Online Aggregation of the Forwarding Information Base: Accounting for Locality and Churn

Marcin Bienkowski Nadi Sarrar Stefan Schmid Steve Uhlig

***Abstract*—This paper studies the problem of compressing the Forwarding Information Base (FIB), but taking a wider perspective. Indeed, FIB compression goes beyond sheer compression, as the gain in memory use obtained from the compression has consequences on the updates that will have to be applied to the compressed FIB.**

We are interested in the situation where forwarding rules can change over time, e.g., due to BGP route updates. Accordingly, we frame FIB compression as an online problem, and design competitive online algorithms to solve it. In contrast to prior work which mostly focused on static optimizations, we study an online variant of the problem where routes can change over time, and where the number of updates to the FIB are taken into account explicitly. The reason to consider this version of the problem is that leveraging temporal locality while accounting for the number of FIB updates helps to keep routers CPU load low and reduces the number of FIB updates to be transferred, e.g., from the network-attached Software-Defined Network (SDN) controller to a remote switch.

This paper introduces a formal model which is an interesting generalization of several classic online aggregation problems. Our main contribution is an *O(w)*-competitive algorithm, where *w* is the length of an IP address. We also derive a lower bound which shows that our result is asymptotically optimal within a natural class of algorithms, based on so-called *sticks*.

***Keywords*-Prefix Aggregation, Competitive Analysis, Software-Defined Networking**

I. Introduction

At the heart of any router (and switch) lies a so-called *Forwarding Information Base* (FIB) containing the router's *forwarding rules*. A routing decision for a given packet is made on the basis of these rules and the destination IP address of a packet. A fast rule lookup requires the FIB to be stored in a fast (and expensive) memory on the line cards.

For quite some time now, the number of FIB rules at the Internet core routers has been growing. In 2017, the FIB size exceeds 700k.[1] New forwarding rules emerge primarily because of the growth of the Internet itself, trends for advertising more specific routes [20] (e.g., for traffic engineering purposes [29]), or the increasing demand for virtual networks [6], [16]. The migration to IPv6 is not expected to mitigate the address space disaggregation problem [7]. The increasing memory requirement comes at a significant cost for ISPs [31], as this memory is expensive and power-hungry [21].

Marcin Bienkowski is with University of Wrocław, Poland.
Nadi Sarrar is with TU Berlin, Germany.
Stefan Schmid is with University of Vienna, Austria and Aalborg University, Denmark.
Steve Uhlig is with Queen Mary University, UK.
Manuscript sent February 2017.
[1]See e.g., http://bgp.potaroo.net/as6447/.

An interesting solution to alleviate the problem — before possible long-term solutions are deployed — is the *aggregation/compression of the FIB*, i.e., the replacement of the existing set of rules by an *equivalent but smaller* set. The aggregation of FIB rules has the appealing property that it is a *purely local solution* in the sense that it does not affect neighboring routers and it can be done by a simple software update [30].

While the compression of the FIB is beneficial in terms of memory, it also entails a potential overhead: As the FIB of a router changes dynamically over time — typically several thousands rules are modified each second on a BGP core router [9] — the rule compression may lead to a situation where already aggregated FIB entries need to be disaggregated again, resulting in a larger number of rule updates. There is a certain cost associated with each such update: First, each update entails some changes in the local data structure, which needs to be rebuilt. Moreover, transmissions of control messages is problematic as the communication channel between route processor and line card (resp. between the controller and the switch) can become a bottleneck [19]. In the worst-case, the updates even have to be transmitted over a network featuring varying latencies: in the context of Software-Defined Networks (SDN), the *SDN controller* [18] needs to send updates to a remote OpenFlow switch.

Accordingly, we in this paper make the case for, and initiate the study of, FIB aggregation algorithms which simultaneously try to maximize the compression ratio and minimize the number of updates to the compressed FIB. In particular, we are interested in the practically relevant *dynamic* setting, where forwarding rules can change over time, e.g., due to BGP route updates. Accordingly, we aim to design *competitive online algorithms*.

Next, and before discussing our contributions and related work, we introduce our model in more detail.

A. The Model

An (IP) *address* is a binary string of length w (e.g., $w = 32$ for IPv4 and $w = 128$ for IPv6) or equivalently an integer from $[0, 2^w - 1]$. An (IP) *prefix* is a binary string of length at most w; we denote the empty prefix by ε. A prefix *contains* all addresses that start with it, i.e., it corresponds to a *range* of addresses of the form $[k \cdot 2^i, (k+1) \cdot 2^i - 1]$, where $w - i$ is the prefix length and $k \geq 0$ is an integer.

Forwarding Information Base (FIB). We consider a packet forwarding router (or a switch) with a set of *ports* (also known as the *next-hops*). A *Forwarding Information Base (FIB)* is

a set of *forwarding rules* used by the router; each rule is a *(prefix,port)* pair (p, c). For the presentation, we will refer to the ports by *colors*, i.e., assume a unique color for each port. For any packet processed by the router, a decision is made on the basis of its destination IP address x using the *longest prefix match* policy [19]: among the FIB rules $\{(p_i, c_i)\}_i$, the router chooses the longest p_i being a prefix of x, and forwards the packet to port c_i. We assume that there are no two rules with the same prefixes and different ports. If no rule matches, the packet is dropped.

For instance, consider a FIB containing four rules $\{(\varepsilon, a), (00, b), (1, c), (11, a)\}$, where a, b, and c are ports. It could be replaced by an equivalent FIB containing the rules $\{(\varepsilon, a), (00, b), (10, c)\}$. In this compression process, we require *strong forwarding correctness* [30], i.e., we require that the forwarding and dropping behavior remain the same.

Finally, we call two (different) rules *dependent* if the ranges represented by them overlap (i.e., one of these ranges is contained in the other) and *independent* otherwise.

Costs and Competitive Analysis. We consider a simple router which consists of two parts: the *controller* (typically implemented on the route processor) and the *(compressed) FIB* (stored in a fast and expensive memory), cf. Figure 1. The controller keeps a copy of the *uncompressed FIB (U-FIB)* and receives dynamic updates to this structure, e.g., due to various events from the *Border Gateway Protocol, BGP*. More precisely, we assume continuous time; at any time t, a single forwarding rule may change its color (port). In particular, we do not allow new rules to be inserted to U-FIB nor old rules to be deleted from it. Thus, the input is a sequence of such color changes called *events*. Right after a change occurs, the controller must ensure that the U-FIB and the FIB are equivalent. To this end, the controller may insert, delete or update (change color) individual rules in the FIB. The controller can also issue these commands at any point of time, e.g., for a delayed compression of the FIB. We associate a fixed cost α to any such change of a single rule. We emphasize that α is a fixed parameter but not necessarily a constant.

Note that we use a fixed parameter α independently of the change type issued by the controller (insert, delete, color update) to keep the model simple and general: α is not specific to any particular FIB data structure (e.g., trie or cache), but may also be used to model the cost of transmitting a control packet between an SDN controller and the OpenFlow switch. (See also [14], [25].) We will refer to the total cost paid this way as *update cost*, and the amount paid by an algorithm ALG in a time interval I is denoted by U-COST$_I$(ALG). More generally, we ignore IP lookup cost in our model, as it depends on both the choice of data structure and the hardware support (e.g., TCAM) provided by the target forwarding engine, and are often negligible in practice [19, chapter 15].

The second type of cost we want to optimize is the size of the FIB, which — following [8] — is defined as the number of FIB forwarding rules. This modeling is justified by state-of-the-art approaches (see, e.g., [19, chapter 15]), where the size of such a structure is usually proportional to the number of entries in the FIB. For an algorithm ALG and time t, we denote the number of FIB rules at time t by SIZE$_t$(ALG). The total memory cost in a time interval I is then defined as M-COST$_I$(ALG) $= \int_I$ SIZE$_t$(ALG) dt.

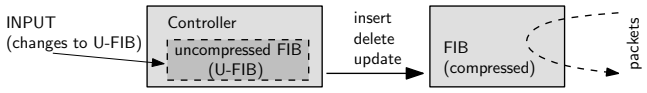


Fig. 1. Controller and FIB: the controller updates the rules in the FIB. This paper focuses on online algorithms for the controller.

In both objective functions (U-COST and M-COST), we drop time interval subscripts when referring to the total cost during the runtime of an algorithm. This paper focuses on minimizing the sum of these two costs, i.e., COST(ALG) $=$ U-COST(ALG) $+$ M-COST(ALG). Note that the parameter α can be used to put more emphasis on either of the two costs.

We assume a conservative standpoint and consider algorithms that do not have any knowledge of future prefix changes, and need to decide *online* on where and when to aggregate. Not relying on predictions is reasonable as it makes our approach robust to the often unpredictable behavior of the route updates in the modern Internet [13]. We use the standard yard-stick of online analysis [5], i.e., we compare the cost of the online algorithm to the cost of an optimal offline algorithm OPT that knows the whole input sequence in advance. We call an online algorithm ALG *ρ-competitive* if there exists a constant ξ, such that for any input sequence the following holds: COST(ALG) $\leq \rho \cdot$ COST(OPT) $+ \xi$. The competitive ratio of an algorithm is the minimum ρ, for which the algorithm is ρ-competitive.

Trie Representation. Throughout this paper, we represent both the U-FIB and the FIB as one-bit tries containing all the prefixes from the forwarding rules. This affects merely the description: we do not assume anything about the actual implementation of the U-FIB/FIB structures.

A Note about IP Lookup. In our model, we do not take into account the impact a FIB compression may have on IP lookup times, because they are affected only to a very limited extent. The state-of-the-art data structures used for IP lookup (see, [19, chapter 15] and the references therein) use a variety of tree-like constructs augmented with additional information. This allows for lookup times of order $O(\log w)$, with practical implementations using 2-3 memory lookups on average. Unfortunately, little is known about proprietary data structures actually used in the routers of different vendors.

B. Related Work

The increasing FIB size problem has received much attention recently, and many approaches have been explored to represent FIBs more efficiently [11], [12] or cache FIB entries on cheaper memory [2], leveraging Zipf's law [25]. FIB aggregation is a well-known technique to mitigate router memory consumption, and accordingly, it has been studied intensively and in different contexts. In particular, there are known fast algorithms for optimal FIB aggregation of table

snapshots, e.g., the seminal *Optimal Routing Table Constructor (ORTC)* [8] as well as others [22], [27]. However, these algorithms are static and do not support the efficient handling of incremental updates. Therefore, a re-computation of the optimally aggregated FIB is needed on each forwarding rule change. This is computationally expensive and can lead to high churn.

Several papers deal with this problem by proposing heuristics that simultaneously try to limit the number of updates to the FIB while maintaining a good compression rate, including SMALTA [28], FIFA [14], and others [10], [15], [17], [30]. However, none of these works give a formal bound on the achievable performance over time, neither with respect to the number of updates to the aggregated FIB nor to the aggregation gain. They also do not consider the temporal and spatial locality of churn for their benefit.

In this light, the paper closest to ours is [4], which studies online algorithms for FIB compression under the assumption of independent prefixes (both in U-FIB and in FIB). Without prefix dependencies, the nature of the problem is more related to online ski rental and technically different: achieving a constant competitive ratio is simple, but what the optimal constant is remains an open question. The authors present a 3.603-competitive solution.

On the practical side, we envision our algorithms to be operated in architectures such as [23], [24], [25]. For details on these architectures and prototype implementations, we refer the reader to [24].

Bibliographic Note. Earlier versions of this paper were presented at the IEEE ICDCS 2014 [3] and the IEEE Globecom 2014 [26] conferences.

C. Our Contribution

Forwarding Information Bases (FIB) in Internet routers can be compressed (or aggregated) to at least half of their original size, as shown by previous studies [8]. However, the permanent stream of updates to the FIB due to routing updates complicates FIB aggregation in practice: keeping an optimally or near-optimally aggregated FIB in face of these routing updates is algorithmically challenging. Thus, a sensible trade-off has to be found between the aggregation gain and the complexity of handling routing updates.

This paper presents the first formal study of the tradeoff between FIB compression and update churn under dependent prefixes. After providing some empirical insights on the spatial and temporal locality of routing updates, revealing opportunities for improving this trade-off in online FIB aggregation, we present the online algorithm HIMS (HIDE INVISIBLE AND MERGE SIBLING). HIMS achieves an asymptotically optimal competitive ratio for a natural class of algorithms based on *sticks*. Sticks capture the subset of prefixes that are subject to optimization without violating forwarding correctness. HIMS (1) removes unnecessary and "invisible" prefixes from the FIB, and (2) merges FIB prefixes that are forwarded to the same port and describe adjacent IP address spaces.

We prove that HIMS achieves a competitive ratio of $O(w)$; the performance is hence independent of the update cost α (which need not to be a constant). Furthermore, we derive a tight lower bound of $\Omega(w)$ on the competitive (resp. approximation) ratio of stick-based online (resp. offline) algorithms.

D. Organization

The remainder of this paper is organized as follows. In Section II, we present an empirical study of FIB locality and churn, motivating our work. Our competitive algorithm is described in Section III and analyzed in Section IV. We describe how to implement our algorithm in Section V. Section VI concludes our contribution and discusses future work.

II. LOCALITY AND CHURN: AN EMPIRICAL MOTIVATION

Before presenting our algorithmic approach, we take a closer look at the locality properties of FIB churn. These properties are the empirical motivation for our dynamic FIB aggregation algorithms.

A. Spatial and Temporal Locality of FIB Aggregation

In order to study FIB locality properties, also over time, we use the following simple methodology. First, we split the FIB in its usual trie representation into subtrees, which are aggregated only when they are considered stable. When a subtree has not been affected by updates for a pre-defined time period (β seconds), we use ORTC [8] to optimally aggregate the subtree. Then, on routing updates, the subtree is reverted to its original (disaggregated) representation before the update is applied. We consider real BGP update streams and identify the trade-offs associated with the parameters γ (spatial locality) and β (temporal locality). In all of our simulations, we verify that FIB and U-FIB are equivalent.

Concretely, in our methodology, the tree is split horizontally into two parts. The upper part, which we call TOP, contains the less specific prefixes and remains untouched. Hence, as the TOP is not subject to aggregation, routing updates to the TOP can be applied immediately as they come (one update to the TOP results in one update to the FIB). The lower part contains the more specific prefixes and is aggregated selectively. The parameter γ defines at which depth (prefix length) to draw the line that separates the TOP from the more specific part of the tree. All prefixes with a prefix length $\geq \gamma$ belong to the more specific part.

The more specific part of the tree is split vertically into subtrees, called BOTTOMS. All the nodes with prefix length $= \gamma$ represent root nodes of individual BOTTOMS. A BOTTOM which has not seen any updates for a predefined time period is aggregated using the ORTC algorithm. We aggregate BOTTOMS independently from the TOP: no next-hop information, which can change over time, is being inherited from the TOP when aggregating a BOTTOM. When forwarding packets, however, BOTTOMS do depend on next-hop information from the TOP in case of holes in a BOTTOMS address space. To handle this correctly, we rely on a variant of ORTC that ensures full congruency between U-FIB and FIB, meaning that all holes in a BOTTOM's address space are retained to allow the TOP's entries to determine the next-hop for a packet.

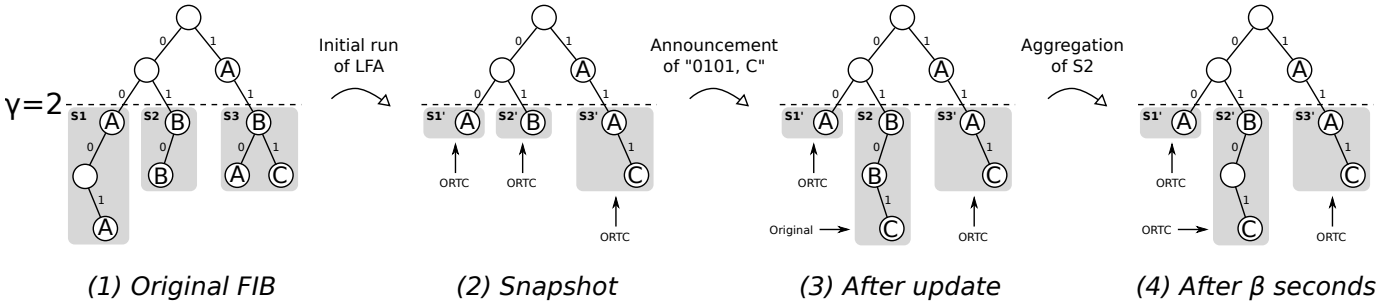


Fig. 2. Locality-aware FIB Aggregation (LFA) Methodology.

The parameter β specifies the time in number of seconds after which a BOTTOM is aggregated in the absence of updates. For each BOTTOM, a timestamp is maintained that indicates the time of its most recent update. Upon incoming updates to a BOTTOM, we distinguish two cases:

1) BOTTOM *aggregated:* In case the affected BOTTOM is aggregated, the BOTTOM is reverted to its non-aggregated (untouched) version prior to applying the update.
2) BOTTOM *untouched:* Updates are applied as-is to non-aggregated BOTTOMS.

In both cases, the BOTTOM's update timestamp is set to the time of the update. A priority queue maintains pointers to each untouched BOTTOM, ordered by the time of the most recent update. A timer keeps track of the tail of the queue and aggregation is applied to those BOTTOMS that have an update timestamp $\leq$ current time - β.

Figure 2 illustrates the components of our methodology for $\gamma = 2$. The trie represents a FIB, trie levels represent prefix lengths starting at zero, and letters represent ports. Empty nodes do not have a corresponding entry in the FIB. The leftmost figure (Figure 2 (1)) highlights how γ is used to separate the TOP from the BOTTOMS S1 to S3. Initially (see Figure 2 (2)), all BOTTOMS are aggregated using ORTC while reducing the total number of prefixes from 8 to 5. In the figures, we append a prime symbol to the BOTTOM identifiers when they are aggregated, hence we now have the BOTTOMS S1' to S3'.

Next, in Figure 2 (3), we consider an update affecting S2'. Prior to applying the update, S2' is reverted to its disaggregated form S2. Then, the update can be applied. In this example the update reflects a prefix announcement which is handled by the insert procedure. Algorithm 1 provides pseudo-code for the insert procedure used in our methodology. We omit the delete procedure as it is similar to the insert one, except for Lines 2 and 10 where *TrieInsert()* should be replaced with *TrieDelete()*. After S2 remains unchanged for β seconds, S2 is aggregated again in Step (4).

We note, that this approach introduces a certain amount of memory overhead (e.g., the priority queues) and additional computations for performing the aggregation. However, this affects only the route controller (an embedded system in a router or a separate route server), where memory and CPU resources are abundant. The goal of this work (and several related FIB aggregation papers) is to reduce the memory needed on line-cards and to pay for it as little as possible in terms of additional churn, i.e., the amount of updates to the FIBs on the line-cards. This is particularly relevant in the context of Software Defined Networks (SDN), where the latency and capacity of the communication channel between the forwarding device and the route controller can be limited [23].

Algorithm 1 Methodology: Insert(p: prefix, o: next-hop)

```
1: if p < γ then
2:     TrieInsert(p, o)
3: else
4:     S ← Stick(p)
5:     if IsAggregated(S) then
6:         RevertToOriginal(S)
7:     else
8:         Dequeue(S)
9:     end if
10:    TrieInsert(p, o)
11:    SetTimestamp(S)
12:    Enqueue(S)
13: end if
```

B. Analysis of Churn Locality

We have designed our methodology specifically to facilitate studies of the locality of churn in the FIB. More specifically, our methodology allows to (1) quantify the aggregatability of dependency-free[2] regions of the FIB, (2) monitor the locality of churn over time, and (3) study the trade-offs of the parameters γ and β.

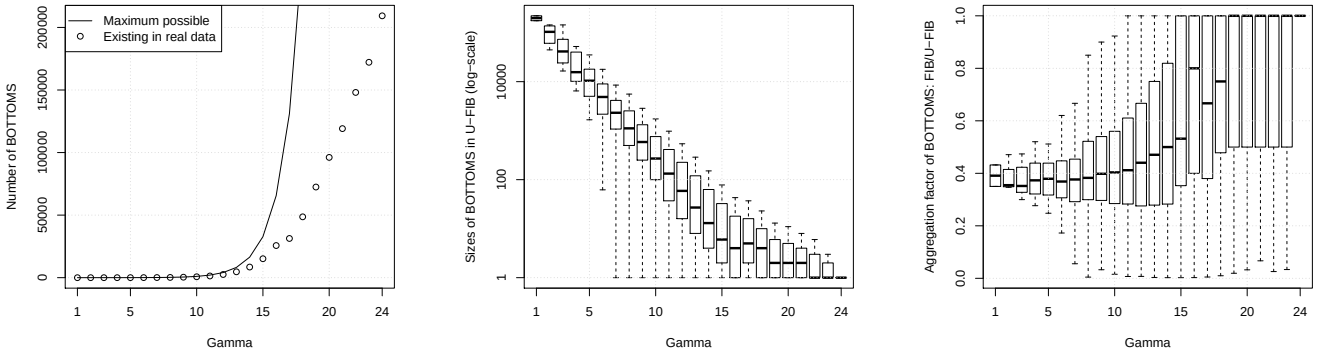
Aggregability of BOTTOMS. We rely on snapshots of real routing tables to study the general aggregability of BOTTOMS and the dependency on γ. We obtained the routing table dumps from RouteViews[3] [1]. Due to space limitations and because the results are similar[4], we present results based on a single routing table snapshot from a large US Internet service provider. This routing table contains almost $400,000$ entries with more than 900 unique next-hop ASes. The high number of next hops can be considered as an upper bound on true number of next hops in FIBs.

At first, in Figure 3(a), we show the number of BOTTOMS as a function of γ. The figure compares the maximum possible

[2]A dependency-free region of a FIB is a group of prefixes that does not have more specifics, but less specifics may (and typically do) exist.

[3]Due to limitations in the data we approximate ports with next-hop ASes.

[4]We ran our analysis on about 30 routing table dumps from each year between 2009 to 2012 and observed similar results.



(a) Number of existing BOTTOMS as a function of γ.

(b) Distribution of BOTTOM sizes in U-FIB as a function of γ.

(c) Per-BOTTOM aggregation gain as a function of γ.

Fig. 3. A first look at the impact of γ.

number of BOTTOMS for a given γ with the number of existing BOTTOMS in our snapshot. Figure 3(a) shows that the number of existing BOTTOMS is substantially smaller than the maximum possible. This means that despite the near exhaustion of the current IPv4 address space, IPv4 FIBs are sparsely populated in terms of their filling of the tree data structure.

Figure 3(b) shows the impact of γ on the distribution of U-FIB BOTTOM sizes, i.e., the numbers of prefixes in non-aggregated BOTTOMS. We observe that both the average and the maximum BOTTOM size decreases as γ increases. For values of γ larger than 7, the minimum BOTTOM size goes to 1, indicating that at least one BOTTOM contains no more than a single prefix. Figure 3(c) shows the per-BOTTOM aggregation factor as a function of γ. For $\gamma \leq 15$, BOTTOMS can be aggregated to half their original size, while larger values of γ result in worse aggregation factors. We observe a non-monotonic behavior in Figure 3(c) for $\gamma \geq 16$. This is a result of the strong dependency of ORTC on the structure of a BOTTOM for the efficiency of its aggregation.

We conclude that values of $\gamma \leq 15$ will lead to good aggregation factors without incurring a high overhead for tracking and keeping the state of large numbers of BOTTOMS, while at the same time achieving median BOTTOM sizes of more than one.

With Figure 4, we complete our routing table snapshot analysis. Figure 4 shows, as a function of γ, the total number of prefixes in the FIB. We further decompose the FIB size into its TOP and BOTTOM components. For $\gamma \leq 15$, the TOP contributes only limited numbers of prefixes while the prefixes from the BOTTOM components dominate the total size of the FIB, which is more than 60% off of the size of the U-FIB. This is consistent with our results in Figure 3(c), in which we show that the aggregation gain suffers when γ grows beyond 15. Furthermore, we observe a steep increase in the size of TOP for $\gamma \geq 20$. At the same time, we see limited aggregation possibilities of BOTTOMS. As a result, the total size of the FIB grows until it reaches the size of the U-FIB, see the dashed line on Figure 4.

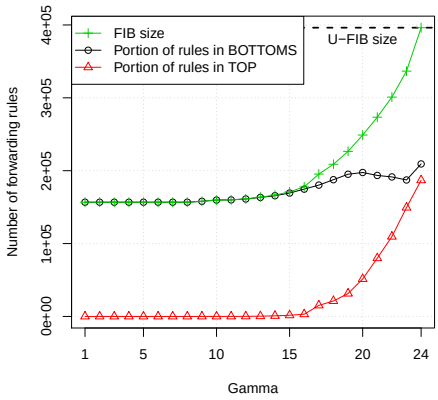
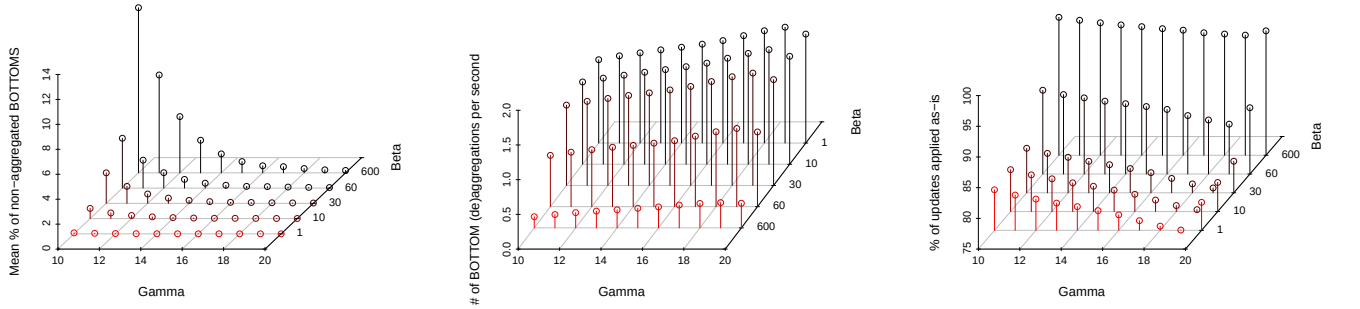


Fig. 4. Size of aggregated BOTTOMS and TOP as a function of γ.

In summary, based on our analysis, a sensible region of γ in current IPv4 routing tables appears to lie below 16. The results in Figure 4 are particularly encouraging as they show that even for γ up to 18 the total size of the FIB can be reduced by at least 50%. This supports the approach of aggregating BOTTOMS individually, as the achieved aggregation factors are close to those from optimal aggregation of the entire FIB [28], [8].

Trade-offs Over Time. Now that we have expectations from our analysis about the impact of γ on the achieved aggregation factors, we analyze the sensitivity for various γ and β. For that, we take a single dataset retrieved from a Canadian ISP router that contains more than 400,000 routing table entries. We obtain the routing table snapshot along with a stream of more than 400,000 BGP updates which covers a period of seven hours. This router has almost 200 unique next-hop ASes. We verified that the results presented are similar to those from different routers on different days. In the following results, we consider values of γ ranging from 10 to 20, and values of β of 1, 10, 30, 60, and 600 seconds. We chose these values of β because they capture the scales at which BGP routing events take place [9].

In Figure 5(a), we show the fraction of BOTTOMS that are non-aggregated over time. This is a particularly important



(a) Percentage of non-aggregated BOTTOMS.

(b) Number of BOTTOM (dis)aggregations.

(c) Percentage of routing updates that are applied as-is.

Fig. 5. Locality trade-offs with γ and β.

metric to consider as it provides intuition about the locality of routing table updates. Non-aggregated BOTTOMS represent those that have seen updates within the last β seconds. The results indicate, that for $\gamma \geq 14$ and $\beta \leq 60s$ the fraction of non-aggregated BOTTOMS is very low. On average, less than 0.4% of the BOTTOMS are not aggregated (inferred by further data inspections).

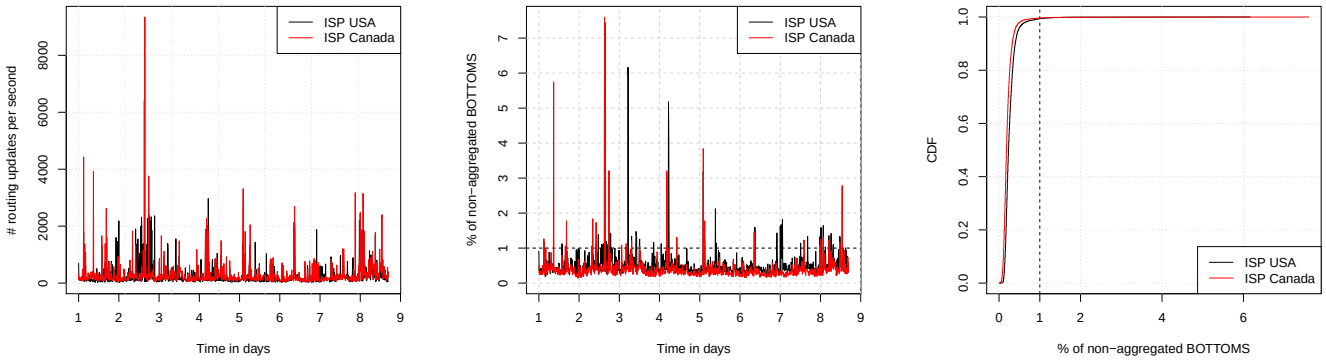
Another metric to consider is the number of (dis)aggregations of BOTTOMS over time. This metric tells us how often updates hit aggregated BOTTOMS, requiring to disaggregate them before applying the update, and how often BOTTOMS are aggregated after a stable period of β seconds. In Figure 5(b) we show the average number of BOTTOM (dis)aggregations per second as a function of γ and β. For improved visual presentation we reverted the ordering of values on the y-axis. The results show that even for a value of β as small as $1s$, the average number of BOTTOM (dis)aggregations per second does not exceed 3. We also observe that this metric strongly depends on β as the results show a steep increase when considering β from $600s$ to $1s$.

Finally, we study the impact of γ and β on the fraction of routing table updates which can be applied as they come. This includes all routing table updates that affect either the TOP, or non-aggregated BOTTOMS. Figure 5(c) shows the fraction of such routing table updates as a function of the parameters γ and β. We observe that as β decreases, this fraction also decreases. This is expected since smaller values of β limit the ability to leverage update locality over time. On the other hand, the behavior of γ is non-trivial. As γ increases, the TOP increases, while non-aggregated BOTTOMS decrease (Figure 5(a)). The net effect we observe is a decrease of the number of updates that can be applied as-is. This happens because the number of updates to the TOP increases very slowly with γ, while the fraction of non-aggregated BOTTOMS decreases much faster with γ. The reason for this behavior is that smaller BOTTOMS have a higher likelihood of being aggregated, as they are less likely to be affected by routing updates.

A Sensible Trade-Off. We now combine the insights from our earlier results and extract the most sensible trade-off in the selection of γ and β. Our results suggest that γ should not be larger than 15 to achieve good aggregation gains. The results from our online experiments suggest that γ should be ≥ 14 to maintain a low number of non-aggregated BOTTOMS for $\beta \leq 60s$. For $\gamma = 14$, Figure 5(a) suggests that β should be no larger than $60s$, while Figures 5(b) and 5(c) show benefits in choosing a large value of β. In summary, our analyses indicate that the most appropriate values are $\gamma = 15$ and $\beta = 60s$, under the churn in the studied Internet routing tables.

While our results, which we verified using routing table data from multiple vantage points and years, provide a surprisingly clear suggestion for the choice of values for γ and β, this work poses the question of how to algorithmically adapt these parameters over time.

Performance Over Time. To better understand the properties over time, with $\gamma = 15$ and $\beta = 60s$, we now perform experiments based on more than one week worth of routing table updates. The results are shown in Figure 6 for two ISP routers, one from Canada and one from the USA. We plot the workload in Figure 6(a) as the time-series of the number of BGP updates per second. We show the maximum value for every 10 minute time interval to stress how bursty BGP updates can be. We notice several routing events which cause more than 2,000 routing table updates per second. In Figure 6(b), we plot the corresponding fraction of non-aggregated BOTTOMS over time. Again, to give importance to the high values, we show the maximum out of every 10 minute time bin. The auto-correlation (not shown) between the original time-series used in Figures 6(a) and 6(b) shows the impact of β: We observe a strong correlation within time lags of 60, while larger time lags show a much smaller correlation. Finally, we show in Figure 6(c) the CDF of the fractions of non-aggregated BOTTOMS in one second time intervals. Contrary to Figures 6(a) and 6(b) that show maximum values over 10 minute bins, Figure 6(c) provides a representative perspective on the ability of keeping most of the FIB compressed over



(a) Input data: number of U-FIB updates per second (max of every 10 minute bin).

(b) Percentage of non-aggregated BOTTOMS per second (max of every 10 minute bin).

(c) Distribution of percentages of non-aggregated BOTTOMS per second.

Fig. 6. Performance over time.

time. In more than 99% of the one second time intervals, for both routers, less than 1% of the BOTTOMS are non-aggregated. We are therefore able to leverage the locality in how the updates affect the FIB structure, by keeping most of it compressed.

Putting it all together. Our results show that there is strong locality in the routing table updates with respect to their spatial and temporal properties. This locality should be exploited by FIB aggregation algorithms, even under the bursts of BGP routing updates.

III. Aggregation Algorithm HIMS

With the empirical motivation in mind, and given our insights into the locality and churn of FIB aggregation, we now introduce the formal model and present a competitive algorithm. In particular, our algorithm will revolve around the concept of *sticks*, as introduced in the following. In particular, sticks can be seen as a natural generalization of the *bottom* and *top* concepts used in our empirical study, allowing us to naturally leverage locality. We start with some basic definitions.

A. Basic Concepts

Recall that we we represent both the U-FIB and the FIB as one-bit tries containing all the prefixes from the forwarding rules. Each node of the tree (corresponding to some prefix p) has an associated *color* c if there is a forwarding rule (p, c); a node without any associated color is called *blank*. We identify nodes with the prefixes they represent and with the address ranges their prefix implies. In particular, we call two nodes *adjacent* if the address ranges they cover are adjacent.

We call a node v a U-FIB (FIB) *rule* if it is colored in the U-FIB (in the FIB). For any node v (also a blank one), we denote its *least colored ancestor* (the ancestor farthest from the root) in the U-FIB and in the FIB by $\text{lca}_\text{U}(v)$ and $\text{lca}_\text{F}(v)$, respectively.

We assume that each non-leaf node has exactly two children. We call a non-root node *left (right)* if it is a left (resp. right) son of its parent. For U-FIB, we assume minimal tries, i.e., tries without blank sibling leaves (they may contain blank leaves, though). In the trie representation of the FIB, minimality is not assumed.

Color Determination and Superfluous Nodes. Observe that the coloring of the U-FIB (FIB) implies the coloring of the whole address space $[0, 2^w - 1]$: each address has the color of the prefix that would be applied as a forwarding rule. We say that such a node v *determines* the color of address j in the U-FIB (FIB). Unlike in the U-FIB, the node that determines the color of a given address in the FIB may change with time. For succinctness of the description, we slightly extend the address space, incorporating two blank addresses -1 and 2^w.

We call a U-FIB rule that does not determine the color of any address *superfluous*. As the color changes of superfluous nodes can be ignored, without loss of generality, we may assume that the U-FIB does not contain any such nodes: they can be removed from the FIB by an algorithm at the very beginning and at constant cost. We hence have the following property.

Observation III.1. *Any input event (i.e., a color change) changes the coloring of the address space, and hence any algorithm has to react by modifying the* FIB *and paying at least* α.

Taking Dependencies into Account. Keeping dependent prefixes in the FIB is crucial to achieve a good competitive ratio. It is tempting to consider a simple online algorithm that just observes the coloring of the address space induced by the current U-FIB rules, and tries to reflect that state using independent prefixes in the FIB. Unfortunately, it appears that no such algorithm can achieve a non-trivial competitive ratio.

Lemma III.2. *Let* ALG *be a (possibly offline) algorithm that never keeps any dependent prefixes in its* FIB. *Then, there exists an input sequence, for which* $\text{COST}(\text{ALG}) = \Omega(2^w) \cdot \text{COST}(\text{OPT})$.

Proof: Consider a U-FIB that is represented by a full

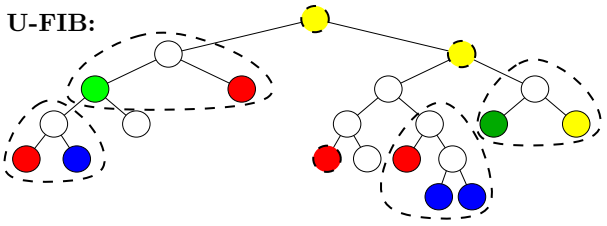


Fig. 7. Partition of the U-FIB into sticks. Superfluous nodes are already removed. Stick boundaries are marked with dashed lines.

binary tree of height w, with root colored initially green and each second leaf colored black. All other nodes are blank. Now the input sequence contains changes of the root color from green to red and back to green. By simply copying the state of the U-FIB to its FIB, OPT pays α for each change to the U-FIB. On the other hand, each change of the U-FIB induces $\Omega(2^w)$ changes to the coloring of the address space. To reflect these changes in its FIB, each time ALG has to pay $\Omega(2^w) \cdot \alpha$. Hence, U-COST(ALG) $= \Omega(2^w) \cdot$ U-COST(OPT). This relation implies the desired bound, since by frequently changing the U-FIB the adversary renders the memory costs negligible. ■

B. Sticks and Active Nodes

Sticks emerge after removing superfluous nodes, and naturally decompose the U-FIB into multiple node groups. The stick decomposition remains invariant throughout the runtime of the algorithm. Each stick is a maximal subtree (where leaves of this subtree may be U-FIB internal nodes), such that all subtree leaves are colored and all its internal nodes are blank, cf. Figure 7.

To this end, we first group all rules of the U-FIB into sets $L_1, L_2, L_3, \ldots$. Each L_i is a maximal (in terms of cardinality) set of colored nodes corresponding to adjacent address ranges, whose union is a range that can be represented by a single node v_i. In other words, if all these nodes were of the same color c, they could be compressed into a single node v_i of color c. Note that this partitioning does not depend on the order in which we gather nodes into sets L_i.

A stick S_i is then defined to contain all nodes "between v_i and L_i inclusively": all nodes in the tree rooted at v_i that are either in L_i or are ancestors of L_i. L_i, $S_i \setminus L_i$, and v_i are called the leaves, the internal nodes and the root of stick S_i, respectively. Note that the stick leaves are not necessarily tree leaves. When L_i is a singleton, S_i is also a singleton and is called a *trivial* stick. As U-FIB does not contain superfluous nodes, all sticks are disjoint and all internal nodes of a stick are blank in the U-FIB. We call nodes that belong to any stick *active*.

Stick Optimizations. Our algorithm will leverage the stick concept to perform locality-aware optimizations. In a nutshell, our algorithm tries to merge nodes of the same color within a single stick. That is, if all the leaves of a single non-trivial stick have the same color (in the U-FIB) for some period of time, then in the FIB they should become blank and the root of the stick should have that color assigned. Furthermore, we perform intra-stick optimizations when possible, e.g., if some adjacent nodes of a single stick are of the same color and can be replaced by a single colored node. However, optimizing the sticks alone may still yield a poor performance. Consider, for example, a U-FIB containing a non-superfluous red root (being a trivial stick) and many non-adjacent red leaves, all being trivial sticks. In this example, the red nodes below the root are "invisible", i.e., they could be deleted without changing the correctness of the forwarding table. The optimally compressed FIB contains only the root.

C. The Algorithm

For our algorithm, we need to define, for any active node, two counters that depend on time and the coloring of the U-FIB. Consider any (active) node u belonging to some stick S. If u is a leaf of S, then let $L(u) = \{u\}$, otherwise let $L(u)$ contain all leaves of S that are descendants of u. Furthermore, if u is not a root of a stick, $p(u)$ denotes its parent in the trie, otherwise $p(u)$ is undefined.

1) For any node u, the *counter* $C_u(t)$ measures for how long, until time t (uninterruptedly), all nodes of $L(u)$ were of the same color. Hence, for a stick leaf u, $C_u(t)$ simply measures the time since the last change of u's color.
2) The second counter is used to hide invisible nodes. Assume that $\text{lca}_\text{U}(u)$ exists. The *counter* $H_u(t)$ measures how long, until time t (without interruption), all nodes of $L(u) \cup \{\text{lca}_\text{U}(u)\}$ were of the same color. When $\text{lca}_\text{U}(u)$ does not exist, $H_u(t) = 0$ for any time t.

Since multiple nodes cannot change colors simultaneously, any color change of a stick leaf u causes the resetting of the C and H counters on the path from u to the root of a stick containing u. Similarly, the color change of $\text{lca}_\text{U}(u)$ resets all H counters from the stick containing u. Note that $C_u(t) \geq H_u(t)$ and, if $p(u)$ is defined, $C_{p(u)}(t) \leq C_u(t)$ and $H_{p(u)}(t) \leq H_u(t)$.

Algorithm Definition. We now present our algorithm HIMS (HIDE INVISIBLE AND MERGE SIBLING). In the FIB of HIMS, the inactive nodes are always blank. For any active node u, HIMS decides whether it should be colored and, if so, with which color.

An active node u is a FIB rule at time t if and only if all the following three conditions hold:

1) $H_u(t) < \alpha$,
2) $C_u(t) \geq \alpha$ or u is a stick leaf,
3) $C_{p(u)}(t) < \alpha$ or u is a stick root.

If u is decided to be FIB rule, then its color is the color of the nodes in $L(u)$. Note that this color is well defined (either u is a leaf and then $L(u)$ is a singleton, or $C_u(t) > 0$).

Example HIMS Execution. To get some understanding of the behavior of HIMS, let us take a look at two extreme cases. For a trivial stick consisting of a single node u, the second and the third condition always hold as u is both a stick leaf and a stick root. Therefore, once u and $\text{lca}_\text{U}(u)$ are of the

same color, the algorithm simply waits until $H_u(t)$ reaches α and then removes (the invisible) u from the FIB. On the other hand, for a stick S that has no colored ancestors in the U-FIB, $H_u(t) = 0$ for any $u \in S$. Thus, the actions of HIMS on S depend only on the C counters inside S. For example, if all the stick leaves in the U-FIB are unicolor for time α, then only the root of S remains present in the FIB of HIMS.

IV. ANALYSIS OF HIMS

We start by presenting the framework of our analysis, while most of the technical details are given in the subsequent sections. To explain the idea behind bounding the memory cost of HIMS, let us consider a specific adversarial strategy: given the state of the U-FIB at time t, the adversary does not change anything for a certain time period. Then, in the time interval $(t, t+\alpha]$, HIMS may perform some optimizations, but after time $t+\alpha$, HIMS will not introduce any further changes to the FIB. Furthermore, it is possible to show that at time $t+\alpha$, the size of the algorithm's FIB is an $O(w)$-approximation of the optimal FIB size.

Clearly, we cannot expect an adversary to behave as described above, as it has many more options. Nevertheless, we can show that if we take the U-FIB and FIB snapshots at any particular time t, then either some compressions were already performed by HIMS or the changes to the U-FIB are quite recent, i.e., they occurred during time interval $(t-\alpha, t]$. In either case, we are able to construct a lower bound on OPT's cost, and hence relate M-COST(HIMS) to COST(OPT).

For a formal proof, we introduce a concept of *rainbow points*. A rainbow point is an address-time pair (a, t), denoting that at time t address $a \in [-1, 2^w - 1]$ has a different color than address $a+1$ (where blank is treated as an additional color). We call two rainbow points *different* if their addresses are different. Rainbow points measure the spatio-temporal complexity of the coloring of the address space: even OPT has to represent this coloring by its own FIB and pay for them.

Lemma IV.1. *If k pairwise different rainbow points exist in some time interval I of length α, then* $\text{COST}_I(\text{OPT}) \geq \lceil k/2 \rceil \cdot \alpha$.

Proof: A rainbow point (a, t) is a witness for a rule that had to be present at time t in the OPT's FIB and whose range either ended with address a (at the right) or $a+1$ (at the left). Therefore, k different rainbow points are the witnesses of at least $\lceil k/2 \rceil$ distinct rules that were present in the FIB at some times from I. Any such rule was either present in the FIB throughout the whole interval I or it was inserted or deleted at some time of I. In either case, such a rule contributes α to $\text{COST}_I(\text{OPT})$. ∎

It remains to show how to find sufficiently many rainbow points: We just need to consider the snapshots of HIMS's FIB at certain times. In Section IV-A, we will show the following result.

Lemma IV.2. *Set any time t at which the* FIB *of* HIMS *does not change. There are $\Omega(\text{SIZE}_t(\text{HIMS})/w)$ pairwise different rainbow points in interval $(t-\alpha, t]$.*

Finally, we need to bound the number of updates HIMS performs in the FIB. Using a potential function argument, we map each FIB update either to a change of the U-FIB or to a time period of length at least α this rule spent in the FIB. By ensuring that a single U-FIB update is not mapped to more than $O(w)$ times, we obtain the following result (proven formally in Section IV-B).

Lemma IV.3. *For any input sequence with m color changes,* $\text{U-COST}(\text{HIMS}) = O(\text{M-COST}(\text{HIMS})) + O(w) \cdot m \cdot \alpha + O(\alpha) \cdot \text{SIZE}(\text{U-FIB})$.

Theorem IV.4. HIMS *is $O(w)$-competitive.*

Proof: First, we bound M-COST(HIMS). We partition the entire runtime of the algorithm into disjoint intervals $I_1, I_2, \ldots, I_\ell$ of length α. At any such interval I_j, we identify a time $t_j \in I_j$ at which there is no change in the FIB and the size of the FIB of HIMS is the largest; let $k_j = \text{SIZE}_{t_j}(\text{HIMS})$. Clearly, $\text{M-COST}_{I_j}(\text{HIMS}) \leq k_j \cdot \alpha$.

For any $j \geq 2$, let $r_j = (t_j - \alpha, t_j]$. The number of rainbow points in interval r_j is $\Omega(k_j/w)$ by Lemma IV.2, and thus, by Lemma IV.1, $\text{COST}_{r_j}(\text{OPT}) = \Omega(k_j \cdot \alpha / w)$. The intervals r_j may overlap, however any time belongs to at most two such intervals (as the distance between every second t_j is at least α). Thus, $\sum_{j=2}^{\ell} \text{M-COST}_{I_j}(\text{HIMS}) \leq \sum_{j=2}^{\ell} k_j \cdot \alpha = O(w) \cdot \sum_{j=2}^{\ell} \text{COST}_{r_j}(\text{OPT}) = O(w) \cdot \frac{1}{2} \cdot \text{COST}(\text{OPT})$. Finally, the memory cost in the first interval is at most $k_1 \cdot \alpha \leq \alpha \cdot \text{SIZE}(\text{U-FIB})$, and hence $\text{M-COST}(\text{HIMS}) = O(w) \cdot \text{COST}(\text{OPT}) + \alpha \cdot \text{SIZE}(\text{U-FIB})$.

Now by Lemma IV.3 and Observation III.1, $\text{U-COST}(\text{HIMS}) = O(\text{M-COST}(\text{HIMS})) + O(w) \cdot \text{COST}(\text{OPT}) + O(\alpha) \cdot \text{SIZE}(\text{U-FIB})$. Thus, in total, $\text{COST}(\text{HIMS}) = \text{M-COST}(\text{HIMS}) + \text{U-COST}(\text{HIMS}) = O(w) \cdot \text{COST}(\text{OPT}) + O(\alpha) \cdot \text{SIZE}(\text{U-FIB})$. As the term $O(\alpha) \cdot \text{SIZE}(\text{U-FIB})$ is a constant independent of the input sequence, HIMS is $O(w)$-competitive. ∎

A. Finding Rainbow Points (Proof of Lemma IV.2)

We start with some basic properties of the HIMS algorithm. We call a node u that satisfies the second and the third conditions given in the description of HIMS a (FIB) *semi-rule*. If a semi-rule u satisfies also the first condition, it is clearly a FIB rule, otherwise we call it (FIB) *hidden rule*.

Claim IV.5. *Consider any stick S. For any stick leaf u, let A_u contain all the nodes on the path from u to the stick root. At any time t, A_u contains exactly one semi-rule. If $H_u(t) < \alpha$, then A_u contains exactly one* FIB *rule.*

Proof: Consider the sequence $u_1 = u, u_2, \ldots, u_s$ of all nodes of A_u sorted from the leaf u of S to the root of S. The first part of the lemma follows in a straightforward manner by observing that the values of $C_{u_i}(t)$ are non-increasing with i. As the $H_{u_i}(t)$ values are also non-increasing with i, $H_u(t) < \alpha$ implies that $H_{u_i}(t) < \alpha$ for any i, and therefore the only semi-rule in A_u is in fact a rule. ∎

Claim IV.6. *Fix time t. Fix a stick leaf node u that changes color in the* U-FIB *at time $t' \in (t-\alpha, t)$. Fix an address a*

contained in the range of u. If at time t, in the FIB *of* HIMS*, the color of a is determined by u or its ancestor, then in the* U-FIB *the color of a is determined by u.*

Proof: For the sake of contradiction, assume that in the U-FIB there are descendants $u_1, u_2, \ldots, u_s$ of u containing a. Let S_1 be the stick containing u_1; clearly, u belongs to a different stick than S_1. As u changes color at t', $H_{u_1}(t') = 0$, and therefore $H_{u_1}(t) \leq t - t' < \alpha$. Then, by Claim IV.5, either u_1 or one of its ancestors from S_1 is present as a colored node in the FIB. As such a node lies below u in the FIB trie, neither u nor any of its ancestors can determine the color of a in the FIB at time t. ∎

Relating pairs of FIB rules to rainbow points. The following lemma captures the core properties of the optimizations performed by HIMS. It states that if at some time the FIB contains two "neighboring" nodes, then either they cannot be aggregated by HIMS at all, or it was not possible to aggregate them in the nearest past. In either case, we provide a witness (a rainbow point).

Lemma IV.7. *Fix any time t at which the* FIB *of* HIMS *does not change and an address a. Assume that at time t the colors of a and $a+1$ are determined in the* FIB *by two distinct rules u and v, respectively. Assume that one of the following three cases occurs: (i) u and v are siblings; (ii) u is a left node and v is its ancestor; (iii) v is a right node and u is its ancestor. Then, there exists a rainbow point (a, t'), for some $t' \in (t-\alpha, t]$.*

Proof: We assume that addresses a and $a+1$ have the same color c at time t, as otherwise they would immediately constitute the rainbow point (a, t). We consider the three cases listed in the lemma assumptions and show that in either case U-FIB contains a stick leaf x that changes color at a time $t_c \in (t-\alpha, t)$, such that x is either u, or v, or their descendant, and contains either a or $a+1$ (but not both).

1) If u and v are siblings, then — by the way we defined sticks — they belong to the same stick. This implies that their parent p also belongs to the same stick. Hence, by the definition of HIMS, $C_p(t) < \alpha$. This means that (i) at time t, all nodes from $L(p) = L(u) \cup L(v)$ are of the same color c, (ii) at time $t_c = t - C_p(t)$ one of these nodes changed its color from c'; let x be this node. Without loss of generality, assume that $x \in L(u)$. This implies that $C_u(t) = C_p(t) < \alpha$. As u is a FIB rule at time t, u has to be a stick leaf and hence $x = u$.
2) If v is an ancestor of u, then by Claim IV.5, they belong to different sticks. Let $v' = \text{lca}_{\text{U}}(u)$. Node v' is either equal to v or is its descendant. Note that v' is a stick leaf. As u is a left node, v' contains the address $a+1$. As u is a FIB rule at time t, $H_u(t) < \alpha$, i.e., there is a node $x \in L(u) \cup \{v'\}$ that changed color at time $t_c = t - H_u(t) \in (t-\alpha, t)$. It remains to show that x fulfills our requirements. It is clearly the case when $L(u) = \{u\}$, because then x can be then either v' or u. Assume now that $L(u)$ is not a singleton set, i.e., u is not a stick leaf. As u is a FIB rule at time t, $C_u(t) \geq \alpha$, which implies that no node from $L(u)$ changed the color during the time interval $(t-\alpha, t)$. Thus, in this case $x = v'$.
3) If u is an ancestor of v, the argument is symmetric to the previous case.

By Claim IV.6, x determines the color either of a or $a+1$ in the U-FIB. Without loss of generality, assume that it determines the color of a and that at time $t_c > t-\alpha$ it changes color from c' to c. This means that there exists a sufficiently small $\epsilon > 0$, such that $(t_c - \epsilon, t_c + \epsilon) \subset (t-\alpha, t)$ where a has color c' in time interval $(t_c - \epsilon, t_c)$ and color c in $(t_c, t_c + \epsilon)$. As the color of $a+1$ is not determined by node x and there are no simultaneous changes of colors, there is a time $t' \in (t_c - \epsilon, t + \epsilon) \subset (t-\alpha, t)$ when addresses a and $a+1$ have different colors: (a, t') is our desired rainbow point. ∎

Relating global FIB state to rainbow points. To show Lemma IV.2, we fix time t and perform the following grouping of the leaves of the FIB of HIMS. We sweep the leaves from left to right, partitioning them into groups $G_1, G_2, G_3, \ldots$ In the grouping process, we put two consecutive leaves u, v (possibly representing non-adjacent address ranges) into the same group G_i when either (i) both u and v are left nodes and v is a descendant of the right sibling of u, or (ii) both u and v are right nodes and u is a descendant of the left sibling of v. In the former case, we call a group *left*, in the latter — *right*. (Note that a group can consist of a single leaf only if the orientation of leaves change, but can also have up to w many members.)

Claim IV.8. *If there are h groups of leaves in the* FIB*, then the number of all* FIB *rules is $O(h \cdot w)$.*

Proof: For any group G_i, we denote by K_i the set of all leaves of G_i plus the union of their (not necessarily colored) ancestors. As each FIB rule is in at least one set K_i, it is sufficient to show that the number of elements of any set K_i is at most $O(w)$. Recall that, by the definition of G_i, K_i has at most one leaf on each level; let ℓ be the maximal such level. It suffices to show that there is exactly one ancestor on each of the levels $0, 1, \ldots, \ell - 1$. Such a claim follows by a simple backward induction on the levels. Level $\ell-1$ contains exactly one ancestor being the parent of the ℓ-th level leaf of K_i. Now fix level $j < \ell - 1$. Note that level $j+1$ contains a single ancestor (by the inductive assumption) and possibly a leaf, and these nodes are siblings (by the construction of the groups). Hence these nodes of K_i have a single parent at level j, which concludes the proof of the claim. ∎

We are now ready to prove Lemma IV.2, i.e., the relation between the number of FIB entries at time t and the number of different rainbow points in interval $(t-\alpha, t]$.

Proof of Lemma IV.2: Let $I = (t-\alpha, t]$. We group all the leaves as described above into h groups $G_1, G_2, \ldots, G_h$. For any group G_i, we denote the leftmost address covered by a leaf from G_i by a_i and its rightmost one by b_i. By Claim IV.8, it is sufficient to show that the number of rainbow points is $\Omega(h)$. If $h < 4$, then we simply consider the last colored address, b_h: as $b_h + 1$ is blank, (b_h, t) is a rainbow point and the lemma follows.

In the following, we thus assume that $h \geq 4$. We focus on a consecutive pair of groups G_i and G_{i+1}, such that at least one of the conditions hold: (i) G_i is a left group, (ii) G_{i+1} is a right group. Note that among all $h-1$ pairs of consecutive groups, at least every second pair (i.e., at least $(h-2)/2 = \Omega(h)$ pairs) has this property. Thus, it remains to show that for such a pair of groups, we may find a unique rainbow point in I.

We denote the rightmost leaf of G_i by v_i and the leftmost leaf of G_{i+1} by v_{i+1}. Without loss of generality, we can assume that G_i is a left group, which means that v_i is a left node. (The case when G_{i+1} is a right group is symmetric, i.e., we start our construction with v_{i+1} and we reverse the roles of left and right nodes). If $b_i + 1$ is blank, then (b_i, t) is our rainbow point; otherwise let u_i be the node that determines the color of the address b_i+1. We consider three cases depending on the relation between the levels (i.e., depth in the trie) of u_i and v_i, henceforth referred to by $\text{lev}(u_i)$ and $\text{lev}(v_i)$, respectively.

1) $\text{lev}(u_i) < \text{lev}(v_i)$. As v_i is a left node, the address ranges of v_i and u_i cannot be adjacent, and therefore u_i is an ancestor of v_i. By Lemma IV.7, there exists a rainbow point (b_i, t_i), where $t_i \in I$.
2) $\text{lev}(u_i) = \text{lev}(v_i)$. Then, u_i is the right sibling of v_i. By Lemma IV.7, there exists a rainbow point (b_i, t_i), where $t_i \in I$.
3) $\text{lev}(u_i) > \text{lev}(v_i)$. Then, u_i is a left node, whose leftmost address is $b_i + 1$. Note that u_i cannot be a FIB leaf as then it would belong to G_i. Furthermore, v_{i+1} is the leftmost leaf of the subtree rooted at u_i, i.e., $\text{lev}(v_{i+1}) > \text{lev}(u_i) > \text{lev}(v_i)$. This implies that v_{i+1} has to be a right node as otherwise it would belong to G_i. Furthermore, v_{i+1} has an ancestor (node u_i) in the FIB. Let $u_{i+1} = \text{lca}_\text{F}(v_{i+1})$ (it can be either u_i or some of its descendants). As v_{i+1} is a right node and is the leftmost leaf of u_{i+1}, node u_{i+1} determines the color of $a_{i+1} - 1$. Hence, by Lemma IV.7, there exists a rainbow point $(a_{i+1} - 1, t_i)$, where $t_i \in I$.

■

B. Bounding the Update Cost (Proof of Lemma IV.3)

We bound the update cost over an input sequence using amortized analysis. For any node u, we define its potential at time t as

$$F_u(t) = \begin{cases} 5\alpha + 2 \cdot \min\{H_u(t), \alpha\} & \text{if } u \text{ is a FIB rule} \\ 6\alpha & \text{if } u \text{ is a FIB hidden rule} \\ 0 & \text{otherwise} \end{cases}$$

Let the total potential at time t be defined as $\Phi(t) = \sum_u F_u(t)$, where we sum over all (active) nodes from the FIB trie.

In this section, we abuse the notation, and use U-COST(HIMS) to denote the *amortized* cost of its updates, defined as the actual cost plus the change in the potential. We show how to bound this amount in all possible cases.

Lemma IV.9. *For any time interval $I = (t_0, t_1)$ with no updates to the* FIB*, it holds that* $\text{U-COST}_I(\text{HIMS}) \leq 2 \cdot \text{M-COST}_I(\text{HIMS})$.

Proof: There is no actual update cost. The increase of the total potential is $\Delta\Phi = \Phi(t_1) - \Phi(t_0) \leq 2 \cdot (t_1 - t_0) \cdot k$, where k is the number of FIB rules kept within I. Finally, $\text{M-COST}_I(\text{HIMS}) = (t_1 - t_0) \cdot k$, and therefore the lemma follows. ■

Now, we analyze the amortized update cost at any time t when the FIB is updated by HIMS. Such update is caused either by some counters reaching α or by a color change in the U-FIB that resets some counters. For the analysis, we assume that these events occur separately. Namely, we split time t into three stages:

1) In the first stage, HIMS behaves as if there was no U-FIB color update, and processes only the changes caused by some H counters that reached α.
2) In the second stage, HIMS processes the changes induced by some C counters that reached α.
3) In the third stage, HIMS processes a (single) event of a color change in the U-FIB.

We bound U-COST(HIMS) in all stages separately (see the three lemmas below). Note that it is possible that some stages are missing, and furthermore, if more than one stage is present, we possibly overestimate the cost of HIMS (for example, we may charge it for inserting a rule because of a change in C counters and then for its removal because of the U-FIB color change, while in reality HIMS would do nothing with this rule).

Lemma IV.10. *The first stage: Assume that some H counters reach value α. Then,* $\text{U-COST}(\text{HIMS}) \leq 0$.

Proof: If a counter H_u of a node u reaches α then if this node was a rule it becomes a hidden rule. (If it was not a rule, its status remains unchanged.) HIMS removes u from the FIB paying α and the change in the potential is $6\alpha - (5\alpha + 2 \cdot \min\{H_u, \alpha\}) = -\alpha$, i.e., the amortized update cost associated with u is zero. The lemma follows by summing the amortized cost over all affected nodes u. ■

Lemma IV.11. *The second stage: Assume that C counters reach value α. Then,* $\text{U-COST}(\text{HIMS}) \leq 0$.

Proof: By the definition of HIMS, growing C counters can only create a semi-rule that is an ancestor of a previously existing semi-rule. Thus, if a new semi-rule u is created, then $\ell \geq 2$ semi-rules (lying in the subtree rooted at u) are removed. These actions costs at most $(\ell+1) \cdot \alpha$ (the cost might be lower than $(\ell + 1) \cdot \alpha$, because there is no actual cost involved in creating and removing hidden rules). On the other hand, as potentials for semi-rules are always between 5α and 7α, the change in the total potential is at most $7\alpha - \ell \cdot 5\alpha$. Therefore, the amortized cost is at most $(\ell + 1 + 7 - 5\ell) \cdot \alpha = (8 - 4\ell) \cdot \alpha \leq 0$. By applying this reasoning to all newly created semi-rules, the proof follows. ■

Lemma IV.12. *The third stage: Assume a node u changes its color in the* U-FIB*. Then,* $\text{U-COST}(\text{HIMS}) = O(w \cdot \alpha)$.

Proof: By the stick definition, u is a stick leaf. Its color change affects two group of nodes.

First, pick any node v, such that $\text{lca}_\text{U}(v) = u$. Nodes v

with this property are active nodes belonging to sticks that are "immediately below" u, i.e., there are no sticks between u and them. The H counter of any such node v is reset to zero. If v was a hidden rule, then it becomes reinserted to the FIB (as a FIB rule). Otherwise, v remains unchanged. In the former case, the actual cost associated with v is α, while the potential change is $5\alpha + 2 \cdot \min\{H_v, \alpha\} - 6\alpha = -\alpha$. Thus, in total, the amortized cost is zero.

Second, the change of color of u also causes all the C and H counters on the path from u to the root of the stick containing u (inclusively) to be reset to zero. From the definition of HIMS, only the nodes on this path and their children are affected, i.e., only those nodes may be inserted, deleted and have their potential changed. As there are $O(w)$ such nodes and their change in the potential is at most 7α, the total amortized cost is $O(w \cdot \alpha)$. ∎

Now we combine the lemmas above to bound the amortized cost of HIMS updates in the general case.

Proof of Lemma IV.3: Let m be the number of color changes in the input. If we sum the guarantees of Lemma IV.9 to Lemma IV.12, we obtain that the total amortized cost is bounded by $O(\text{M-COST}(\text{HIMS})) + O(w \cdot \alpha \cdot m)$. Finally, we observe that the initial potential is $5\alpha \cdot \text{SIZE}_0(\text{HIMS}) = 5\alpha \cdot \text{SIZE}(\text{U-FIB})$, which contributes a constant $O(\alpha) \cdot \text{SIZE}(\text{U-FIB})$ to the total cost. ∎

C. Lower Bound

We can show that Theorem IV.4 is the best we can hope for, at least for the natural class of so-called *stick-based* algorithms: informally speaking, these algorithms do not create dependent prefixes within a single stick. We will derive an $\Omega(w)$ lower bound for such (online or offline) algorithms.

More formally, we consider the U-FIB without superfluous nodes. Recall that without loss of generality, we may assume that an algorithm removes them at the very beginning. Then, we call an algorithm *stick-based* if (i) it never keeps an inactive node (a node outside of a stick) in the FIB, and (ii) for any two active nodes from a single stick that are in a ancestor-descendant relation, it keeps at most one of them in the FIB. Clearly, HIMS fulfills these properties and is hence an instance of a stick-based algorithm.

Theorem IV.13. *The competitive ratio of any stick-based algorithm* ALG *(even an offline one) is* $\Omega(w)$.

Proof: It is sufficient to consider a U-FIB containing a single stick S with $w+1$ leaves, corresponding to adjacent address ranges of lengths $2^{w-1}, 2^{w-2}, 2^{w-3}, \ldots, 2^2, 2^1, 2^0, 2^0$, i.e. each length except for the last one occurs exactly once. The root of S coincides with the root of the whole trie. The coloring of S is constant: the first w leaves are always black and the last one is red. In this case, ALG has to use at least $w + 1$ entries in the FIB to represent such a U-FIB. On the other hand, OPT could just keep two entries in the FIB: the last red entry from U-FIB of length 2^0 and the black root (of length 2^w). Note that in the long run, the initial update cost of OPT becomes negligible and thus $\text{COST}(\text{ALG}) \geq \text{M-COST}(\text{ALG}) = \Omega(w) \cdot \text{OPT}$. ∎

V. IMPLEMENTING HIMS

In this section, we show that HIMS can be implemented efficiently in the route processor. We will focus on the algorithmic aspects, and refer the reader to existing work [23], [24], [25] on details about possible architectures in which we envision HIMS to execute as well as the performance of prototype implementations (which depends more on the specific hardware architecture).

Our approach consists of two stages. First, we show how to maintain a data structure that keeps the set of all semi-rules at all times. Additionally, for each semi-rule u, we store its color, i.e., the color of leaves of $L(u)$. Second, we show how to augment this data structure, so that at any time we know which of the semi-rules are rules and should be kept in the FIB.

Lemma V.1. *It is possible to maintain the set of semi-rules (in the route processor) using a data structure of size* $O(\text{SIZE}(\text{U-FIB}))$*, so that any sequence of events to* U-FIB *can be processed in time* $O(w)$ *on average.*

Proof: For maintaining the set of all semi-rules, we need to track two types of events: (i) a U-FIB color change may force some C counters to be reset to zero and (ii) some counters may reach the value of α. (Note that H counters are irrelevant for computing semi-rules.) Any such change in the counter of a node v affects the state (i.e., being or not being a semi-rule) of at most three nodes: v itself, and if v is not a stick leaf then also its two children. Thus, for a single event, we may update our semi-rules set in constant time.

We first observe that the number of active nodes (for which we want to track the C counters) is at most twice the number of U-FIB rules. Second, instead of storing counters C_u, we just keep the timestamps of the last reset of C_u to zero. These values are sorted in non-increasing order, kept in a linked queue Q with additional references from and to the nodes of the U-FIB trie. Finally, we store a pointer q to the place in Q that splits Q into two parts: the left one keeping counters strictly smaller than α and the right one with counters that are at least α.

When a counter is reset to zero, it is moved to the front of the list with its timestamp updated to the current time. To track the second type of events, we set an alarm to the time when the first counter would reach α. (This is the counter immediately to the left of q.) When the alarm goes off, we shift q to the left accordingly and set the alarm for the next element. It is possible that many counters, say k, reach α simultaneously, in which case we shift q by k positions. By using a standard amortization argument, we may assign a constant cost to counter resets and zero cost to the event where a counter reaches α.

Finally, we observe that the resetting of counters can only occur when there is a color change (of a stick leaf u) in the U-FIB. Such a change affects at most $w + 1$ counters of the nodes on a path from u to the root of the stick u belongs to. Thus, the total cost of maintaining the structure is at most $O(w)$ times larger than the number of color changes in the input sequence. ∎

To compute the set of rules instead of semi-rules, one could try to keep H counters in a similar data structure. However, the color change of a single node may affect H counters of virtually all possible nodes. (Consider the U-FIB described in Lemma III.2, where the root frequently changes color from red to black, and back.) Below, we show that while sometimes we indeed need to make a lot of updates, their number can be asymptotically bounded by the number of updates to the FIB.

Theorem V.2. HIMS *can be implemented in the route processor using a data structure of size* $O(\text{SIZE}(\text{U-FIB}))$*, so that any sequence of* m_1 *events to the* U-FIB *that entails* m_2 *updates to the* FIB *can be processed in expected time* $O(m_1 \cdot w + m_2)$.

Proof: We want to augment the data structure described in Lemma V.1. From the definition of HIMS, we know that a semi-rule is *not* a FIB rule if and only if $H_u \geq \alpha$, which is equivalent to $C(\text{lca}_{\text{U}}(u)) \geq \alpha$ and the color of $\text{lca}_{\text{U}}(u)$ being the same as the color of u. These conditions can be checked in constant time when the semi-rule is created. When a semi-rule is deleted (and was a rule), it is also removed from the FIB. Below, we show that we may also keep track when a node, which is uninterruptedly a semi-rule, starts or ceases to be a FIB rule.

To this end, for any node v and color c, we keep the set of all semi-rules u of color c, such that $\text{lca}_{\text{U}}(u) = v$. We denote such a set by $P(v, c)$, and we denote the union of these sets over possible colors by $P(v)$. To maintain these sets, for each semi-rule u we keep a bidirectional pointer to $\text{lca}_{\text{U}}(u)$. To keep the memory requirement asymptotically the same as for storing semi-rules only, we just store those sets $P(v, c)$ that are non-empty. This can be achieved by keeping a hashing table for each node v, whose keys are colors c and values are non-empty sets $P(v, c)$.

Now, whenever the counter of v is reset to zero or reaches α, we may easily enumerate those semi-rules (and only them) that need to be added to or removed from the FIB. Namely, when $C_v \geq \alpha$ and v has color b, all semi-rules from $P(v, b)$ are hidden rules and all semi-rules from $P(v) \setminus P(v, b)$ are FIB rules. When $C_v < \alpha$, all semi-rules from $P(v)$ are FIB rules. Therefore, the additional time overhead for modifying the data structure is proportional to the number of FIB updates. ■

VI. SUMMARY AND OPEN QUESTIONS

This paper studied the classical problem of FIB compression, but taking a wider perspective than the state-of-the-art. We considered the situation where forwarding rules can change over time, e.g., due to BGP route updates. Accordingly, we framed FIB compression as an online problem, and designed competitive online algorithms to solve it.

We introduced a formal model, by generalizing several classic online aggregation problems. We designed a *O(w)*-competitive algorithm, where *w* is the length of an IP address. We also derived a lower bound, showing that our result is asymptotically optimal within a natural class of algorithms, based on so-called *sticks*. Finally, we showed that our algorithm can be implemented efficiently in the route processor.

Our work opens several interesting directions for future research. In particular, the optimality of our competitive ratio only holds for a restricted class of algorithms. Generalizing our lower bound or proving that this is not possible would help better understand the nature of the problem. Finally, regarding the design of *offline algorithms*: while it is quite easy to see that under certain circumstances, optimal solutions can be computed in time $f(\alpha) \cdot n^{O(1)}$ where n denotes the number of prefixes and f is a function of α, it remains an open question whether a polynomial time algorithm exists. Another interesting direction for future research regards the extension to more general packet classifications, in *multiple* dimensions: this is a non-trivial generalization of our work.

On the practical front, the field is currently missing a rigorous methodology and benchmark to compare different algorithms, on different hardware platforms. As the time to install FIB rules depends on the forwarding hardware and the software managing it (i.e., the firmware of the routers/switches), the results will be tied to the specific hardware that is used. An example for a methodology that evaluates SDN-enabled switches and real hardware forwarding platforms and that would be comparable to our situation can be found in [23]: as would be the case with our solution in this paper, we found in [23] that features supported by different hardware platforms widely differ, and the FIB management software has very different behavior.

Having that said, our algorithms as discussed in the paper are quite efficient, and come with low time and memory requirements. Moreover, if performance becomes an issue, the algorithms may be scaled to a certain extent, which however depends on the granularity at which FIB updates can be flushed to the hardware. A rigorous study of these aspects constitutes another interesting subject for future research.

Acknowledgments. We thank Robert Wuttke from TU Berlin, Fred Baker from Cisco/IETF and Magnús M. Halldórsson from Reykjavik University for their valuable inputs and feedback. Marcin Bienkowski was partially supported by the Polish National Science Centre grant 2016/22/E/ST6/00499, and Stefan Schmid by the Danish Villum project *ReNet*.

REFERENCES

[1] University of Oregon Route Views Project. http://www.routeviews.org/.
[2] M. Bienkowski, J. Marcinkowski, M. Pacut, S. Schmid, and A. Spyra. Online tree caching. In *Proc. 29th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 329–338, 2017.
[3] M. Bienkowski, N. Sarrar, S. Schmid, and S. Uhlig. Competitive fib aggregation without update churn. In *Proc. 34th Int. Conf. on Distributed Computing Systems (ICDCS)*, pages 607–616, 2014.
[4] M. Bienkowski and S. Schmid. Competitive FIB aggregation for independent prefixes: Online ski rental on the trie. In *Proc. 20th Colloq. on Structural Information and Communication Complexity (SIROCCO)*, pages 92–103, 2013.
[5] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
[6] T. Bu, L. Gao, and D. Towsley. On characterizing BGP routing table growth. *Computer Networks*, 45:45–54, 2004.
[7] L. Cittadini, W. Muhlbauer, S. Uhlig, R. Bush, P. Francois, and O. Maennel. Evolution of internet address space deaggregation: myths and reality. *IEEE Journal on Selected Areas in Communications*, 28:1238–1249, 2010.
[8] R. P. Draves, C. King, S. Venkatachary, and B. D. Zill. Constructing optimal IP routing tables. In *Proc. IEEE INFOCOM*, pages 88–97, 1999.

[9] A. Elmokashfi, A. Kvalbein, and C. Dovrolis. BGP churn evolution: a perspective from the core. *IEEE/ACM Trans. on Networking*, 20(2):571–584, 2012.

[10] E. Karpilovsky, M. Caesar, J. R. amnd Aman Shaikh, and J. E. van der Merwe. Practical network-wide compression of IP routing tables. *IEEE Trans. on Network and Service Management*, 9:446–458, 2012.

[11] K. Kogan, S. I. Nikolenko, P. Eugster, A. Shalimov, and O. Rottenstreich. Fib efficiency in distributed platforms. In *Proc. 24th Int. Conf. on Network Protocols (ICNP)*, pages 1–10, 2016.

[12] K. Kogan, S. I. Nikolenko, O. Rottenstreich, W. Culhane, and P. Eugster. Exploiting order independence for scalable and expressive packet classification. *IEEE/ACM Trans. on Networking*, 24(2):1251–1264, 2016.

[13] J. Li, M. Guidero, Z. Wu, E. Purpus, and T. Ehrenkranz. BGP routing dynamics revisited. *SIGCOMM Computer Communication Review*, 37:5–16, 2007.

[14] Y. Liu, B. Zhang, and L. Wang. Fast incremental FIB aggregation. In *Proc. IEEE INFOCOM*, pages 1213–1221, 2013.

[15] Y. Liu, X. Zhao, K. Nam, L. Wang, and B. Zhang. Incremental forwarding table aggregation. In *Proc. Global Communications Conference (GLOBECOM)*, pages 1–6, 2010.

[16] L. Luo, G. Xie, S. Uhlig, L. Mathy, K. Salamatian, and Y. Xie. Towards TCAM-based scalable virtual routers. In *Proc. 8th Int. Conf. on Emerging Networking Experiments and Technologies (CoNEXT)*, pages 73–84, 2012.

[17] L. Luo, G. Xie, S. Uhlig, L. Mathy, K. Salamatian, and Y. Xie. A trie merging approach with incremental updates for virtual routers. In *Proc. IEEE INFOCOM*, pages 1222–1230, 2013.

[18] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *SIGCOMM Computer Communication Review*, 38:69–74, 2008.

[19] D. Medhi and K. Ramasamy. *Network Routing: Algorithms, Protocols, and Architectures*. Morgan Kaufmann Publishers Inc., 2007.

[20] X. Meng, Z. Xu, B. Zhang, G. Huston, S. Lu, and L. Zhang. Ipv4 address allocation and the bgp routing table evolution. *SIGCOMM Computer Communication Review*, 35(1):71–80, 2005.

[21] M. Moradi, F. Qian, Q. Xu, Z. M. Mao, D. Bethea, and M. K. Reiter. Caesar: High-speed and memory-efficient forwarding engine for future internet architecture. In *Proc. ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pages 171–182, 2015.

[22] G. Rétvári, J. Tapolcai, A. Kőrösi, A. Majdán, and Z. Heszberger. Compressing IP forwarding tables: Towards entropy bounds and beyond. In *Proc. of the ACM SIGCOMM*, pages 111–122, 2013.

[23] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. Moore. OFLOPS: An Open Framework for OpenFlow Switch Evaluation. In *Passive and Active Measurements Conference (PAM)*, pages 85–95, 2012.

[24] N. Sarrar, A. Feldmann, S. Uhlig, R. Sherwood, and X. Huang. Fibium: Towards hardware accelerated software routers. Technical Report 9, Deutsche Telekom Laboratories, An-Institut der Technischen Universitaet Berlin, 2010.

[25] N. Sarrar, S. Uhlig, A. Feldmann, R. Sherwood, and X. Huang. Leveraging zipf's law for traffic offloading. *SIGCOMM Computer Communication Review*, 42(1):16–22, 2012.

[26] N. Sarrar, R. Wuttke, S. Schmid, M. Bienkowski, and S. Uhlig. Leveraging locality for FIB aggregation. In *Proc. Global Communications Conference (GLOBECOM)*, pages 1930–1935, 2014.

[27] S. Suri, T. Sandholm, and P. R. Warkhede. Compressing two-dimensional routing tables. *Algorithmica*, 35(4):287–300, 2003.

[28] 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. of the Int. Conf. on Emerging Networking Experiments and Technologies (CoNEXT)*, page 29, 2011.

[29] F. Valera, I. Van Beijnum, A. Garcıa-Martınez, and M. Bagnulo. *Multipath BGP: motivations and solutions*, pages 238–256. Cambridge University Press, 2011.

[30] X. Zhao, Y. Liu, L. Wang, and B. Zhang. On the aggregatability of router forwarding tables. In *Proc. IEEE INFOCOM*, pages 848–856, 2010.

[31] X. Zhao, D. J. Pacella, and J. Schiller. Routing scalability: an operator's view. *IEEE Journal on Selected Areas in communications*, 28(8):1262–1270, 2010.

Marcin Bienkowski obtained a PhD degree from the University of Paderborn, Germany, where he worked in the Algorithms and the Complexity Theory group. He is currently an associate professor and the head of the Combinatorial Optimization Group at the Computer Science Institute of the University of Wroclaw, Poland. His research interests are focused on online and approximation algorithms, especially for network problems.

Nadi Sarrar obtained a Computer Science PhD from TU Berlin, Germany. His research focuses on Internet protocols, network architectures and data analysis. Currently, Nadi Sarrar is building blockchains and distributed ledger applications at Axoni.

Stefan Schmid is a Professor at University of Vienna, Austria. He received his MSc (2004) and PhD degrees (2008) from ETH Zurich, Switzerland. In 2009, Stefan Schmid was a postdoc at TU Munich and the University of Paderborn, between 2009 and 2015, a senior research scientist at the Telekom Innovations Laboratories (T-Labs) in Berlin, Germany, and afterwards an Associate Professor at Aalborg University. His research interests revolve around the fundamental and algorithmic problems of networked and distributed systems.

Steve Uhlig obtained a PhD degree in Applied Sciences from the University of Louvain, Belgium, in 2004. He is currently the Professor of Networks and Head of the Networks Research group at Queen Mary University of London, UK. His current research interests are focused on Internet measurements, software-defined networking, and content delivery.