (99.78%)	7,575,754 (100.00%)
2025 (total: 7,915,985)						
FAQS	0	2,380	3,533,304 (44.64%)	6,281,020 (79.35%)	7,904,171 (99.85%)	7,915,982 (100.00%)
Juniper	0	4,437	3,437,483 (43.42%)	5,562,219 (70.27%)	7,900,390 (99.80%)	7,915,985 (100.00%)

TABLE VI. IPv6 FIB BURST STATISTICS. MIN/MAX ENTRIES CHANGED PER UPDATE AND THE SHARE OF UPDATES WITH EXACTLY 0, ≤ 1, AND ≤ 10 CHANGES.

Algorithm	Min	Max	FIB Δ=0	FIB $\Delta\leq$1	FIB $\Delta\leq$10	All updates
2010 (total: 248,819)						
FAQS	0	4	238,462 (95.84%)	248,559 (99.90%)	248,819 (100.00%)	248,819 (100.00%)
Juniper	0	4	238,462 (95.84%)	248,531 (99.88%)	248,819 (100.00%)	248,819 (100.00%)
2015 (total: 25,567,953)						
FAQS	0	48	22,320,334 (87.30%)	25,071,682 (98.06%)	25,566,712 (100.00%)	25,567,953 (100.00%)
Juniper	0	3,107	22,319,928 (87.30%)	25,049,526 (97.97%)	25,563,687 (99.98%)	25,567,953 (100.00%)
2020 (total: 60,628,560)						
FAQS	0	577	42,210,579 (69.62%)	57,492,471 (94.83%)	60,621,338 (99.99%)	60,628,560 (100.00%)
Juniper	0	694	42,170,909 (69.56%)	55,947,943 (92.28%)	60,620,299 (99.99%)	60,628,560 (100.00%)
2025 (total: 190,577,485)						
FAQS	0	1,429	118,500,313 (62.18%)	173,639,172 (91.11%)	190,565,922 (99.99%)	190,577,485 (100.00%)
Juniper	0	1,484	118,396,053 (62.12%)	170,465,505 (89.45%)	190,560,245 (99.99%)	190,577,485 (100.00%)

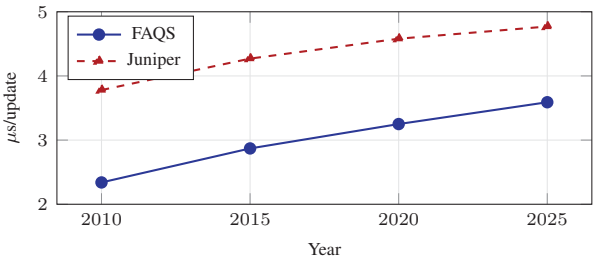


Fig. 6. Average per-update processing time, IPv6 (2010–2025). The gap persists at $\sim$1.2–1.4 μs as prefix volume grows 100-fold.

metric operators historically optimized—final table size—but far apart on the metrics that govern control-plane behavior. Both roughly halve the IPv4 table and both scale IPv6 compression with growth. Yet FAQS processes updates faster in every year and produces fewer, smaller bursts. Because both reach essentially the same compressed state, the difference is not *what* they compute but *how much work* they do to maintain it under churn.

B. Why the research design wins under churn

Two properties explain the gap. First, FAQS confines each update to the local subtree around the change and uses a bottom-up early-exit test, so the average update touches few nodes and changes few installed rows. Juniper instead packs prefixes into the fewest possible rows and, to preserve that packing, recompresses the entire ancestor path on every change; the price is re-shadowing and reinstallation of many entries per update. Second, this makes the two algorithms scale differently: FAQS's per-update cost is largely independent of global table size (flat IPv4 latency), whereas Juniper's grows super-linearly with depth and diversity as logical prefixes entangle with hardware entries. These observations match the churn literature, which holds that locality-preserving, change-bounding designs yield smaller bursts and more predictable update costs [28].

The worked example in Section IV-D makes the mechanism concrete and quantitative. FAQS's per-update cost is governed by the distance from the changed node to the first ancestor whose selected next hop is unaffected—usually a handful of levels, because in a locality-rich trie a next-hop change rarely propagates far. Juniper's cost is instead governed by the full trie height H (≤ 32 for IPv4, ≤ 128 for IPv6), regardless of how local the change actually was. That single structural fact explains the shapes in Figures 5–6: FAQS's IPv4 latency is almost flat because its touched-region size does not grow with the table, whereas Juniper's rises steadily and, in IPv6, pulls away as H-deep recompressions become both more frequent and more likely to move an installed row. The same fact drives the burst distributions: the nine-point IPv4 gap in the $\Delta\leq 1$ bin (Table V) and Juniper's 2015 IPv6 maximum of 3,107 versus

FAQS's 48 (Table VI) are the aggregate footprint of precisely the full-path-versus-early-exit difference the two-node-versus-four-node trace illustrates. In other words, the microscopic example and the fifteen-year measurements are two views of one mechanism.

C. Implications for IPv6 scalability

IPv6 is where the choice matters most. Its larger, sparser space and its faster growth in specific de-aggregations (for policy and multihoming) deepen and diversify the trie [28]. In our 2015–2025 IPv6 results this shows up as widening latency and occasional extreme Juniper bursts, while FAQS holds low average latency and bounded bursts. As operators lean further into IPv6, an algorithm whose update cost tracks the locality of the change—not the size of the table—offers markedly more predictable control-plane behavior.

D. What each side can take from the other

The comparison is not a verdict that industry compression is wrong; its packing goal is well matched to scarce TCAM rows on a fixed chip. The finding is narrower and more useful: the tradeoff industry accepts—slower, burstier updates in exchange for a compact table—is avoidable. A locality-preserving research design reaches the same table size while decoupling update cost from table growth. For vendors, adopting an early-exit, subtree-local update path would retain the compression they already achieve while cutting per-update latency and TCAM write bursts. For researchers, the parity in final size is a reminder that compression ratio is a largely solved, saturated metric on real traces; the open ground is update predictability at IPv6 scale.

More broadly, the result reframes what "good enough" means for a deployed FIB algorithm. For two decades the headline metric has been compression ratio, and on that axis a simple industry heuristic and a decade of research converge. But compression ratio is a property of the *steady state*, whereas an operator experiences the algorithm through its *transients*—the latency and the size of the hardware write burst on every one of hundreds of millions of updates per year (Tables V–VI). A design that is optimal in steady state but pays $O(H)$ per transient is, at IPv6 depth, strictly dominated by one that is near-optimal in steady state and sub-linear per transient. The practical takeaway for the two communities is symmetric: vendors can retain their compact tables while adopting an early-exit update path, and researchers should report transient behavior—per-update latency distributions and burst tails—as a first-class result alongside compression ratio, because that is where the remaining, operationally decisive differences now live.

E. Limitations

Three caveats bound our claims. First, both algorithms are Python re-implementations validated for forwarding correctness with VeriTables, not vendor-optimized production code; absolute microsecond figures would differ in a line-card runtime, though the two share an environment so the *ratios* should reflect the algorithms. Second, a production system must also handle control-plane integration and hardware constraints we did not model; a fair next step is porting FAQS into a router control plane and benchmarking against vendor software on real hardware. Third, we study a single AS. AS3356 was chosen for its scale and representativeness, but ASes with different peering or policy may churn differently.

VII. Conclusion and Future Work

We presented the first fifteen-year, head-to-head comparison of a research FIB aggregation algorithm (FAQS) and an industry-style compression algorithm (Juniper-inspired) on real IPv4 and IPv6 BGP traces from AS3356. Across every measurement year the two achieve comparable compression—roughly halving the IPv4 table—but FAQS processes updates faster (3.79 vs. 4.55 μs for IPv4 in 2025, with a larger, faster-growing gap in IPv6) and produces fewer, smaller FIB bursts (2025 IPv4 maximum 2,380 vs. 4,437). The advantage is algorithmic: locality and early exit decouple update cost from table growth, whereas full-path recompression couples it to depth and diversity. With IPv6 tables growing exponentially and TCAM remaining costly, that difference translates directly into lower hardware pressure and steadier forwarding.

The broader point is that FIB aggregation research has matured to where its designs offer concrete, measurable operational benefits over deployed practice without sacrificing compression. Future work includes obtaining a reference vendor implementation (or vendor pseudocode) for a more conclusive comparison, extending the evaluation to additional ASes with varied peering and policy, and prototyping a research-grade update path inside production router software. We offer this longitudinal study as a data-driven basis for bringing locality-preserving aggregation from the literature into the next generation of router software.

References

[1] G. Huston, "BGP in 2024: The BGP table," The ISP Column, APNIC/Potaroo, Jan. 2025, [Online]. Available: https://bgp.potaroo.net/.

[2] B. Quoitin, S. Uhlig, C. Pelsser, L. Swinnen, and O. Bonaventure, "Interdomain traffic engineering with BGP," *IEEE Communications Magazine*, vol. 41, no. 5, pp. 122–128, May 2003, doi: https://doi.org/10.1109/MCOM.2003.1200112.

[3] G. Varghese, *Network Algorithmics: An Interdisciplinary Approach to Designing Fast Networked Devices*. San Francisco, CA, USA: Morgan Kaufmann, 2005.

[4] R. Draves, C. King, V. Srinivasan, and B. Zill, "Constructing optimal IP routing tables," in *Proc. IEEE INFOCOM*, vol. 1, New York, NY, USA, Mar. 1999, pp. 88–97, doi: https://doi.org/10.1109/INFCOM.1999.749256.

[5] Z. A. Uzmi, M. Nebel, A. Tariq, S. Jawad, R. Chen, A. Shaikh, J. Wang, and P. Francis, "SMALTA: Practical and near-optimal FIB aggregation," in *Proc. 7th Conf. Emerging Networking Experiments and Technologies (CoNEXT)*, Tokyo, Japan, Dec. 2011, pp. 29:1–29:12, doi: https://doi.org/10.1145/2079296.2079325.

[6] Y. Liu, B. Zhang, and L. Wang, "FIFA: Fast incremental FIB aggregation," in *Proc. IEEE INFOCOM*, Turin, Italy, Apr. 2013, pp. 1213–1221, doi: https://doi.org/10.1109/INFCOM.2013.6566913.

[7] Y. Liu and G. Grigoryan, "Toward incremental FIB aggregation with quick selections (FAQS)," in *Proc. 17th IEEE Int. Symp. Network Computing and Applications (NCA)*, Cambridge, MA, USA, Nov. 2018, pp. 1–8, doi: https://doi.org/10.1109/NCA.2018.8548101.

[8] N. Fevrier, "FIB compression — optimizing your routing tables," Juniper Networks, The Feed (video), Feb. 2023, [Online]. Available: https://www.juniper.net/us/en/the-feed/topics/wan/fib-compression-optimizing-your-routing-tables.html.

[9] ——, "PTX and ACX7000 FIB compression," Juniper Networks Tech-Post, Sep. 2022, [Online]. Available: https://community.juniper.net/blogs/nicolas-fevrier/2022/09/19/ptx-fib-compression.

[10] Y. Liu, G. Grigoryan, J. Li, G. Sun, and T. Tauber, "VeriTable: Fast equivalence verification of multiple large forwarding tables," *Computer Networks*, vol. 168, p. 106981, 2020, earlier version in Proc. IEEE INFOCOM, 2018, pp. 621–629. doi: https://doi.org/10.1016/j.comnet.2019.106981.

[11] V. Srinivasan and G. Varghese, "Fast address lookups using controlled prefix expansion," *ACM Transactions on Computer Systems*, vol. 17, no. 1, pp. 1–40, Feb. 1999, doi: https://doi.org/10.1145/296502.296503.

[12] M. A. Ruiz-Sánchez, E. W. Biersack, and W. Dabbous, "Survey and taxonomy of IP address lookup algorithms," *IEEE Network*, vol. 15, no. 2, pp. 8–23, Mar./Apr. 2001, doi: https://doi.org/10.1109/65.912716.

[13] University of Oregon, "The RouteViews project," 2025, [Online]. Available: http://www.routeviews.org/.

[14] V. Fuller, T. Li, J. Yu, and K. Varadhan, "Classless inter-domain routing (CIDR): An address assignment and aggregation strategy," Internet Engineering Task Force (IETF), Request for Comments RFC 1519, Sep. 1993, doi: https://doi.org/10.17487/RFC1519.

[15] J. Postel, "Internet protocol," Internet Engineering Task Force (IETF), Request for Comments RFC 791, Sep. 1981, doi: https://doi.org/10.17487/RFC0791.

[16] S. E. Deering and R. M. Hinden, "Internet protocol, version 6 (IPv6) specification," Internet Engineering Task Force (IETF), Request for Comments RFC 8200, Jul. 2017, doi: https://doi.org/10.17487/RFC8200.

[17] RIPE NCC, "BGPdump: A utility for parsing MRT routing files," RIS Project, 2024, [Online]. Available: https://github.com/RIPE-NCC/bgpdump.

[18] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian, "Delayed Internet routing convergence," *IEEE/ACM Transactions on Networking*, vol. 9, no. 3, pp. 293–306, Jun. 2001, doi: https://doi.org/10.1109/90.929852.

[19] Y. Liu, X. Zhao, K. Nam, L. Wang, and B. Zhang, "Incremental forwarding table aggregation," in *Proc. IEEE Global Communications Conference (GLOBECOM)*, Miami, FL, USA, Dec. 2010, pp. 1–6, doi: https://doi.org/10.1109/GLOCOM.2010.5683158.

[20] X. Zhao, Y. Liu, L. Wang, and B. Zhang, "On the aggregatability of router forwarding tables," in *Proc. IEEE INFOCOM*, San Diego, CA, USA, Mar. 2010, pp. 848–856, doi: https://doi.org/10.1109/INFCOM.2010.5462137.

[21] N. Sarrar, M. Bienkowski, S. Schmid, S. Uhlig, and R. Wuttke, "Exploiting locality of churn for FIB aggregation," Technische Universität Berlin, Tech. Rep. 2012/12, 2012.

[22] M. Bienkowski, N. Sarrar, S. Schmid, and S. Uhlig, "Brief announcement: Dynamic forwarding table aggregation without update churn — the case of dependent prefixes," in *Proc. 27th Int. Symp. Distributed Computing (DISC)*, Jerusalem, Israel, Oct. 2013, full version: "Competitive FIB Aggregation without Update Churn," in Proc. IEEE ICDCS, 2014.

[23] N. Sarrar, R. Wuttke, S. Schmid, M. Bienkowski, and S. Uhlig, "Leveraging locality for FIB aggregation," in *Proc. IEEE Global Communications Conference (GLOBECOM)*, Austin, TX, USA, Dec. 2014, pp. 1930–1935, doi: https://doi.org/10.1109/GLOCOM.2014.7037090.

[24] M. Degermark, A. Brodnik, S. Carlsson, and S. Pink, "Small forwarding tables for fast routing lookups," in *Proc. ACM SIGCOMM*, Cannes, France, Sep. 1997, pp. 3–14, doi: https://doi.org/10.1145/263105.263133.

[25] G. Rétvári, J. Tapolcai, A. Kőrösi, A. Majdán, and Z. Heszberger, "Compressing IP forwarding tables: Towards entropy bounds and beyond," in *Proc. ACM SIGCOMM*, Hong Kong, China, Aug. 2013, pp. 111–122, aCM SIGCOMM Comput. Commun. Rev., vol. 43, no. 4.

[26] B. Mihálka, A. Kőrösi, and G. Rétvári, "Compressing virtual forwarding information bases using the trie-folding algorithm," in *Advances in Communication Networking (EUNICE 2014)*, ser. Lecture Notes in Computer Science, vol. 8846. Cham, Switzerland: Springer, 2014, pp. 121–133, book doi: https://doi.org/10.1007/978-3-319-13488-8.

[27] H. Asai and Y. Ohara, "Poptrie: A compressed trie with population count for fast and scalable software IP routing table lookup," in *Proc. ACM SIGCOMM*, London, U.K., Aug. 2015, pp. 57–70, doi: https://doi.org/10.1145/2829988.2787474.

[28] A. Elmokashfi, A. Kvalbein, and C. Dovrolis, "BGP churn evolution: A perspective from the core," in *Proc. IEEE INFOCOM*, San Diego, CA, USA, Mar. 2010, pp. 1208–1216, doi: https://doi.org/10.1109/INFCOM.2010.5461924.