FIFA: Fast Incremental FIB Aggregation

Yaoqing Liu
yliu6@memphis.edu
The University of Memphis

Beichuan Zhang
bzhang@cs.arizona.edu
The University of Arizona

Lan Wang
lanwang@memphis.edu
The University of Memphis

***Abstract*—The fast growth of global routing table size has been causing concerns that the Forwarding Information Base (FIB) will not be able to fit in existing routers' expensive line-card memory, and upgrades will lead to higher cost for network operators and customers. FIB Aggregation, a technique that merges multiple FIB entries into one, is probably the most practical solution since it is a software solution local to a router, and does not require any changes to routing protocols or network operations. While previous work on FIB aggregation mostly focuses on reducing table size, this work focuses on algorithms that can update compressed FIBs quickly and incrementally. Quick update is critical to routers because they have very limited time to process routing updates without impacting packet delivery performance. We have designed three algorithms: FIFA-S for smallest table size, FIFA-T for shortest running time, and FIFA-H for both small tables and short running time, and operators can use the one best suited to their needs. These algorithms significantly improve over existing work in terms of reducing routers' computation overhead and limiting impact on the forwarding plane while maintaining a good compression ratio.**

I. Introduction

The global routing table size has been increasing faster than ever in a super-linear trend, mostly due to the practice of multihoming and traffic engineering [1]. This has caused serious concerns in both academia and industry ([2][3]). Once the FIB becomes so large that it can no longer fit in the fast memory of routers' line cards, ISPs have to upgrade their line cards, eventually making Internet services more expensive. While a number of solutions ($e.g.$, [4][5][6][7][8][9][10]) have been proposed to solve the routing table scalability problem in the long run by changing the routing architecture, ISPs need practical solutions soon, and FIB aggregation is considered one of the most practical solutions [11].

FIB aggregation reduces FIB size by combining entries whose prefixes are numerically aggregatable and whose next-hops are the same. It is a software solution that can be applied to a single router without upgrading the hardware, changing the control plane, or affecting packets' forwarding paths. Thus it can be deployed incrementally and selectively in a network at operators' discretion. One of the fundamental tradeoffs in FIB aggregation is between aggregated table size and computation overhead. Spending too much CPU cycles in aggregating the table will delay the downloading of the table into the line cards, which may lead to packet loss or incorrect forwarding. Existing work (*e.g.*, [12][13][14][15]) has demonstrated that FIB aggregation can reduce table size by as much as 70% with moderate computation, but these efforts have not focused on reducing routers' overhead in all aspects.

The most challenging problem in FIB aggregation is to quickly apply updates to the already aggregated table and still maintain good compression ratio. When a router receives a routing update, it has very limited amount of time to process the update and install the new FIB. When the FIB is already aggregated, one routing table change may lead to updating multiple FIB entries, because it may change the aggregatability of those entries. In some cases, there can be thousands or even tens of thousands FIB entries to be updated, even if there is only a single routing table change. Therefore in this work, we focus on reducing FIB aggregation's overhead in the following aspects: (1) reducing the overall time of processing a stream of updates; (2) speeding up the process to re-aggregate an entire FIB, if a scheme requires such re-aggregations; and (3) reducing the average and maximum number of FIB changes caused by any individual routing table change, so as to reduce the time it takes to push those changes to the line card.

To this end, we have designed three algorithms: FIFA-S for smallest table size, FIFA-T for shortest running time, and FIFA-H for both small tables and short running time. They take advantage of some intrinsic properties of an aggregated FIB trie to speed up the incremental update process. Among them, FIFA-S and FIFA-H do not need to run full re-aggregations, and FIFA-T performs fast re-aggregation on the existing aggregated trie. Moreover, they use a prioritized set of next-hop selection rules to improve the stability of the aggregated FIB, thus reducing the number of FIB changes per routing table change. Our evaluation shows that they outperform state-of-art algorithms in both speed and FIB stability.

The remainder of the paper is organized as follows. Section II gives an overview of FIB aggregation. Sections III presents the design of FIFA. Section IV evaluates the performance improvement of FIFA over existing work. Section V discusses related work, and Section VI concludes the paper.

II. Background on FIB Aggregation

A. RIB and FIB

A ***Routing Information Base (RIB)*** stores all IP routing information, and is responsible for next-hop selection from multiple available routes received from different peers. A subset of the RIB information, i.e., the address prefixes and their selected next-hops, are installed in the ***Forwarding Information Base (FIB)*** for fast lookup during data forwarding.

Definition 1. *Given an IP address d and a FIB F, let $LPM(F, d)$ denote d's Longest Prefix Match, and $nexthop(F, p)$ denote the next-hops for prefix p. We define $nexthop(F, d) = nexthop(F, LPM(F, d))$. It is possible that d does not have any match in the FIB, i.e., $LPM(F, d) = NULL$, and packets destined to d will be dropped.*

As an example, Table I(a) shows a FIB F with five entries. For address $141.225.48.7$, $LPM(F, 141.225.48.7) =$

This work was supported by NSF Grants 0721645 and 0721863.

TABLE I. FIB ENTRIES BEFORE AND AFTER AGGREGATION

(a) Original FIB Entries

Label	Prefix	Next-hop
A	141.225.0.0/16	1
B	141.225.64.0/18	1
C	141.225.32.0/19	1
D	141.225.96.0/19	2
E	141.225.48.0/20	2

(b) Aggregated FIB Entries

Label	Prefix	Next-hop
A	141.225.0.0/16	1
D	141.225.96.0/19	2
E	141.225.48.0/20	2

$141.225.48.0/20$, and $nexthop(F, 141.225.48.7) = nexthop(F, 141.225.48.0/20) = \{2\}$.

B. FIB Aggregation and Forwarding Correctness

FIB aggregation is to aggregate a FIB into one with fewer number of entries while ensuring "forwarding correctness", *i.e.*, the aggregated FIB should not change the next-hops that packets take to reach their destinations. All the FIB aggregation algorithms proposed in this paper satisfy *strong forwarding correctness* as defined below. Note that even if two algorithms satisfy the same type of forwarding correctness, they may reduce a FIB into different sizes depending on what aggregation opportunities they exploit.

Definition 2. *Given a FIB F, another FIB F' satisfies **Strong Forwarding Correctness** with respect to F if and only if the following conditions hold: (1) any non-routable address in F will remain non-routable in F',* i.e.*, if $LPM(F, d) = NULL$, then $LPM(F', d) = NULL$; (2) the next-hop of any routable address in F will remain the same in F',* i.e.*, if $LPM(F, d) \neq NULL$, $nexthop(F', d) = nexthop(F, d)$. If only the second condition holds, we say that F' satisfies **Weak Forwarding Correctness** with respect to F (this means a non-routable address in F can become routable in F').*

C. Optimal Routing Table Constructor (ORTC)

Our algorithms are based on the Optimal Routing Table Constructor (ORTC) [12], a one-time aggregation algorithm that minimizes the FIB size with strong forwarding correctness. The basic ORTC algorithm uses a binary tree to store FIB entries and traverses the tree three times to produce the aggregated FIB. As we will show in Section III-A, ORTC can be implemented using a patricia trie [16] with two tree traversals. However, for ease of illustration, we describe the basic ORTC algorithm using a binary tree and three passes.

We use the FIB in Table I(a) as our example. Figure 1(a) shows the initial binary tree with seven nodes. Five of the nodes, A, B, C, D, and E, correspond to the FIB entries in Table I(a). We call them "real" nodes, while the other two nodes, F and H, are called "auxiliary" nodes.

The first pass is a depth-first traversal in pre-order to normalize the tree, so that all the nodes have zero or two children. The expanded nodes have the same next-hops as their nearest ancestors that are real nodes. Figure 1(b) depicts the process for pass 1. Node G, I, J and K are the expanded leaf nodes, and they have the same next-hops as their nearest real ancestors A, C, A, and B, respectively.

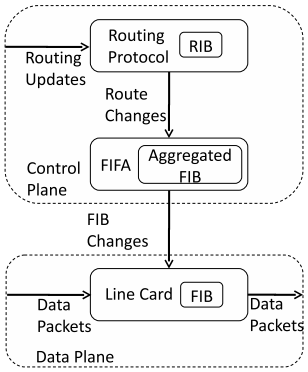


Fig. 2. Relationship between FIFA and other router components

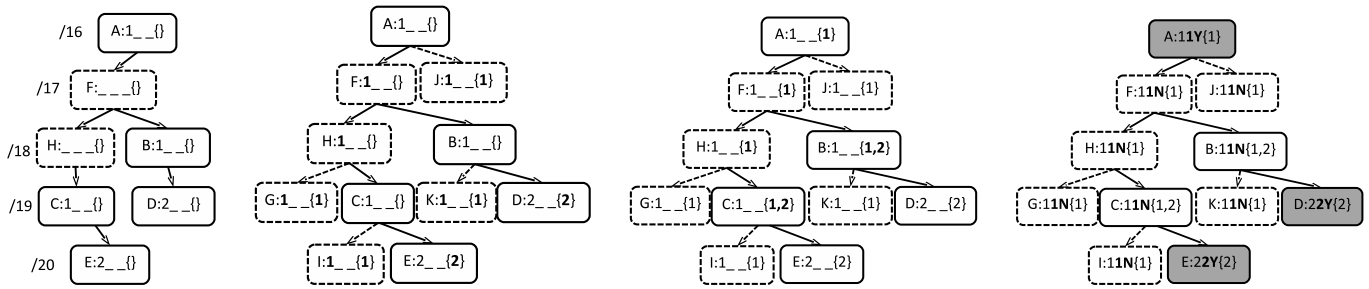
The second pass is a depth-first traversal in post-order to merge next-hops, in which two children merge their next-hop sets to form their parent's next-hop set. If the two children have one or more common next-hops, the merging uses an intersection operation, otherwise, it uses a union operation. Figure 1(c) depicts the merging process. For example, E and I have no common next-hops, so their parent C's next-hop set is $\{1,2\}$, the union of $\{1\}$ and $\{2\}$. Another example is H, whose next-hop set $\{1\}$ is the intersection of C's next-hop set $\{1,2\}$ and G's next-hop set $\{1\}$.

The third pass is a depth-first traversal in pre-order to select each node's next-hop and form the aggregated FIB. The root node's next-hop is randomly selected from its next-hop set (the original next-hop may be preferred for stability). From then on, if a node's selected next-hop h appears in its child's next-hop set, then the child should have h as its selected next-hop, so that the child will not be loaded into the FIB. Otherwise, the child's next-hop is randomly selected from its next-hop set, and the child will be loaded into the FIB.

Figure 1(d) shows the results after pass three. Root A has 1 as its selected next-hop. Since its children F and H have 1 in their next-hop set, they also have 1 as their selected next-hops and, as such, they will not appear in the aggregated FIB. On the other hand, D's selected next-hop (2) is different from that of its parent B (1), so it must be put into the aggregated FIB. Table I(b) shows the final prefixes and their next-hops.

III. DESIGN

We aim to develop FIB aggregation algorithms that are practical to use in a real production network. First, they should reduce the FIB size sufficiently to postpone the upgrading of FIB memory in line cards by several years. Second, they should handle route changes fast as a router may need to handle a large number of routing changes during routing convergence. Third, they should not incur a large number of FIB changes per routing update. According to Francois *et al*. [17], the time required to update a FIB entry in a real router is about $100\mu s$. Since one route change may result in multiple FIB changes on an aggregated FIB, it would be desirable to minimize such FIB changes. Finally, we would like to maintain *strong forwarding correctness* (see Section II) to avoid potential looping problems associated with weak forwarding correctness.



(a) Initial Tree (b) After pass 1 (c) After pass 2 (d) After pass 3

Fig. 1. ORTC Aggregation Algorithm. There are four fields for each node from left to right: original next-hop, selected next-hop, FIB status (Y: IN_FIB, N: NON_FIB), and next-hop set. A bold font denotes a field updated in the current step. A solid rectangle denotes a *real* node from the unaggregated FIB. A dashed rectangle denotes an auxiliary node generated either as a glue node or for optimization purposes. A grey node denotes a node with IN_FIB status.

When a router starts up, FIFA uses our improved version of ORTC ([12] and Section II) to build the initial aggregated FIB. When a new routing update arrives, first the routing protocol will update the RIB and then FIFA will apply each resulting route change to the aggregated FIB, which may generate one or more FIB changes. FIFA then installs these FIB changes in the line card. Figure 2 illustrates this process.

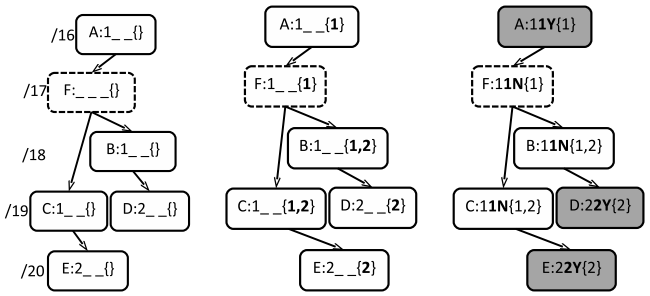
FIFA is composed of three algorithms, FIFA-S, FIFA-T and FIFA-H, and ISPs can choose one based on their concerns. ***FIFA-S*** keeps the FIB size smallest among the three, with very light FIB bursts and no FIB re-aggregation. ***FIFA-T*** is the fastest among the three, with relatively small number of FIB changes and fast re-aggregation. ***FIFA-H*** is a hybrid approach combining the advantages of both FIFA-S and FIFA-T. It has medium time cost compared to the other two schemes, and much lighter FIB burst than FIFA-T. Moreover, it does not perform any re-aggregations.

In the rest of this section, we describe our improved version of ORTC and the three FIFA algorithms.

A. Improving ORTC Efficiency

FIFA is based on the ORTC algorithm, but it addresses two inefficiencies in the latter: (1) the basic ORTC algorithm traverses the FIB tree three times, so it can be quite slow for a large FIB; and (2) ORTC uses a binary tree structure that could consume more memory than necessary when there are large gaps between address prefixes and the large number of tree nodes means slower tree traversals. We improved ORTC using two passes on a Patricia Trie [16]. A Patricia Trie is a space-optimized tree in which a child prefix can be longer than its parent prefix by more than one, thus eliminating unnecessary internal nodes. For example, Figure 3(a) shows the Patricia Trie representation of Table I(a) – node C has a prefix length 19 while its parent F has a prefix length of 17. We tested both implementations using RouteView's data [18]. For the routing table of router 4.69.184.193 on 1/1/2011 (332,588 entries), our implementation is 2.5 times faster and uses only 44% of the memory consumed by the original implementation.

In order to distinguish the patricia trie-based ORTC algorithm from the basic ORTC, we use Round One (Figure 3(b)) and Round Two (Figure 3(c)) to represent its new passes. *Round One* is a depth-first traversal in post-order to merge next-hops (as in pass two) without normalizing the tree (otherwise we get a complete binary tree). *Round Two* is a depth-first traversal in pre-order to select next-hops (as in pass three) and it adds new tree nodes to maintain forwarding correctness.



(a) Initial Trie (b) After round one (c) After round two

Fig. 3. Improved ORTC Aggregation Algorithm using Patricia Trie

For Round One, in order to merge the next-hops correctly without expanding the trie, we compute a node's next-hop set by merging what would be the next-hop sets of its *imaginary* children if there is a complete binary tree. Let $S(n)$ be the next-hop set of node n, $S_l(n)$ and $S_r(n)$ be the next-hop set of n's imaginary left and right child, respectively. Then $S(n) = merge(S_l(n), S_r(n))$. In Figure 3(b), $S(n)$ is the last value associated with each node.

Below we explain how to compute $S_l(n)$. Let $H(n)$ be the original next-hop of n, and d be the difference between the prefix length of a node and that of its *actual* left child. There are four possible cases: no left child, $d = 1$, $d = 2$, and $d > 2$. In each case, the calculation follows a simple rule explained below (all the examples refer to the FIB tree in Figure 3(b)). The rules can be proven by expanding the part of trie that includes the parent and the child into a complete binary structure and applying the merging rules to it.

1) No left child: S_l is derived from the original next-hop of the parent node, since the child was to be created from tree normalization. For example, C has no left child, so $S_l(C) = \{H(C)\} = \{1\}$.
2) $d = 1$: S_l is the next-hop set of the actual left child. For example, $d = 1$ for A and F, so $S_l(A) = S(F) = \{1\}$.
3) $d = 2$: S_l is the merged next-hops of the parent's original next hop and the actual left child's next-hop set. For example, $d = 2$ between F and C, so $S_l(F) = \{H(F)\} \cap S(C) = \{1\} \cap \{1, 2\} = \{1\}$.

4) $d > 2$: S_l is a set containing only the original next-hop of the parent node.

We then obtain $S_r(n)$ using the same procedure, and calculate $S(n)$ by merging $S_l(n)$ and $S_r(n)$. For example, since $d = 1$ between F and B, $S_r(F) = S(B) = \{1, 2\}$. Therefore, $S(F) = S_l(F) \cap S_r(F) = \{1\} \cap \{1, 2\} = \{1\}$.

Round Two goes through similar steps as pass three to select the next-hop of each node. In addition, it creates a new node when $H(n) \neq H'(n)$, where $H(n)$ and $H'(n)$ are the original and selected next-hop of node n, and one of the following two conditions is satisfied:

1) $d \geq 2$: if n has a left (right) child with prefix length greater than n's length by at least 2, then a left (right) child under n is created.
2) One child is missing: if n has no left (or right) child, then a left (or right) child under n is created.

Otherwise, we do not need to create new nodes. After the two rounds, we obtain the same set of aggregated FIB entries as the original ORTC does with much fewer nodes in general. For example, In Figure 3(c), we did not create any new node because $H(n) = H'(n)$ for all the nodes, and only six nodes are created compared to 11 nodes in Figure 1(d).

B. FIFA-S

FIFA-S keeps the aggregated FIB size optimal after every update. A naive way to do so is to perform the ORTC aggregation on the entire FIB trie upon every update, but this would be too time-consuming. A better approach is to update only those parts of the FIB trie that may have been impacted by the update. We follow this approach in both the optimal size update handling algorithm (BasicOptSize) we proposed in 2010 [14] and FIFA-S, but FIFA-S is eight times faster than BasicOptSize (see Section IV) and its heaviest FIB burst (i.e., number of FIB changes caused by a single route change) is only 1/10 of that in BasicOptSize. Below we first describe how BasicOptSize works and then show FIFA-S' improvements.

The BasicOptSize algorithm goes through the following steps, after applying the update to the corresponding node:

1) Step A: on the subtree rooted at the *updated node*, merge the next-hops using a depth-first traversal in post-order. This is basically a *Round One* operation on a subtree;
2) Step B: for each ancestor node *above* the updated node, merge its next-hops until the node's new next-hop set is the same as its old next-hop set. We call this node the "highest changed node";
3) Step C: on the subtree rooted at the *highest changed node*, select the next-hops using a depth-first traversal in pre-order. This is a *Round Two* operation on a subtree.

In the longer version of this paper [19], we prove that the above steps can keep the aggregated FIB optimal.

To illustrate BasicOptSize, we add a new entry to our example FIB (H in Table II(a)). Figure 4 shows the FIB trie after updating node H (its type is changed to REAL and its next-hop from 1 to 3). Figure 5 shows Steps A, B and C. After

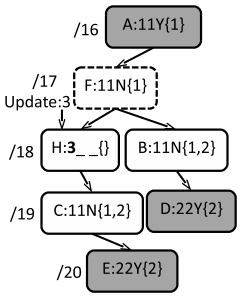


Fig. 4. Updating node H upon receiving a new route

TABLE II. UNAGGREGATED AND AGGREGATED FIB ENTRIES AFTER AN UPDATE

(a) Original FIB Entries

Label	Prefix	Next-hop
A	141.225.0.0/16	1
B	141.225.64.0/18	1
C	141.225.32.0/19	1
D	141.225.96.0/19	2
E	141.225.48.0/20	2
H	141.225.0.0/18	3

(b) Aggregated FIB Entries

Label	Prefix	Next-hop
A	141.225.0.0/16	1
D	141.225.96.0/19	2
E	141.225.48.0/20	2
G	141.225.0.0/19	3

Step C, the aggregated FIB contains three of the five original entries, A, D, and E, and a new entry G (Table II(b)).

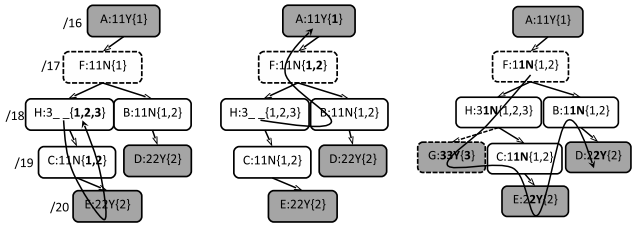
Figure 5 shows that Step B and C need to update the entire subtree rooted at H and F, respectively. To reduce the number of nodes visited on these subtrees, FIFA-S takes advantage of the following two properties (see [19] for their proofs):

Property 1*: The result of Step A will be the same without updating any subtrees rooted at $REAL$ nodes.*

Property 2*: The result of Step C will be the same without updating any subtree with these properties: (1) the next-hop sets did not change in any nodes on the subtree in Step A; and (2) the selected next-hop of the subtree root did not change.*

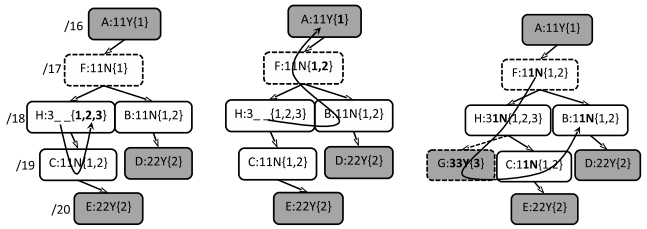
Moreover, FIFA-S adopts the following rules and it has considerably fewer FIB changes than BasicOptSize (Section IV).

Property 3*: In Step C, selecting a node's next-hop from its next-hop set using the following prioritized rules can reduce*



(a) Step A (b) Step B (c) Step C

Fig. 5. Steps in BasicOptSize



(a) Step A' (b) Step B' (c) Step C'

Fig. 6. Steps in FIFA-S

the number of FIB changes: (a) the next-hop selected by the nearest ancestor with IN_FIB status (this is for FIB size optimization); (b) the old selected next-hop; (c) the original next-hop; and (d) if none of those are found in the next-hop set, sort the set and pick the first one instead of random selection.

We call the improved procedures *Step A', B' and C'*. Figures 5 and 6 show that (1) BasicOptSize and FIFA-S have the same aggregation results; (2) E was skipped in Step A'; and (3) D and E were skipped in Step C'.

We present the pseudo code of FIFA-S in Procedures 1 - 6. Function *mergeNexthopsBelowNode* is Step A', *mergeNexthopsAboveNode* is Step B' and *selectNexthop* is Step C'. Note that we associate a flag *optimal* with each node to indicate whether Step C' is needed in the subtree of this node.

In the next two sections, we show how Properties 1-3 can be used in FIFA-T and FIFA-H to reduce computation overhead and keep the aggregated trie stable.

Procedure 1 $main(type)$ function

```
1: Build initial FIB trie based on unaggregated FIB
2: Run improved ORTC on the FIB trie to obtain aggregated FIB
3: for each update do
4:     if Announcement then
5:         Lookup the corresponding node and create it if non-existent
6:         Update the next-hop of the current node
7:         node.type ← REAL
8:     else
9:         Lookup the corresponding node and return if non-existent
10:        Remove the next-hop of the current node
11:        node.type ← AUXILIARY
12:    if (type = ALG_FIFA_T) ∨ (type = ALG_FIFA_H) then
13:        node.optimal ← 0 for all ancestors of the current node
14:    switch (type)
15:    case ALG_FIFA_S:
16:        FIFA_S(node)
17:    case ALG_FIFA_T:
18:        FIFA_T(node)
19:    case ALG_FIFA_H:
20:        FIFA_H(node)
21:    end switch
```

Procedure 2 $FIFA\_S(node)$ function

```
1: realAncestor ← nearestRealAncestor(node)
2: mergeNexthopsBelowNode(node, realAncestor)
3: highestNode ← mergeNexthopsAboveNode(node, ALG_FIFA_S)
4: infibAncestor ← nearestINFIBAncestor(highestNode)
5: selectNexthop(highestNode, infibAncestor)
```

C. FIFA-T

FIFA-T aims to shorten the FIB update time by localizing the changes on the FIB trie while maintaining strong forwarding correctness. The trade-off is that the FIB size will

Procedure 3 $mergeNexthopsBelowNode(node, realAncestor)$

```
1: node.optimal ← 0
2: l ← node.l
3: r ← node.r
4: if (l ≠ NULL) ∧ (l.type ≠ REAL) then
5:     mergeNexthopsBelowNode(l, realAncestor)
6: if (r ≠ NULL) ∧ (r.type ≠ REAL) then
7:     mergeNexthopsBelowNode(r, realAncestor)
8: if (node.type ≠ REAL) then
9:     node.originalNexthop ← realAncestor.originalNexthop
10: node.mergedNexthops ← merge(l, r)
```

Procedure 4 $mergeNexthopsAboveNode(node, type)$

```
1: parent ← node.parent
2: while parent do
3:     if (type = ALG_FIFA_H) ∧ (parent.length ≤ CAP) then
4:         Return node
5:     old ← parent.mergedNexthops
6:     new ← merge(parent.l, parent.r)
7:     if old = new then
8:         Return node
9:     node ← parent
10:    parent ← node.parent
11:    if type = S then
12:        node.optimal ← 0
13: Return node
```

Procedure 5 $selectNexthop(node, ancestor)$

```
1: oldStaus ← node.status
2: oldNexthop ← node.selectedNexthop
3: if ancestor.selectedNexthop ∈ node.mergedNexthops then
4:     node.selectedNexthop ← ancestor.selectedNexthop
5:     node.staus ← NON_FIB
6: else
7:     if oldNexthop ∈ node.mergedNexthops then
8:         node.selectedNexthop ← oldNexthop
9:     else if node.originalNexthop ∈ node.mergedNexthops then
10:        node.selectedNexthop ← node.originalNexthop
11:    else
12:        node.selectedNexthop ← node.mergedNexthops[0]
13:    node.staus ← IN_FIB
14: updateFIB(oldStatus, oldNexthop, node)
15: if (oldNexthop = node.selectedNexthop) ∧ (node.optimal = 1)
    then
16:    Return
17: if node.status = IN_FIB then
18:    ancestor ← node
19: if (node.l = NULL) ∧ (node.r = NULL) then
20:    Return
21: if node.selectedNexthop ≠ node.originalNexthop then
22:    generateNewNode(node)
23: if node.l ≠ NULL then
24:    selectNexthop(node.l, ancestor)
25: if node.r ≠ NULL then
26:    selectNexthop(node.r, ancestor)
27: node.optimal ← 1
```

Procedure 6 $updateFIB(oldStatus, oldNexthop, node)$

```
1: if oldStatus ≠ node.status then
2:     if node.status = NON_FIB then
3:         Delete the prefix and next-hop from FIB
4:     else
5:         Add the prefix and next-hop to FIB
6: else if oldNexthop ≠ node.selectedNexthop then
7:     if node.status = IN_FIB then
8:         Update the corresponding next-hop to FIB
```

not be optimal. As more updates come, the FIB size will increase until it reaches a threshold, *e.g.*, 90% of the FIB memory in the line card. At this point, a re-aggregation is performed on the FIB trie from the root to optimize the FIB size. On the surface, FIFA-T is very similar to the minimal time update handling algorithm (BasicMinTime) we proposed [14]. However, there are two important differences that make FIFA-T more efficient: (a) FIFA-T utilizes the three properties described in Section III-B; and (b) FIFA-T's re-aggregation is performed on the aggregated FIB trie, but BasicMinTime has to destroy the old aggregated FIB trie, and build a new one from the unaggregated FIB. Our results show that it uses 40% less time than BasicMinTime and generates only 1.1 FIB changes per routing update.

FIFA-T works as follows: (1) before the threshold is reached, perform the following (the less efficient procedures in BasicMinTime are called *Step X* and *Y*) –

- Step X': on the subtree rooted at the *updated node*, merge the next-hops using a depth-first traversal in post-order, skipping REAL nodes and their subtrees (based on Property 1);
- Step Y': on the subtree rooted at the *updated node*, select the next-hops (following rules based on Property 3) using a depth-first traversal in pre-order, skipping REAL nodes with optimal flag set to 1 as well as their subtrees (based on Property 2).

(2) when the threshold is reached, re-aggregate the trie from its root incorporating the three properties to obtain an optimal trie. The pseudo code is in Procedures 1, 7, and 3 - 6.

Procedure 7 $FIFA\_T(node)$ function

```
1: if Threshold then
2:     Do re-aggregation on the FIB trie from the root
3: else
4:     realAncestor ← nearestRealAncestor(node)
5:     mergeNexthopsBelowNode(node, realAncestor)
6:     infibAncestor ← nearestINFIBAncestor(node)
7:     selectNexthop(node, infibAncestor)
```

Figure 7 illustrates the differences between BasicMinTime and FIFA-T, e.g., node E was skipped in Step X' and Y'.

D. FIFA-H

In addition to FIFA-S and FIFA-T, we propose FIFA-H, a hybrid scheme that achieve a good balance among aggregation speed, FIB size and number of FIB changes. In this approach, a FIB size threshold and a CAP are set at the beginning. For each update, FIFA-H performs three steps - U, V, W (or W') as follows (Figure 8):

- Step U: merge the next-hops below the updated node (same as Step A' in FIFA-S and Step X' in FIFA-T);
- Step V: merge the next-hops above the updated node up to the highest changed node whose prefix length is less than or equal to CAP, called the CAP node, which limits the computation overhead and the number of FIB changes compared to FIFA-S;
- Step W or W': if the threshold is not reached, this step (W) performs next-hop selection on the subtree rooted at the current updated node (*SaveTime* mode). Otherwise, this step (W') will start from the CAP node for next-hop selection (*ReduceSize* mode).

FIFA-H incurs less computation time and fewer FIB changes than FIFA-S, and has smaller FIB bursts than FIFA-T (Section IV). It has no lengthy re-aggregations, thus avoiding potential problems during re-aggregation, e.g., packet losses.

Procedure 8 $FIFA\_H(node)$ function

```
1: realAncestor ← nearestRealAncestor(node)
2: mergeNexthopsBelowNode(node, realAncestor)
3: capNode ← mergeNexthopsAboveNode(node, ALG_FIFA_H)
4: if ThresholdReached then
5:     node ← capNode
6: infibAncestor ← nearestINFIBAncestor(node)
7: selectNexthop(node, infibAncestor)
```

IV. Evaluation

In this section, we evaluate the performance improvement of FIFA over BasicOptSize, BasicMinTime, as well as SMALTA [15], another ORTC-based FIB aggregation scheme. We also compare the three FIFA algorithms so that users can choose the right algorithm based on their own needs. We verified the correctness of our results by checking that every address has the correct next-hop after aggregation.

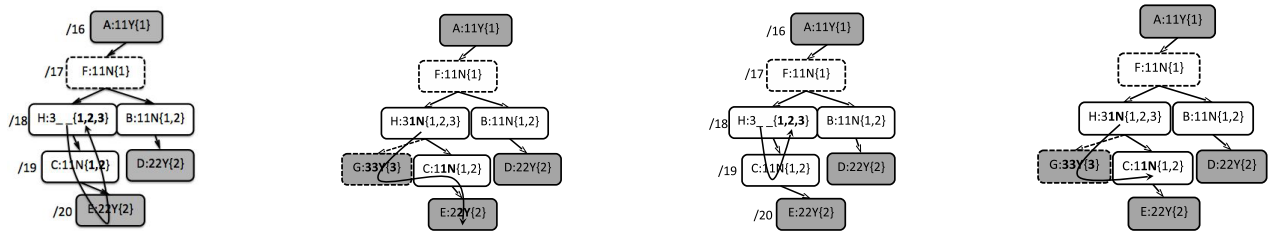
A. Methodology

We used RIBs and routing updates from 01/01/2011 to 12/31/2011 in the Routeviews [18] route-views2 data archive. Since the routing updates do not contain next-hop IP address information, we use next-hop ASes to approximate next-hop routers (as in [13], [14]) and we have used internal routing information from a tier-1 ISP to verify that our approach closely approximates the results using IGP next-hops. In order to show the worst-case performance, we present the results from 4.69.184.193, a router in the tier-1 ISP *Level 3*, because this router has the most number of AS neighbors (2876 and 3151 on 01/01/2011 and 12/31/2011, respectively) among all 36 routers. In general, more neighbors means more next-hops the prefixes can have, which may lead to lower FIB aggregation performance. In practice, a router has tens or at most hundreds of interfaces.

We use the following four performance metrics: (1) FIB Size: total number of entries in FIB; (2) Time Cost: time to apply routing changes to the FIB including re-aggregation time, if any; (3) FIB Changes: total number of FIB updates caused by all routing updates; and (4) FIB burst: number of FIB changes caused by one route change. The evaluation was done on a machine with an Intel Core 2 Quad 2.83GHz CPU.

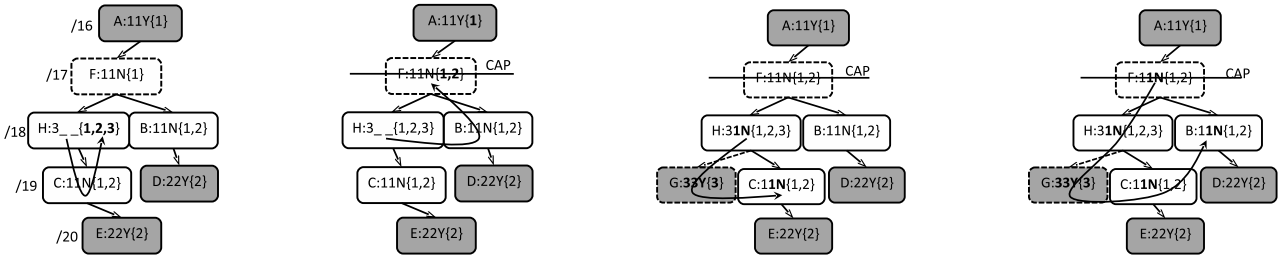
B. Summary of Findings

FIFA-S improves the time efficiency of BasicOptSize by 8.22 times and keeps the FIB size optimal. It is mostly useful when the FIB memory size is close to its optimal aggregated size, when FIFA-T will trigger too many re-aggregations. FIFA-T is the fastest among the three schemes; it is suitable when the FIB memory is much larger than the optimal aggregated size. FIFA-H is a well-balanced scheme with medium running time and FIB burst size. Compared with SMALTA,



(a) Step X in BasicMinTime (b) Step Y in BasicMinTime (c) Step X' in FIFA-T (d) Step Y' in FIFA-T

Fig. 7. Steps in BasicMinTime and FIFA-T. Step X and Y are steps for BasicMinTime, while Step X' and Y' are steps for FIFA-T.



(a) Step U (b) Step V (c) Step W (d) Step W'

Fig. 8. FIFA-H: the first two steps are Step U and V. The third step is Step W before reaching the threshold and Step W' after reaching the threshold.

all FIFA algorithms are faster and have smaller FIB bursts. In addition, ***FIFA-T and FIFA-H incur fewer total number of FIB changes than SMALTA***. Technically, there are mainly three factors leading to the fast FIB aggregation in FIFA. First of all, SMALTA requires rebuilding the FIB aggregation tree from scratch during re-aggregation, which is time consuming. In FIFA-T, the re-aggregations are very fast, because it does not require rebuilding the FIB aggregation. In FIFA-S and FIFA-H, there is no re-aggregation. Secondly, SMALTA uses a binary tree data structure, but we use a patricia trie. The binary tree has many more nodes to traverse than the patricia trie, thus incurring more computation overhead. Thirdly, FIFA has the two important features which are not applicable to SMALTA and they can further help reduce more redundant node accesses but keep the same aggregated size.

C. FIFA-S vs. BasicOptSize

We first compare FIFA-S with BasicOptSize. Figure 9(a) shows the FIB size. Since both schemes achieve optimal FIB size, their lines overlap with each other ending below 150,000. The top line shows the unaggregated FIB size, which increased from 332,588 to 378,728 during the year. In other words, ***either scheme reduced the FIB size by about 60%***. If the unaggregated FIB size increases at the current rate (about 13.9%), it will take 7.5 more years for the aggregated FIB size to reach the current unaggregated FIB size (as of 12/31/2011).

Figure 9(b) shows that ***FIFA-S is more than 8.22 times faster*** (108s in total or 2μs/update) ***than BasicOptSize*** (888s in total or 16.4μs/update). The time cost of FIFA-S is very close to the bottom line, which corresponds to the time cost to update an unaggregated FIB. This suggests that it is feasible to deploy FIFA-S in an operational router.

Figure 9(c) shows that the total number of FIB changes in FIFA-S is only about 1.8 times of that in an unaggregated FIB, and this ratio is very stable.

Table III shows the FIB burst distribution. In both schemes, about 98% of the FIB bursts have no more than 10 FIB changes. Moreover, ***FIFA-S' largest FIB burst (568) is less than 10% of that in BasicOptSize (6,226)***.

D. FIFA-T vs. BasicMinTime

In Figure 10 and Table IV, we compare FIFA-T with BasicMinTime and observe the following: (a) their FIB size oscillates between the optimal size and the configured threshold, and FIFA-T triggers only 9 fast re-aggregations during the entire year (Figure 10(a)); (b) ***FIFA-T uses 40% less time than BasicMinTime*** (Figure 10(b)); and (c) FIFA-T's largest FIB burst is much smaller than that in BasicMinTime (Table IV).

E. Comparison among FIFA Algorithms and with SMALTA

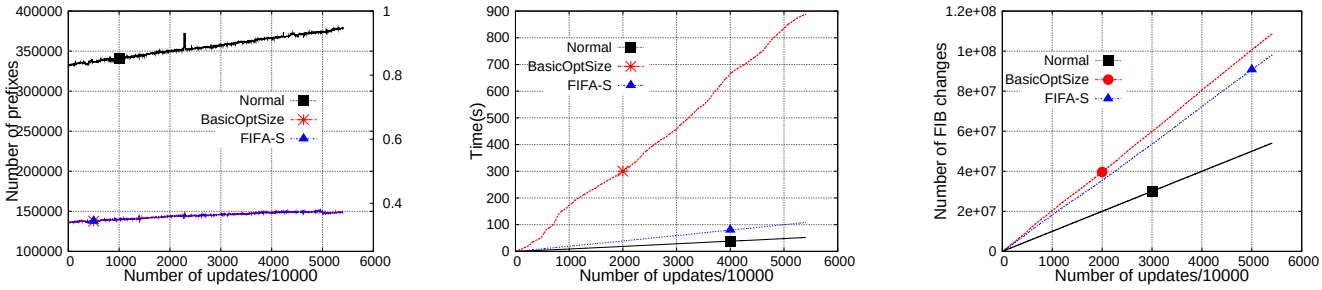
Below we compare FIFA algorithms and SMALTA.

a). FIB Size: Figure 11(a) shows that (1) FIFA-S has the smallest FIB size; (2) FIFA-T and SMALTA oscillate between the optimal size and the threshold, and (3) FIFA-H tends to stay around the threshold with no re-aggregation.

b). Time Cost: Figure 11(b) shows that (1) SMALTA takes the most time – 237.23s, which includes 11 full tree re-aggregations (158.62s or 14.3s per re-aggregation); (2) FIFA-T is the fastest – 66s, which includes 9 fast tree re-aggregations with 0.2s for each (FIFA-T is 70 times faster than SMALTA in re-aggregation efficiency); and (3) FIFA-S (108s) and FIFA-H (100s) have similar time cost.

c). FIB Changes: Figure 11(c) shows that (1) FIFA-T and FIFA-S have the lowest and highest total number of FIB changes, respectively; (2) SMALTA has slightly more total number of FIB changes than FIFA-H.

d). FIB Bursts: Table V shows that (1) most route changes cause zero or one FIB change, and about 99% FIB bursts have

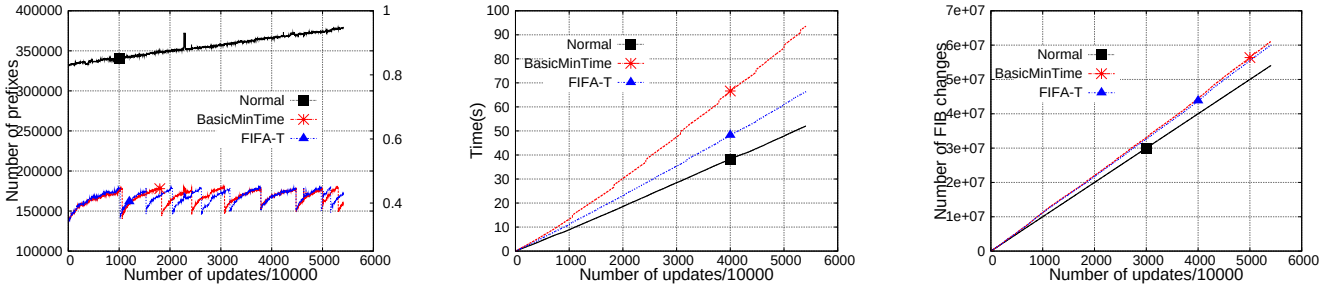


(a) FIB Size (b) Time Cost (c) FIB Changes

Fig. 9. FIFA-S vs. BasicOptSize (*Normal* refers to no aggregation)

TABLE III. FIB BURST DISTRIBUTION COMPARISON BETWEEN FIFA-S AND BASICOPTSIZE

Burst Size	Min	Max	Median	=0	≤1	≤10	All
BasicOptSize	0	6,226	1	6,914,934 (12.78%)	38,185,015 (70.58%)	52,795,683 (97,59%)	54,095,965 (100%)
FIFA-S	0	568	1	6,961,449 (12.87%)	38,645,578 (71.43%)	53,318,607 (98.56%)	54,095,965 (100%)



(a) FIB Size (b) Time Cost (c) FIB Changes

Fig. 10. FIFA-T vs. BasicMinTime (*Normal* refers to no aggregation)

TABLE IV. FIB BURST DISTRIBUTION COMPARISON BETWEEN FIFA-T AND BASICMINTIME

Burst Size	Min	Max	Median	=0	≤1	≤10	All
BasicMinTime	0	149,815	1	6,080,983 (11.24%)	49,611,562 (91.71%)	53,913,320 (99.66%)	54,095,973 (100%)
FIFA-T	0	69,526	1	6,150,664 (11.36%)	49,704,177 (91.88%)	53,919,736 (99.67%)	54,095,965 (100%)

less than 10 FIB changes; (2) FIFA-T usually has small FIB bursts, but they can get very large (69,526); (3) with FIFA-S and FIFA-H, the FIB bursts have at most 568 and 1,182 FIB changes, respectively; and (4) SMALTA has the largest FIB burst (72,856).

F. FIB Lookup Performance and Memory Saving Efficiency

We use the software reference design of Tree Bit Map [20] to obtain FIB lookup time and memory usage. We found that (1) the FIB lookup time on the aggregated table is slightly better than that on the full table; and (2) the FIB aggregation ratio calculated based on actual memory usage is slightly higher than that calculated based on the number of FIB entries, but still provides considerable memory savings. More specifically, the former ranges from 21.2% to 52.2% with a median of 49.6%, while the latter ranges from 13.28% to 39.57% with a median of 36.74%.

V. RELATED WORK

Optimal Routing Table Constructor(ORTC) [12] was proposed by Draves *et al.* 1997. It aggregates a routing table into its optimal size using three passes over a binary tree, while maintaining strong forwarding correctness. However, this algorithm does not include any update handling mechanism.

Zhao *et al.* proposed four aggregation algorithms (Level1 - Level4) [13]. Level1 and Level2 algorithms maintain strong forwarding correctness, but they do not optimize the FIB size. Level3 and Level4 achieve weak forwarding correctness, by introducing extra routable space.

In 2009, Karpilovsky proposed an incremental FIB update algorithm [21] based on ORTC. This algorithm is similar to FIFA-S. However, it needs three passes to handle an update, and it normalizes all affected ancestors of an updated node, both of which introduce considerable computational overhead.

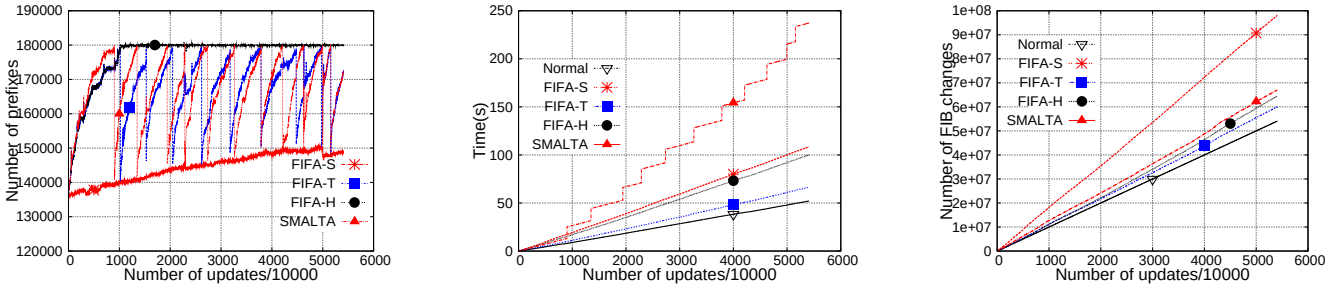
In 2010, we proposed two incremental FIB aggregation algorithms [14] based on ORTC. We showed that FIFA-S and FIFA-T outperform these algorithms in Section IV.

Another very relevant work is SMALTA [15], in which Uzmi *et al.* implemented a different update handling algorithm based on ORTC. They utilize a binary tree rather than Patricia Trie and update only affected nodes without optimizing the subtree rooted at the updated node.

Li *et al.* proposed an FIB aggregation scheme with multiple selectable next-hops [22], which is geared toward FIBs

TABLE V. FIB BURST DISTRIBUTION COMPARISON AMONG THREE FIFA SCHEMES AND SMALTA

Burst Size	Min	Max	Median	=0	≤1	≤10	All
FIFA-S	0	568	1	6,961,449 (12.87%)	38,645,578 (71.43%)	53,318,607 (98.56%)	54,095,965 (100%)
FIFA-T	0	69,526	1	6,150,664 (11.36%)	49,704,177 (91.88%)	53,919,736 (99.67%)	54,095,965 (100%)
FIFA-H	0	1,182	1	6,232,328 (11.52%)	48,997,278 (90.57%)	53,784,161 (99.42%)	54,095,965 (100%)
SMALTA	0	72,856	1	4,456,410 (8.23%)	48,297,973 (89.28%)	53,873,603 (99.58%)	54,095,976 (100%)



(a) FIB Size (b) Time Cost (c) FIB Changes

Fig. 11. FIFA Algorithms vs. SMALTA (*Normal* refers to no aggregation)

with multiple selectable next-hops for each prefix.

Simple Virtual Aggregation [23] installs virtual prefixes which are shorter than real prefixes, such as /6, to legacy routers to control FIB size growth. It can reduce the FIB size on most routers, while the routers that announce the virtual prefixes still need to maintain many specific prefixes.

VI. CONCLUSION

FIB aggregation is a promising direction in controlling the growing FIB size. We have proposed FIFA, a suite of three FIB aggregation algorithms, with significant performance improvement over existing algorithms in terms of time cost, total number of FIB changes and FIB burst size. For our next step, we plan to investigate how to automatically switch among the algorithms based on a set of constraints, e.g., memory size, aggregated FIB size and maximum FIB burst size.

VII. ACKNOWLEDGMENT

We thank Zartash Afzal Uzmi for sharing his SMALTA code with us. We also thank Lixia Zhang and the anonymous reviewers for their feedback.

REFERENCES

[1] V. Fuller, "Scaling Issues with Routing+Multihoming," http://www.vaf.net/~vaf/apricot-plenary.pdf.

[2] "IRTF Routing Research Group." [Online]. Available: http://www.irtf.org/charter?gtype=rg&group=rrg

[3] "IETF Global Routing Operations (GROW)." [Online]. Available: http://www.ietf.org/dyn/wg/charter/grow-charter.html

[4] R. Atkinson and S. Bhatti, "ILNP Architectural Description," Work in Progress, http://tools.ietf.org/html/draft-irtf-rrg-ilnp-arch-06.

[5] D. Jen, M. Meisel, H. Yan, D. Massey, L. Wang, B. Zhang, and L. Zhang, "Towards A Future Internet Architecture: Arguments for Separating Edges from Transit Core," in *Proc. ACM HotNets*, 2008.

[6] D. Jen, M. Meisel, D. Massey, L. Wang, B. Zhang, and L. Zhang, "APT: A Practical Tunneling Architecture for Routing Scalability," UCLA, Technical Report 080004, 2008. [Online]. Available: http://www.cs.ucla.edu/~meisel/apt-tech.pdf

[7] V. Khare, D. Jen, X. Zhao, Y. Liu, D. Massey, L. Wang, B. Zhang, and L. Zhang, "Evolution towards global routing scalability," *IEEE Journal on Selected Areas in Communications*, vol. 28, no. 8, pp. 1363–1375, Oct. 2010.

[8] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis, "Locator/ID Separation Protocol (LISP)," Mar. 2009, Work in Progress, http://tools.ietf.org/html/draft-farinacci-lisp-12.

[9] P. Francis, X. Xu, H. Ballani, D. Jen, R. Raszuk, and L. Zhang, "FIB Suppression with Virtual Aggregation," Jul. 2012, Work in Progress, http://tools.ietf.org/html/draft-francis-intra-va-06.

[10] H. Ballani, P. Francis, C. Tuan, and J. Wang, "Making Routers Last Longer with ViAggre," in *Proc. NSDI*, 2009.

[11] X. Zhao, D. J. Pacella, and J. Schiller, "Routing scalability: an operator's view," *IEEE Journal on Selected Areas in Communications*, vol. 28, no. 8, pp. 1262–1270, Oct. 2010.

[12] R. Draves, C. King, S. Venkatachary, and B. D. Zill, "Constructing Optimal IP Routing Tables," in *Proc. IEEE INFOCOM*, 1999.

[13] X. Zhao, Y. Liu, L. Wang, and B. Zhang, "On the Aggregatability of Router Forwarding Tables," in *Proc. IEEE INFOCOM*, 2010.

[14] Y. Liu, X. Zhao, K. Nam, L. Wang, and B. Zhang, "Incremental forwarding table aggregation," in *Proc. IEEE Globecom*, 2010.

[15] 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. CoNEXT*, 2011.

[16] "Net-Patricia Perl Module." [Online]. Available: http://search.cpan.org/dist/Net-Patricia/

[17] P. Francois, C. Filsfils, J. Evans, and O. Bonaventure, "Achieving sub-second IGP convergence in large IP networks," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 3, pp. 35–44, Jul. 2005.

[18] Advanced Network Technology Center and University of Oregon, "The RouteViews project." [Online]. Available: http://www.routeviews.org/

[19] Y. Liu, B. Zhang, and L. Wang, "Fast Incremental FIB Aggregation (FIFA)," http://www.cs.memphis.edu/~lanwang/fifa-tr.pdf.

[20] W. Eatherton, G. Varghese, and Z. Dittia, "Tree bitmap: hardware/software ip lookups with incremental updates," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 2, pp. 97–122, Apr. 2004. [Online]. Available: http://doi.acm.org/10.1145/997150.997160

[21] E. M. Karpilovsky, "Reducing Memory Requirements for Routing Protocols (thesis)," Princeton University, Technical Report TR-861-09, 2009. [Online]. Available: http://www.cs.princeton.edu/research/techreps/TR-861-09

[22] Q. Li, D. Wang, M. Xu, and J. Yang, "On the scalability of router forwarding tables: Nexthop-Selectable FIB aggregation," in *Proc. IEEE INFOCOM*, 2011.

[23] R. Raszuk, J. Heitz, A. Lo, L. Zhang, and X. Xu, "Simple Virtual Aggregation (S-VA)," Work in Progress, http://tools.ietf.org/search/draft-ietf-grow-simple-va-10.