Incremental Forwarding Table Aggregation

Yaoqing Liu yliu6@memphis.edu
Xin Zhao Kyuhan Nam {zhaox,airkh}@email.arizona.edu
Lan Wang lanwang@memphis.edu
Beichuan Zhang bzhang@arizona.edu

***Abstract*— The global routing table size has been increasing rapidly, outpacing the upgrade cycle of router hardware. Recently aggregating the Forwarding Information Base (FIB) emerges as a promising solution since it reduces FIB size significantly in the short term and it is compatible with any long-term architectural solutions. Because FIB entries change dynamically with routing updates, an important component of any FIB aggregation scheme is to handle routing updates efficiently while shrinking FIB size as much as possible. In this paper, we first propose two incremental FIB aggregation algorithms based on the ORTC scheme. We then quantify the tradeoffs of the proposed algorithms, which will help operators choose the algorithms best suited for their networks.**

I. Introduction

The global Internet routing table has been growing at an alarming rate [8], which appears to outpace the increase in memory size, especially for the special type of memory used in router line cards for fast lookup. Moreover, it forces ISPs to upgrade router hardware at a faster pace, which not only causes higher operational cost to the ISPs, but also makes issues such as power consumption and lookup speed more prominent.

A promising solution to the routing table size problem is FIB aggregation, which combines multiple entries in the forwarding table (FIB) without changing the next hops for data forwarding. This approach is particularly appealing because it can be done by a software upgrade at a router and its impact is limited within the router. It does not require changes to routing protocols or router hardware, nor does it affect multi-homing, traffic engineering, or other network-wide operations. FIB aggregation is a local solution that can be quickly implemented and deployed in the short-term. In the long run, it can co-exist and complement architectural solutions.

The feasibility of FIB aggregation depends on the solution to one critical issue – *how to design* ***incremental FIB aggregation schemes*** *to efficiently handle routing changes*, while reducing FIB size and maintaining correct forwarding behavior. In fact, several router vendors have raised their concern for the overhead of handle routing updates when we discussed FIB aggregation with them. In the simplest approach, one can re-aggregate the FIB from scratch after each routing update. However, routing updates may arrive rapidly under certain conditions. If every update triggers a full aggregation, the computation overhead would be extremely high and the route processor may not be able to process all the routing updates in time. Therefore, any practical FIB aggregation scheme must be able to perform "*incremental update handling*". More specifically, an ideal algorithm should limit its computation to only those FIB entries impacted by each routing update, thus shortening the route processing time and FIB update time.

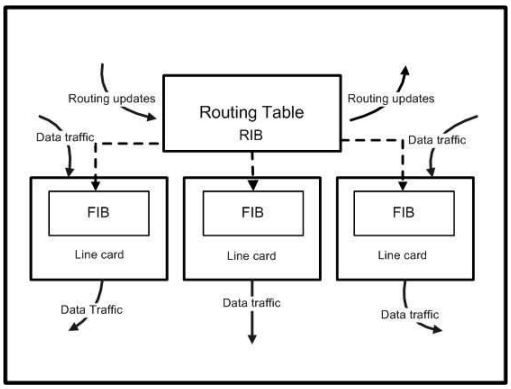


Fig. 1. RIB and FIB

Several FIB aggregation algorithms have been proposed, including Optimal Routing Table Constructor (ORTC) [5], Suri *et al.*'s scheme [9], and our own Level-1 to Level-4 aggregation [11]. However, incremental FIB aggregation has not been studied in depth in the previous work.

In this paper, we add update handling capability to ORTC [5] to make it an incremental FIB aggregation scheme. More specifically, we propose two update handling algorithms for ORTC, one optimizing the FIB size and the other with a short computation time. We then quantify the pros and cons of the proposed update handling algorithms, which will help operators choose the right algorithms best suited for their networks.

We organize the paper as follows. Section II defines some basic terminology and Section III discusses related work. We describe the original ORTC algorithm in Section IV and our proposed incremental FIB aggregation algorithms in Section V. We present our evaluation methodology and results in Section VI – VIII. Section IX concludes the paper.

II. Background

An ***IP address prefix*** summarizes all the IP addresses that share some common bits at the beginning. We use the notation a/l to represent those addresses whose first l bits are equal to a. For example, $141/8$ represents all the addresses from $141.0.0.0$ to $141.255.255.255$. In particular, $0/0$ represents all IP addresses. Given two prefixes $p = a/l$ and $p' = a'/l'$, if $a' = a\{0,1\}^+$ and $l' > l$, we refer to p' as a more specific or longer prefix of p.

A ***Routing Information Base (RIB)*** is the repository in which all IP routing information is stored (Figure 1). For each address prefix, there may be multiple available routes and one or several best routes. Routes may be added or deleted in response to routing updates and the best route(s) will be re-calculated.

A ***Forwarding Information Base (FIB)*** is derived from a RIB but stored in line cards for fast lookup (Figure 1). Line cards usually uses memory with high access speed, which is more expensive than normal memory. A FIB (F) is comprised of a set of forwarding entries, *i.e.*, $F = \{(p, h)\}$, where h is a set of nexthop addresses for forwarding packets to any address in prefix p. We further define $nexthop(F, p)$ to be the nexthops for prefix p according to F.

Given an IP address d and a FIB F, an address prefix $p = a/l \in F$ is the ***Longest Prefix Match (LPM)*** for d, *i.e.*$p = LPM(F, d)$, if and only if the following conditions hold: (1) $d = a\{0, 1\}^*$, and (2) $\forall p' = a'/l' \in F$, if $d = a'\{0, 1\}^*$, then $l' < l$. We define $nexthop(F, d) = nexthop(F, LPM(F, d))$. It is possible that d does not have any match in the FIB, which means $LPM(F, d) = \emptyset$ and packets to d will be dropped.

The most important requirement for FIB aggregation is to ensure "forwarding correctness", *i.e.*, an aggregated FIB should not change the paths that packets take to reach their destinations. We formally define this requirement below.

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 $LPM(F, d) = \emptyset$, then $LPM(F', d) = \emptyset$; (2) the nexthop of any routable address in F will remain the same in F', *i.e.*, if $LPM(F, d) \neq \emptyset$, $nexthop(F', d) = nexthop(F, d)$. If we require only the second condition to hold, F' is said to satisfy ***Weak Forwarding Correctness*** with respect to F. Note that, in this case, a non-routable address in F could become routable in F', resulting in ***extra routable space***.

III. Related Work

Several long-term routing scalability solutions have been proposed in the IRTF Routing Research Group. For example, LISP[6], APT[7], and Ivip[10] use Map-and-Encap to separate edge prefixes from the Internet core. However, implementing these proposals requires changing the routing architecture and modifying protocols. In contrast, FIB aggregation is a local solution. It can be implemented via software upgrade and deployed by individual ISPs and routers. It also complements the long-term solutions.

In [5], Draves *et al.* designed an algorithm that aggregates a FIB to the furthest extent without introducing extra routable space, *i.e.*, ORTC is the optimal algorithm under the strong forwarding correctness requirement. Suri *et al.* extended the ORTC work by considering each routing table entry as a 3-tuple (src, dest, action) [9]. They used dynamic programming to optimize the routing table size. Herrin [2] suggested yet another aggregation algorithm which may introduce extra routable space. Extra routable space is the address space that is not routable in the original FIB but routable in the aggregated FIB. In [11], we designed four levels of FIB aggregation, each level with higher aggregation ratio but also higher algorithmic complexity. By exploiting the tradeoff between extra routable space and aggregation ratio, our Level-4A algorithm can compress FIBs more than ORTC does.

Previous FIB aggregation algorithms, with the exception of ours, do not handle dynamic routing updates efficiently. Although we did introduce an incremental update algorithm in [11], we did not do a thorough investigation. In this paper, we study different approaches to converting a static FIB aggregation scheme to an incremental one. To this end, we use ORTC as an example, since it clearly illustrates the tradeoffs.

IV. FIB Aggregation using ORTC

Routing tables are usually stored in a tree-like data structure, such as a Patricia Trie [1] or an M-trie [4], with the $0/0$ prefix at the root and the most specific prefixes at the leaf level. By augmenting the tree nodes with a few additional fields, a FIB aggregation algorithm can traverse the tree to produce the aggregated FIB.

Draves *et al.* proposed the Optimal Routing Table Constructor (ORTC) algorithm [5], which minimizes FIB size while achieving strong forwarding correctness. Their original algorithm was based on a Binary Tree data structure. We implemented it using a Patricia Trie, which is more memory efficient and more commonly used for storing routing tables.

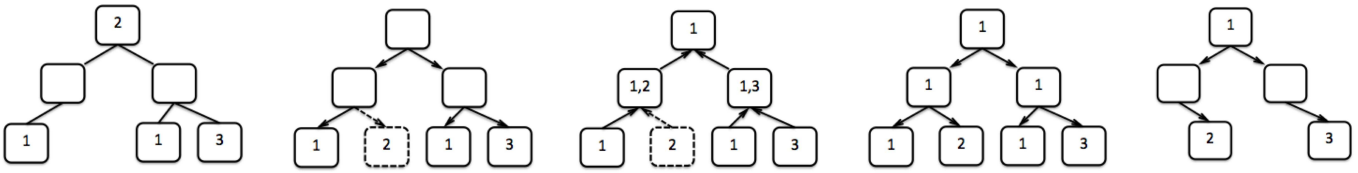
A. Original ORTC Algorithm

The original ORTC algorithm assumes that the routing table is stored in a binary tree. It traverses the tree three times to produce the optimal FIB (Figure 2). The first two tree passes may be combined into one step in an implementation.

The first tree pass (Figure 2(b)) expands the tree so that every node has zero or two children. Each expanded leaf node has the same next hop as that of its nearest ancestor (the ancestor has to be a real prefix in the routing table). This expansion "de-aggregates" the routing table – the routing table is now composed of only the most specific prefixes (*i.e.*, leaf nodes) and their nexthop information, which is an important preparation for the next two passes.

The second pass (Figure 2(c)) is a bottom-up process that calculates the most prevalent next hops at every level of the routing table. If two children share one or more common next hops, their common nexthops will be stored at the parent node as the parent's candidate nexthop set. Otherwise, the union of the children's nexthops will be stored at the parent node as the parent's candidate nexthop set. In other words, suppose l and r are two children of the parent p, then p's candidate nexthop set is computed by either UNION(l, r) if it is not empty or INTERSECTION(l, r) operations otherwise. This process repeats up to the root of the tree.

The third pass (Figure 2(d)) is a top-down process in which each node selects one nexthop from the candidate nexthop set computed by the second pass. An important rule for this step is that, whenever its parent's nexthop appears in its candidate nexthop set, a node will choose its parent's nexthop as its own nexthop so that its information does not have to be installed in the FIB. However, if the parent's nexthop is not a member of the node's candidate nexthop set, a nexthop will be selected randomly from the candidate nexthop set and the node will be



(a) Initial Tree (b) Pass 1 (c) Pass 2 (d) Pass 3 (e) Prefixes in FIB

Fig. 2. ORTC Algorithm (The initial routing table has 4 prefixes, with their nexthop addresses shown in each node. The aggregated FIB has 3 prefixes.)

tagged as IN_FIB. After this step, all the nodes with the tag IN_FIB will be placed into the FIB.

B. Patricia Trie Implementation

Following other open source router implementations (*e.g.*, gated and MRTD), we used Patricia Trie [1] to implement our algorithms. A Patricia Trie is a binary tree, but it does not require children's prefixes to be longer than their parents' by exactly one. As such, it can reduce memory consumption by eliminating internal nodes whenever possible. For example, in order to store the prefixes 0/0 and 001/3, a regular binary tree needs four nodes including two internal nodes 0/1 and 00/2, while a Patricia Trie does not require the two internal nodes. For each tree node, we store its *original nexthop*, *node type*, *candidate nexthop set*, *selected nexthop*, as well as fields used in tree traversal.

In a binary-tree implementation, the tree is first expanded and then the children's nexthops are merged to calculate their parents' candidate nexthop set. This is much more challenging to implement correctly in a Patricia Trie, as we need to avoid creating internal nodes to the extent possible (otherwise it becomes a binary tree). We have found ways to merge nexthops correctly without creating internal nodes, when children's prefixes are longer than their parent's by more than one. We also postpone expanding the tree until the third pass, so that we only create those leaf nodes that will be in the FIB. For brevity, we do not present the details in this paper.

V. ORTC-based Incremental FIB Aggregation

The incremental update handling capability is missing from the original ORTC work but important to router operations in practice. We designed two such algorithms for ORTC. Both of our update handling algorithms ensure strong forwarding correctness. The first one aims to reduce the amount of time for the route processor to process the routing message and update the FIB at each line card. We call it the "Minimal Time" scheme. The second one maintains the optimal aggregated FIB size, which we call the "Optimal Size" scheme.

There are several common operations used by these two schemes. Below we first define these operations, and then use them to describe the algorithm for each scheme.

- *updateDescendantsCandidate(p)*: starting from the bottom of the sub-trie rooted at p, update all descendant prefixes' candidate nexthop sets. This is the same as the second pass of ORTC, except that it stops at the prefix p.
- *updateAffectedAncestors(p)*: starting from the parent of prefix p, update all ancestor prefixes' candidate nexthop set as done in the second pass of ORTC until reaching an ancestor prefix whose candidate nexthop set is the same as before. The last ancestor updated will be returned from this function, we use PA to refer to the returned ancestor node.
- *updatePrefix(p)*: update prefix p's candidate nexthops as done in the second pass of ORTC.
- *updateDescendantsSelected(p)*: starting from prefix p, compute selected nexthop for each prefix in the sub-trie, as done in the third pass of ORTC.

With the above basic operations, we first describe the update procedure for an unaggregated FIB and then describe the two update handling algorithms for an FIB aggregated using ORTC.

A. Unaggregated FIB update Scheme

Unaggregated FIB update scheme runs the normal update operations, e.g. add, update and withdraw, on the original FIB table without any aggregation for each coming update message. Upon receiving an announcement of p, the router will look up the FIB table to check if p exists in the FIB. If so, then update the next hop according to the announcement; Otherwise, add a new prefix with the corresponding next hop into the FIB table. Upon receiving a withdrawal of p, the router will look up the corresponding prefix from its FIB table, then remove this prefix with its next hop. This scheme is used in current routers, and can always guarantee the forwarding correctness.

B. Minimal Time Scheme

This scheme essentially reruns the ORTC algorithm on the subtree rooted at the prefix being updated, even though there may be more aggregation opportunities above the prefix. Therefore, this scheme will not optimize the FIB size, but rather it aims to achieve a good balance between processing time and FIB size.

Upon receiving an announcement of prefix p, if p does not exist, then p is inserted into the RIB, otherwise p's information is updated if necessary. This is followed by calling *updateDescendantsCandidate(p)*, *updatePrefix(p)*, and *updateDescendantsSelected(p)* to re-aggregate the subtree rooted at p (see an example in Figure 3).

Upon receiving a withdrawal of prefix p, the prefix is removed from FIB if it is IN_FIB. Note that we do not remove the prefix from the RIB, but instead label it as a $FAKE$ node, since it may be re-announced later. Then its original

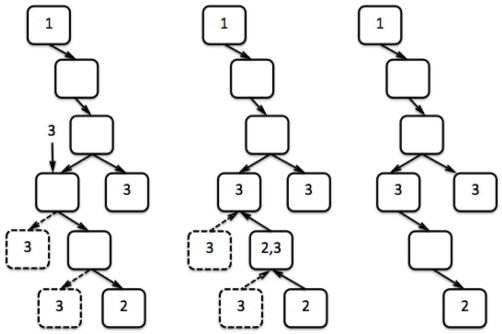


Fig. 3. Minimal Time Scheme Example. The number in each square is the nexthop for that address prefix. First, a new prefix with the nexthop 3 is inserted into the routing table, so there are a total of 4 prefixes in the tree. Then, we expand the subtree rooted at the new prefix and re-calculate the candidate nexthop sets for each node on the subtree. This is followed by selection of the nexthop. The final result is that the same four prefixes remain on the tree even though two of them are aggregatable.

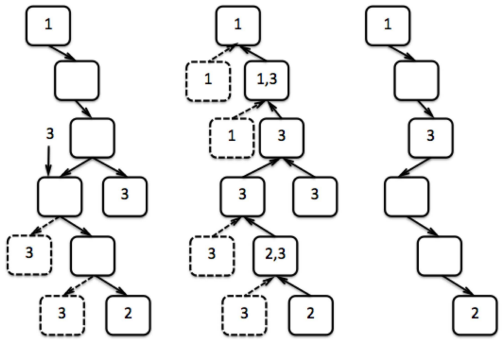


Fig. 4. Optimal Size Scheme Example. First, a new prefix with the nexthop 3 is inserted into the routing table, so there are a total of four prefixes in the tree. Then, we expand the subtree rooted at the new prefix and re-calculate the candidate nexthop sets from the leaf nodes under the new prefix towards the root (tree expansion is done when necessary). This is followed by selection of the nexthop from the highest level node whose candidate nexthop set has changed. The final result is that only three prefixes remain on the tree.

nexthop is updated to be the same as its nearest ancestor's original nexthop. We then use *updateDescendantsCandidate(p)*, *updatePrefix(p)*, *updateDescendantsSelected(p)* to update the subtree rooted at p.

C. Optimal Size Scheme

This scheme needs to produce exactly the same result as running the ORTC full aggregation algorithm. However, in order to reduce computation time, it must restrict the tree traversals to only those nodes whose state will likely change (otherwise, this is not *incremental* update handling). More specifically, starting from the prefix being modified, we recalculate the candidate nexthop set of its ancestors until reaching a node whose candidate nexthop set does not change. We then re-aggregate the subtree rooted at this node. The detailed algorithm is similar to the previous one, so we only highlight their differences below.

For either an announcement or withdrawal of p, instead of calling *updateDescendantsSelected(p)*, we use *updateAffectedAncestors(p)* and *updateDescendantsSelected(PA)*, where PA is returned by the previous operation (see an example in Figure 4). This ensures that we update those ancestors of p whose optimal nexthops (for the purpose of aggregation) are affected by this routing update. All the other operations are the same as those in the Minimal Time Scheme.

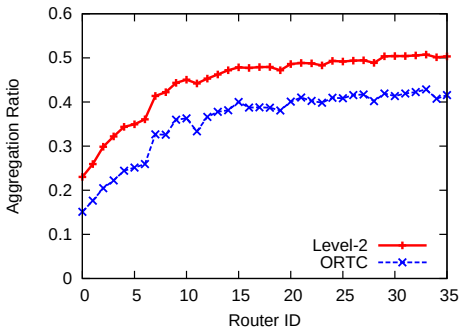


Fig. 5. Aggregation Ratio (ORTC vs. Level-2)

VI. Performance Metrics

We use the following metrics to compare different algorithms (in the following definitions, F and F' are the original FIB and aggregated FIB respectively):

a) Aggregation Ratio (r): the ratio between the aggregated FIB size and the original FIB size, *i.e.*, $r = |F'|/|F|$. A smaller aggregation ratio means more reduction in FIB size.

b) Computation Time (c): the time cost for aggregating the initial FIB (c_1) and that for updating the aggregated FIB (c_2). One may think that a route processor will inevitably be slowed down by the computation associated with FIB aggregation, but a good news is that an aggregated FIB may require less time to update than an unaggregated FIB [11].

VII. Evaluation of Original ORTC Algorithm

Today a typical BGP routing table has hundreds of thousands of entries. Some routers in large ISPs even have more than one million entries including both BGP and IGP routes. The DFZ routing tables have grown by several orders of magnitude since ORTC was originally proposed in 1999. Therefore, we first need to evaluate the feasibility of the basic ORTC algorithm.

We obtained BGP routing tables from 36 peers at the route-views.oregon-ix.net collector of the RouteViews project [3]. We then extracted the prefixes and their nexthop ASs from the routing tables. Note that we cannot directly obtain the IP nexthop address from the BGP routing tables. However, we have used private data containing both routing tables and forwarding tables from a Tier-1 ISP to verify this methodology and found that the results do not differ much when we use the nexthop AS in place of the IP nexthop. More justification of our methodology can be found in our earlier paper [11].

Our evaluation has been done on a Linux machine with an Intel Core 2 Quad 2.83GHz CPU. One router vendor told us that their routers' CPU processing power is similar to that of high-end laptops. Therefore, our computation time results are reasonable indicators of how long it will take the routers to perform the aggregation.

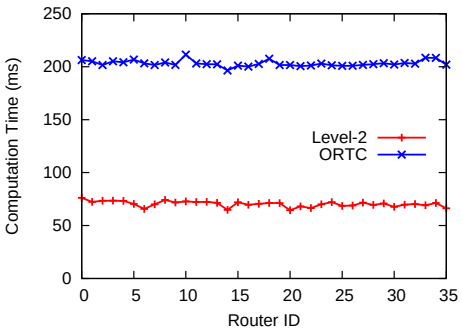


Fig. 6. Computation Time (ORTC vs. Level-2)

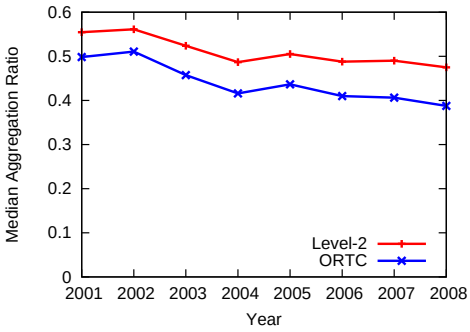


Fig. 7. Median Aggregation Ratio over Time (ORTC vs. Level-2)

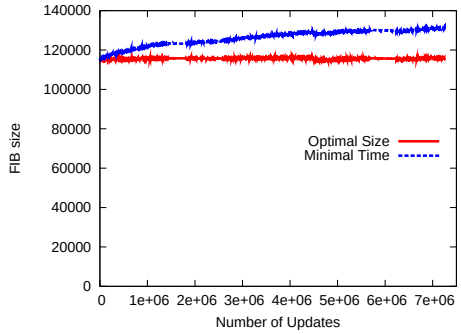


Fig. 8. FIB Size under Incremental FIB Aggregation Schemes over One Month for Router 4.68.1.166

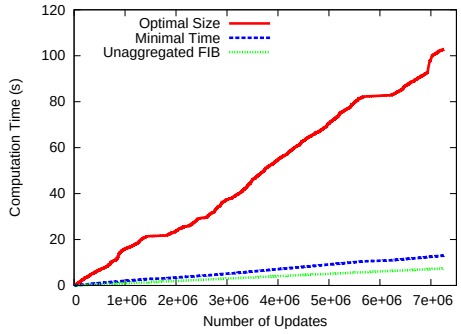


Fig. 9. Cumulative Computation Time of Incremental FIB Aggregation Schemes over One Month for Router 4.68.1.166

Figure 5 shows the aggregation ratio of ORTC when it is applied to the routing tables collected on Dec. 31, 2008. One can observe that the aggregation ratio varies from 0.15 to 0.42, with a median of 0.39. In other words, the aggregated FIB size can be 15% to 42% of the original FIB size. The specific aggregation ratio depends on how many different nexthops that a router has and how the nexthops are distributed among the prefixes. In general, routers with fewer nexthops have better aggregation ratios. For reference, we also include the aggregation ratio of our Level-2 algorithm [11], which ensures strong forwarding correctness as ORTC does. The Level-2 aggregation ratio ranges from 0.23 to 0.50 with a median of 0.48. As expected, ORTC has better aggregation ratios than Level-2, since the former optimizes the FIB size.

On the other hand, Figure 6 shows that the Level-2 algorithm's computation time is between $64ms$ and $76ms$ with a median of $71ms$, while ORTC requires $196ms$ to $211ms$ with a median of $202ms$ to finish a full aggregation process. In other words, the Level-2 algorithm is two times faster than ORTC. This is mainly because the former traverses the routing table only once, while the latter requires at least two passes.

Finally, as shown in Figure 7, the median aggregation ratio of ORTC has decreased from 0.5 in 2001 to below 0.4 in 2008 and the Level-2 algorithm exhibits a similar trend, suggesting that the forwarding tables have become more amenable to aggregation over time. This may be due to the increasingly popular practices of traffic engineering and multi-homing, which typically introduce more covered prefixes.

VIII. Evaluation of ORTC-based Incremental FIB Aggregation

We evaluated the two proposed update handling schemes using BGP tables and updates collected by RouteViews in December 2008. The number of BGP updates from a router ranges from a few million to tens of millions in this month. We present the results for the router 4.68.1.166, which is located in the Tier-1 ISP Level-3 Communications. The other routers have similar results.

Figure 8 shows how the FIB size changes over the one-month period for the peer 4.68.1.166. The unaggregated FIB size was $267,108$ on Dec. 1, 2008 and $270,927$ on Dec. 31, 2008. The ORTC full aggregation algorithm reduces the initial FIB size to $114,733$, representing an aggregation ratio of 0.43. After processing $7,254,478$ BGP updates, the Optimal Size scheme achieves a FIB size of $116,041$ (the bottom curve), maintaining the aggregation ratio of 0.43. In contrast, with the Minimal Time scheme, the FIB size increases to $131,210$ (the top curve), *i.e.*, 13% larger than that of the Optimal Size scheme.

If the FIB size increases continuously under the Minimal Time scheme, it may exceed the memory size on a line card. One solution is to perform a full aggregation whenever the FIB size reaches a certain threshold. For example, if we set the threshold to 130,000, then this particular router's FIB needs to be re-aggregated in 20 to 30 days. The threshold value depends on the actual memory size on the line cards.

As shown in Figure 9, the computation time of the Optimal Size scheme is $103s$ for processing the 7.25 million updates

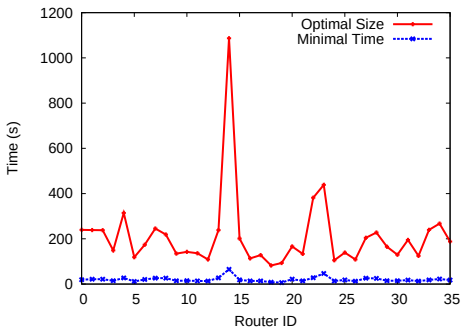


Fig. 10. Computation Time of Incremental FIB Aggregation Schemes for All Routers

(the top curve), which means $14\mu s$ per update. It is orders of magnitude faster than rerunning the full aggregation scheme, which would take $200ms$ per update. On the other hand, the Minimal Time scheme takes only $13s$ to process all the updates (the middle curve), *i.e.*, $1.8\mu s$ per update. *It is seven times faster than the Optimal Size scheme*. Moreover, this scheme only takes $5s$ more than the Unaggregated FIB scheme (the bottom curve), which is the normal update scheme for FIB without any aggregation applied, for the total 7.25 million updates. Namely, the Minimal Time scheme with aggregation only takes about $0.7\mu s$ longer than current update mechanism without aggregation per update.

As for the computational complexity and memory usage in the worst case, we need to consider how many nodes each scheme would traverse. In the Minimal Time scheme, if the routing update is for a prefix located at the root node, we need to update all the tree nodes beneath the root node, which is the entire tree. This incurs the same overhead as the original ORTC (two tree passes). However, this worst case only happens when the updated prefix is located at the root. In the Optimal Size scheme, the worst case may happen even when the prefix in question is not at the root – we need to update its ancestors until reaching one that is not affected by the change and this process may ultimately reach the root node in the worst case. In practice, the actual computation time and memory usage depend on the specific routing updates and the FIB. For reference, Figure 10 shows the processing time of all the 36 routers' updates in Dec. 2008. Minimal Time scheme typically takes about tens of seconds, while Optimal Size scheme takes hundreds of seconds in general.

Because FIB updates require real-time processing so that traffic can be forwarded to the correct nexthop, these results suggest that the Minimal Time scheme may be more preferable in a real network setting, although it requires a slightly larger FIB memory.

IX. Conclusions

As the Internet keeps growing rapidly, routers are facing a tough task: forwarding huge amount of traffic at line rate and still operates within their memory limit. FIB aggregation is a promising solution to the problem of increasing table size, but any FIB aggregation scheme must run fast enough not to slow down packet forwarding. ORTC is a FIB aggregation algorithm that gives the minimum table size possible under strong forwarding correctness, but it takes about $200ms$ to run, which is way too long for real operation, especially when there are many routing changes. In this paper, we have designed two algorithms that can incrementally update the aggregated FIB table upon a routing change. These algorithms take $14\mu s$ or $1.8\mu s$ per update, dramatically reducing the processing time and making FIB aggregation practical for real operation.

We also compare ORTC's performance with a simple aggregation algorithm (Level-2), and quantify performance difference between the two ORTC update handling algorithms. The results illustrate that FIB aggregation is a typical tradeoff between memory requirement and processing cycles. ORTC achieves the minimum table size possible under strong forwarding correctness. Compared with the simple Level-2 algorithm, ORTC compresses the table about 10% more, but takes almost 3 times longer. Between the two ORTC update handling algorithms, one maintains minimum table size all the time, another trades about 13% table size for a speedup of 7 times. Network operators will be the one to decide which tradeoff to make depending on their router configurations and network requirement, and our results provide quantitative information to help the decision making.

Acknowledgments

This work was supported by NSF Grants 0721645 and 0721863. We thank Richard Draves for sharing his ORTC code, as well as the anonymous reviewers for their feedback.

References

[1] Net-Patricia Perl Module. http://search.cpan.org/dist/Net-Patricia/.
[2] Opportunistic Topological Aggregation in the RIB-FIB Calculation? http://www.ops.ietf.org/lists/rrg/2008/threads.html#01880.
[3] Advanced Network Technology Center and University of Oregon. The RouteViews project. http://www.routeviews.org/.
[4] S. Ahmand and R. Mahapatra. M-trie: an efficient approach to on-chip logic minimization. In *Proc. IEEE/ACM International conference on Computer-aided design*, 2004.
[5] R. Draves, C. King, S. Venkatachary, and B. D. Zill. Constructing Optimal IP Routing Tables. In *Proc. IEEE INFOCOM*, 1999.
[6] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis. Locator/ID Separation Protocol (LISP). Work in Progress, http://tools.ietf.org/html/draft-farinacci-lisp-12, Mar. 2009.
[7] D. Jen, M. Meisel, D. Massey, L. Wang, B. Zhang, and L. Zhang. APT: A Practical Tunneling Architecture for Routing Scalability. Technical Report 080004, UCLA, 2008.
[8] D. Meyer, L. Zhang, and K. Fall. Report from the IAB Workshop on Routing and Addressing. *RFC 4984*, 2007.
[9] S. Suri, T. Sandholm, and P. Warkhede. Compressing Two-Dimensional Routing Tables. *Algorithmica*, 35:287–300, 2003.
[10] R. Whittle. Ivip (Internet Vastly Improved Plumbing) Architecture. draft-whittle-ivip-arch-02, August 2008.
[11] X. Zhao, Y. Liu, L. Wang, and B. Zhang. On the Aggregatability of Router Forwarding Tables. In *Proc. IEEE INFOCOM*, 2010.