Exploiting Locality of Churn for FIB Aggregation

Nadi Sarrar*
Marcin Bienkowski†
Stefan Schmid*
Steve Uhlig‡
Robert Wuttke*

*TU Berlin / Telekom Innovation Labs
†University of Wroclaw
‡Queen Mary, University of London

TU Berlin Technical Report: 2012-12
ISSN: 1436-9915

Exploiting Locality of Churn for FIB Aggregation

Nadi Sarrar*, Marcin Bienkowski†, Stefan Schmid*, Steve Uhlig‡ and Robert Wuttke*
*TU Berlin / Telekom Innovation Labs
Email: {nadi,stefan,robert}@net.t-labs.tu-berlin.de
†University of Wroclaw
Email: mbi@ii.uni.wroc.pl
‡Queen Mary, University of London
Email: steve@eecs.qmul.ac.uk

***Abstract*—Snapshots of the Forwarding Information Base (FIB) in Internet routers can be compressed (or aggregated) to at least half of their original size, as shown by previous studies. In practice however, the permanent stream of updates to the FIB due to routing updates complicates FIB aggregation: keeping an optimally aggregated FIB in face of these routing updates is algorithmically challenging. A sensible trade-off has to be found between the aggregation gain and the number of changes to the aggregated FIB. This paper is the first to investigate whether the spatial and temporal locality properties of updates to the tree-like FIB data structure can be leveraged by online FIB aggregation.**

Our contributions include (a) an empirical study of the locality of updates in public Internet routing data, (b) the specification and simulations of our Locality-aware FIB Aggregation algorithm (LFA), and (c) a competitive analysis that sheds light on the performance of online algorithms under worst-case update streams. Our results show that even a simple algorithm like LFA can effectively exploit the locality of FIB churn to keep low the number of updates to the aggregated FIB, as most FIB updates affect only a small number of regions in the FIB.

TOPIC: Router and switch design.
METHODOLOGY: Optimization models and methods.

I. Introduction

At a high level, Internet routers are built around a route processor which runs routing protocols and makes routing decisions, and a forwarding plane which forwards packets according to the decisions of the route processor. The crucial link between the two components is the *Forwarding Information Base* (FIB), containing the *forwarding rules*. The route processor inserts and deletes FIB entries according to its route computations. The forwarding plane uses the FIB to perform an IP destination lookup on each incoming packet. This requires the FIB to support very fast IP destination lookups so that packets can be forwarded at line-rate. In addition, the FIB needs to support frequent updates to the forwarding rules due to the churn in the BGP routing table. Finally, the number of forwarding rules that a FIB needs to keep is growing over time, putting extra pressure on the FIB memory capacity[1]. To fulfill all these requirements, FIB memory used in modern routers is very expensive and power-hungry. It is also considered a main limiting factor in terms of a router's lifetime [1].

[1]The BGP routing table contains more than $400,000$ entries as of mid-2012, up 15% from just a year ago.

A natural and local solution to mitigate the problem — before possible long-term solutions are deployed — is the *aggregation* (or *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 realized entirely in software [2].

While the compression of the FIB is beneficial in terms of memory, it also entails a potential overhead: As the FIB contents of a router change over time — several hundreds of rules are modified each second on average [3] — , FIB aggregation may lead to a situation where already aggregated FIB entries need to be deaggregated again [4], resulting in additional updates to the aggregated FIB. There is a certain cost associated with each such update as the internal FIB structures have to be updated, delaying the corresponding changes to the forwarding plane. Hence, FIB management strategies, including FIB aggregation algorithms, should aim at limiting as much as possible the number of FIB updates.

Contributions. We present and evaluate the *Locality-aware FIB aggregation algorithm* (LFA). LFA exploits the spatial and temporal locality of churn and aggregates slices of the FIB (STICKS) adaptively to amortize update costs with aggregation gains. We provide an empirical analysis of LFA under publicly available Internet routing data to investigate the trade-offs associated with LFA. Motivated by the observed locality properties, we specify a set of rules to aggregate STICKS in a timed and amortizing manner. We present a rigorous competitive analysis of the obtainable deterministic performance under an arbitrary stream of FIB updates. In particular, in online algorithms jargon, we show that the proposed rules achieve a competitive ratio of $O(w^2)$ where w denotes the trie depth. We are not aware of any other formal analysis of this trade-off in the literature.

The remainder of this paper is organized as follows. We start in Section II with background information and the terminology, a description of our assumptions, and a discussion of the related work. LFA is introduced in Section III where we also report on our empirical results on FIB churn locality. We formally introduce and analyze a second approach for locality-aware aggregation in Section IV. In Section V we discuss further

work and then conclude in Section VI.

Due to space constraints, we only sketch some formal proofs in the paper. A full version of this paper can be found online [5] and will be made available on ArXiv.

II. Terminology, Assumptions, and Related Work

Terminology. 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 *matches* all addresses that have the prefix as their first bits.

We consider an Internet router with a number of network interfaces, or *ports*. A *Forwarding Information Base* (FIB) is a set of *forwarding rules* used by the router for its packet forwarding operations, where each such rule is a *prefix-port* pair (p, c). A *port* in this context represents all information needed for the router to forward IP packets to a given destination. For the presentation, we will sometimes refer to the ports by *next-hops*, or *colors*, i.e., we assume a unique color for each port. For every incoming packet the router performs a destination lookup based on the destination IP address x of the packet. The destination lookup is a *longest prefix match*: Among the forwarding rules $\{(p, c)\}$, the router finds the longest p being a prefix of x, and forwards the packet to port c. If no rule matches, i.e., no route to the destination is known, 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. This FIB 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* [2], i.e., we require that the forwarding and dropping behavior remain the same.

We denote the original set of forwarding rules by OT. The OT is updated according to the routing protocols by the route processor. Prior to downloading the OT into the FIB of the router we apply FIB aggregation: The forwarding rules of the OT are replaced by an equivalent but smaller set, denoted by AT. Hence, the FIB of the router contains the AT.

We assume continuous time; at any time t, a single forwarding rule may change its color. The input is a sequence of such changes called *events*. After a change occurs, the route processor must ensure that the AT is equivalent to the OT. To this end, the route processor may apply updates to the forwarding rules in the FIB. The commands may also be issued at any later point in time (e.g., for delayed FIB aggregation).

Assumptions. We take a pragmatic standpoint and study 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 seems to be a reasonable assumption considering the behavior of the route updates in the current Internet [6].

Related Work. There are known fast algorithms for optimal FIB aggregation of table snapshots, for example the Optimal Routing Table Constructor (ORTC) [7] and others [8]. We rely heavily on ORTC in this paper, as it is provably optimal for FIB aggregation. However, as these algorithms do not support efficient handling of incremental updates, a re-computation of the optimally aggregated FIB on each forwarding rule change is needed. This is computationally expensive and can lead to high churn. There are several papers that 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 [4] and others [9], [2]. 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 AT, nor to the aggregation gain. They also do not consider to use churn locality for their benefit.

III. Empirical Analysis

In this section we study the locality of FIB churn based on real Internet routing data. From our results, we discuss and quantify the potential benefit of FIB aggregation techniques that treat churny regions of the FIB differently to those with limited churn. We propose an online FIB aggregation algorithm called Locality-aware FIB Aggregation (LFA). LFA aims at aggregating stable parts of the FIB while keeping the less stable ones untouched to limit update overhead. We start by describing the LFA algorithm. Then, we present experimental results based on routing table snapshots that provide a first look at the trade-offs that are associated with LFA. Finally, we evaluate the locality of churn under consideration of the trade-offs and parameters of LFA, based on publicly available streams of BGP updates. Our results indicate that there is substantial locality in routing updates which can be exploited by FIB aggregation algorithms.

A. LFA: Locality-aware FIB Aggregation

LFA operates as follows. The FIB in its usual trie representation is split into subtrees (STICKS) which are aggregated only when they are considered stable. This is when a STICK has not been affected by updates for a pre-specified time period (β seconds), the Optimal Routing Table Constructor (ORTC) [7] is used to optimally aggregate this STICK. Before a STICK is updated due to a routing update, it is reverted to its original (deaggregated) representation before the update is applied. We simulate LFA on real BGP update streams and identify the trade-offs associated with its parameters α (spatial locality) and β (temporal locality). In all of our simulations we verify that the AT indeed is equivalent to the OT.

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

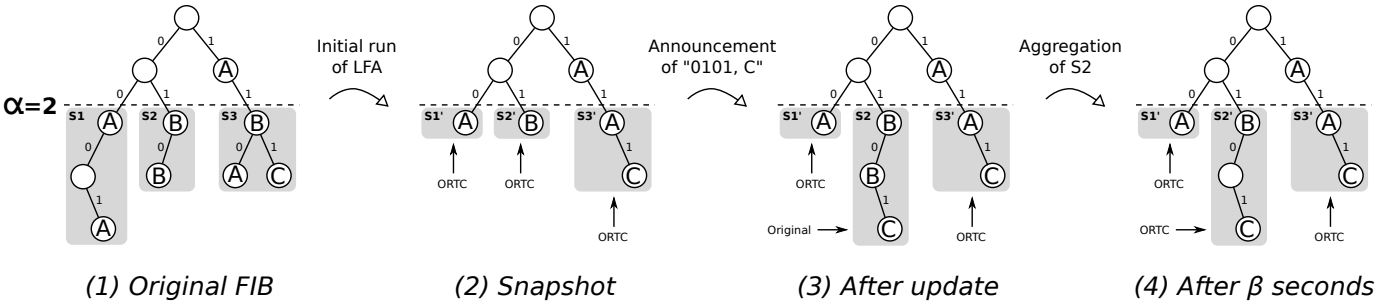


Fig. 1. Locality-aware FIB Aggregation (LFA)

The more specific part of the tree is split vertically into subtrees, called STICKS. All the nodes with prefix length $= \alpha$ represent root nodes of individual STICKS. A STICK which has not seen any updates for a pre-defined time period is aggregated using the ORTC algorithm. In LFA, STICKS are aggregated independently from the GROUND: no next-hop information, which can change over time, is being inherited from the GROUND when aggregating a STICK. Also, as original and aggregated STICKS are congruent in their forwarding information, there is no dependency of the GROUND on the present state of a STICK (aggregated or non-aggregated).

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

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

In both cases, the STICK's last update timestamp is set to the time of the update. A priority queue maintains pointers to each untouched STICK, sorted by the time of their most recent update. A timer is implemented for keeping track of the tail of the queue and aggregation is applied to those STICKS that have a last update timestamp $\leq$ current time - β.

Figure 1 illustrates the algorithmic components of LFA for $\alpha = 2$. The trie represents a FIB, trie levels represent prefix length starting at zero, and letters represent ports. Empty nodes do not have a corresponding entry in the FIB. The first figure highlights how α is used to separate the GROUND from the STICKS S1 to S3. Initially, in (2), all STICKS 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 STICK identifiers when they are aggregated, hence we now have the STICKS S1' to S3'.

Next, in Figure 1 (3), we consider an update that affects S2'. Prior to applying the update, S2' is reverted to its deaggregated form S2. After that, the update can be applied. In this example the update reflects a prefix announcement which is handled by LFA's insert procedure. Algorithm 1 provides pseudo-code for LFA's insert procedure. We leave out the delete procedure as it is similar to the insert one, except lines 2 and 10 call *TrieDelete()* instead of *TrieInsert()*[2]. After S2 remains unchanged for β seconds, S2 is aggregated again in step (4).

Algorithm 1 LFA-Insert(P, N)

```
1:  if P < α then
2:      TrieInsert(P, N)
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, N)
11:     SetTimestamp(S)
12:     Enqueue(S)
13: end if
```

B. Analysis of churn locality

LFA has been especially designed to facilitate studies of the locality of churn in the FIB. More specifically, LFA allows us to (1) quantify the aggregatability of dependency-free[3] regions of the FIB, (2) monitor the locality of churn over time, and (3) study the trade-offs of the parameters α and β of LFA.

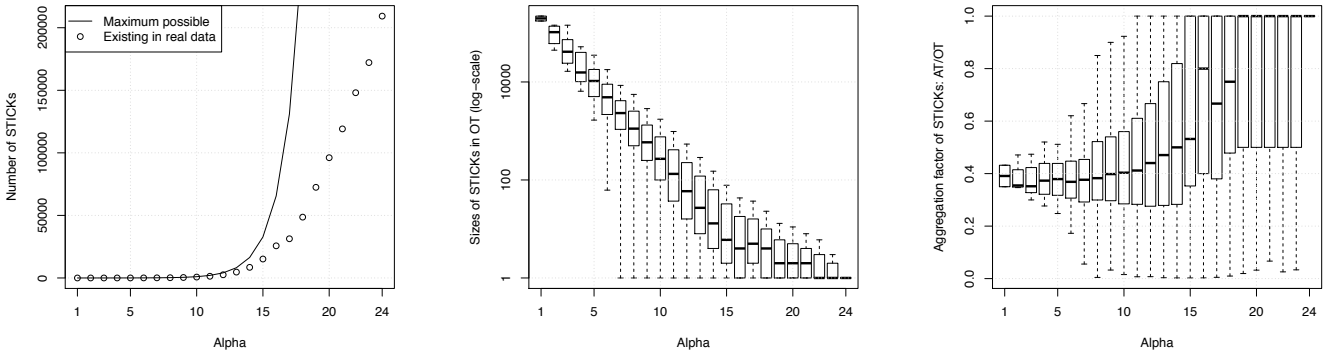
Aggregatability of STICKS. We rely on snapshots of real routing tables to study the general aggregatability of STICKS and the dependency on α. We obtained the routing table dumps from RouteViews[4] [10]. As the results for the different routing tables we analyzed are similar[5], we present results based 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.

[2]We note that real implementations of LFA's insert/delete procedures must be able to instantiate and destroy STICKS, that is when a first prefix of a STICK is announced, or when the only prefix of a STICK is withdrawn, respectively.

[3]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.

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

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



(a) Number of existing STICKS as a function of α. (b) Distribution of STICK sizes in OT as a function of α. (c) Per-STICK aggregation gain as a function of α.

Fig. 2. A first look at the impact of α in LFA.

At first, in Figure 2(a), we show the number of STICKS as a function of α. The figure compares the maximum possible number of STICKS for a given α with the number of existing STICKS in our snapshot. To get the maximum possible number of STICKS, we assume that all STICKS rooted at α have at least one prefix. Figure 2(a) shows that the number of existing STICKS 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 2(b) shows the impact of α on the distribution of OT STICK sizes, i.e., the numbers of prefixes in non-aggregated STICKS. We observe that both the average and the maximum STICK size decreases as α increases. For values of α larger than 7, the minimum STICK size goes to 1, indicating that at least one STICK contains no more than a single prefix. Figure 2(c) shows the per-STICK aggregation factor as a function of α. For $\alpha \leq 15$, STICKS can be aggregated to half of their original size, while bigger values of α result in worse aggregation factors. We observe a non-monotonic behavior in Figure 2(c) for $\alpha \geq 16$. This is a result of the strong dependency of ORTC on the structure of a STICK for the efficiency of its aggregation. This dependency is more visible when STICKS are very small.

We conclude that values of $\alpha \leq 15$ will lead to good aggregation factors without incurring a high overhead for tracking and keeping the state of a large numbers of STICKS, while at the same time achieving median STICK sizes of more than one. It is a necessary (but not sufficient) requirement for a STICK to be larger than one in size in order for it to be effectively aggregatable.

With Figure 3, we complete our routing table snapshot analysis. Figure 3 shows, as a function of α, the total number of prefixes in the AT. We further decompose the AT size into its GROUND and STICK components. For $\alpha \leq 15$, the GROUND contributes only limited numbers of prefixes while the prefixes from the STICK components dominate the total

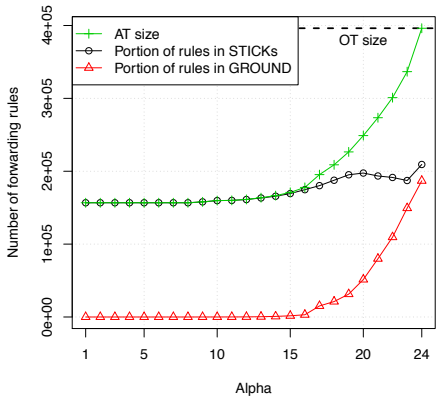
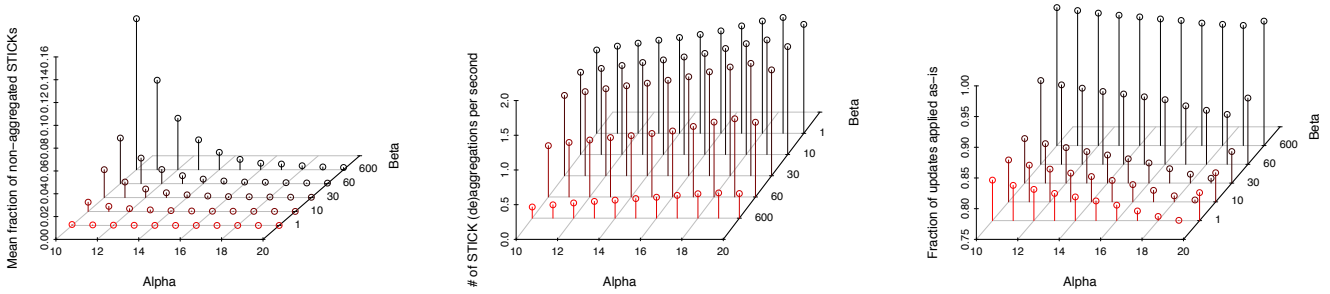


Fig. 3. Size of aggregated STICKS and GROUND as a function of α.

size of the AT, which is more than 60% off of the size of the OT. This is consistent with our results in Figure 2(c), in which we show that the aggregation gain suffers when alpha grows beyond 15. Furthermore, we observe a steep increase in the size of GROUND for $\alpha \geq 20$. At the same time, we see limited changes in the STICK sizes. As a result, the total size of the AT grows until it reaches the size of the OT, see the dashed line on Figure 3.

In summary, based on our analysis, a sensible region of α in the general case of current IPv4 routing tables appears to lie below 16. The results in Figure 3 are particularly encouraging for LFA as they show that even for α up to 18 the total size of the FIB can be reduced by at least 50%. This gives us evidence in the approach of aggregating STICKS individually, as the achieved aggregation factors are similar to those from optimal aggregation of the complete FIB [7], [4].

Trade-offs over time. Now that we have expectations from our analysis about the impact of α on the achieved aggregation



(a) Fraction of non-aggregated STICKS.

(b) Number of STICK (de)aggregations.

(c) Fraction of routing updates that are applied as-is, which is when the update affects the GROUND or a non-aggregated STICK.

Fig. 4. LFA trade-offs with α and β.

factors, we now analyze the online performance of LFA under changing α and β. For that, we rely again on publicly available data from the RouteViews project [10]. We rely on a single dataset taken 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 cover 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 α 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 [3].

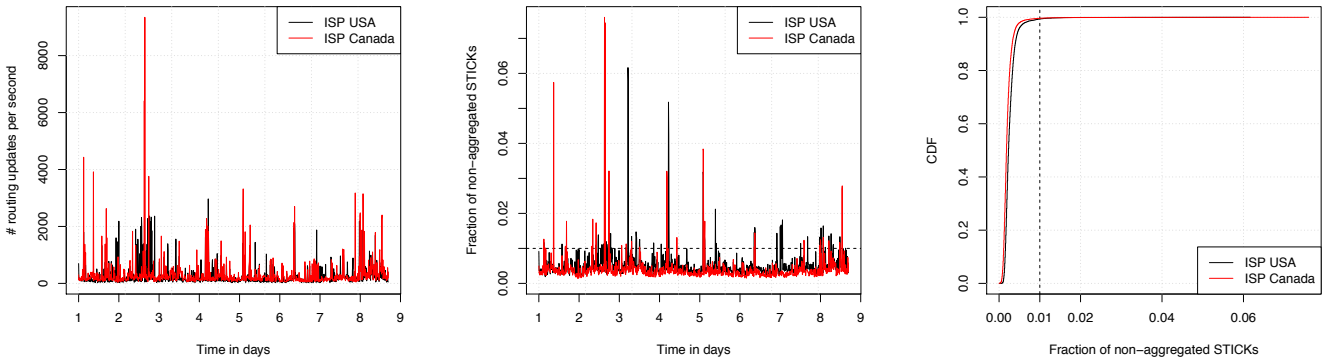
In Figure 4(a), we show the fraction of STICKS that are non-aggregated over time. This is a particularly important metric to consider as it provides some intuition about the locality of routing table updates. Non-aggregated STICKS represent those that have seen updates within the last β seconds. The results in Figure 4(a) show, that for $\alpha \geq 14$ and $\beta \leq 60s$ the fraction of non-aggregated STICKS is very low: On average, less than 0.4% of the STICKS are not aggregated.

Another metric to consider is the number of (de)aggregations of STICKS over time. This metric will tell us how often updates hit aggregated STICKS, requiring to deaggregate them before applying the update, and how often STICKS are aggregated after a stable period of β seconds. In Figure 4(b) we show the average number of STICK (de)aggregations per second as a function of α 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 STICK (de)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 α and β on the fraction of routing table updates which can be applied immediately at no extra cost due to deaggregations of STICKS. This includes all routing table updates that affect either the GROUND, or non-aggregated STICKS. Figure 4(c) shows the fraction of such routing table updates as a function of the parameters α and β. We observe that as β decreases, this fraction also decreases. This is expected since smaller values of β limit the ability of LFA to leverage update locality over time. On the other hand, the behavior of α is non-trivial. As α increases, the GROUND increases, while non-aggregated STICKS decrease (Figure 4(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 GROUND increases very slowly with α, while non-aggregated STICKS decreases much faster with α. The reason for this behavior is that smaller STICKS 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 α and β. Our results suggest that α should not be larger than 15 to achieve good aggregation gains. The results from our online experiments suggest that α should be ≥ 14 to maintain a low number of non-aggregated STICKS for $\beta \leq 60s$. For $\alpha = 14$, Figure 4(a) suggests that β should be no larger than $60s$, while Figures 4(b) and 4(c) show benefits in choosing a large value of β. In summary, our analyses indicate that the most appropriate values are $\alpha = 15$ and $\beta = 60s$.

Performance over time. To better understand the performance of LFA with $\alpha = 15$ and $\beta = 60s$, we now perform experiments based on more than one week worth of routing table updates. The results are shown in Figure 5 for two ISP routers, one from Canada and one from the USA. We plot the workload in Figure 5(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



(a) Number of routing table updates per second. This plot shows the maximum of every 10 minute time bin.

(b) Fraction of non-aggregated STICKS per second with $\alpha = 15$ and $\beta = 60s$. This plot shows the maximum of every 10 minute time bin.

(c) Distribution of fractions of non-aggregated STICKS per second.

Fig. 5. LFA performance over time.

than 2,000 routing table updates per second. In Figure 5(b) we plot the corresponding fraction of non-aggregated STICKS over time. Again, to give importance to the high (bad) 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 5(a) and 5(b) exhibits 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 5(c) the CDF of the fractions of non-aggregated STICKS in one second time intervals. Contrary to Figures 5(a) and 5(b) that show maximum values over 10 minute bins, Figure 5(c) provides a representative perspective on the ability of LFA to keep most of the FIB compressed over time. In more than 99% of the one second time intervals for both routers, less than 1% of the STICKS are non-aggregated. LFA is 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 can be exploited by FIB aggregation algorithms such as LFA, even under the bursts of routing table updates generated by BGP.

IV. A Competitive Analysis

Motivated by the empirical results discussed in the previous section, we investigate the trade-off between AT size and update cost from a formal *competitive analysis* perspective. In particular, we propose a simple online scheme HIMS (for "hide invisible merge siblings") to aggregate subtrees or STICKS in a local manner such that update costs are amortized.

Competitive analysis [11] is a framework to study the performance of an *online algorithm* in the *worst case*. Essentially, an online algorithm is an algorithm that does not have any knowledge of future BGP updates, and needs to decide *online* on where and when to aggregate the AT. The yard-stick to evaluate the quality of an online algorithm ALG is an optimal *offline algorithm* OPT which knows the whole input sequence in advance.

Definition 4.1 (Competitive Ratio): We call an online algorithm ALG *ρ-competitive* if there exists a constant k, such that for any input sequence it holds that $\text{COST}(\text{ALG}) \leq \rho \cdot \text{COST}(\text{OPT}) + k$. The competitive ratio of an algorithm is the *infimum* over all possible ρ such that the algorithm is ρ-competitive.

For our formal description, we focus on color changes only and assume continuous time where at any time t, a forwarding rule may change a port (i.e., *color*). The input to the algorithms is a sequence of such changes called *events*. As before, the original FIB is referred to as OT, the aggregated one as AT.

We associate a fixed cost γ to any AT rule change. The total cost incurred is called *update cost*; the cost incurred by an algorithm ALG in a time interval I is denoted by $\text{U-COST}_I(\text{ALG})$. The second type of cost we want to optimize is the size of the AT — the number of rules in the AT. For an algorithm ALG and time t, we denote the size of the AT at time t by $\text{SIZE}_t(\text{ALG})$. The total memory cost in a time interval I is then defined as $\text{M-COST}_I(\text{ALG}) = \int_I \text{SIZE}_t(\text{ALG})\, dt$. This paper focuses on minimizing the sum of these two costs, i.e., $\text{COST}(\text{ALG}) = \text{U-COST}(\text{ALG}) + \text{M-COST}(\text{ALG})$. Note that the parameter γ can be used to control the trade-off between the two costs.

We represent both the OT and the AT as one-bit tries. This affects merely the presentation, as we do not make assumptions about the actual implementation of the OT/AT data structures. We assume that each non-leaf node has exactly two children. Each node in the trie (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 assume minimal tries, that is, tries without blank sibling leaves

(they may contain blank leaves, though).

For any node v, we denote the subtree rooted at v by $T(v)$. We call a non-root node *left (right)* if it is a left (right resp.) son of its parent. Sometimes it is convenient to identify the nodes with the address ranges they represent. In particular, we call two nodes *adjacent* or *overlapping* if the address ranges they cover are adjacent or overlapping, respectively. Note that the latter case implies that nodes are in an ancestor-descendant relation and one range is in fact contained in the other.

Moreover, we will make use of the following definitions.

Definition 4.2 (Rules and Least Colored Ancestor): We call a node v a OT (AT) *rule* if it is colored in the OT (in the AT). For any node v (also a blank one), we denote its *least colored ancestor* (the non-blank ancestor farthest from the root) in the OT and in the AT by $\text{lca}_{\text{U}}(v)$ and $\text{lca}_{\text{F}}(v)$, respectively.

We start by pointing out that the coloring of the OT (AT) implies the coloring of the 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 OT (AT). Unlike in the OT, the node that determines the color of a given address in the AT may change with time. For presentation, we slightly extend the address space, incorporating two blank addresses -1 and 2^w.

We call a rule in the OT which does not determine the color of any address *superfluous*. As the color changes of such nodes can be ignored, we may assume that the OT contains no such nodes as they can be removed at a constant cost by an algorithm at the very beginning. Additionally, such removal yields the following property.

Observation 4.3: Any input event (a color change) changes the coloring of the address space, and hence any algorithm has to react by modifying the AT and paying at least γ.

For our online aggregation algorithm, we will again partition the nodes of the OT trie into STICKS. In doing so, we define STICKS differently to STICKS in LFA. In HIMS we allow STICKS to reside in "inner" parts of the trie, rather than focusing on STICKS connecting leaves only. Concretely, we group *all* nodes into STICKS and a remaining set of blank nodes. Each STICK will contain at least one colored node and possibly some blank nodes, and represents a "subtree" (not necessarily connecting leaves) that can be aggregated independently.

To this end, we first group all rules of the OT into sets $L_1, L_2, L_3, \ldots$. Each L_i is a maximal (w.r.t. to cardinality) set of colored nodes whose covered address spaces are pairwise non-overlapping, such that if all its nodes were of the same color c, they could be compressed to 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 defined to contain all nodes "between v_i and L_i inclusively", i.e., all nodes from $T(v_i)$ that are either from L_i or are ancestors of L_i, cf. Fig. 6. 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. When L_i is a singleton, S_i is also a singleton and is called *trivial* STICK. As OT does not contain superfluous nodes, all STICKS are disjoint and all internal nodes of a STICK are blank. We call nodes that belong to a STICK *active*.

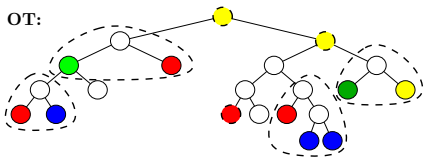


Fig. 6. Partition of the OT into STICKS. Superfluous nodes are already removed. STICK boundaries are marked with dashed lines.

In order to study churn-aware FIB compression from a competitive analysis perspective, we distill the basic time and space locality properties of the LFA algorithm, and introduce a simple approach called HIMS ("hide invisible merge siblings"). The HIMS online algorithm operates over STICKS; In our context, HIMS is just a small set of basic rules, which seek to amortize update costs over time.

HIMS keeps a subset of active nodes in the AT, while all the inactive nodes are always blank. For any active node, it defines two counters that are functions of time and depend on the coloring of the OT. Fix any node u belonging to some STICK S. If u is a leaf of S, then $L(u) = \{u\}$, otherwise $L(u)$ contains 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 how long till time t (uninterruptedly) all nodes of $L(u)$ have 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 till time t (uninterruptedly) all nodes of $L(u) \cup \{\text{lca}_{\text{U}}(u)\}$ have the same color. When $\text{lca}_{\text{U}}(u)$ does not exist, $H_u(t) = 0$ for any time t.

Any color change of a STICK leaf u causes resetting 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 also that $C_u(t) \geq H_u(t)$, $C_{p(u)}(t) \leq C_u(t)$ and $H_{p(u)}(t) \leq H_u(t)$ if $p(u)$ is defined.

The algorithm HIMS keeps an active node u as a rule in the AT at time t if and only if the following three conditions hold:

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

The color of u is the color of nodes in $L(u)$. Note that this color is unique (either u is a leaf and then $L(u)$ is a singleton,

or $C_u(t) > 0$). When a change in the AT is enforced by these rules, HIMS makes the minimal necessary amount of changes to achieve the desired state of the AT.

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, the algorithm simply waits till $H_u(t)$ reaches γ and then removes (the invisible) u from the AT. On the other hand, for a STICK S that has no colored ancestors in the OT, $H_u(t) = 0$ for any $u \in S$.

The following theorem states our main formal result: The HIMS strategy achieves a good performance even compared to an optimal offline algorithm.

Theorem 4.4: HIMS is $O(w^2)$-competitive, where w is the binary address length.

More precisely, as we will see in the proof, the competitive ratio only depends on the *maximal prefix length* (rather than the entire address length). For example, the longest prefix observed in our empirical study in Section III is 24 bits.

Proof Sketch. Due to space constraints, we only sketch the proof. A full version of the proof can be found online [5].

First, we prove that due to the slow pace of changes triggered by HIMS, we may always charge an AT update either to a change of the OT, or to a time period with a length of at least γ that this (or a related) rule spent in the AT. By assuring that no rule will be charged more than $O(w)$ times, we obtain the following bound.

Lemma 4.5: For any input sequence, U-COST(HIMS) $= O(w) \cdot (\text{M-COST(HIMS)} + m \cdot \gamma)$, where m is the number of color changes in the input sequence.

It remains to compare M-COST(HIMS) to COST(OPT). To this end, we introduce the 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 the address $a + 1$ (where blank is treated as an additional color). We say that rainbow point (a, t) *occurs* at time t. We call two rainbow points *different* if their addresses are different. Rainbow points measure the spatial-temporal complexity of the coloring of the address space; also OPT has to represent this coloring by its own AT.

Lemma 4.6: If there are γ pairwise different rainbow points occurring in some time interval I of length γ, then $\text{COST}_I(\text{OPT}) \geq \lceil k/2 \rceil \cdot \gamma$.

It remains to show how to find sufficiently many rainbow points on the basis of the snapshot of the AT at certain times. The following lemma captures the core properties of the optimizations performed by HIMS, essentially stating that if at some time the AT contains two "neighboring" nodes, then either they cannot be aggregated by HIMS now, or it was not possible to aggregate them in the nearest past. In either case, we provide a witness (a rainbow point) to support such a claim.

Lemma 4.7: Fix any time t at which the AT of HIMS does not change, and an address a. Assume the colors of a and $a+1$ are determined in the AT by two nodes u and v, respectively. For any of the following three cases: (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; there exists a rainbow point (a, t'), where $t' \in (t - \gamma, t]$.

Lemma 4.8: Fix any time t at which the AT of HIMS does not change, let γ be the number of entries in this AT, and let I denote interval $(t - \gamma, t]$. Then the number of pairwise different rainbow points in I is $\Omega(k/w)$.

Proof: We only sketch the proof. First, we group the leaves in the AT of HIMS. We sweep the leaves from left to right, partitioning them into groups $G_1, G_2, \ldots, G_h$. In the grouping process, we put two consecutive leaves u, v (potentially 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*. One may observe that for a single group G_i the number of leaves from G_i along with the numbers of their ancestors is at most $O(w)$, and thus $h = \Omega(k/w)$.

We call a pair of two consecutive groups G_i and G_{i+1} *non-critical* if G_i is a right group and G_{i+1} is a left one, otherwise such a pair is *critical*. Among all $h-1$ consecutive group pairs, at least $\Omega(h)$ are critical. It suffices to show that for any critical pair of groups G_i and G_{i+1}, either (b_i, t_i) or $(a_{i+1} - 1, t_i)$ is a rainbow point for some $t_i \in I$, where b_i is the rightmost address covered by a leaf from G_i, and we denote the leftmost address covered by a leaf from G_{i+1} by a_i.

We denote the rightmost leaf of G_i by v_i and the leftmost leaf of G_{i+1} by v_{i+1}. We assume that address $b_i + 1$ is non-blank, otherwise we would immediately obtain that (b_i, t) is a rainbow point. As G_i and G_{i+1} are a critical pair, at least one of the two conditions holds: (i) v_i is a left node, or (ii) v_{i+1} is a right node; we assume the former without loss of generality. 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 $\text{lev}(u_i)$, $\text{lev}(v_i)$, 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 4.7, there is 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 4.7, there is 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 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. 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 4.7, there is a rainbow point $(a_{i+1} - 1, t_i)$, where $t_i \in I$.

■

Lemma 4.6 combined with Lemma 4.8 allows to state that M-COST(HIMS) $= O(w) \cdot$ COST(OPT). On the other hand, by Observation 4.3, OPT has to pay the term $O(m \cdot \gamma)$ occurring in the statement of Lemma 4.5, and thus, U-COST(HIMS) $= O(w^2) \cdot$ COST(OPT). This yields Theorem 4.4.

V. DISCUSSION AND FUTURE WORK

In this paper we ignored the impact that FIB aggregation may have on IP destination lookup times, because they are affected by this only to a limited extent. The state-of-the-art data structures used for destination lookups (see, [12, chapter 15] and the references therein) use a large variety of tree-like constructs augmented with additional information. This allows for lookup times in the order of $O(\log w)$, with practical implementations using 2-3 memory lookups on average. Additionally, little is known about proprietary data structures actually used in the routers of different vendors.

We believe our work is particularly relevant when FIB aggregation is to be implemented in systems which have a notable delay between the route processor and the forwarding engine. This is the case on very large routers, as well as on remotely controlled switches such as in large enterprise networks [13] or data centers [14]. The resulting FIB update processing delays can lead to limitations in the number of updates per second that can be applied. This is particularly relevant with Software-defined Networks (SDN) as well as centralized control planes [15], and has been shown to be a limitation in a number of existing OpenFlow implementations [16]. However, our work can help better scale these approaches, and can be combined with other techniques that leverage the traffic properties [17].

A more theoretical direction for future research regards the study of competitive *bi-criteria* guarantees of the aggregation trade-offs, and the investigation of how the HIMS scheme can be improved and generalized. Even in an offline setting, the design of optimal (approximation) algorithms remains an open question. While it is easy to see that under certain circumstances, e.g., when there can be at most one color change per time unit, the problem is *fixed parameter tractable*, i.e., 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 more general approximation algorithms exist.

VI. SUMMARY

In this paper we studied the spatial and temporal locality of routing table updates in the tree-like FIB data structure based on publicly available BGP traces. We proposed an online FIB aggregation algorithm, called Locality-aware FIB Aggregation (LFA), that leverages the locality properties in routing table updates. We evaluated the ability of LFA to keep the FIB compressed under the constant stream of BGP routing updates and showed that it is able to keep most of the FIB compressed most of the time. We also carried a competitiveness analysis of online FIB compression algorithms and obtained a non-trivial approximation ratio, which also holds for the computationally hard offline problem.

ACKNOWLEDGMENTS

The authors would like to thank Magnús M. Halldórsson (Reykjavik University) and Anja Feldmann (TU Berlin / Telekom Innovation Labs) for their valuable feedback on this work.

REFERENCES

[1] D. Meyer, L. Zhang, and K. Fall, "Report from the IAB Workshop on Routing and Addressing," RFC 4984 (Informational), Internet Engineering Task Force, September 2007.

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

[3] A. Elmokashfi, A. Kvalbein, and C. Dovrolis, "Bgp churn evolution: a perspective from the core," *IEEE/ACM Trans. on Networking*, vol. 20, no. 2, pp. 571–584, Apr. 2012.

[4] 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 ACM CoNEXT*, 2011.

[5] Technical Report for INFOCOM submission, "Exploiting locality of churn for FIB aggregation," in *http://www.net.t-labs.tu-berlin.de/∼stefan/infocom12fib.pdf*, 2012.

[6] J. Li, M. Guidero, Z. Wu, E. Purpus, and T. Ehrenkranz, "BGP routing dynamics revisited," *SIGCOMM Comput. Commun. Rev.*, vol. 37, pp. 5–16, 2007.

[7] R. P. Draves, C. King, S. Venkatachary, and B. D. Zill, "Constructing Optimal IP Routing Tables," in *Proc. of the IEEE INFOCOM*, 1999, pp. 88–97.

[8] S. Suri, T. Sandholm, and P. R. Warkhede, "Compressing two-dimensional routing tables," *Algorithmica*, vol. 35, no. 4, pp. 287–300, 2003.

[9] Y. Liu, X. Zhao, K. Nam, L. Wang, and B. Zhang, "Incremental forwarding table aggregation," in *Proc. of the GLOBECOM*, 2010, pp. 1–6.

[10] "University of Oregon Route Views Project," http://www.routeviews.org/.

[11] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

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

[13] C. Kim, M. Caesar, and J. Rexford, "Floodless in seattle: a scalable ethernet architecture for large enterprises," in *Proc. ACM SIGCOMM*, 2008, pp. 3–14.

[14] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: a distributed control platform for large-scale production networks," in *Proc. of OSDI*, 2010.

[15] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe, "Design and implementation of a routing control platform," in *Proc. of NSDI*, 2005, pp. 15–28.

[16] 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*, 2012.

[17] N. Sarrar, S. Uhlig, A. Feldmann, R. Sherwood, and X. Huang, "Leveraging Zipf's Law for Traffic Offloading," in *ACM SIGCOMM CCR*, 2012.