ABSTRACT
In the era of high-performance cloud computations, networks need to ensure fast packet forwarding. This task is carried by TCAM forwarding chips that perform line-rate Longest Prefix Matches in a Forwarding Information Base (FIB). However, with the increasing number of prefixes in IPv4 and IPv6 routing tables the price of TCAM increases as well. In this work, we present a novel FIB compression technique by adding an aggregation layer into a FIB caching architecture. In Combined FIB Caching and Aggregation (CFCA), cache-hit ratio is maximized up to 99.94% with only 2.50% entries of the FIB, while the churn in TCAM is reduced by more than 40% compared to low-churn FIB aggregation techniques.

CCS CONCEPTS
• Networks → Network architectures;

KEYWORDS
Routing, FIB caching, FIB aggregation, TCAM overflow

1 INTRODUCTION
A network router looks up the Forwarding Information Base (FIB or a Forwarding Table) for selecting the next-hop of a packet via Longest Prefix Matching (LPM). With the mass adoption of distributed computing and hence increasing network traffic, performing LPM at a line-rate speed requires expensive Ternary-Content Addressable Memory (TCAM) that provides parallel lookup through hundreds of thousands of FIB entries in a single CPU cycle. In the meantime, the size of a global FIB at backbone routers reached 820K IPv4 and 80K IPv6 routes in 2020 [1]. The super-linear growth is mostly attributed to IPv4 prefix fragmentation for traffic engineering and multi-homing purposes [26, 37]. While the IPv4 routing table is projected to exceed 1M in 2024, the size of an IPv6 table will at least double within the next 5 years [12]. FIB growth poses challenges for network operators. First, a larger FIB means more costly TCAM memory with higher energy consumption [16, 18, 21, 27, 35]. Next, migrating from old to new hardware leads to higher operational cost [37]. Based on our market analysis, the price of the Cisco ASR 9000 Series Line Card that can support up to 1M of TCAM entries can make up to 80% of the total cost of a router. Alternatively, increasing the space for IPv4 prefixes on TCAM at the cost of reducing the allocations for IPv6 prefixes [28] is problematic due to the rates of IPv6 growth.

FIB aggregation has been proposed to tackle these challenges by merging adjacent or overlapped prefixes with the same next-hop. FIB aggregation, however, increases BGP churns in which a BGP update causes multiple entry insertions, deletions, and updates. This is particularly detrimental to TCAM as an insertion may require up to 1,000 operations [16]. Another technique to deal with the high cost of TCAM is FIB caching that utilizes the high skewness between the number of popular and unpopular routes as studied in [11, 20, 30]. For example, Programmable FIB Caching Architecture (PFCA) [14] achieves 99.8% cache-hit ratio with only 3% of the total routes in TCAM, where the primary FIB table (or FIB cache) resides. FIB caching techniques convert original prefixes to fragmented non-overlapping prefixes in order to avoid cache hiding where a less specific prefix hides more specific one in a secondary FIB table, causing errors in LPM matching [11, 14, 20, 23]. Many adjacent prefixes generated by this conversion, however, occupy additional space, increase cache-miss ratios, and ultimately increase lookup latencies.

We integrate FIB aggregation and caching for small FIB size with efficient use of TCAM and a higher cache-hit ratio, but not compromising LPM matching correctness. To this end, we design Combined FIB Caching and Aggregation (CFCA in short) by introducing an aggregation layer into the Route Manager of the control plane in PFCA. Unlike other
An example of FIB aggregation is given in Table 1. Aggregability of a FIB depends on the number of present adjacent or overlapping prefixes with the same next-hop. ORTC-based algorithms [10, 24] reduce the size of a FIB by up to 80%. However, on average, optimal aggregation almost doubles the number of changes on a FIB to preserve its compressed state. FAQS [22] significantly reduces the number of FIB changes caused by BGP updates applied against a compressed FIB. Nevertheless, in the worst case, a single BGP update in FAQS produces almost 6500 changes in a FIB. Such a large number of changes in Ternary Content-Addressable Memory (TCAM) is detrimental since writing in TCAM memory is an expensive and complex procedure.

### Table 1: Example of FIB aggregation

<table>
<thead>
<tr>
<th>Label</th>
<th>Original Prefix</th>
<th>Aggregated Prefix</th>
<th>Next hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>129.10.124.0/24</td>
<td>129.10.124.0/24</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>129.10.124.0/27</td>
<td>129.10.124.0/27</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>129.10.124.64/26</td>
<td>129.10.124.0/24</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>129.10.124.192/26</td>
<td>129.10.124.0/24</td>
<td>2</td>
</tr>
</tbody>
</table>

**FIB aggregation**. Numerous studies [11, 20, 23, 30] show that most of the network traffic is carried to a small set of destination Autonomous Systems. This traffic includes real-time applications, social media networks, video streaming and requires fast forwarding at the backbone routers. With FIB caching, TCAM memory acts as a fast cache for a FIB and contains only prefixes carrying the large amounts of the traffic. The challenges for FIB caching include popular prefix selection, victim cache entry selection, latency for cache-missed LPM lookups. Finally, a FIB caching solution must prevent cache hiding, where a less specific prefix hides more specific one in a secondary FIB table, causing errors in LPM matching. PFCA [14] leverages Protocol Independent Switch Architecture (PISA) [7] to build a FIB caching solution with minimized latency and line-rate cache table management.

**Our contributions** are as follows:

1. We design a FIB aggregation algorithm compatible with FIB caching. CFCA's aggregation preserves the forwarding behavior of a FIB without introducing overlapping routes. As a result, CFCA outperforms existing FIB caching techniques by achieving 99.94% cache-hit ratio with only 2.50% of the FIB entries in a TCAM cache.
2. We design CFCA's incremental BGP update handling algorithm. In CFCA, only 0.625% of BGP updates trigger changes in TCAM; the total churn in TCAM is reduced by 40% compared to a standard FIB aggregation. In the meantime, the time cost for BGP update handling by CFCA exceeds PFCA's by only 0.09µs (11.6%).
3. We evaluate CFCA using traffic traces with more than 3.5 billion of packets and 45,600 BGP updates. Our simulation demonstrates the advantages of CFCA over standard FIB caching and aggregation approaches in terms of cache-miss ratio and cache churn.

## 2 BACKGROUND

**FIB aggregation**. An example of FIB aggregation is given in Table 1. Aggregability of a FIB depends on the number of present adjacent or overlapping prefixes with the same next-hop. ORTC-based algorithms [10, 24] reduce the size of a FIB by up to 80%. However, on average, optimal aggregation almost doubles the number of changes on a FIB to preserve its compressed state. FAQS [22] significantly reduces the number of FIB changes caused by BGP updates applied against a compressed FIB. Nevertheless, in the worst case, a single BGP update in FAQS produces almost 6500 changes in a FIB. Such a large number of changes in Ternary Content-Addressable Memory (TCAM) chips is detrimental since writing in TCAM memory is an expensive and complex procedure.

For example, a single entry insertion can require up to 1000 operations in a TCAM [16].

**FIB caching**. Numerous studies [11, 20, 23, 30] show that most of the network traffic is carried to a small set of destination Autonomous Systems. This traffic includes real-time applications, social media networks, video streaming and requires fast forwarding at the backbone routers. With FIB caching, TCAM memory acts as a fast cache for a FIB and contains only prefixes carrying the large amounts of the traffic. The challenges for FIB caching include popular prefix selection, victim cache entry selection, latency for cache-missed LPM lookups. Finally, a FIB caching solution must prevent cache hiding, where a less specific prefix hides more specific one in a secondary FIB table, causing errors in LPM matching. PFCA [14] leverages Protocol Independent Switch Architecture (PISA) [7] to build a FIB caching solution with minimized latency and line-rate cache table management. PFCA handles cache hiding through prefix extensions, i.e., converting a FIB into a set non-overlapping prefixes and preserving such state through BGP updates. Our analysis of the resulting FIB in the cache showed that it is compressible by at least 15%. There are two sources of the redundant prefixes that can be compressed:

1. BGP-announced adjacent prefixes, that share the same next hop in the original FIB.
2. Prefixes, generated after a prefix extension procedure, which increases the number of entries by 40% [15].

Freeing cache memory may decrease the cache churn and increase the cache-hit ratio. Yet the optimal aggregation algorithms [10] can not be directly applied to a cache or the main FIB, as they may introduce overlapping prefixes and trigger cache hiding. For example, suppose the original FIB consists of entries shown in Table 1(a). Applying above-mentioned FIB aggregation algorithms halves this FIB (see Figure 1(b)) but creates a risk of cache hiding. Specifically, if the longer prefix 129.10.124.192/26 is not in a cache, matching the IP address 129.10.124.192 in a cache results into the less specific prefix 129.10.124.0/24, which violates the Longest Prefix Matching rule for packet forwarding.

**Programmable data plane**. Protocol-Independent Switch Architecture (PISA) [7] represents a relevant platform for building and deploying novel routing solutions. Unlike Openflow [25], PISA implies the programmability of both the control and the data planes. In this work, we build CFCA...
Figure 1: Portable Switch Architecture

Figure 2: The architecture of CFCA

with regard to the specification of the Portable Switch Architecture (PSA) [3], one of the PISA targets proposed by the P4.org Architecture Working Group [2]. We show the design of PSA in Figure 1. The white blocks represent the programmable components of PSA, such as the ingress/egress parser/deparser, packet processing pipelines, and the control plane. The gray blocks represent the hardware blocks, responsible for packet replication, bufferization and queuing. The programmable blocks are configured via the data plane programming P4 language [33]. P4 language allows developers to customize a packet parser and deparser, match-action tables, ingress/egress flows in the data plane. In addition, with the help of P4, a PSA data plane may contain user-defined registers, meters, counters and perform hashing operations.

3 DESIGN OF CFCA

As shown in Figure 2, CFCA is built on top of Programmable FIB Caching Architecture (PFCA) [14]. The control plane of CFCA is responsible for collecting network routes and managing the tables in the data plane, namely Level-1 (L1) cache in TCAM, Level-2 (L2) cache in SRAM, and the rest of the FIB in DRAM. To avoid cache hiding (see Section 2 for details), the Route Manager (RM) extends the FIB routes received from the network into a set of non-overlapping prefixes (see Figure 3). The main departure of CFCA from PFCA lies with the addition of the FIB aggregation module that compresses the extended prefix table of the RM into a smaller set of non-overlapping prefixes. The prefixes are aggregated via a binary prefix tree where prefixes and their next-hops are represented as nodes and the sibling nodes with the same next-hop merge into the parent node (see Figure 4). Important, the FIB aggregation module can incrementally handle BGP updates and push those updates into FIB tables with no risk of cache hiding. The FIB entries are provided with traffic counters to help detect frequently-accessed prefixes. A traffic counter is incremented for each match, and when a threshold is reached, the entry migrates to a faster cache table. In addition, the router keeps track of less popular routes (also known as light traffic hitters) in the cache for future cache replacement. The rest of the section discusses details of the RM, the FIB aggregation module, the data plane workflow, and the light traffic hitters detection algorithm.

3.1 Control plane workflow

The Route Manager (RM) first processes route updates from neighbor routers, and passes these updates to the FIB aggregation module. The RM then performs prefix migration by moving popular prefixes into fast memory and unpopular prefixes to slow memory. The FIB aggregation module serves as a filter between the RM and the data plane. It keeps the routing table in a compressed state by performing incremental FIB aggregation without introducing new overlapping routes nor changing the forwarding behavior. In the following sections, we discuss how to achieve this task at each stage of computation.

3.1.1 Initial FIB installation. At the initialization stage, CFCA’s RM receives the current prefix table’s snapshot - a Routing Information Base (RIB) - and extends it to get a set of non-overlapping prefixes, in order to avoid cache hiding due to the rule dependencies (see Section 2). Prefix extension (see Figure 3 as an example) is done by adding the prefixes to the binary prefix tree and generating additional nodes in order to get a full binary prefix tree. Such a tree in CFCA has the following properties:
(1) The root node root of the tree corresponds to the prefix 0/0 and is assigned with a default next-hop value.
(2) Each node n in the tree may have zero or two children.
(3) The left (right) child of a node n.l (n.r) corresponds to a more specific prefix with an appended 0(1) bit.
(4) The set of non-overlapping prefixes is composed by all the leaf nodes of the tree.

Each node n in CFCA’s full binary tree has the following parameters:

(1) The current table of a node, n.t. Possible values are NONE, Level-1 (L1) or Level-2 (L2) cache, or DRAM.
(2) The prefix corresponding to that node, n.p.
(3) REAL/FAKE flag n.rf, that indicates if a node was originated from the RIB (REAL node) or was generated as a result of prefix extension (FAKE node).
(4) The original next-hop n.o, taken from the RIB. The FAKE nodes inherit the original next-hops of their REAL parent nodes.
(5) The selected next-hop n.s, set by CFCA’s FIB aggregation algorithm.
(6) FIB status n.f: IN_FIB or NON_FIB which indicates if the corresponding prefix and its selected next-hop should be installed into the data plane.

Although prefix extension does not change the forwarding behavior of a prefix table, it may significantly increase its size. CFCA’s aggregation algorithm mitigates this negative effect by recursively “folding” adjacent prefixes that share the same next-hop into a larger prefix and installing into the FIB only one prefix per branch, thus preventing cache hiding. We show the pseudo-code of the initial FIB aggregation in Algorithm 1. The compression is done within a single post-order traversal over binary prefix tree starting from the root node. Consider an example with entries from Table 1a. The initial binary prefix tree for these entries contains 5 leaf nodes (see Figure 4(a)). However, at the left branch, nodes B and G share the same next-hop from the RIB; thus, they merge into E. F selects its next-hop from B and G. Since that next-hop is equal to C’s next-hop, F and C similarly merge into E. At the right branch, leaves I and D have different next-hops from the RIB and do not merge; consequently, nodes E and H do not “fold” into A and the traversal terminates. As a result, instead of 5 prefixes, only 3 non-overlapping prefixes (from the nodes E, I and D) will be installed into the FIB. More specifically, at each node n, CFCA is running two operations: (1) Setting the selected next-hop value of n; (2) Setting the FIB status of the left and right children of n, namely n.l and n.r. Below is the detailed description of both operations.

(1) Setting the selected next-hop value n.s for the node n. If n is a leaf node, then n.s is equal to the original next-hop value n.o. In case n is an internal node, its selected next-hop depends on the selected next-hops of n.l and n.r:
   a. If n.l.s= n.r.s, then n.s= n.l.s;
   b. Otherwise, n.s=0.

(2) Setting the FIB status of the left and right child nodes of n (n.l.f and n.r.f). This operation is performed for the internal nodes of the prefix binary tree. In case n.s is not equal to 0, then both n.l.f and n.r.f are set to NON_FIB. Otherwise, if n.s=0, the FIB status of a child node is set to IN_FIB if its own selected next-hop value is not equal to 0. Initially, the entries corresponding to IN_FIB nodes are pushed into the DRAM memory of the data plane. While the traffic is passing through the data plane, those entries may migrate to L2 and L1 cache if they become popular.

Intuitively, a non-zero selected next-hop value n.s of a node n means that all the leaf descendant nodes of n have the original next-hop value equal to n.s. Hence the FIB entries corresponding to those leaf prefixes can be compressed into a single entry. We call the nodes with a non-zero selected next-hop as points of aggregation. In the meantime, a zero value of n.s means that not all of the descendant leaf nodes of n have the same original next-hop value. If a left or a right child node of n is a point of aggregation, then it is assigned with
Algorithm 1: aggrInit

Input: n, where n ← node, n.o ← its original next-hop; n.s ← its selected next-hop; n.f ← its FIB status; n.t ← the table assigned to the corresponding route, n.l, n.r ← its left and right children.

1. if n is a leaf node then
   2. n.s = n.o
3. else if n is an internal node then
   4. aggrInit(n.l)
   5. aggrInit(n.r)
   6. if n.l.s = n.r.s then
      /* First case: children nodes share the same next-hop */
      7. n.s ← n.l.s
      8. n.l.f ← n.r.f ← NON_FIB
   else /* Second case: children nodes have different next-hops */
      9. n.s = 0
   10. if n.l.s ≠ 0 then
       /* The left child of n is a point of aggregation */
       11. n.l.f ← IN_FIB, n.l.t ← DRAM
       12. Push the prefix and the selected next-hop of n.l into the DRAM memory
   else
      13. n.l.f = NON_FIB
   14. if n.r.s ≠ 0 then
       /* The right child of n is a point of aggregation */
       15. n.r.f ← IN_FIB, n.r.t ← DRAM
       16. Push the prefix and the selected next-hop of n.r into the DRAM memory
   else
      17. n.r.f ← NON_FIB

IN_FIB status. At this stage, our algorithm guarantees that the prefix region n.p is fully covered with a set of IN_FIB descendant nodes of n. Thus, n and its all ancestor nodes should be assigned with NON_FIB status.

CFCA’s initial aggregation algorithm consists of a single post-order traversal of a binary tree, hence its complexity is O(N), where N is the number of nodes in a binary tree. The number of nodes N is equal to 2 * N_p – 1, where N_p is the total number of non-overlapping prefixes (i.e., leaf nodes). In the worst case, each original prefix p in a FIB can produce up to w non-overlapping prefixes, where w is p’s prefix length. BGP update handling of CFCA has a similar complexity, as it leverages post-order traversals for keeping a FIB in a compressed state. We demonstrate it further in this subsection.

3.1.2 BGP update handling. The RM receives BGP updates (new entry announcements, next-hop updates, and entry withdrawals) from a BGP daemon running on the control plane. The RM applies these updates to the binary prefix tree by updating or adding a node to it. Next, CFCA re-aggregates the sub-tree rooted at the updated (or added) node, as well as the parents of that node (see Figure 5 for the workflow overview). However, in most cases, the update affects only a few nodes of a tree. Below we give a detailed description of how CFCA handles different types of BGP updates.

- Announcement of a new next-hop NH for an existing prefix p in a FIB. A next-hop update of a prefix p may affect not just p itself, but also the prefixes generated after prefix extension. The resulting changes may require next-hop updates in the data plane, as well as de-aggregation or re-aggregation of prefixes overlapping with p. Thus, CFCA needs to perform partial traversals of a tree branch on which p resides. Next, we describe this process in details.

After the RM finds the node n corresponding to p in the prefix binary tree, it assigns the new value of the next-hop to n.o, the original next-hop of n. Next, it partially traverses the branch rooted at n in a post-order manner (see Algorithm 2), avoiding the branches rooted at REAL nodes and visiting FAKE nodes only. Intuitively, the REAL descendants of n are not affected by the next-hop update of n and traversing them is useless. The FAKE nodes, on the contrary, are generated


Algorithm 2: postOrderUpdate

**Input:** \( n, NH \), where \( n \) ← node, \( n.o \) ← its original next-hop; \( n.l, n.r \) ← its left and right children; \( n.rf \) its REAL/FAKE flag, \( NH \) is the new next-hop value.

1. if \( n.rf = \text{FAKE} \) then
   2. \( n.o \) ← \( NH \)
   3. postOrderUpdate(\( n.l \))
2. if \( n.rf = \text{FAKE} \) then
   3. \( n.o \) ← \( NH \)
   4. postOrderUpdate(\( n.r \))
3. setSelectedNextHop(\( n \))
4. setFIBstatus(\( n \))

Algorithm 3: setSelectedNextHop

**Input:** \( n \), where \( n \) ← node, \( n.s \) ← its selected next-hop.

1. if \( n \) is a leaf node then
   2. \( n.s \) = \( n.o \)
2. else if \( n \) is an internal node then
   3. if \( n.l.s = n.r.s \) then
      4. \( n.s \) ← \( n.l.s \)
   5. else
      6. \( n.s \) = 0

During prefix extension in CFCA. Since they inherit next-hop values from \( n \), they are updated with every change to \( n.o \). During the traversal, at each traversed node, CFCA:

1. Updates the node’s original next-hop with \( NH \).
2. Sets the selected next-hop of the node in the same manner as in the **Initial FIB Aggregation**. More specifically, for the leaf nodes, the selected next-hop is equal to their original next-hops. For the internal nodes, the selected next-hop is equal to 0 if their children nodes have different selected next-hop values. The procedure *setSelectedNextHop(\( n \)) is shown in Algorithm 3.*
3. Sets the FIB status of the node’s children (if they exist) and pushes the next-hop changes to the data plane. More specifically, if a node has a non-zero selected next-hop, its children should not be present in the FIB. Otherwise, the children nodes with a non-zero selected next-hops (i.e., points of aggregation) should stay or be installed into the data plane. Note that the RM tracks the current location of entries with the value of \( n.r.t \) and thus is able to push the updates to the correct data plane destination (L1, L2 caches or DRAM). We give the pseudo-code of *setFIBstatus(\( n \)) in Algorithm 4.*

Once the post-order traversal is done, the ancestors of the node \( n \) should be traversed in a bottom-up manner. There are two reasons why such procedure is necessary: first, the ancestors’ selected next-hop value depends on their children’s selected next-hops. Second, in CFCA, the FIB status of a node is defined by its parent. Since the changes in a tree propagate through selected next-hops, the bottom-up traversal can terminate if a node’s selected next-hop does not change. Similarly to the post-order traversal, at each visited node \( n \), CFCA calls both *setSelectedNextHop(\( n \)) and setFIBstatus(\( n \)). We show the pseudo-code of *bottomUpUpdate(\( n \)) in Algorithm 5.* Note that *bottomUpUpdate(\( n \)) is called only if the value of \( n.s \) is changed after calling *setSelectedNextHop(\( n \)).

Although a new route announcement and a route withdrawal require more operational steps by the RM, they mostly rely on the same routines as with a route update, i.e., *postOrderUpdate(\( n, NH \)) and bottomUpUpdate(\( n \)). We show it below in this section.
Algorithm 5: bottomUpUpdate

Input: \( n \), where \( n \rightarrow \) node.
1 repeat
2 \( n \leftarrow \) parent node of \( n \)
3 oldSelectedNextHop \( \leftarrow n.s \)
4 setSelectedNextHop\((n)\)
5 setFIBstatus\((n)\)
6 until oldSelectedNextHop = \( n.s \)

- **Announcement of a new route** \((p, NH)\), where \( p \) is the prefix and \( NH \) is its next-hop value. Two cases may happen. In the first case, a node \( n \) associated with \( p \) already exists in the binary prefix tree. CFCA handles such announcements similarly to the next-hop updates. The RM updates \( n \)'s original next-hop value, changes its type to REAL (if it was FAKE prior to the update) and runs \( postOrderUpdate(n, NH) \) and \( bottomUpUpdate(n) \) routines.

In the second case, the node \( n \) for \( p \) should be first added to the binary prefix tree. For that goal, the RM traverses the tree accordingly to the bits of \( p \), to find \( n_0 \), the existing leaf ancestor for \( n \). Next, the RM installs the node \( n \) while generating FAKE sibling nodes for each level of the tree between \( n_0 \) and \( n \). Such generation is necessary to avoid cache hiding. These siblings nodes will inherit their original next-hop value from \( n_0 \) and their prefix values will belong to the prefix range of \( n_0 \) with exclusion of \( p \). We call this procedure prefix fragmentation, since it fragments the prefix of \( n_0 \) into a set of non-overlapping prefixes with \( p \). The pseudo-code of prefix fragmentation is given in Algorithm 6.

The newly added nodes need to be aggregated, thus CFCA calls the initial aggregation aggrInit starting from the node \( n_0 \). Finally, in case the selected next-hop of the node \( n_0 \) is changed, CFCA runs \( bottomUpUpdate(n_0) \) to re-aggregate the upper part of the branch, until the first unchanged ancestor.
- **Withdrawal of a prefix** \( p \). Suppose \( n \) is the node that represents \( p \) in the binary prefix tree. CFCA handles prefix withdrawals as follows:
  1. The Route Manager sets the type of \( n \) to FAKE.
  2. The node inherits its original next-hop value from the parent node. This value might be a default next-hop, inherited from the root node of the tree, or an arbitrary next-hop from a less specific prefix existing in the original RIB.
  3. Similarly to the next-hop updates, the aggregation module runs \( postOrderUpdate(n, NH) \) and \( bottomUpUpdate(n) \) routines.

Intuitively, the prefix withdrawal operation in CFCA can be represented as a node’s original next-hop update with that node’s parent’s original next-hop value. In the meantime, CFCA ensures the compacted state of the binary prefix tree, by detecting and removing sibling FAKE leaf nodes from the data structure.

In Figure 6, we illustrate BGP update handling by CFCA. We continue using the entries from Table 1a. In this example, the RM received two BGP updates: a next-hop update for the prefix 129.10.124.64/26 and an announcement of a new prefix 129.10.124.128/25. For the first update, CFCA reaggregates the tree with bottom-up traversal from the node \( C \), after which the nodes \( E \), \( F \) and \( C \) change their FIB status. For the announcement of the 129.10.124.128/25, the algorithm first changes the type of the corresponding node \( H \) from FAKE to REAL. Next, it performs the post-order and bottom-up
As we show later in this paper, LTHD can be built in the TCAM. In the case of a match, the matching prefix’s counter (L1) and Level-2 (L2) caches for future cache replacement.

If so, the prefix is being installed into the L1 cache. Similarly to PFCA, the essential part of the CFCA’s data plane is the Light Traffic Hitters Detection module (LTHD). The workflow of CFCA’s data plane is presented in Figure 7. Next, we describe the Light Traffic Hitters Detection module in more detail.

### 3.3 Light Traffic Hitters Detection

The light traffic hitters are the least popular prefixes in the cache. When an entry from a slower memory needs to be installed into a full cache, an efficient FIB caching solution should perform line-rate light hitters selection among tens of thousands of FIB routes.

As CFCA is designed on top of PFCA [14], it uses the same mechanism to collect the light traffic hitters in the data plane. The Light Traffic Hitters Detection module (LTHD) is able to find the least popular prefixes without real-time scanning through the cache. Its design is similar to “Heavy-Hitters” detection proposed by Sivaraman et al. in [31]. To collect the least popular prefixes, LTHD uses d hash tables $T_1, T_2, ..., T_d$. Each of the tables is assigned with a specific hash key $h_1, h_2, ..., h_d$. On a cache hit, the matched prefix p and its counter value c is pipelined through the hash tables. Consequently, LTHD selects the most popular prefix among p and a subset of prefixes in tables $T_1, T_2, ..., T_d$ and replaces that prefix with p, unless p is not the most popular one.

Consider an example in Figure 8 with a 3-stage LTHD and a pipelined prefix $p_1=11011$ and its counter $c_1=3$. At the first stage, $p_1$ is compared against the third entry of the table $T_1$, because its hash $h_1(p_1)=3$. Since the third entry’s counter is less than $c_1$, $p_1$ is carried to the second stage. There, $h_2(p_2)=2$ and $T_2$’s second entry with the prefix $p_2=01101$ has the counter $c_2>c_1$. Thus, $p_1$ replaces $p_2$ and $p_2$ is carried to the last stage. Finally, $h_3(p_3)=1$ and the first entry in the Table $T_3$ has a counter $c_3>c_2$. Thus, $p_2$ replaces $p_3=01101$ in $T_3$ as a less popular prefix. $p_3$ is removed from the LTHD and will not be considered as a potential victim cache entry.
We evaluate the performance of CFCA by comparing it to pure FIB caching (PFCA) and FIB aggregation algorithms with incremental BGP updates handling (FAQS and FIFA-S):

- **PFCA** is a FIB caching-only architecture for a programmable switch described in [14]. We use PFCA for comparison in terms of cache-miss ratio and BGP-related cache churn.
- **FAQS** [22] is a FIB aggregation algorithm with minimal data plane churn and suboptimal aggregation ratio.
- **FIFA-S** [24] is an ORTC-based FIB aggregation algorithm with an optimal aggregation ratio.

To emulate the operation of a programmable switch with CFCA we designed its data plane prototype with P4 language. As works on the P4 compiler for PSA are in progress [32], our prototype was designed for the Simple Switch architecture [34], and thus had limited capabilities compared to the original design of CFCA. For example, “Simple Switch” lacks the generation of packet digests in a data plane, i.e., messages that can be forwarded to a control plane, which is necessary for detecting the heavy hitters at the latter. While our future work will focus on designing the full P4 prototype of CFCA, in this paper, we evaluate the efficiency of CFCA’s combined FIB caching and aggregation using a trace-driven simulator written in C.

### 4.1 Experimental setup

We use RouteViews [4] to build a routing table that contains 599,298 entries. Our experiment simulates router operations for about an hour by processing a mixed trace of 45,600 BGP updates of RouteViews and an anonymized traffic trace from CAIDA [8] with 3.5 billion packets. For performance evaluation, we measure the following metrics:

- **Cache-miss ratio** per 100K packets for Level-1 (L1) and Level-2 (L2) caches.
- The number of cache installations/evictions in L1 and L2 caches.
- The number of BGP updates applied to L1 and L2 caches compared to the total number of BGP updates.

We tune both CFCA and PFCA in the same way as follows:

1. We set the number of stages in the Light Traffic Hitters Detection (LTHD) module to four.
2. Each stage has a hash table of size 10.
3. Initially, L1 and L2 caches are empty; however, they are quickly filled up as the initial threshold for cache installations in DRAM and Level-2 cache are 1 and 15 matches, respectively.
4. Once the L1 and L2 caches are full, we set the thresholds to 100 matches per minute for DRAM and to 300 matches per minute for L2 cache.
5. We compare the performance of CFCA and PFCA for different sizes of L1 and L2 caches, namely 5K with 10K, 10K with 15K, and 15K with 20K.

We verified the correctness of BGP update handling by CFCA, PFCA, FAQS and FIFA-S with VeriTable [15], a tool designed for fast verification of forwarding equivalence of multiple forwarding tables.

### 4.2 CFCA vs FIB Caching (PFCA)

The results that compare CFCA with PFCA are shown in Table 2.

#### 4.2.1 Cache-miss ratio

We first measure cache-miss ratios. Here, the larger values are, the more IP lookups are to be performed at slow memory causing packet forwarding delays and even losses. It is ideal to have the L1 cache miss ratio is minimized. In CFCA, the average cache-miss ratio is 1.13% when the L1 cache contains only 0.83% entries of the FIB. Also, if the L1 cache contains more prefixes (2.50% of the FIB), the cache-miss ratio drops drastically to 0.058%, which is more than five times less than PFCA’s average cache-miss ratio (0.316%). Figure 9 shows this cache-miss ratio improvement through FIB aggregation (see the red line for the average). The number of L1 cache-miss lookups in

![Figure 9](image-url)
Table 2: CFCA vs FIB caching with PFCA. Evaluation summary

<table>
<thead>
<tr>
<th>CFCA/ PFCA</th>
<th>L1 size</th>
<th>L1 size</th>
<th>L2 size</th>
<th>L2 misses, %</th>
<th>L1 misses, %</th>
<th>L1 installations</th>
<th>L2 installations</th>
<th>L1 BGP churn</th>
<th>L1 BGP burst</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.83</td>
<td>5,000</td>
<td>10,000</td>
<td>1.133</td>
<td>0.153</td>
<td>109,070</td>
<td>623,994</td>
<td>88</td>
<td>2</td>
</tr>
<tr>
<td>CFCA</td>
<td>1.67</td>
<td>10,000</td>
<td>15,000</td>
<td>0.144</td>
<td>0.017</td>
<td>18,108</td>
<td>150,547</td>
<td>156</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>2.50</td>
<td>15,000</td>
<td>20,000</td>
<td>0.058</td>
<td>0.004</td>
<td>17,277</td>
<td>37,842</td>
<td>285</td>
<td>6</td>
</tr>
<tr>
<td>PFCA</td>
<td>0.83</td>
<td>5,000</td>
<td>10,000</td>
<td>4.135</td>
<td>1.258</td>
<td>302,337</td>
<td>664,580</td>
<td>51</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>1.67</td>
<td>10,000</td>
<td>15,000</td>
<td>0.873</td>
<td>0.211</td>
<td>72,400</td>
<td>118,847</td>
<td>133</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>2.50</td>
<td>15,000</td>
<td>20,000</td>
<td>0.316</td>
<td>0.071</td>
<td>32,242</td>
<td>58,246</td>
<td>195</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 3: CFCA L1 cache vs FAQS/FIFA-S

<table>
<thead>
<tr>
<th>CFCA/ FAQS</th>
<th>Compression ratio, %</th>
<th>FIB</th>
<th>BGP</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFCA</td>
<td>2.50</td>
<td>21,495</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>FAQS</td>
<td>28.04</td>
<td>36,386</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>FIFA-S</td>
<td>25.08</td>
<td>48,672</td>
<td>33</td>
<td></td>
</tr>
</tbody>
</table>

Figure 10: L1 cache in CFCA and PFCA

CFCA rarely exceeds 0.4%, while the number of cache-misses is negligible for L2. In contrast, the cache-miss ratio of PFCA reaches nearly 1% in the worst case.

4.2.2 Cache installations and evictions. In CFCA, the control plane aggregates FIB entries before pushing them into the data plane, and this aggregation dramatically decreases both L1 and L2 cache installation and eviction rates. The low installation and eviction rates help save energy and improve performance as writing on TCAM is computationally expensive [16]. We show the L1 cache installations for both PFCA and CFCA in Figure 10a. We set the sizes of the L1 and L2 caches to 15,000 and 20,000 entries, respectively. A cache entry installation does not require eviction until the cache is fully initialized (see the dashed lines in the figure). Note that both CFCA and PFCA apply the cold start strategy, i.e., L1 and L2 cache tables are initially empty and are filled up with traffic later. Once the cache tables become full, cache replacement is needed for a new cache installation, and we use Light Traffic Hitters Detection to identify victims for eviction.

4.2.3 BGP updates impact. One drawback of FIB aggregation is the potential cache churn increase in TCAM due to the BGP updates applied to a compressed FIB. Table 2 shows that BGP-related cache churn in the L1 cache of CFCA is slightly increased compared to PFCA, because PFCA does not aggregate the FIB. However, the percentage of TCAM updates in CFCA remains low. As an example, only 0.625% of all BGP updates result in TCAM updates when the L1 cache size is 15,000 (see Figure 10b). In addition, we evaluate BGP burst in L1, i.e., the maximum number of FIB changes caused by a single BGP update. Interestingly, CFCA shows lower BGP burst in all the configurations as shown in Table 2. The FIB aggregation compresses prefixes generated by the prefix extension, and therefore a single next-hop update of a parent prefix affects fewer entries in TCAM than PFCA with more extended prefixes present in TCAM.

4.3 CFCA vs FIB Aggregation (FAQS/FIFA-S)

FIB aggregation minimizes its disadvantages when applied together with FIB caching. Table 3 shows that the total churn, which includes L1 installations, evictions, and next-hop updates, in CFCA with 15,000 entries is far less than in FAQS, that is a FIB aggregation scheme with the minimal BGP-related cache churn. For ORTC-based algorithms with optimal aggregation ratio like FIFA-S, the total churn is even larger. Moreover, CFCA does not produce large BGP bursts, which we attribute to the observation that BGP updates are mostly related to unpopular routes. The only strength of FIB aggregation over FIB caching is that all the routes stay in a cache. In FIB caching, on the other hand, the lookups that do not find a prefix match in a cache are forwarded to lower memory obviously increasing forwarding delay. CFCA mitigates this drawback as it reduces the cache-miss ratio to 0.058% for the L1 cache with only 2.50% entries in the FIB.

4.4 Evaluating a heavier trace

In the previous subsections, we showed the performance of CFCA under a traffic load with the mean transmission rate of
about 1M packets per second. To test the behavior of CFCA under a heavier load, we collected a more recent (01/17/2019) 1-hour trace from CAIDA [8] with 7.7 billions of packets and a 1-hour BGP update trace with 1,364,963 routing updates. We also used a larger routing table with 725,344 entries and L1/L2 cache with 20,000 and 30,000 entries respectively.

Due to significantly increased packet frequency, the average cache-miss ratio for this trace is higher (~0.27%). However, CFCA keeps it within 1% (see Figure 11). For this trace, we also compared the BGP update handling time by CFCA vs PFCA, FAQS and FIFA-S (see Figure 12). CFCA’s control plane takes only 0.78µs per a BGP update on average, slightly slower than PFCA (0.69µs per update), due to the additional FIB aggregation. In the meantime, the time cost of FIB aggregation algorithms is much higher (2.11µs and 3.50µs per update for FAQS and FIFA-S respectively).

5 RELATED WORK

FIB aggregation. Given an arbitrary FIB table, the Optimal Routing Table Construction algorithm (ORTC) [10] builds an equivalent routing table with a minimal number of entries. Bienkowski et al. in [6] present the formal study on the trade-offs between FIB compression optimality and the cost of BGP updates and propose HIMS, an online FIB aggregation algorithm. FAQS [22] minimizes the data plane churn and BGP update handling complexity by sacrificing FIB compression ratio.

FIB caching. The idea of caching entries in a forwarding table goes back to the early stages of the Internet. In [17], Raj Jain et al. showed the "source locality" of packets and proposed caching FIB entries for active flows and prefetching entries for the corresponding reverse flows. Route caching was studied for Layer-4 switching as well [36]. Kim et al. give an overview of caching popular flows of IP routing table in [20]. Sarrar et al. in [30] propose Traffic-aware Flow Offloading (TFO) strategy that outperforms LRU for a cache with less than 10,000 entries. In [29], authors apply rule caching for lossy compression of packet classifiers and show that 3% of the original classifiers are sufficient for handling 99% of the traffic. The paper does not elaborate on a chosen real-time cache victim selection strategy. Katta et al. in Cacheflow [19] build dependency graphs and identify independent groups of rules to avoid cache hiding. Such an approach requires the control plane to perform graph computations each time a flow becomes popular and needs to be installed in the cache. Similarly, Bienkowski et al. in [5] propose an online Tree Caching algorithm, that requires no prefix extensions. Evaluation of Cacheflow showed 97% cache-hit rate with 10%-sized TCAM; the authors of Tree Caching do not provide experimental results but instead give a theoretical analysis of its algorithm. Both algorithms are suited for a data center’s Software-Defined Networks with medium-sized FIBs.

IHARC [9] speeds up software IP lookup by utilizing CPU cache for route caching. It merges prefixes in the cache by selecting index bits that point to different cache sets with maximum mergeability. However, IHARC’s architecture has several shortcomings. First, IHARC leverages a 3-level NART (Network Address Routing Table) data structure, with prefixes flattened to the lengths 8, 24 and 32. This significantly increases the size of a routing table and consequent update churn. Second, the index selection routine is an expensive operation invoked at each routing update. The updates require IHARC to entirely invalidate its caches for maintaining forwarding correctness, increasing cache-miss ratios. In [13], the authors improve IHARC with selective cache invalidations and reduced recomputations of index bits.

PFCA [14] leverages the programmable data plane in order to build a FIB caching architecture with two levels of cache. In this work, we enhance PFCA with a FIB aggregation module. Our novel approach, CFCA, achieves remarkably high cache-hit ratios with small cache sizes. While incrementally handling the routing updates, CFCA significantly decreases total FIB churn in TCAM compared to the standard FIB caching and aggregation algorithms.

6 CONCLUSION

With the migration of computations into clouds, network operators face more challenges with packet forwarding. Specifically, they need to ensure fast prefix lookups in FIBs while amounts of traffic and the size of the global routing table continue to grow. In this work, we present CFCA, Combined FIB Caching and Aggregation, which uses two FIB compressing techniques to offload the expensive TCAM memory used for FIB from unpopular routes. Evaluation of CFCA on large traffic trace and BGP update traces shows remarkably low cache-miss ratios and FIB churn compared to the existing FIB caching solutions. In particular, CFCA achieves up to 99.94% cache-hits and stabilizes the cache by more than 46% with 2.5% of the entire FIB in the cache. Finally, compared to FIB aggregation algorithms, CFCA avoids large bursts of TCAM updates, reducing costly TCAM churn by 40% on average.
ACKNOWLEDGMENTS
We would like to thank our shepherd, Dr. Kartik Gopalan, and the reviewers for their helpful comments.

REFERENCES