Competitive FIB Aggregation for Independent Prefixes: Online Ski Rental on the Trie

Marcin Bienkowski[1⋆] and Stefan Schmid[2]

[1] Institute of Computer Science, University of Wrocław, Poland
[2] Telekom Innovation Laboratories & TU Berlin, Germany

Abstract. This paper presents an asymptotically optimal online algorithm for compressing the *Forwarding Information Base* (FIB) of a router under a stream of updates (namely insert rule, delete rule, and change port of prefix). The objective of the algorithm is to dynamically aggregate forwarding rules into a smaller but equivalent set of rules while taking into account FIB update costs. The problem can be regarded as a new variant of ski rental on the FIB trie, and we prove that our deterministic algorithm is 3.603-competitive. Moreover, a lower bound of 1.636 is derived for any online algorithm.

1 Introduction

An Internet router typically stores a large number of *forwarding rules*: Given a packet's IP address, the router uses the so-called *Forwarding Information Base* (FIB) to determine the forwarding port (or next-hop) of the packet. These very time critical FIB lookups require a fast and expensive memory on the line card, which constitutes a major cost factor of today's routers. It is expected that the virtualization trend in the Internet will further increase the memory requirements [2,9], and also IPv6 does not mitigate the problem [3].

A simple and local solution to reduce the FIB size is the *aggregation (compression) of the FIB*, i.e., the replacement of the existing set of rules by an *equivalent but smaller* set. This solution does not affect neighboring routers and it can be done by a simple software update [18]. However, aggregation may come at the cost of a higher FIB update churn (e.g., see [5]): upon certain BGP updates, aggregated FIB entries may have to be disaggregated again. Frequent FIB updates are problematic as upon each update, internal FIB structures have to be rebuilt to ensure routing consistency. In particular, update costs are also critical in the context of *Software-Defined Networks (SDN)* (e.g., based on *OpenFlow* [10]), as the network controller is remote from the switch and FIB updates may have to be transmitted over a bandwidth-limited network [15].

While this problem is currently discussed intensively in the networking community [7,17], only heuristics and static algorithms have been proposed so far. We, in this paper, assume the perspective of competitive and worst-case analysis, and present a solution which *jointly optimizes* the FIB compression ratio and the number of FIB updates.

⋆ Supported by MNiSW grant number N N206 368839, 2010-2013.

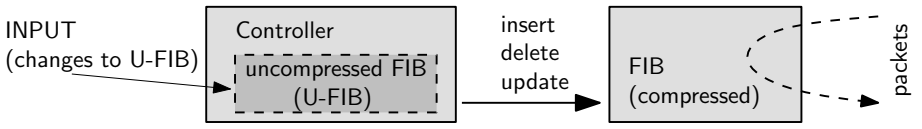

Fig. 1. Controller and FIB: the controller updates the rules in the FIB. This paper focuses on online algorithms for the controller.

1.1 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 *matches* all addresses that start with it, i.e., it corresponds to a *range* of addresses of the form $[k \cdot 2^i, (k+1) \cdot 2^i - 1]$.

Forwarding Rules. We consider a packet forwarding router with a set of *ports* (or *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 [11]: 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.

The router contains two parts: the *controller* (either implemented on the route processor, or an SDN controller) and the *(compressed) FIB* (stored in a fast and expensive memory), cf. Fig. 1. The controller keeps a copy of the *uncompressed FIB (U-FIB)* and receives a stream of 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, (1) a new forwarding rule may be *inserted*, (2) an existing rule *deleted*, or (3) a prefix may change its forwarding port (color *update*). A sequence of such *events* constitutes the *input* to our problem.

Right after a change occurs, the controller must ensure that the U-FIB and the FIB are equivalent, i.e., their forwarding and dropping behavior is the same. In this paper, we will make the simplifying assumption that the FIB prefixes are *independent*: the FIB does not contain any prefixes which overlap in their address range. To this end, the controller may insert, delete or update (change color) individual rules in the FIB. The controller may also issue these commands at any point of time (e.g., for a delayed compression of the FIB).

Costs. We associate a *fixed* cost α with any such change of a single rule in the FIB. Note that we represent the update cost as a constant to keep the model general: α is not specific for any particular FIB data structure (e.g., trie, cache, or Multibit Burrows-Wheeler [12]), but may also model the cost of transmitting

a control packet between an SDN controller and the OpenFlow switch. (See also [7].) The total cost paid this way is called *update cost*; the amount paid by an algorithm ALG in a time interval I is denoted by U-COST$_I$(ALG).

The second type of cost we want to optimize is the size of the FIB, which — following [4] — is defined as the number of FIB forwarding rules. This modeling is justified by state-of-the-art approaches (see, e.g., [11, 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.

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.

Competitive analysis. We assume a conservative 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 chaotic behavior of the route updates in the modern Internet [6]. We use the standard yard-stick of online analysis [1], 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 γ, such that for any input sequence it holds that COST(ALG) $\leq \rho \cdot$ COST(OPT) $+ \gamma$. The competitive ratio of an algorithm is the *infimum* over all possible ρ such that the algorithm is ρ-competitive.

Empirical Motivation. The motivation for our simplifying assumption of independent prefixes is twofold. First, an algorithm to solve the independent case can be applied to the independent subtrees. Moreover, empirical data shows that while Internet routers typically define a default route (an empty prefix), the prefix hierarchy is typically very flat [15]: prefixes hardly overlap with more than one other prefix. As of February 2013, the Internet-wide BGP routing table contains more than 440k prefixes. In table dumps obtained from *RouteViews* [13], we observe that around half of all prefixes do not have any less specifics, and on average, a prefix has 0.64 less specifics.

Furthermore, in our modeling, we neglect 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, [11, chapter 15]) use a large 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 the average). Additionally, little is known about proprietary data structures actually used in the routers of different vendors.

1.2 Related Work

There are known fast algorithms for optimal FIB aggregation of table snapshots, for example the Optimal Routing Table Constructor (ORTC) [4] and others [16]. 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 [17] and others [7,8,18]. Moreover, some authors even proposed to only store a *subset* of rules in the FIB, leveraging Zipf's law [15]. 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 to use churn locality for their benefit.

The closest paper to ours is [14] by Sarrar et al. The authors first study the temporal and spatial locality of churn in the trie empirically, and then present an $O(w^2)$-competitive online algorithm for tries with *dependent* prefixes. It is worth noting that in the dependent case, for a large class of online algorithms, there exists a $\Omega(w)$ lower bound. This indicates that the independent prefix variant might be inherently simpler.

1.3 Our Contribution

The main contribution of this paper is a deterministic online algorithm for the FIB aggregation problem (with independent prefixes) that jointly optimizes the FIB size (by rule aggregation) as well as the number of updates to the FIB (by timed waiting). We prove that our algorithm is 3.603-competitive under a worst-case sequence of rule updates (events: *insert*, *delete*, port *change*). Furthermore, we show that there provably does not exist any online algorithm with a competitive ratio smaller than 1.636.

Technically, the problem can be regarded as a variant of online ski rental on a trie: The presented algorithm Block seeks to aggregate prefixes slowly over time, amortizing aggregation costs with the memory benefits.

2 Basic Properties

Trie Representation. Throughout this paper, we will represent both the U-FIB and the FIB as one-bit tries containing all the prefixes from the forwarding rules. This affects merely the presentation: we do not assume anything about the actual implementation of the U-FIB/FIB structures. We assume that each non-leaf node has exactly two children. 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 assume minimal tries, that is, tries without blank sibling leaves (they may contain blank leaves, though).

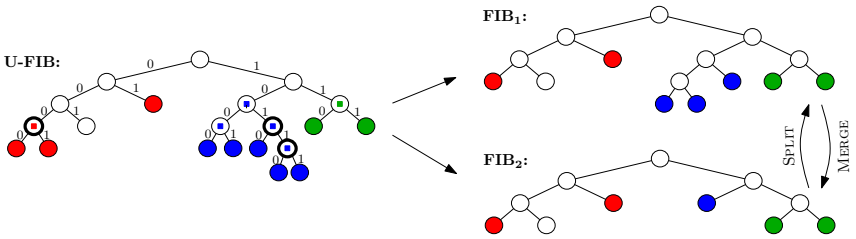

Fig. 2. One-bit tries representing a U-FIB and two possible compressions to the FIBs. Mergeable nodes are marked with small squares in the center. Nodes consolidated for the creation of FIB_1 are drawn with bold lines.

For any node v, we denote the subtree rooted at v by $T(v)$. A non-root node we call *left (right)* if it is a left (respectively right) son of its parent. Sometimes it is convenient to identify the nodes with the address ranges they represent.

Mergeable Nodes. Let's first assume that the state of the U-FIB is static, i.e., we are not processing an input, and study the structural properties of the possible FIB aggregations. An internal (blank) node v from U-FIB is called *c-mergeable* if all leaves of $T(v)$ are of color c. Sometimes, we will simply say that a node is *mergeable* instead of c-mergeable. Clearly, if a node v is c-mergeable, then all internal nodes from $T(v)$ are c-mergeable.

Mergeable nodes are the key to all compression patterns possible. Namely, any possible FIB aggregation is defined by choosing any set A of mergeable, pairwise non-overlapping nodes. For any c-mergeable node $v \in A$, we remove all descendants of v (recall that each of them is either an internal c-mergeable node or a leaf of color c) and color v with c. In the U-FIB, we call v and all the internal (mergeable) nodes of $T(v)$ *consolidated*, cf. Fig. 2.

Hence, all an algorithm may do is to choose which mergeable nodes to consolidate and when. At runtime, an algorithm may change the FIB incrementally, by consolidating or unconsolidating nodes. The only restriction is that all the (mergeable) descendants of consolidated nodes have to be consolidated as well. Therefore, there are two possible operations an algorithm may perform.

- *A merge operation* (at a mergeable, unconsolidated node v) changes the state of all, say k, mergeable, unconsolidated nodes from $T(v)$ to consolidated. These k nodes were internal ones, and hence the operation involves replacing $k+1$ leaves in the FIB by a node v, i.e., induces the update cost of $(k+2)\cdot\alpha$.
- *A split operation* has the reverse effect and is described by a tree (rooted at v) of k' consolidated nodes whose state is changed to unconsolidated. It is possible that this tree does not contain all the consolidated nodes of $T(v)$. In the FIB, this involves replacing node v by $k'+1$ nodes, and hence the associated update cost is $(k'+2)\cdot\alpha$.

The size of the FIB is tightly related to the number of consolidated nodes. Precisely speaking, we denote the size of the U-FIB by Size(U-FIB), the

number of mergeable nodes at time t by $M(t)$, and nodes consolidated by an algorithm ALG by $S^{\text{ALG}}(t)$. Then, $S^{\text{ALG}}(t) \leq M(t)$ and $\text{SIZE}_t(\text{ALG}) = \text{SIZE}(\text{U-FIB}) - S^{\text{ALG}}(t)$.

So far, we only studied the compression of a static U-FIB. When at some time t a prefix (v, c) changes color to c', the following changes to the U-FIB occur. If the sibling of v has color c', then the parent of v and possibly some of its ancestors may become c'-mergeable. If the sibling of v has color c, then all the c-mergeable ancestors of v (if any) become non-mergeable. If some of these nodes were consolidated, a split operation involving these nodes is forced.

Event Costs. We fix any algorithm ALG and take a closer look at its update cost, U-COST(ALG). Whenever ALG processes a single event at time t, it has to update the FIB paying α. However, this cost can be avoided if ALG performs a related merge or split operation (as defined in Sect. 2) immediately at time t. For example, if a color update of node v changes the state of the parent of v to mergeable, ALG may merge the parent of v at the same time and pay only 3α for the cost of merging without paying the event cost of α. Another example is a forced split.

Lemma 1. *Assume that an online algorithm* ALG *is R-competitive if we neglect event costs. Then, it is $(R+1/3)$-competitive if we take these costs into account.*

Proof. We partition U-COST(ALG) into the sum of O-COST(ALG), the cost of all (merge or split) operations performed by ALG, and E-COST(ALG), the cost of events on nodes not accompanied by immediate operations on the same nodes. In these terms, COST(ALG) = M-COST(ALG)+O-COST(ALG)+E-COST(ALG).

Fix any input sequence. By the assumption of the lemma, M-COST(ALG) + O-COST(ALG) $\leq R \cdot (\text{M-COST}(\text{OPT}) + \text{O-COST}(\text{OPT})) + \gamma$ for some constant γ. Let k be the number of events in the input sequence. Clearly, E-COST(ALG) $\leq k \cdot \alpha$. On the other hand, for any event, an optimal offline algorithm OPT either performs a merge or a split, which increases O-COST(OPT) by at least 3α, or does not perform any spatial operations on the trie, which increases E-COST(OPT) by α (color change only). Thus, $\frac{1}{3} \cdot \text{O-COST}(\text{OPT}) + \text{E-COST}(\text{OPT}) \geq k \cdot \alpha$, and hence,

$$\begin{aligned}
\text{COST}(\text{ALG}) &\leq R \cdot (\text{M-COST}(\text{OPT}) + \text{O-COST}(\text{OPT})) + \gamma + k \cdot \alpha \\
&\leq R \cdot \text{M-COST}(\text{OPT}) + (R + 1/3) \cdot \text{O-COST}(\text{OPT}) \\
&\quad + \text{E-COST}(\text{OPT}) + \gamma \\
&\leq (R + 1/3) \cdot \text{COST}(\text{OPT}) + \gamma \ ,
\end{aligned}$$

which completes the proof. □

3 The Algorithm BLOCK

Our algorithm BLOCK is very simple: With any node v, we associate a counter C_v which is a function of time. If v is c-mergeable at time t, then $C_v(t)$ measures

how long (uninterruptedly) v is in this state; otherwise $C_v(t) = 0$. The algorithm Block is parameterized with two constants, $A \geq B$; we derive an optimal choice for these constants later.

As soon as there is a non-consolidated node v whose counter is $A \cdot \alpha$, Block merges the tree $T(u)$ rooted at the ancestor u of v which is closest to the trie root and whose counter is at least $B \cdot \alpha$. (It is possible that $u = v$.) Furthermore, Block splits only when forced, i.e., when a consolidated node changes state to non-mergeable.

Lemma 2. *Fix any values of A and B. Neglecting event costs,* Block(A, B) *is* $\max\{(A+6)/A, (B+4)/B, (A+6)/(A+2-B), (B+4)/2, (A+4)/2\}$*-competitive.*

Proof. The proof pursues an accounting approach: We charge the non-event costs of any algorithm to particular nodes and relate the cost of Block and Opt on chosen subsets of nodes. Recall that whenever an algorithm merges or splits a subtree, it pays $(k+2) \cdot \alpha$, where k is the number of leaves of this subtree. We assign the cost of 3α to the root of this tree and α to all its remaining internal nodes. Thus, whenever an algorithm consolidates a mergeable node (or changes the state back from consolidated) it pays 3α or α. Furthermore, we assume that only nodes that are mergeable but not consolidated are counted towards the size of FIB. This way, at time t, we underestimate the actual memory cost by the number of non-mergeable nodes, Size(U-FIB) $- M(t)$. This amount is however the same for any algorithm. Therefore, if the algorithm is R-competitive using charged costs, it is also R-competitive in the actual cost model.

We first take a mergeable period (of length τ) of node v during which it is not consolidated by Block; we compare the costs of Opt and Block for v in this period. In this case, Block pays just τ for the memory cost. As Block does not consolidate v, $\tau < A\alpha$. If Opt decides to merge v, it pays at least α for merging v and at least α for splitting it. Otherwise, it pays τ for the memory cost. In total, the ratio of the Block and Opt costs is at most

$$R_1 = \tau / \min\{\tau, 2\alpha\} \leq \max\{1, \tau/2\alpha\} \leq \max\{1, A/2\} \ .$$

Now, we compare the costs on nodes that are consolidated (by a single merging operation) by Block. More precisely, we consider any time t at which Block merges a tree T rooted at node u. We analyze the total cost of Block and Opt for all mergeable periods of all nodes from T, such that these periods contain time t. Let $k+1$ be the number of all consolidated nodes from T (i.e., $k \geq 0$). We denote the value of counters of these nodes at time t by C_i. For convenience, we assume these values are sorted, i.e., $C_i \leq C_{i+1}$. By the definition of the algorithm, $B\alpha \leq C_1 \leq C_2 \ldots C_k \leq C_{k+1} = A\alpha$. On the considered mergeable periods of nodes of T, Block pays $\sum_{i=1}^{k+1} C_i$ (memory cost) plus $(k+3) \cdot \alpha$ (merging cost), plus $(k+1) \cdot 3\alpha$ (splitting cost as in the worst case the nodes are split individually). Thus, in total,

$$\textsc{Cost}_T(\textsc{Block}) \leq 2\alpha + \textstyle\sum_{i=1}^{k+1}(C_i + 4\alpha) \leq (A+6)\alpha + \sum_{i=1}^{k}(C_i + 4\alpha) \ .$$

Now, we compute the cost of Opt on the same mergeable periods. Let $\mathcal{K}$ be the smallest time interval containing all these periods. By the algorithm definition, $\mathcal{K}$ starts at time $t - A\alpha$. We consider three cases depending on Opt actions within $\mathcal{K}$. In each bound, after computing the Block-to-Opt ratio, we immediately use the relation $(a+b)/(c+d) \le \max\{a/c, b/d\}$.

Case 1. Within $\mathcal{K}$, Opt does not merge any subtree rooted at a node from T nor at any of ancestors of u. In this case, no node of T becomes consolidated by Opt, and hence Opt just pays memory costs, i.e., $\textsc{Cost}_T(\textsc{Opt}) = \sum_{i=1}^{k+1} C_i = A\alpha + \sum_{i=1}^{k} C_i$. Therefore, the Block-to-Opt cost ratio is at most

$$R_2 = \frac{(A+6)\alpha + \sum_{i=1}^{k}(C_i + 4\alpha)}{A\alpha + \sum_{i=1}^{k} C_i} \le \max\left\{\frac{A+6}{A}, \frac{B+4}{B}\right\} .$$

Case 2. Within $\mathcal{K}$, Opt does not merge any subtree rooted at a node in T, but does merge a subtree rooted at an ancestor of u, say u'. By the algorithm definition, $C_{u'}(t) < B \cdot \alpha$ (otherwise Block would choose u' as the root for the merging operation). This means that Opt merges not earlier than at time $t - B\cdot\alpha$, so the corresponding counters of the nodes from T are at least $C_i - B\alpha$. Hence, the cost of Opt associated with node i is at least $C_i - B\alpha$ (memory cost) plus 2α (merging and splitting). Therefore, $\textsc{Cost}_T(\textsc{Opt}) \ge \sum_{i=1}^{k+1}(C_i - B\alpha + 2\alpha) = (A+2-B)\alpha + \sum_{i=1}^{k}(C_i + (2-B)\alpha)$, and the ratio in this case is

$$R_3 = \frac{(A+6)\alpha + \sum_{i=1}^{k}(C_i + 4\alpha)}{(A+2-B)\alpha + \sum_{i=1}^{k}(C_i + (2-B)\alpha)} \le \max\left\{\frac{(A+6)}{A+2-B}, \frac{B+4}{2}\right\} .$$

Case 3. Within $\mathcal{K}$, Opt merges some subtree rooted at a node from T. In this case, we split the indices of nodes of T into two sets: a set S of nodes that become consolidated sometime within $\mathcal{K}$ and a set N of nodes that remain unconsolidated for the whole period $\mathcal{K}$. Clearly, $|S| + |N| = k + 1$. For the node from S being the root of the merged subtree, Opt pays at least $3\alpha + \alpha$ (merging and splitting cost) and for the remaining nodes from S at least $\alpha + \alpha$. For any node from N, Opt pays at least C_i (memory cost). Hence, in total, $\textsc{Cost}(\textsc{Opt}) = 2\alpha + \sum_{i\in S} 2\alpha + \sum_{i\in N} C_i$, and the cost ratio is

$$R_4 = \frac{2\alpha + \sum_{i\in S}(C_i + 4\alpha) + \sum_{i\in N}(C_i + 4\alpha)}{2\alpha + \sum_{i\in S} 2\alpha + \sum_{i\in N} C_i} \le \max\left\{1, \frac{A+4}{2}, \frac{B+4}{B}\right\} .$$

As we all possible cases are considered above, the competitive ratio is at most $\max_{1\le i\le 4} R_i$. The lemma follows by substituting the actual values of R_i. □

Tedious case analysis and elementary algebra shows that the choice of parameters minimizing the guarantee of Lemma 2 is $A = \sqrt{13} - 1 \approx 2.606$ and $B = \frac{2}{3}A \approx 1.737$. Taking event costs into account (cf. Lemma 1), we obtain the following result.

Theorem 1. *The competitive ratio of* $\textsc{Block}(\sqrt{13}-1, 2\sqrt{13}/3-2/3)$ *is at most* $(\sqrt{13}+3)/2 + 1/3 \approx 3.603$.

Note that there is a simpler version of the Block algorithm that consolidates only one node at a time, i.e., uses $A = B$. In such case, the optimal choice of the parameters is $A = B = 2$, and thus such algorithm is $(4 + 1/3)$-competitive.

4 Handling Insertions and Deletions

So far, we show how to handle color updates to the U-FIB. In this section, we show that it is possible to handle also insertions of new rules to the U-FIB and deletions of old rules from the U-FIB. Recall that the algorithm Block simply watches the changes in the mergeability of U-FIB nodes. For completing the definition of Block, it is therefore sufficient to show how insertions and deletions affect that aspect.

- A prefix (v, c) is inserted to the U-FIB. If node v already existed in the tree as a blank node, v must be a leaf. In this case, zero or more ancestors of v become c-mergeable. If, however, node v did not exist in the tree, then v is inserted as a leaf with a blank sibling. The set of mergeable nodes does not change in this case.
- A prefix (v, c) is deleted from the U-FIB. Node v becomes blank and all ancestors of v (including the root) become non-mergeable. As we require the tree to be minimal, we may have to perform an optional pruning of blank nodes. However, this does not change the mergeability of any node.

The only detail that has to be changed in the analysis of Block is that now the size of the U-FIB is not constant but is a function of time, denoted $\textsc{Size}_t(\text{U-FIB})$ at time t. Then in the proof of Lemma 2, by using charged costs, we underestimate the actual memory size by $\textsc{Size}_t(\text{U-FIB}) - M(t)$, where $M(t)$ is the number of nodes mergeable at t. However, as in the original proof, this amount is the same for Block and Opt, and thus R-competitiveness using charged costs implies the R-competitiveness in the actual cost model. Thus, we obtain the following result.

Theorem 2. *The competitive ratio of* $\textsc{Block}(\sqrt{13}-1, 2\sqrt{13}/3-2/3)$ *is at most* $(\sqrt{13}+3)/2 + 1/3 \approx 3.603$ *also when insertions and deletions may occur in the input sequence.*

5 Lower Bound

The algorithm Block was designed with two objectives in mind: (i) to balance the memory cost and the update cost, (ii) to exploit the possibility of merging multiple tree nodes simultaneously at a lower price. An online algorithm is bound to choose sub-optimally in both of these aspects: we will show a lower bound of 1.636 on the competitive ratio of any online algorithm.

The analysis of Block suggests a straightforward lower bound: We keep a tree of two prefixes $\{0, 1\}$. By changing the color of one of them, the adversary changes the state of root from non-mergeable to mergeable, and back. When

the root becomes mergeable, the algorithm may consolidate it at some time, but right after that happens, the adversary turns the root non-mergeable, enforcing a split. An analogous approach can be found in many online problems, most notably in the ski-rental problem [1]. However, unlike in the ski-rental problem, we cannot obtain the lower bound of 2 by this adversarial strategy. The first obstacle is that the memory cost (the equivalent of renting skis) is always at least 1 (even for Opt). The second obstacle are event costs that are sometimes paid also by Opt. An exact analysis would yield a lower bound of 1.5. We may improve this bound by making the tree slightly larger.

Theorem 3. *Any online algorithm* Alg *has a competitive ratio of at least* $18/11 \approx 1.636$.

Proof. The set of prefixes in the U-FIB will be constant and equal to $\{00, 01, 1\}$, initially with the colors red, green and green, respectively. A strategy of the adversary consists of phases. At the end of each phase, the state of the U-FIB will be the same as the initial one, and the Alg-to-Opt cost ratio on any phase will be at least 18/11. The adversary may generate a sequence consisting of an arbitrary number of phases, thus ensuring that the cost Alg cannot be hidden in the additive constant γ in the definition of the competitive ratio.

In a single phase starting at time t, the adversary changes the color of prefix 00 to green, making both internal nodes of the tree mergeable. Note that in such a situation there are three possible states of Alg: a low state (no node consolidated), a middle state (the lower mergeable node consolidated) and a top state (both mergeable nodes consolidated). Two cases are possible:

- Alg changes state for the first time (either to the top or to the middle one) at time $t' \leq t + \alpha$.
- Alg does not change state in the interval $[t, t+\alpha]$. In this case, let $t' > t+\alpha$ be the first time when it changes its state to the top one (it may change state to the middle one before t').

Note that if none of the two described events occurs, then Alg never changes its state to the top one. In this case, its FIB size is at least 2 whereas the optimal possible is 1. This would immediately imply a lower bound of 2 on the competitive ratio.

At time $t' + \epsilon$, the adversary changes the color of prefix 00 back to red, forcing Alg to change the state back to the low one, and ending the phase. As the adversary may choose ϵ to be arbitrarily small, in the analysis we assume $\epsilon = 0$.

To analyze the performance of Alg in a single phase, we set $\ell = t' - t$. The cost of Opt is upper-bounded by the minimum of costs of two possible strategies: (i) do nothing, (ii) change the state to top at time t and then back to low at time $t' + \epsilon$. The cost for the former strategy is 3ℓ (memory cost) plus 2α (event cost), while the cost for the latter is ℓ (memory cost) plus 4α (merging cost) + 4α (splitting cost). Altogether, $\textsc{Cost}(\textsc{Opt}) \leq \min\{2\alpha+3\ell, 8\alpha+\ell\}$. We consider two cases.

1. The first event occurs, i.e, $\ell \leq \alpha$. Then ALG pays at least 3α (merging cost) and another 3α (splitting cost). Additionally, the memory cost is 3ℓ as ALG is in the low state till it merges anything. Furthermore, if ALG merges after time t, then it has to pay event cost α at time t. Thus, we consider two subcases:
   (a) Algorithm merges already at time t, i.e., $\ell = 0$. Then, COST(OPT) $\leq 2\alpha$, COST(ALG) $= 6\alpha$, and hence the ratio is $R_1 = 3$.
   (b) Algorithm merges after time t. Then, COST(OPT) $\leq 2\alpha + 3\ell$ while COST(ALG) $= 7\alpha + 3\ell$. The ratio is then at least
$$R_2 = \frac{7\alpha + 3\ell}{2\alpha + 3\ell} \geq \frac{7\alpha + 3\alpha}{2\alpha + 3\alpha} = 2 \ .$$
2. The second event occurs, i.e., $\ell > \alpha$. We consider two subcases.
   (a) ALG changes state only once, at time ℓ. Then, it pays α (event cost) plus 3ℓ (memory cost) plus 4α (merging cost) plus 4α (splitting cost). The ratio is then
$$R_3 = \frac{9\alpha + 3\ell}{\min\{2\alpha + 3\ell, 8\alpha + \ell\}} \geq \frac{18}{11} \approx 1.636 \ .$$
   (b) ALG changes state more than once. Then ALG is in low state at least till time α and in state low or middle at times between α and t'. Therefore, it pays at least α (event cost) plus $3 \cdot \alpha + 2 \cdot (\ell - \alpha)$ (memory cost) plus $3\alpha + 3\alpha$ (merging cost) + 4α (splitting cost). In this case, the ratio is
$$R_4 = \frac{12\alpha + 2\ell}{\min\{2\alpha + 3\ell, 8\alpha + \ell\}} \geq \frac{18}{11} \approx 1.636 \ .$$

Altogether, in either case, the competitive ratio is at least 18/11. □

6 Conclusions

This paper studied a novel online aggregation problem arising in the context of (classical or SDN) router optimization. The described online algorithm that provably achieves a low, constant competitive ratio. Since the derived lower bound is not tight, the main open technical question regards closing the gap between the upper and lower bound.

Acknowledgments. The authors would like to thank Magnús M. Halldórsson, Nadi Sarrar and Steve Uhlig for many interesting discussions.

References

1. A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
2. T. Bu, L. Gao, and D. Towsley. On characterizing BGP routing table growth. *Comput. Netw.*, 45:45–54, 2004.

3. L. Cittadini, W. Muhlbauer, S. Uhlig, R. Bushy, P. Francois, and O. Maennel. Evolution of internet address space deaggregation: myths and reality. *IEEE J.Sel. A. Commun.*, 28:1238–1249, 2010.
4. R. P. Draves, C. King, S. Venkatachary, and B. D. Zill. Constructing optimal IP routing tables. In *Proc. of the 18th IEEE Int. Conference on Computer Communications (INFOCOM)*, pages 88–97, 1999.
5. A. Elmokashfi, A. Kvalbein, and C. Dovrolis. BGP churn evolution: a perspective from the core. *IEEE/ACM Transactions on Networking*, 20(2):571–584, 2012.
6. J. Li, M. Guidero, Z. Wu, E. Purpus, and T. Ehrenkranz. BGP routing dynamics revisited. *ACM SIGCOMM Computer Communication Review*, 37:5–16, 2007.
7. Y. Liu, B. Zhang, and L. Wang. Fast incremental FIB aggregation. In *Proc. of the 32nd IEEE Int. Conference on Computer Communications (INFOCOM)*, 2013.
8. Y. Liu, X. Zhao, K. Nam, L. Wang, and B. Zhang. Incremental forwarding table aggregation. In *Proc. of the Global Communications Conference (GLOBECOM)*, pages 1–6, 2010.
9. L. Luo, G. Xie, S. Uhlig, L. Mathy, K. Salamatian, and Y. Xie. Towards TCAM-based scalable virtual routers. In *Proc. of the 8th Int. Conf. on Emerging Networking Experiments and Technologies (CoNEXT)*, pages 73–84, 2012.
10. N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38:69–74, 2008.
11. D. Medhi and K. Ramasamy. *Network Routing: Algorithms, Protocols, and Architectures*. Morgan Kaufmann Publishers Inc., 2007.
12. G. Rétvári, Z. Csernátony, A. Korösi, J. Tapolcai, A. Császár, G. Enyedi, and G. Pongrácz. Compressing IP forwarding tables for fun and profit. In *Proc. of the 11th ACM Workshop on Hot Topics in Networks (HotNets)*, pages 1–6, 2012.
13. RouteViews Project. `http://www.routeviews.org/`, 2013.
14. N. Sarrar, M. Bienkowski, S. Schmid, S. Uhlig, and R. Wuttke. Exploiting locality of churn for FIB aggregation. Technical Report 2012/12, Technische Universität Berlin, 2012.
15. N. Sarrar, S. Uhlig, A. Feldmann, R. Sherwood, and X. Huang. Leveraging Zipf's law for traffic offloading. *ACM SIGCOMM Computer Communication Review*, 42(1):16–22, 2012.
16. S. Suri, T. Sandholm, and P. R. Warkhede. Compressing two-dimensional routing tables. *Algorithmica*, 35(4):287–300, 2003.
17. 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 7th Int. Conf. on Emerging Networking Experiments and Technologies (CoNEXT)*, pages 29:1–29:12, 2011.
18. X. Zhao, Y. Liu, L. Wang, and B. Zhang. On the aggregatability of router forwarding tables. In *Proc. of the 29th IEEE Int. Conference on Computer Communications (INFOCOM)*, pages 848–856, 2010.