On the Aggregatability of Router Forwarding Tables

Xin Zhao* zhaox@email.arizona.edu
Yaoqing Liu† yliu6@memphis.edu
Lan Wang† lanwang@memphis.edu
Beichuan Zhang* bzhang@arizona.edu

***Abstract*—The rapid growth of global routing tables has raised concerns among many Internet Service Providers. The most immediate concern regarding routing scalability is the size of the Forwarding Information Base (FIB), which seems to be growing at a faster pace than router hardware can support. This paper focuses on one potential solution to this problem – FIB aggregation, i.e., aggregating FIB entries without affecting the forwarding paths taken by data traffic. Compared with alternative solutions to the routing scalability problem, FIB aggregation is particularly appealing because it is a purely local software optimization limited within a router, requiring no changes to routing protocols or router hardware. To understand the feasibility of using FIB aggregation to extend router lifetime, we present several FIB aggregation algorithms and evaluate their performance using routing tables and updates from tens of networks. We find that FIB aggregation can reduce the FIB table size by as much as 70% with small computational overhead. We also show that the computational overhead can be controlled through various mechanisms.**

I. Introduction

The global Internet routing table has been growing at an alarming rate ([23], [14], [22]), driven in part by the increasing number of organizations connected to the Internet, and in part by the increasing practices of multi-homing and traffic engineering. This rapid increase in routing table size appears to outpace the increase in memory size, especially for the type of memory used in line cards for fast lookup. Moreover, it forces ISPs to upgrade their routers 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.

This routing scalability problem has raised concerns in both industry and research communities, as documented in the report from the IAB Workshop on Routing and Addressing [23]. Several solutions have been proposed under the IRTF RRG [3] and IETF GROW [2] working groups. To address the root cause of the scalability problem, fundamental changes to the Internet routing architecture and protocols are called for. However, deploying architectural changes is likely to take a long time, as illustrated by past examples like IPv6. While architectural changes may benefit the Internet in the long run, short-term solutions are needed as the problem is serious and imminent. In particular, ISPs urgently need to reduce their forwarding table size. Forwarding tables are derived from routing tables and router configurations, thus their size increases as routing tables grow. However, forwarding tables use high performance memory that is more expensive and more difficult to scale than the memory used to hold routing tables. Therefore, their size is a more immediate concern to ISPs and vendors.

*Computer Science Department, The University of Arizona, USA.
†Computer Science Department, The University of Memphis, USA.

This paper investigates the feasibility of a purely local solution: FIB aggregation, which is to combine multiple entries in the forwarding table 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. It is important to note that FIB aggregation is not a replacement for the long-term architectural solutions because it does not address the root causes of the routing scalability problem. Instead, *FIB aggregation is a local solution that can be quickly implemented and deployed in the short-term, and in the long run, it can co-exist and complement architectural solutions*.

The idea of FIB aggregation is rather intuitive, but to our best knowledge, no study has systematically evaluated its potential benefits or costs. FIB aggregation is an opportunistic technique – its effectiveness depends on what prefixes are present in the table, how many of them can be numerically represented by a single prefix, and how many of them share the same next-hop. The benefits of FIB aggregation come with certain costs, such as extra CPU cycles. The costs also depend on the actual aggregation algorithms, and how routing changes are handled to update the aggregated forwarding table. A thorough understanding of FIB aggregation is needed in order to decide whether it is a viable solution.

This paper conducts a systematic analysis and evaluation of FIB aggregation to understand its gains and costs. We recognize that there can be different levels of aggregation, each representing different tradeoffs between table size reduction and computation complexity. We design and implement five algorithms at different aggregation levels, and evaluate them using publicly available routing tables from tens of networks. The results show that the lowest-level aggregation can reduce table size by 30%-50%, making the table same size as two and half years ago, while the highest-level aggregation can reduce the table size by 70%, making the table same size as eight years ago. The computation time of one aggregation run ranges from tens of milliseconds to a few hundred milliseconds on a commodity Linux machine. Although these numbers may not reflect the computation time on a router, they reflect the relative speed of different levels of aggregation. To handle routing changes, we design and implement algorithms to incrementally update the aggregated FIB upon a change. The full aggregation algorithm is only invoked when the router CPU load is low or the FIB size becomes above a threshold, thus its computation time is amortized over time. The evaluation using one-month BGP routing updates shows that compared with unaggregated

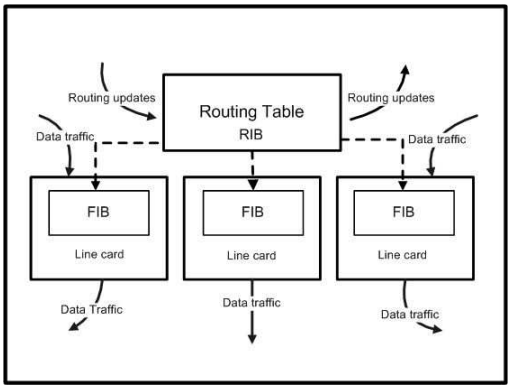


Fig. 1. RIB and FIB

FIB, the computation overhead of maintaining aggregated FIB over time is small.

Our algorithms have assumed a generic tree structure to store the routing tables, and we have not attempted to optimize either the algorithm or the implementation. Our goal is to show that, without special optimization, FIB aggregation in general is a viable solution to the scalability problem with good table size reduction and small computational overhead. When FIB aggregation is adopted in real networks, the algorithms and implementations can always be optimized for specific hardware, operating systems, and routing table data structures.

The rest of the paper is organized as follows. Section II gives an overview of FIB aggregation, why it may be effective and what tradeoffs are involved. Section III presents various aggregation techniques and algorithms. Section IV evaluates the full aggregation algorithms as well as its incremental update techniques. Section V discusses related work, and Section VI concludes the paper.

II. FIB Aggregation

There are two types of tables used by routers: *Routing Information Base (RIB)* for routing and *Forwarding Information Base (FIB)* for forwarding. RIB is stored in the main memory of a route processor. The route processor receives and processes routing update messages and runs routing protocols, *e.g.*, OSPF [24] and BGP [26], to compute the RIB. Each RIB entry contains the destination IP prefix and associated route information. For example, BGP maintains full AS path and many other attributes for each prefix in RIB. FIB is derived from RIB and router configurations. It is stored in line cards, whose job is to forward data packets. Therefore, FIB usually uses high performance memory, which is more expensive and more difficult to scale. For each destination IP prefix, the FIB has an entry to store the next-hop IP, next-hop MAC address and outgoing interface for fast data forwarding. Figure 1 illustrates these different components in a router.

Despite growth constraints such as strict address allocation policies [23], the routing tables in the default free zone (DFZ) have been growing at an alarming rate in recent years. Currently, a DFZ router stores hundreds of thousands of routes or even a million in tier-1 ISPs. This is in part due to the sheer growth of the Internet, and in part due to the lack of aggregation. When a customer network multi-homes to multiple providers for resilient Internet connectivity, the customer's address prefix(es) must be visible in the global routing table in order to be reachable through any of its providers, thus breaking down provider-based aggregation [8]. Traffic engineering is another contributing factor. For example, a network may try to influence the paths of specific incoming traffic flows by splitting its prefix into several longer ones and injecting them at different network attachment points. Splitting prefixes is also used as a defense mechanism against prefix hijacking. Growing table size leads to increasing FIB tables, RIB tables, and routing churns. Among these problems, ISPs and vendors are more concerned about the FIB size than RIB size, because it is more difficult to scale up the memory in line cards than in route processors [14].

The conventional way of reducing routing table size is to aggregate the RIB, which will also reduce FIB size. However, RIB aggregation has very limited adoption in the Internet. At a prefix's origin network, there is little incentive to aggregate the prefix, because the gain of aggregating a small number of self-originated prefixes does not make much difference to the table size. At the same time, the origin network actually has incentives, such as multi-homing and traffic engineering, to split the prefix. At a remote site, aggregation opportunity is limited since two prefixes must have the same *path attributes* in order to be aggregated in RIB. Otherwise their path information will be lost and protocol functions may be affected. Forcing aggregation of prefixes that have different paths would also defeat multi-homing and traffic engineering intended by the prefix origin networks.

FIB aggregation eliminates and aggregates entries in a FIB based on the next-hop router information while ensuring forwarding correctness. For example, it can remove prefix $P1$ from the FIB if its super-prefix $P2$ uses the same next-hop as $P1$. It may also introduce a new entry to the FIB after removing multiple entries that share the same next-hop.

FIB aggregation may be more effective than RIB aggregation since it only requires prefixes to have the same *next-hop* in order to be aggregated. For example, considering that a Los Angeles router connects to a Tokyo router, which in turn connects to a Beijing router and a Shanghai router. The Los Angeles router may reach prefixes announced by China Telecom via different paths, some via Beijing and some via Shanghai. However, in its FIB, most these prefixes take the Tokyo router as the next-hop, making them aggregatable.

The effectiveness of FIB aggregation depends on how prefixes are distributed over next-hop routers. Generally speaking, the fewer neighbors a router has, the better aggregation it may achieve. In the extreme case that all prefixes share the same single next-hop, aggregation is maximized. According to Li *et al.* [20], although some routers have high degrees up to a couple of hundreds, most connections are with their end-customers, which represent only a small percentage of the address space. The routers still use a small number of transit neighbors to reach most address prefixes.

Besides sharing the same next-hop, prefixes also need to be

numerically aggregatable. This is possible due to two factors. First, in IP address allocation, large blocks of Internet addresses are first allocated to Regional Internet Registries and then they further allocate the addresses to networks within the same region. Thus prefixes announced out of the same regions tend to be numerically aggregatable. Second, for prefixes split for traffic engineering or other purposes, a router near the origin network is likely to take different next-hops, but a router further away from the origin network is more likely to have the same next-hop towards these numerically aggregatable prefixes.

Therefore, although FIB aggregation is opportunistic and the aggregation degree varies from router to router, there are inherent properties of the Internet that can make FIB aggregation effective. If FIB aggregation is indeed effective in reducing table size, its most appealing feature is that the impact is limited within a router's data plane. It does not change any routing protocols, or any router's routing decisions. Data traffic still flows on the same router paths. Therefore, it can co-exist with almost any new routing protocols, including those architectural solutions to the routing scalability problem in the long run.

The idea of FIB aggregation is not new. It was mentioned as a potential strategy in "Preliminary Recommendation for a Routing Architecture" [21]. Through personal exchanges, we have learned that one small vendor has implemented a simple FIB aggregation scheme (similar to our Level-1 aggregation). There is also a patent for a FIB aggregation algorithm [9]. Draves *et al.* designed an optimal aggregation algorithm when no extra routable space is allowed [11]. However, to our best knowledge, we are the first to present an in-depth analysis of different levels of FIB aggregation, and systematically evaluate its effectiveness and overhead.

III. FIB Aggregation Techniques and Algorithms

There are two main questions in designing FIB aggregation techniques: how to aggregate the full FIB and how to update an aggregated FIB upon a routing change. We consider four levels of full FIB aggregation, each associated with different tradeoffs. We also propose a few techniques to reduce the computation time in updating the FIB. The algorithms presented in this section assume the routing table is stored in a tree structure. Though our implementation uses patricia trie, the algorithms should apply to any tree data structure. Note that our algorithms do not build any additional trees just for aggregation; we simply use the existing trees that the RIB and FIB already have. For a network device that uses non-tree data structure to implement routing tables, the general techniques discussed here still apply, but the algorithmic implementation will differ.

FIB aggregation should ensure packet delivery and not change the paths that packets take, which we call **forwarding correctness**. We define two types of forwarding correctness as follows.

- *Strong Forwarding Correctness*: The longest-match lookup of any destination address should end up with the same next-hop before and after the aggregation.
- *Weak Forwarding Correctness*: For destination addresses that have non-NULL next-hops before the aggregation, longest-match lookup should end up with the same next-hop after the aggregation.

There are four levels of FIB aggregation. The first two satisfy strong forwarding correctness, the last two satisfy weak forwarding correctness. We will discuss the tradeoffs and implications after presenting the aggregation techniques.

A. Full FIB Aggregation

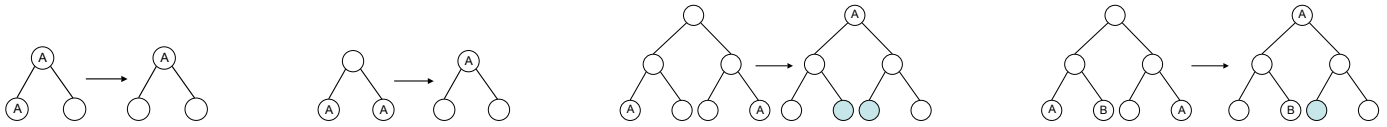
a) Level 1 Aggregation: this technique is illustrated in Figure 2(a). The simplest form of aggregation is to remove prefixes that share the same next-hop with their immediate ancestor prefixes, in which case we say that the "covered prefix" has the same next-hop as the "covering prefix" and can be removed from FIB. Addresses that previously match the covered prefix now will match the covering prefix and still get the same next-hop. Previously non-routable packets, whose table lookup ends up with NULL next-hop, will still be non-routable. This aggregation does not introduce any new prefix nor extra routable space into the table.

The algorithm implementing this technique simply traverses the tree recursively from the root node in *postorder*. When it arrives at a node with a prefix, it compares this prefix's next-hop with its immediate ancestor prefix's next-hop. If they have the same next-hop, it labels the current node NON-FIB, otherwise labels it IN-FIB. The immediate ancestor prefix's next-hop is updated and remembered during the tree traversal. Eventually every prefix node is labeled as either NON-FIB or IN-FIB, and all IN-FIB prefixes comprise the aggregated FIB. The aggregation is done recursively throughout the entire table. The computation time is O(n), where n is the total number of nodes in the tree.

b) Level 2 Aggregation: this technique is illustrated in Figure 2(b). In addition to performing Level 1 aggregation, Level 2 combines sibling prefixes that share the same next-hop into a parent prefix. If the parent node already has a prefix with a different next-hop, then the aggregation cannot be done. Or if the parent node already has a prefix with the same next-hop, then it is part of Level 1 aggregation. Therefore, Level 2 is done when the parent node has no prefix. The net result is to introduce a new prefix to cover two sibling prefixes, but there is no extra routable space introduced, *i.e.*, the aggregated FIB covers the exact address space as the unaggregated FIB.

The algorithm implementing Level 2 aggregation traverses the tree recursively from the root node in *postorder*. Besides doing Level 1 aggregation, when it arrives at a node without a prefix, it compares this node's two children. If both children have prefixes and use the same next-hop, then both children are labeled NON-FIB, and this current node is assigned the parent prefix and labeled IN-FIB. The aggregation is done recursively throughout the entire table. The computation time is O(n), where n is the total number of nodes in the tree.

c) Level 3 Aggregation: this technique is illustrated in Figure 2(c). In addition to performing the Level 1 and 2 aggregation, Level 3 aggregates a set of *non-sibling* prefixes that have the same next-hop into a super prefix. Between these non-sibling prefixes, non-routable space is allowed. For



(a) Level 1: Removing covered prefixes

(b) Level-2: Combining sibling prefixes

(c) Level-3: Allowing extra routable space

(d) Level-4: Allowing holes in the aggregate

Fig. 2. Different Levels of FIB Aggregation. The binary tree represents part of the IP address space. Nodes labeled with letters are prefixes in the routing table, and the letter represents the next-hop for the prefix. Nodes without labels do not have their corresponding prefixes in the routing table. Filled nodes are extra routable space introduced by the aggregation.

example, in Figure 2(c), at the bottom level of the tree, there are two nodes with address prefixes (real nodes) sharing the same next hop. However, these two nodes are separated in the tree by two nodes without address prefixes. The prefixes of the two real nodes can be aggregated into a grandparent prefix. A side effect is that this newly inserted prefix covers previously non-routable space, therefore some previously non-routable traffic (which would have been dropped by this router) will be forwarded along the next-hop of the aggregate prefix. All previously routable traffic is still routed along the same path as before. This behavior satisfies weak forwarding correctness but not strong forwarding correctness.

Level 3 aggregation must be implemented with care to ensure its forwarding correctness. For example, in Figure 2(c), two grandchildren prefixes are aggregated into one grandparent prefix. This would be incorrect if there is already a great-grandparent prefix (not shown in the figure) covering the subtree with a different next-hop B, because that means the two middle nodes at the bottom level are not non-routable space and their next-hops would change from B to A after the aggregation. In order to handle this case without introducing much computation overhead, we decide that in our implementation we only apply this type of aggregation to prefixes that do not have any existing ancestor prefix. In a typical DFZ routing table, about half of all the prefixes have no ancestor and the other half have ancestors. The prefixes that have ancestors can be aggregated by Level 1 and Level 2, therefore our choice does not lose too much aggregation capability.

The algorithm implementing Level 3 aggregation traverses the tree recursively in *postorder*. Besides doing Level 1 and Level 2 aggregation for all nodes, when it arrives at a prefix that does not have any ancestor, it checks whether this prefix has a sibling node that does not have a prefix. If yes, it returns a pointer of this prefix node to its parent node, which will further pass this pointer up along the tree. When an upper level node has two such pointers, one from a left descendant and another from a right descendant, and these two descendants have the same next-hop, then a new prefix is created at this upper level node and labeled IN-FIB, while the two descendant nodes are labeled NON-FIB. If the two descendants have different next-hops, then aggregation cannot be done and they remain IN-FIB. The computation complexity is O(n), where n is the number of nodes in the tree.

d) Level 4 Aggregation: this technique is illustrated in Figure 2(d). In addition to performing Level 1, 2 and 3 aggregation, Level 4 aggregates a set of *non-sibling* prefixes with the same next-hop. The difference from Level 3 aggregation is that, in Level 4, between the non-sibling aggregated prefixes, other prefixes with different next-hops are allowed, while Level 3 only allows non-routable space. For example, in Figure 2(d), a node with next-hop B is allowed to be between the prefixes being aggregated, punching a "hole" in the aggregate prefix. This type of aggregation maintains forwarding correctness and may also introduce extra routable space as Level 3 does. For the same reason as in Level 3, our algorithm only applies this type of aggregation to prefixes that do not have ancestor prefixes.

The seemingly trivial difference between Level 4 and Level 3 actually has significant implication to algorithm design. It allows the maximum flexibility for aggregation. However, taking full advantage of it may also require significant computation time. For example, given a set of non-sibling prefixes with different next-hops, which super-prefix should be inserted? Which next-hop should the super-prefix take? Finally, how should the decision be made without too much computational complexity? In this paper, we present and evaluate two different Level 4 algorithms described as follows.

The Level 4A algorithm traverses the tree recursively *once* in postorder. Besides doing Level 1, 2 and 3 aggregations, when it arrives at a prefix that does not have any ancestor, it returns a pointer of this prefix node to its parent, which will further pass this pointer up along the tree. An upper level node will receive two *lists* of its descendants, one from its left child and the other from its right child. This node combines the two lists to get all its descendants and their next-hops, picks the *most popular* next-hop as its own next-hop and inserts a prefix at this node. All the descendants that use the most popular next-hop will be labeled NON-FIB, and other descendants are labeled IN-FIB. If multiple next-hops tie for the most popular, then one of them is randomly selected. The computation time is O(n), where n is the number of nodes in the tree.

The Level 4B algorithm is based on Herrin's proposal [5]. It traverses the tree *twice*. The first step traverses the tree recursively in postorder, which is like sweeping all tree nodes from bottom up. During this process, the algorithm calculates the most popular next-hop among all descendant prefixes of a node and records this next-hop with the node unless this node already has a prefix with a different next-hop. The second step traverses the tree recursively in preorder, which is like going through all tree nodes from top down. During this process, the algorithm tries to insert new prefixes with the most popular next-hop from all descendants (not just immediate descendants as in Level 4A), as calculated in the previous postorder tree

traversal, and label descendant prefixes NON-FIB or IN-FIB accordingly. When there are multiple equally popular next-hops, we randomly select one. Under certain conditions a newly inserted prefix at a higher level of the tree may be redundant and will be removed. The computation time is O(n), where n is the number of nodes in the tree. It tries to do a more thorough aggregation than Level 4A, but will take longer time since it traverses the tree twice.

e) Extra Routable Space: The difference between weak and strong forwarding correctness is that the former (*e.g.*, Level 3 and 4 aggregations) introduces new prefixes that cover previously non-routable space, therefore some previously non-routable traffic (which would have been dropped by this router) will be forwarded. The impact of extra routable space depends on how much traffic is destined to that address space. In normal operational conditions, the volume of such traffic should be negligible. However, malicious traffic such as port scanning usually explores such non-routable space and in certain cases it may become noticeable. Eventually these packets will be dropped, either because they arrive at a router that does not have a route for these packets, or because the packet's time-to-live expires, but they will consume bandwidth during transit. It is also possible that traffic to the extra routable space falls in a loop. Network operators can choose the level of aggregation that is the best fit to their networks based on the tradeoff among table size reduction, computation time and extra routable space. To limit the size of extra routable space, one can stop aggregation for prefixes whose prefix lengths are shorter than a threshold. We found that the best tradeoff between table size reduction and extra routable space size is achieved when the aggregation stops at prefix length of 15. Moreover, null-routed prefixes can be inserted to remove the extra routable space.

B. Handling Routing Updates

Internet routes change over time, thus the obvious question is how to update the aggregated FIB when there is a change. Re-run the full FIB aggregation will maintain the best aggregation all the time, but it will also incur significant computation overhead. We use the combination of three mechanisms to make sure that the computation cost of updating aggregated FIBs is under the control of operators. First, operators can choose the level of full FIB aggregation that suits their routers the best. Routers with faster CPU and fewer routing updates can use higher level FIB aggregation, otherwise they can use lower level FIB aggregation. Second, we design an algorithm that updates the aggregated FIB incrementally. The algorithm tries to minimize the number of tree nodes that have to be accessed and changed to maintain forwarding correctness after the routing change. It does not attempt to keep table size small. Third, the full FIB aggregation is only invoked when needed, *e.g.*, the table size has crossed a threshold after being incrementally updated for a while, or when the router has free CPU cycles to spare, *i.e.*, the router load is under a threshold.

Processing a routing update includes two steps: updating the RIB and updating the FIB. The second step is straightforward as we just need to apply RIB changes to FIB. Thus we will focus on describing how the RIB is incrementally updated. In general, when a prefix gets a new nexthop, its nearest descendants need to be re-aggregated, and when a prefix is withdrawn, its nearest descendants need to be de-aggregated. The details differ depending on the level of aggregation. We first define a few basic operations, and then use them to describe the incremental update algorithm for each level.

- update-node(p): when an announcement of prefix p is received, insert the corresponding node if it does not exist in RIB, otherwise update its nexthop information if necessary. If p was previously generated by the aggregation process, label it as a real prefix. Let A be p's nearest ancestor, if nexthop(A) == nexthop(p), label p as NON-FIB, otherwise IN-FIB.
- re-aggregate(p): For each D of p's nearest descendants, if nexthop(D) != nexthop(p), label D IN-FIB. Optionally, if nexthop(D) == nexthop(p), label D NON-FIB, which does not affect forwarding correctness but reduces the FIB size at the expense of updating more nodes.
- de-aggregate(p): Let A be p's nearest ancestor. For each D of p's nearest descendants, if nexthop(D) != nexthop(A), label D IN-FIB. Optionally, if nexthop(D) == nexthop(A), label D NON-FIB, which does not affect forwarding correctness but reduces the FIB size at the expense of updating more nodes. When A does not exist, its nexthop is considered to be NULL and the aforementioned actions still hold.

Using the above basic operations, we describe the incremental update algorithm for each level of aggregation as follows.

Level 1: Upon receiving an announcement of prefix p, update-node(p) and re-aggregate(p). Upon receiving a withdrawal of prefix p, de-aggregate(p) and remove p from the RIB.

Level 2: Upon receiving an announcement of prefix p, update-node(p) and re-aggregate(p). Upon receiving withdrawal of prefix p, de-aggregate(p) and remove p. However, if p's nearest ancestor, A, is a generated prefix, we also need to de-aggregate(A) and remove A to prevent extra routable space.

Level 3: Upon receiving an announcement of prefix p, update-node(p) and re-aggregate(p). During the re-aggregation, if any D is a generated prefix and nexthop(D) != nexthop(p), de-aggregate(D) and remove D. This is needed since in Level 3 aggregation, a generated prefix is not supposed to appear as a descendant of any real prefix, otherwise the forwarding correctness may not hold. The processing of withdrawals is the same as in Level 2.

Level 4: The processing of withdrawals is the same as in Level 2. The processing of announcements is mostly the same as in Level 3, except that the de-aggregation will take place even if a node changes from a generated prefix to a real prefix with the same nexthop. In Level 4 aggregation, it is possible that a generated prefix covers another generated prefix. Therefore when a prefix becomes real, its descendants need to be de-aggregated to make sure that there is no generated prefixes underneath. While in Level 3 aggregation, a generated prefix does not cover another generated prefix.

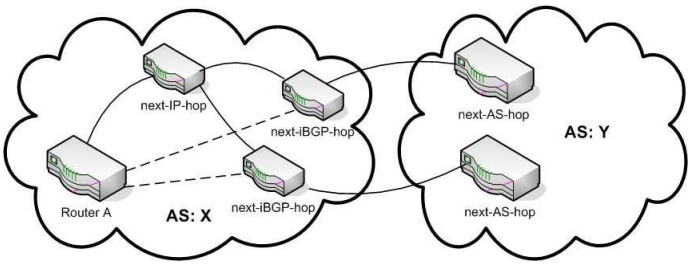


Fig. 3. Next-hop, iBGP neighbor, and next-AS-hop

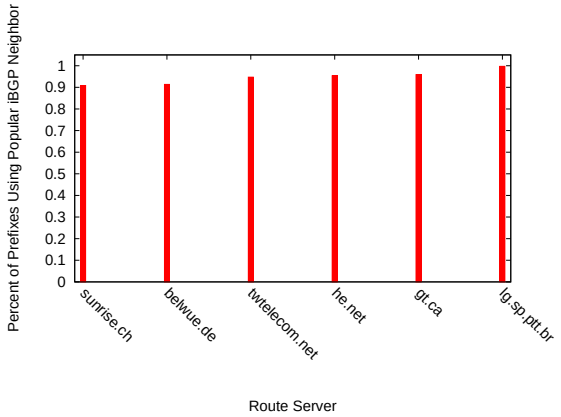


Fig. 4. Prefixes sharing the same next-AS-hop use the same iBGP neighbor.

IV. Evaluation

We use publicly available routing tables from tens of networks to evaluate the various FIB aggregation algorithms for their table size reduction, computing times, and extra routable space. We also use BGP routing updates to evaluate our incremental update algorithm.

A. Methodology

The publicly available BGP routing tables are taken from route servers [1] and the route-views.oregon-ix.net monitor of the RouteViews project [6]. Although these routing tables contain valid next AS hops, they either do not have next-hop router information or do not reflect the diversity of next-hops that an operational router typically has, since the route monitors are not operational routers. Therefore we need to generate realistic next-hops based on known information. Our guideline of this process is trying to overestimate the number of next-hops so that the table reduction results reflect the worst case scenario, and real routers are likely to have better aggregation ratio.

Routing tables downloaded from route servers contain the iBGP neighbor address for each prefix. Assuming intra-domain routing uses a single best path, prefixes that share the same iBGP neighbor will share the same next-hop. Thus we use the iBGP neighbor as the next-hop in evaluations (see Figure 3 for the relationship between next-hop, iBGP neighbor and next-AS-hop). This reflects the worst case scenario since prefixes using different iBGP neighbors may actually use the same next-hop router in reality, which will improve aggregation.

Routing tables downloaded from RouteViews do not even contain iBGP neighbor addresses – they contain only the AS path for each prefix. In this case, we use the next-AS-hop for each prefix to approximate the next-hop router based on the assumption that *prefixes sharing the same next-AS-hop are likely to share the same iBGP neighbor and thus the same next-hop router*. We use tables from route servers to validate this assumption. For each next-AS-hop, if there is only one iBGP neighbor, then all the prefixes using this next-AS-hop share the iBGP neighbor. If there are multiple iBGP neighbors, the one that carries the most prefixes is called "popular," and we expect that most of the prefixes use the popular iBGP neighbors. As shown in Figure 4, more than 90% of the prefixes indeed use the most popular next iBGP neighbor in all the valid route server tables. Note that approximating next-hop router using next-AS-hop tends to *underestimate* the effectiveness of aggregation schemes, since large networks have hundreds to thousands of neighbor ASes, but the number of real next-hops should be much smaller.

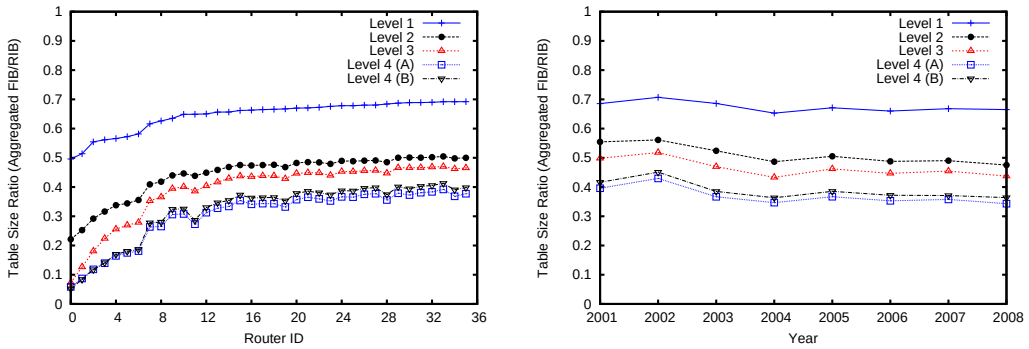
We verify the forwarding correctness of each aggregated FIB by looking up every original RIB prefix and its sub prefixes in the FIB, which should give the same next-hop as that in the RIB. All the results from our FIB aggregation algorithms and incremental update algorithms have been verified to satisfy forwarding correctness.

The evaluation has been done on a Linux machine with an Intel Core 2 Quad 2.83GHz CPU. The implementation uses a single thread and the thread is bound to a single core at runtime. The algorithms are implemented in C and no performance optimization techniques have been attempted. The Patricia trie implementation is taken from the C source code of Perl's Net::Patricia module [4], which in turn was adapted from MRTD's [25] source code.

We use the public BGP routing tables to do the evaluation because these tables come from a diverse set of networks, from tier-1 ISPs to small networks. However, in operational networks, there are other types of routes, such as VPN routes, which can be of a large number too. The FIB aggregation algorithms can be applied to these other types of routes as well, even though this paper does not evaluate the effectiveness of doing so. We plan to obtain forwarding tables from operational routers to further validate our results.

B. Table Size Reduction and Overhead

We apply the four levels of aggregation to 36 routing tables archived at RouteViews on Dec. 31, 2008. Figure 5(a) shows the ratio of aggregated FIB size over original table size. The routers are ordered based on aggregation ratio. One can make the following observations: (1) each level of aggregation can reduce the FIB size more than the previous level, which is expected; (2) even with the simple Level 1 aggregation, the FIB size can be reduced by 30% to 50%; (3) Level 4 aggregation can reduce the FIB size by 60% to over 90% with the median around 66% – some of the tables have almost all the prefixes sharing the same nexthop, leading to very small aggregated table; and (4) Level 4A is slightly better than Level 4B, although the difference is almost negligible. The results for the tables from the route servers are similar. They are not included due to the page limit.



(a) RouteViews Tables, 2008/12/31 (b) RouteViews Tables, 2001 - 2008

Fig. 5. Ratio between Aggregated FIB size and Routing Table Size

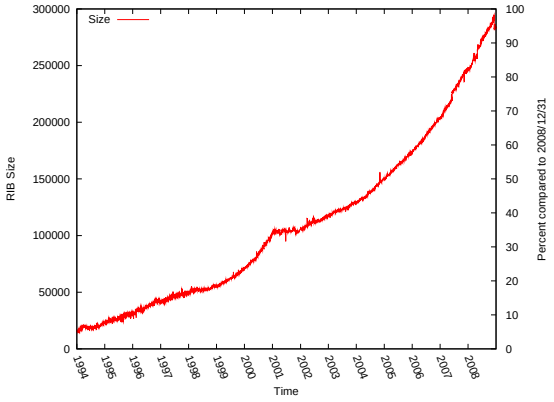


Fig. 6. Historical RIB size

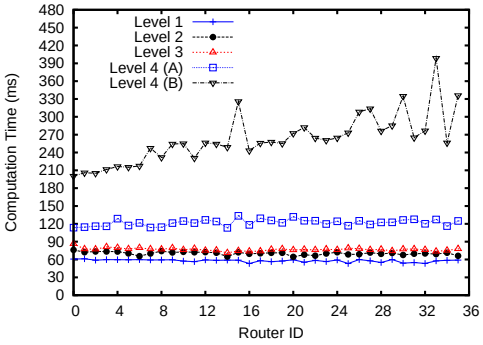


Fig. 7. Computing Time (RouteViews Tables)

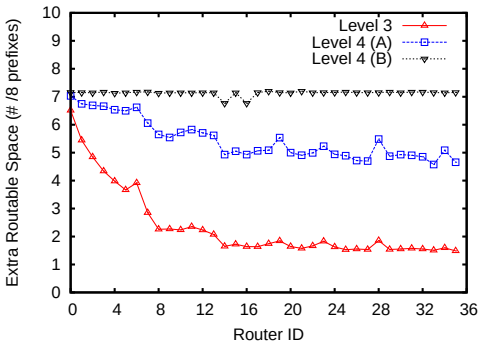


Fig. 8. Extra Routable Address Space (RouteViews Tables)

To evaluate the effectiveness of FIB aggregation over a longer period of time, we apply it to RouteViews routing tables from 2001 to 2008. For each year, we use all the tables available on Dec. 31, and plot median aggregation ratio in Figure 5(b). The result shows an overall slightly decreasing trend, suggesting that the FIB has become more amenable to aggregation over the years. One possible explanation is that the increasing practice of prefix splitting due to multi-homing and traffic engineering has made a larger percentage of FIB entries aggregatable. We plan to further investigate this phenomenon in our future work.

To understand the significance of the table size reduction results, we plot the size of the routing table from 1994 to 2009 (Figure 6) to translate the size reduction into how many years the clock is turned back for a router. The data is obtained from bgp.potaroo.net, a site that tracks the growth of the BGP table size. This figure shows that the FIB size in Nov. 2000 is around 34% of the FIB size on Dec. 31, 2008, which means that if an ISP uses the Level 4 aggregation algorithm, it will still be able to use routers that were deployed in late 2000, assuming table size is the limiting factor.

Figure 7 shows the computing time for each of the 36 routing tables. The Level 1 to 3 algorithms typically take tens of milliseconds, while the Level 4 algorithms take at most a few hundreds milliseconds. An operational router has a different CPU from our commodity Linux machine, and also specialized hardware and software. Thus it is hard to infer a router's computing time from what we report here. Nevertheless, the simplicity of the algorithms and the very short computing time suggest that the computational overhead in an operational router may be small. Moreover, the results can be used to compare the relative speed between different aggregation algorithms. For example, we can observe that Level 4B algorithm is more computationally intensive than the Level 4A algorithm since it traverses the tree one more time.

Figure 8 shows the amount of extra routable space measured by the number of equivalent /8 prefixes. Since Level 1 and 2 algorithms do not introduce any extra routable space, they are not included in the figure. To avoid introducing a large amount of extra routable space, we do not aggregate short prefixes. The exact threshold on the prefix length is a tradeoff between aggregation ratio and extra routable space size. We find that /15 represents a good trade-off, *i.e.*, aggregating prefixes shorter than /15 will only reduce the table size marginally

Algorithms	Total RIB Proc. Time(s)	Avg. RIB Proc. Time (μs)	Total FIB Updates	Total FIB Proc. Time(s)	Avg. FIB Proc. Time(μs)	Total Affected Prefixes in FIB	No. Prefixes Affected Per FIB Update
Un-Aggregated FIB	4.37	0.60	2914020	2.58	0.89	2914020	1.000
Level-1 Aggregation	4.47	0.62	2904623	2.45	0.84	2921339	1.006
Level-2 Aggregation	4.51	0.62	2901197	2.44	0.84	2933968	1.011
Level-3 Aggregation	4.64	0.64	2900302	2.42	0.83	2940223	1.014
Level-4 Aggregation (A)	4.67	0.64	2897384	2.40	0.82	2941992	1.015
Level-4 Aggregation (B)	6.41	0.88	2913988	2.61	0.77	3388764	1.162

TABLE I
PROCESSING ROUTING UPDATES IN DECEMBER 2008

but will introduce a lot of extra routable space. The result presented in Figure 8 caps the aggregation at /15. Level 3 algorithm introduces less extra routable space than Level 4 algorithms, while Level 4B algorithm has more extra routable space than the Level 4A algorithm. This is mainly because the 4B algorithm aggregates prefixes from top to bottom, which introduces more shorter prefixes than the 4A algorithm.

C. Routing Update Handling

To evaluate the incremental update algorithm, we use one month (December 2008) of BGP updates collected by Route-Views from a peer router at a large ISP (Level-3 Communications). There are totally 7,254,478 routing updates during this month, and we make sure there is no BGP session reset or table transfer in that month.

The processing time is obtained for RIB update and FIB update separately. The results are summarized in Table I. We make the following observations: (a) the RIB processing time per routing update increases from 0.6μs without aggregation to 0.62μs for Level 1 aggregation (3.3% increase) and 0.64μs for Level 4A aggregation (6.7% increase). The increase is due to the need to update more than one node in the RIB tree, but the small increase suggests that the extra overhead for updating the RIB is minimal; (b) the total FIB processing time (5th column) decreases by 5% (Level-1) to 7% (Level-4A), despite a slight increase in the total number of affected prefix nodes (7th column). This is because each prefix takes less time to update in an aggregated FIB, leading to a lower total FIB processing time. The lower FIB update time per prefix is likely due to the small FIB size after aggregation, which means faster prefix lookup. In summary, FIB aggregation can reduce both the FIB size and FIB update time, with minimal extra RIB processing time.

Among all the updates, 2,914,020 of them cause changes to unaggregated FIB, *i.e.*, an insertion, removal, or a change to the next-hop of a FIB entry. Note that there can be fewer routing updates that cause changes to an aggregated FIB than the unaggregated FIB (see the 4th column of Table I). For example, the aggregated FIB from Level 4A algorithm has 16,636 fewer updates than the unaggregated FIB. This is due to two reasons. First, some of the route withdrawals are for prefixes already removed from the FIB by the aggregation. Second, the update algorithm minimizes the number of FIB updates at the cost of slightly increased FIB size.

Since the update handling algorithm trades off the FIB size for fewer changes, the FIB needs to be re-aggregated when its size reaches a certain threshold. To estimate how frequently the re-aggregation will be triggered, we measure the growth of the FIB size as our algorithm handles the BGP updates during the month of Dec. 2008 (see Figure 9). The Level 4A aggregated FIB has 104,691 entries on Dec. 1, 2008 (39.2% of the full table size). If the threshold for re-aggregation is set to 150,000 entries (about 55% of the full routing table size), the FIB would be re-aggregated four days later on Dec. 6, 2008. Considering that each full aggregation takes at most a few hundred milliseconds on our commodity PC (perhaps a little longer on a router), incurring this overhead every few days or so should not be a concern for an ISP. Figure 10 confirms that with 150,000 as the threshold re-aggregation indeed happens every few days, for a total number of seven times within this month.

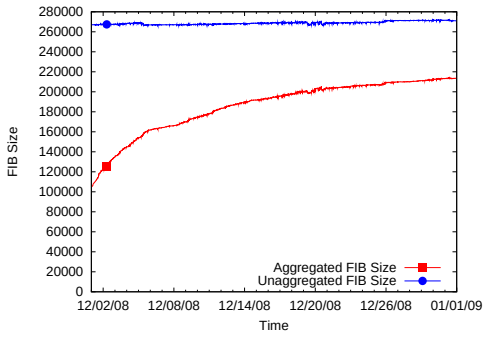


Fig. 9. FIB size after applying Level 4A aggregation algorithm initially and incremental update handling algorithm subsequently

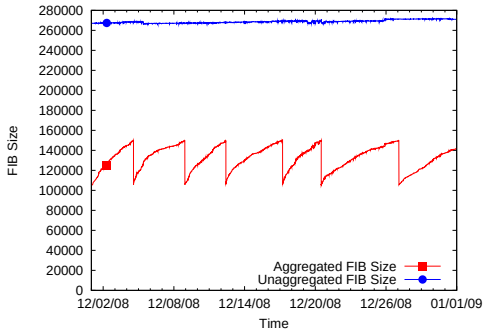


Fig. 10. FIB size with periodic re-aggregation

V. RELATED WORK

The introduction of CIDR [15] in 1993 enabled better aggregation of address prefixes and slowed down the growth of routing table size considerably for a period of time. However, the increasingly pervasive practice of multihoming and traffic engineering has again led to the routing scalability problem.

In response this problem, the IRTF Routing Research Group (RRG) [3] was formed in search for a long-term solution. Many proposals are being discussed on the RRG mailing list and at RRG meetings. In previous work [19], we classified the proposed solutions into two categories, *separation* and *elimination*. One of the separation approaches is Map-and-Encap ([10], [17]). Several recently proposed schemes, *e.g.*, LISP [12], APT [18], Ivip [27], TRRP [16], are realizations of the Map-and-Encap concept. These long-term solutions aim to reduce the routing table size, which inevitably involves changes to the routing architecture and protocols. However, these changes generally take a long time to become a reality. Moreover, they usually change the traffic paths, and incur extra packet processing overhead.

The ORTC algorithm proposed by Draves *et al.* [11] achieves optimal aggregation when no extra routable space is allowed. However, they did not include an update handling mechanism for ORTC. Our work can achieve higher aggregation ratio by introducing a small amount of extra routable space. We also provide different levels of aggregation with different tradeoffs so that operators can choose the scheme best for their routers. Bill Herrin proposed FIB aggregation as a potential strategy in "Preliminary Recommendation for a Routing Architecture" [21]. He described the basic idea of our Level 4B algorithm, which includes our own improvements.

Another proposed approach to reducing FIB size is Virtual Aggregation (VA) ([13], [7]). VA designates a small set of routers (APRs) that announce virtual prefixes, so that other routers do not need to install more specific prefixes under those virtual prefixes in their FIB – they just simply forward packets to the APRs responsible for the corresponding virtual prefixes. It can be independently deployed by one ISP, and does not require changes to the routing architecture or protocols. However, VA requires changes to network-wide router configurations and specialized routers to announce virtual prefixes. Moreover, it could introduce extra delays (stretch) in packet delivery. VA allows operators to control FIB size more explicitly by APR configuration.

VI. Conclusion and Future Work

We have presented an in-depth analysis of FIB aggregation and the results suggest that it is a viable short-term solution to the problem of growing FIB table size. Our aggregation algorithms reduces the FIB size by as much as 70% and requires no hardware changes or network-wide software/configuration changes, thus reducing the need for ISP router upgrades in the short term. During this time, the research community and the industry can design and deploy long-term solutions to reduce both the routing table and the FIB table. Moreover, FIB aggregation can co-exist with any long-term solution to further reduce ISPs' operational costs. We plan to continue our research on FIB aggregation in the following areas. First, we will obtain forwarding tables from operational ISP routers to validate our results. Second, we would like to examine the consequences of the extra routable space introduced by Level 3 and 4 aggregations. Third, we plan to implement our algorithms on a software based router. Finally, we are conducting an in-depth comparison between our algorithms and the ORTC algorithm proposed by Draves *et al.* [11].

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. 0721863 and 0721645. We thank Lixia Zhang, Tony Li, Xiaohu Xu, Keyur Patel, Zartash Uzmi, William Herrin, John Scudder, Danny McPherson and the anonymous reviewers for their insightful comments.

References

[1] BGP4.net Wiki. http://bgp4.net.
[2] IETF Global Routing Operations (GROW). http://www.ietf.org/dyn/wg/charter/grow-charter.html.
[3] IRTF Routing Research Group. http://www.irtf.org/charter?gtype=rg\&group=rrg.
[4] Net-Patricia Perl Module. http://search.cpan.org/dist/Net-Patricia/.
[5] Opportunistic Topological Aggregation in the RIB-FIB Calculation? http://www.ops.ietf.org/lists/rrg/2008/threads.html#01880.
[6] Advanced Network Technology Center and University of Oregon. The RouteViews project. http://www.routeviews.org/.
[7] H. Ballani, P. Francis, C. Tuan, and J. Wang. Making Routers Last Longer with ViAggre. In *NSDI*, 2009.
[8] T. Bu, L. Gao, and D. Towsley. On Characterizing BGP Routing Table Growth. *Computer Networks*, 45(1):45–54, may 2004.
[9] B. Cain. Auto aggregation method for IP prefix/length pairs. http://www.freepatentsonline.com/6401130.html, June 2002.
[10] S. Deering. The Map & Encap Scheme for Scalable IPv4 Routing with Portable Site Prefixes. Presentation, Xerox PARC, March 1996.
[11] R. Draves, C. King, S. Venkatachary, and B. D. Zill. Constructing Optimal IP Routing Tables. In *Proc. IEEE INFOCOM*, 1999.
[12] 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.
[13] P. Francis, X. Xu, H. Ballani, D. Jen, R. Raszuk, and L. Zhang. FIB Suppression with Virtual Aggregation. Work in Progress, http://tools.ietf.org/html/draft-francis-intra-va-01, Oct. 2009.
[14] V. Fuller. Scaling Issues with Routing+Multihoming. http://www.vaf.net/~vaf/apricot-plenary.pdf.
[15] V. Fuller, T. Li, J. Yu, and K. Varadhan. Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy. *RFC 1519*, 1993.
[16] W. Herrin. Tunneling Route Reduction Protocol (TRRP). http://bill.herrin.us/network/trrp.html.
[17] R. Hinden. New Scheme for Internet Routing and Addressing (ENCAPS) for IPNG. *RFC 1955*, 1996.
[18] 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.
[19] 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 *ACM Workshop on Hot Topics in Networks*, 2008.
[20] L. Li, D. Alderson, W. Willinger, and J. Doyle. A first-principles approach to understanding the Internet's router-level topology. In *Proc. of ACM SIGCOMM*, 2004.
[21] T. Li. Preliminary Recommendation for a Routing Architecture. http://tools.ietf.org/html/draft-irtf-rrg-recommendation-02, March 2009.
[22] X. Meng, Z. Xu, B. Zhang, G. Huston, S. Lu, and L. Zhang. IPv4 Address Allocation and BGP Routing Table Evolution. In *ACM SIGCOMM CCR*, Janurary 2005.
[23] D. Meyer, L. Zhang, and K. Fall. Report from the IAB Workshop on Routing and Addressing. *RFC 4984*, 2007.
[24] J. Moy. OSPF Version 2. RFC 2328, SRI Network Information Center, September 1998.
[25] MRTD: The Multi-Threaded Routing Toolkit. http://www.mrtd.net.
[26] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol (BGP-4). *RFC 4271*, Jan. 2006.
[27] R. Whittle. Ivip (Internet Vastly Improved Plumbing) Architecture. draft-whittle-ivip-arch-02, August 2008.