PFCA: A Programmable FIB Caching Architecture

Garegin Grigoryan, Graduate Student Member, IEEE, Yaoqing Liu, and Minseok Kwon

Abstract—Ternary Content-Addressable Memory (TCAM) chips are used to store Forwarding Information Bases (FIB) in modern routers. TCAM provides next-hop lookup for IP packets at the line-rate. However, TCAM is expensive and energy-consuming; in addition, the constant FIB growth may lead to TCAM overflow problem. Yet only a small portion of FIB entries carries the most of network traffic. Thus, FIB caching, namely, installing the most popular entries in a fast memory, e.g., TCAM, may significantly minimize TCAM usage. To date, FIB caching architecture has not been widely deployed in backbone routers due to high cache-miss latency and the lack of an efficient cache replacement strategy. In this work, we leverage the concept of the programmable data plane to design a Programmable FIB Caching Architecture (PFCA) with two levels of cache. We present a pipeline-based algorithm to detect the least popular prefixes in a cache for a victim selection. We tested the prototype of PFCA using real traffic traces and an FIB with more than 599K entries, and showed that PFCA can be implemented using P4 programmable data plane language. Our results show that PFCA achieves 99.8% hit ratio for Level-1 cache with 20K entries and nearly 99.9% hit ratio for Level-2 cache with 40K entries. We also demonstrate that PFCA significantly reduces the number of BGP updates in the cache and thus makes the cache more stable.

Index Terms—Routing scalability, FIB caching, TCAM overflow, IP prefix lookups, programmable data planes, protocol independent switch architecture.

I. INTRODUCTION

FORWARDING Information Base (FIB) is used for IP prefix lookups and packet forwarding based on a packet’s destination IP address. FIBs in backbone routers of many Internet Service Providers (ISPs) contain more than 700K IPv4 entries and the number of entries continues to increase in a superlinear fashion (see Figure 1). To achieve fast next-hop lookups while overcoming Longest Prefix Match (LPM) rule challenges, FIBs are usually installed in Ternary Content-Addressable Memory (TCAM) chips of modern high-end routers. TCAM is a unique type of hardware solution that provides one-cpu-cycle parallel search over hundreds of thousands of FIB entries. However, TCAM chips are significantly more expensive and energy consuming than Static Random Access Memory (SRAM) or Dynamic Random Access Memory (DRAM) chips [1]–[4]. The cost of storage per single bit in TCAM is about 30 times more expensive than in SRAM due to the greater number of transistors [3], [4].

While TCAM chip occupies much larger physical space than SRAM [5], the number of entries that TCAM is able to maintain is significantly smaller compared to both SRAM and DRAM. The size of TCAM available on the market in 2016 was ranging from 1Mb to 72Mb [6]. Due to the high power demands of TCAM chips, the smaller capacity TCAM are the most popular on the market. However, the constant increase of prefixes in the global FIB forces network operators to upgrade their forwarding devices with bigger-size TCAM [7]. Maintaining power-hungry TCAM chips significantly increases operators’ production costs for TCAM usage, including such expenses as providing cooling facilities for the networking hardware. The continuous growth of the global FIB size and widespread adoption of IPv6 may lead to TCAM overflow problem and thereby the Internet service disruptions [8]. These potential consequences drive network operators to buy new high-capacity TCAM chips with higher cost and additional energy consumption or limit the memory allocation for IPv6 routes [9].

There is a number of approaches targeting TCAM overflow and an FIB growth problems. While re-designing the networking protocols and the network architecture in general requires full transition for every participant on the Internet, FIB aggregation [10]–[13] can be applied locally to a router and compress the routing table by more than 70%. However, FIB aggregation has two major issues and thus in not utilized at the production level. First, a single BGP announcement may incur up to several thousand writes in an aggregated FIB [12], [13], while writing operations on TCAM are resource intensive [14]. Second, FIB aggregation does not fundamentally address the high skewness problem between the Internet traffic.
pattern and the number of routes. In fact, merely 1.93% of FIB entries cover more than 99.5% of flows going through ISP network [15].

FIB caching addresses an important property of the Internet traffic by storing the most popular routes in the expensive memory (such as TCAM) and relocating unpopular routes to cheaper memory units (such as DRAM). The switch hardware manufacturers widely applied FIB caching until early 1990s. Specifically, Cisco’s ‘fast switching’ [16] implied storing the recently used prefixes in faster memory. However, later, due to increased amounts of cache misses followed by packet losses at the slower memory buffers, FIB caching was removed from the standard switch architectures [17]. Furthermore, FIB caching solutions lack an efficient procedure to select victim cache entries for cache replacement. Random cache victim selection may result in evictions of popular prefixes from the cache. Thus, popularity-based victim selection is more reliable and accurate. It was shown that LRU (Least Recently Used) policy is the most efficient replacement algorithm [15], [17]. However, LRU is a software solution and is impractical for implementing on high-end routers with large routing tables [18] due to the performance concern. In [18], Liu et al. proposed to use idle timeouts to evict the least popular entries. Yet, such a strategy may lead to TCAM overflow when the number of new cache entries exceeds the number of idle cached flows.

To this end, we revisit the idea of FIB caching (or route caching) in a different perspective from the traditional approaches. We propose to leverage Portable Switch Architecture (PSA) [19] and P4 data plane programming language [20] to minimize cache-miss latency and enhance cache replacement performance. We call our new design a Programmable FIB Caching Architecture (PFCA). In PFCA, we adopt several features of PSA, P4 and programming data planes in general. We enumerate those features below:

1) We take advantage of the programmability of match-action tables to design two different levels of an FIB cache in the data plane. In addition, the rest of an FIB on a slow memory unit stays in the data plane as well.

2) We leverage the programmability of the ingress and egress pipelines in PSA, which allows PFCA to process cache-miss packets entirely in the data plane. Packets that did not find a matching entry in Level-1 cache are directed to Level-2 cache and the slow FIB, located on the slower memory units.

3) Several of the targets for programmable data plane, like TOFINO switches, support parallel packet lookup against multiple match-action tables [20], [21]. In our design, packets that found no matching prefix in Level-1 cache are then matched against Level-2 cache and the slow FIB in parallel, in order to decrease cache-miss latency.

4) PFCA uses P4 meters, registers and hash functions to identify unpopular and popular routes, trigger cache replacement operations and conduct pipeline-based packet processing. More specifically, we detect the heavy flows and collect the light-weight flows for the future cache replacement “on-fly”, without a need for iterating through large forwarding tables.

5) The programmability of PSA switches allows the network operator to tune a switch’ s parameters, such as cache installation thresholds, the cache sizes and light-weight flow filter size in order to adapt the switch to different traffic patterns.

In summary, our key contributions in this work are as follows:

1) We design PFCA, a Programmable FIB Caching Architecture, where cache-miss operations, heavy flow detections and victim entry selections are handled entirely in the data plane. PFCA can be programmed via P4 data plane programming language and deployed on devices with Portable Switch Architecture (PSA).

2) We develop a pipeline-based mechanism for cache victim selections and called it “Light-Hitter Detection Module” (LTHD). We emulate the module in a virtual environment with the P4-programmed virtual switch. We compare LTHD against the random cache victim selection and LRU-based policies and demonstrate the efficiency of our approach in terms of low cache churn (8 cache installations for LTHD vs 7 cache installations for LRU per 1M of packets) and low miss ratios (0.175% for LTHD vs 0.177% for LRU).

3) Our evaluation with the real network traffic traces shows that PFCA uses 3.34% entries (20K) of the full FIB to achieve 99.825% hit ratio for Level-1 cache, and only 0.015% of the total traffic is processed by the slow memory FIB thanks to the Level-2 cache with 6.68% (40K) entries of the full FIB.

4) We identify and demonstrate the advantages of deploying the programmable FIB caching solution, such as significant deduction of expensive writing operations on TCAM caused by BGP route updates.

The rest of the paper is organized as follows: in Section II we give the overview of FIB caching and programmable switches; in Section III, we describe the design of PFCA; in Section IV, we describe the partial prototype of PFCA in P4. We evaluate the performance of PFCA in Section V. In Section VI, we discuss the related work. Lastly, we conclude the paper in Section VIII.

II. BACKGROUND

A. FIB Caching

Typically, a router consists of the two main components: the data plane and the control plane. The data plane is mainly responsible for moving data packets to an egress interface, based on a Longest Prefix Match (LPM) of a packet’s IP destination address. The candidate prefixes are stored in a Forwarding Information Base (FIB), a collection of network destinations - IP prefixes - and the corresponding output network interfaces (next-hops). The control plane is mainly responsible for collecting and distributing routes with peers, as well as installing the best routes into the FIBs under its control. To ensure fast IP lookup in an FIB, hardware manufacturers build the data plane with power-hungry TCAM-based forwarding engines. The continuous growth of the global
FIB results in TCAM overflow problem. Moreover, since a typical FIB contains overlapping prefixes, to ensure correct LPM matching, FIB’s prefixes in TCAM need to be sorted by their prefix lengths. Thus, inserting new prefixes into an FIB during frequent BGP updates may require additional memory’s buffers are full [17]. Second, to the best of our knowledge, there is a lack of an efficient cache victim selection procedure. LRU (Least Recently Used) policy, shown to be the most optimal in [15], [17], is a software solution and can not be implemented for large prefix tables on TCAM. Third, overlapping entries in the FIB cache and the full FIB might lead to the problem known as cache hiding. We give the full description of the cache hiding problem in the following subsection.

B. Cache Hiding of a Prefix

Cache hiding is a natural problem that comes from dropping entries from a routing table that uses the LPM rule to find a match for an IP address. For example, suppose the full FIB contains a more specific prefix $p_A = 128.153.64.0/18$ than a prefix $p_B = 128.153.0.0/16$ existing in a cache. The next-hop assigned to $p_A$ in the full FIB is $A$, while IP packets with addresses matching $p_B$ are forwarded to a next-hop $B$. In such case, a packet with IP destination address $128.153.183.226$ matching $p_A$ will be forwarded to the next-hop $B$ rather than $A$, since in the cached FIB its Longest Prefix Match is the prefix $p_B$. We illustrate this case on Figure 2. Cache hiding may be caused by (1) incorrect initial cache with less specific prefixes than in the full FIB on the slow memory; (2) A BGP prefix announcement, when a more specific prefix than another prefix existing in the cache is installed into the full FIB; (3) Similarly, BGP withdrawal might cause a situation when a cache contains less specific entries than in the full FIB.

When designing PFCA, we aimed to minimize cache-miss latencies, develop an efficient and fast cache victim selection procedure and avoid cache hiding problem. To this end, we leverage the emerging concept of the programmable data plane which we briefly describe next in this paper.

C. Programmable Data Plane

The numerous emerging heavy-workload IT applications require the underlying network to quickly adapt to their demands. Thus there is a need to build more programmable and less hardware-dependent networks. OpenFlow protocol [23] and Software-Defined Networking [24] in general are important tools for building programmable networks focusing on a programmable control plane. However, OpenFlow switches are constrained with the fixed data plane configuration. Differently, the switches standardized with the Portable Switch Architecture (PSA) [19] provide the data plane with a reconfigurable parser, forwarding (or match-action) tables and ingress/egress pipelines and can be fully programmed with P4 data programming language [20]. In addition, PSA switches support packet cloning, packet recirculation, shared metadata between different stages of the packet processing pipeline and provide access to data plane registers and hash calculations.

Figure 3 shows the pipeline of PSA. It consists of programmable blocks, such as ingress and egress pipelines, packet parsers and deparsers; hardware-constrained blocks responsible for a packet replication and buffering. The ingress and egress pipelines may contain registers and match-action tables with packet counters and meters attached to each entry. An output port is assigned to a packet when its’ headers leave the ingress pipeline. Alternatively, a packet can be cloned to multiple output ports or forwarded to the control plane. In addition, PSA data plane can be programmed to form packet digests for further analysis at the control plane.

In our work, we leverage both PSA and P4 to design the data plane architecture for efficient FIB caching. In addition to our data plane program, we use API generated by P4 compiler to build the partial prototype of our FIB caching solution and simulate it in a P4 software switch. Note that the work on PSA compiler is still in progress, however the design proposed in this paper is compatible with the proposed PSA specification [19].

III. DESIGN

A. Overview

We illustrate the design of PFCA in Figure 4. Same as in a typical router, the control plane in PFCA is responsible for collecting and distributing routes, selecting best routes and installing them into the data plane. In addition to that, PFCA control plane contains the Route Manager module, which stores the full FIB, generates the set of non-overlapping routes and distributes them among the tables of the data plane. The data plane of PFCA is based on Programmable Switch Architecture (PSA) and consists of the packet parser, ingress and egress pipelines, packet deparsers and buffers. We construct the ingress pipeline of the PFCA switch with three forwarding tables: (1) Level-1 cache stored in TCAM for
the most popular prefixes; (2) Level-2 cache stored in SRAM for less popular prefixes; (3) The full FIB with unpopular DRAM. The essential part of our design is the Light Traffic Hitters module, that is placed in the ingress pipeline and processes packets that were assigned a next-hop in Level-1 or Level-2 cache.

Overall, PFCA does not require specific hardware compared to standard programmable switches such as Tofino, where the ingress pipeline is built upon different types of memory, such as TCAM and SRAM [25]. The two-layer cache architecture with an SRAM chip in the Level-2 cache minimizes delays in case of Level-1 cache misses. To keep track of the most popular flows, every FIB entry in Level-1 cache, Level-2 cache and the slow FIB is linked with a traffic popularity meter.

PFCA’s workflow is as follows: first, when a packet arrives at the data plane, the parser decouples the Ethernet and IP headers of a packet. Next, the packet’s destination IP address is matched against Level-1 cache, located on a TCAM chip. If there is a hit, the matched prefix is processed through the Light Traffic Hitters Detection module, that collects into its hash tables the least popular prefixes as cache victims. Subsequently, the packet’s Ethernet frame header is rewritten, the packet is passed to the egress pipeline, re-assembled and forwarded to the corresponding next-hop. If there is a Level-1 cache miss, the packet is matched to both Level-2 cache and DRAM table in parallel. The match on Level-2 cache leads to the same procedure as with Level-1 cache hit, except that Light Traffic Hitters Detection operates with the prefixes from Level-2 cache. For a match on DRAM there is no further processing of a packet and it is immediately passed to the egress pipeline.

We leverage parallel table matching of the programmable data plane in order to minimize latencies in case of Level-2 cache miss. The overhead may be a concern here, but note that as we show in Section V, only a tiny part of the traffic (less than 0.2%) is forwarded towards the second stage table matching. For instance, if the total traffic rate is 1 Tbps at the router, then Level-2 cache and DRAM only needs to handle less than 2 Gbps of the traffic, which is an easy task for a typical modern routing device. Therefore, compared with an existing router without caching capabilities, both hardware costs and energy consumption in the new solution will be significantly reduced.

Next, we present the detailed description of the route manager functionality and Light Traffic Hitters Detection module.

B. Route Manager

The Route Manager belongs to the control plane and is responsible for several tasks in PFCA:

1) Control Plane Initialization: The route manager receives the initial FIB entries from the control plane’s BGP instance. Alternatively, the control plane may collect the network routes through OSPF, IS-IS or other network protocols; in this work we assume that the control plane uses BGP. To avoid cache hiding of a prefix (see Section II-B), the Route Manager converts the original prefixes into a set of non-overlapping prefixes. We call this procedure as “prefix extension”. Prefix extension is performed by adding the original prefixes into the full binary tree with the following properties:

1) Each node in the tree may have zero or two children.
2) The root node of the tree corresponds to the root prefix 0/0.
3) The left (right) child of a node corresponds to a more specific prefix with an appended 0 (1) bit.

To add a new prefix $p$ with a length $l$ to the full binary tree, the Route Manager traverses the tree from the root, using the path defined by the bits of $p$. If the path ends at a level $l'$, the Route Manager will generate $l-l'$ sibling nodes to ensure that the new set of leaf nodes will not contain any overlapping prefixes. The nodes generated during prefix extension inherit the next-hops of their closest parent nodes that relate to an original prefix.

Alternatively, we could generate non-overlapping prefixes on-fly, when moving less specific prefixes into the higher-lever memory. However, with such an approach, a “hot” entry migration to a cache might be delayed by prefix extensions due to costly computations at the control plane (see Section III-D for complexity analysis). To avoid these delays, PFCA performs prefix extensions pro-actively upon BGP updates.
Fig. 5. Prefix extension in PFCA. The dashed nodes are generated when adding the prefix 128.153.64.0/18 to the tree.

After all original prefixes are added to the full binary tree, the Route Manager collects the set of non-overlapping prefixes using the leaf nodes of the tree. Each leaf node has four parameters:

1) The value of the prefix associated with a node;
2) The corresponding next-hop;
3) Cache flag, indicating the current location of the prefix in the data plane. Possible values are 0 (DRAM), 1 (Level-1 cache on TCAM) and 2 (Level-2 cache on SRAM);
4) REAL/FAKE flag, indicating if this node and its prefix were generated during prefix extension.

The new set has an equivalent forwarding behavior as the original routing table. Meanwhile, the size of the non-overlapping set is larger than the size of the original routing table. For example, an FIB containing 599,298 prefixes generates 844,108 non-overlapping prefixes [26]. However, the increase in size of the full FIB is not a concern here, as FIB caching dramatically reduces the number of entries in the expensive TCAM memory, as we show in Section V. In addition, since we use only non-overlapping prefixes, the LPM operation on TCAM does not require strict ordering of the table’s entries by their prefix length.

We illustrate prefix extension on Figure 5, where we use prefixes from the cache hiding example given on Figure 2.

2) Data Plane Initialization: Initially, the cache flag values for every node in a tree is set 0 and the corresponding prefixes are installed only into the FIB table on DRAM. There are several ways to fill up the initial caches. One way is simply a traffic-driven initialization ("cold start"), based on a real-time pass-through traffic. More specifically, as network traffic goes through the switch, popular prefixes are being identified and recorded. Furthermore, popular prefixes are being installed only into the FIB table on DRAM. There are several ways to do this. For example, in [18], authors select the less specific routes for the initial cache. In this work, we use "cold start", as it results in a more stable cache than the other approaches.

3) BGP Update Handling: BGP updates (i.e., route announcements and withdrawals) are notified to the Route Manager by the control plane’s BGP daemon. Since caches and a full FIB in the data plane contain extended non-overlapping prefixes, a single BGP update may result in multiple changes. In this subsection we describe how PFCA handles each type of a BGP update.

- For an announcement of a new prefix, the Route Manager first updates its own version of the full FIB (i.e., the prefix binary tree). If the updated prefix p is located on a leaf node, then the Route Manager needs to push the update of p into the data plane. Otherwise, if the updated prefix p is located on an internal node, then the Route Manager performs a tree traversal in order to update all FAKE nodes that inherited their next-hop value from p. For each traversed leaf node, the Route Manager pushes the update into the data plane, according to the cache flag of that node.

- Announcement of a new next-hop for an existing prefix, the Route Manager first updates its own version of the full FIB (i.e., the prefix binary tree). If the updated prefix p is located on a leaf node, then the Route Manager needs to push the update of p into the data plane. Otherwise, if the updated prefix p is located on an internal node, then the Route Manager performs a tree traversal in order to update all FAKE nodes that inherited their next-hop value from p. For each traversed leaf node, the Route Manager pushes the update into the data plane, according to the cache flag of that node.

- For a prefix withdrawal, the Route Manager checks if such prefix p or its fragmented prefixes exist in the data plane and the hash tables of the Light Traffic Hitters Detection module and removes the prefix(es) from there if so. More specifically, the Route Manager traverses the branch of its prefix binary tree rooted at p, and checks the flags of the FAKE leaf nodes that inherited their next-hop from p and performs prefix deletions from the data plane, respectively to the values of the flags.

Although PFCA allows multiple changes in the forwarding tables upon a single BGP update, most of the BGP updates do not affect the popular routes in both caches, as we show in Section V. Therefore, a great advantage of using PFCA is that it can avoid massive expensive writing operations on TCAM and SRAM, and thus significantly boost the network throughput.

We present the graph that illustrates BGP update process on Figure 6. Note that the root node in the binary tree at the Route Manager is assigned with a default next-hop 0.

4) Popular Routes Installation: In our design, the Route Manager performs prefix migration to a faster cache table once a prefix installed in Level-2 cache in SRAM or the slow FIB in DRAM reaches a certain number of matches over a period of time, e.g., 100 hits/s. A threshold can be configured depending on several factors, such as the volume of the traffic and the...
network traffic, the flow monitoring by the control plane is associated with high bandwidth and computational costs.

In this work, we introduce a Light Traffic Hitters Detection module to find the least popular prefixes inside the data plane, without real-time scanning through the entire prefix list in the caches. Our approach is inspired by the algorithm described in [27] for finding $n$ most popular flows (“Heavy-Hitters”). In their work, Sivaraman et al. propose HashPipe, that uses $d$ disjoint hash tables $T_1, T_2, ..., T_d$ in a pipeline for collecting and storing the heaviest flows at a line rate. Each table in HashPipe is associated with an independent hash function $h_1, h_2, ..., h_d$ which returns the $i$th slot in a corresponding table. When a packet arrives at the first table (Stage 1), HashPipe calculates the hash key using $h_1$ function and the flow key $k$. Suppose, the output of $h_1(k)$ is the slot $i$.

If the $i$th slot is empty, then HashPipe installs the flow key $k$ and its corresponding counter value $c$ into the first table. If the slot is occupied by a flow with the same key $k$, then its counter value is updated with $c$. If the slot is occupied by another flow with the key $k'$ and it has a smaller counter value $c' < c$, $k'$ is evicted from the first table and “carried” to the next hash table (Stage 2) along with its counter. The flow key $k$ and the corresponding counter value is installed instead of $k'$. Otherwise, if the flow with the key $k'$ is more popular than $k$, flow is carried to the Stage 2. The whole process is repeated until the last stage.

One side effect of HashPipe is that duplicate entries may appear at different stages of the hash tables. For example, let $h_n(k_j)$ represent the hash value of key $k_j$ in the $n$th hash table. Now, suppose $h_1(k_1) = h_1(k_2)$, where $k_1$ is a more popular flow than $k_2$ and $k_1$ occupies the $i$th slot in the first hash table. Assume at the second stage, the flow $k_2$ is installed in the $j$th slot. Later, the flow $k_2$ becomes more popular than $k_1$, and when the flow $k_2$ is carried through the first hash table, it displaces $k_1$ and its counter from the $i$th slot. Thus, the first and the second tables contain a duplicate entry with the flow $k_2$ (at $i$th and $j$th slots respectively).

Based on the similar idea in HashPipe, we designed a new pipeline of multiple stages to detect the least popular flows (i.e. prefixes) for quick cache replacement. In addition, we addressed hash entry duplicate issues, since (1) the duplicate entries occupy the slots that can be used for other flows; (2) duplicates complicate the hash entry removal upon cache eviction requests. In Light Traffic Hitters, at the first stage, if the $i$th slot is occupied by a flow key $k_1$ with the greater counter value $c_1 > c_2$, where $c_2$ is the counter value for the flow $k_2$, $k_1$ is evicted and replaced by ($k_2$, $c_2$). $k_1$ is then carried to the next stage along with its counter. In contrast, if $k_2$’s counter value is equal or greater than $k_1$’s, $k_2$ is carried to the next stage. We eliminate hash entry duplicates by carrying the first installed flow key and its counter through the stages, until the first duplicate is found (or until the last stage if the duplicate is not found). Note that there is no need to carry the key that was evicted at an earlier stage $n$ and installed at a later stage $m$, $d \geq m > n$, because when the key was installed in the $n$th table, the possible duplicates in any table $T_m$, $d \geq m' \geq m > n$, should have been removed.

C. Light Traffic Hitters Detection

One of the biggest impediments to implement an efficient FIB caching architecture was the lack of an effective mechanism to quickly and accurately identify the unpopular routes for cache replacement. Due to extremely high rates of the available memory on TCAM/SRAM. After prefix is migrated, its counter is set to 0. In case a faster cache table is full, before installing a new entry, the Route Manager randomly picks a victim cache entry from the Light Traffic Hitters Detection module and evicts that entry back to the slower memory (see Figure 7). We describe the cache victim selection process in the next subsection.

Fig. 6. BGP updates processing by the route manager.

Fig. 7. Popular routes installation by the route manager.
We show the pseudo-code of the first stage of Light Traffic Hitters Detection on Algorithm 1. Figure 8 illustrates the new design with an example. At the first stage, a flow $k_1 = 01011$ with the hash $h_1(k_1) = 4$ and counter $c_1 = 2$ is compared to the flow $k_2 = 11101$ with counter $c_2 = 3$, that occupies the 4th slot in the first hash table. Since $c_1 < c_2$, the flow $k_1$ replaces the flow $k_2$. At the next stage, the duplicate of $k_1$ from the 2nd slot is removed; in addition, the flow $k_2$, carried from the first stage, replaces the more popular flow $k_3 = 11111$. At the last stage, $k_3$ does not replace the flow 10010 since its counter $c_3$ is greater by one. Thus, the flow $k_3$ is evicted from the Light Traffic Hitters’ hash tables.

The pipelining in Light Traffic Hitters Detection happens in PFCA on each packet hit in TCAM or SRAM. Note that PFCA uses different hash pipelines for TCAM and SRAM to separate victim prefixes for those forwarding tables.

**D. Complexity**

In this subsection, we analyze the complexity of the algorithms proposed in PFCA. Adding a new prefix with the length $w$ into the binary tree at the Route Manager requires $2 \cdot w$ memory accesses due to prefix extensions at each level of a tree between the root node and the new node. Hence, the running time of a BGP update handling is equal to $O(2 \cdot w)$, when the new prefix is more specific than existing ones. In the meantime, in a worst-case scenario, a BGP update for a less specific prefix $p$ may require a full post-order traversal of the tree rooted at a node corresponding to $p$. Such a traversal runs in $O(2^{n-1})$, where $n$ is the number of entries in the routing table with the extended prefixes. As we show in Section V, the BGP updates in fact rarely affect popular entries in TCAM and thus the slight increase in the BGP update handling delay will not degrade the forwarding correctness of the cache entries.

---

**Algorithm 1 Light Traffic Hitters Detection (Stage 1)**

<table>
<thead>
<tr>
<th>Input</th>
<th>$(k, c)$, where $k \leftarrow$ flow key, $c \leftarrow$ its counter; $d \leftarrow$ number of stages; $h_1, h_2, ..., h_d \leftarrow$ independent hash functions; $T_1, T_2, ..., T_d \leftarrow$ hash tables.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i = h_1(k)$</td>
<td></td>
</tr>
</tbody>
</table>
| if $i$th slot of $T_1$ is empty then | Install $(k, c)$ into $T_1$
| Carry $k$ to remove possible duplicates in $T_2..T_d$. | goto the next stage |
| else if $i$th slot in $T_1$ is occupied by $(k, c_0)$ then | $c_0 = c$ |
| else if $i$th slot in $T_1$ is occupied by $(k', c')$ and $c < c'$ then | Replace $(k', c')$ by $(k, c)$ in $T_1$
| Carry $(k', c')$ for installation in the next stages | goto the next stage |
| Carry $k$ to remove possible duplicates in $T_2..T_d$. | |
| goto the next stage |
| else if $i$th slot in $T_1$ is occupied by $(k', c')$ and $c \geq c'$ then | Carry $(k, c)$ to the next stage. |
| goto the next stage | |

---

**IV. P4 Prototype**

We built a partial prototype of PFCA in P4$_{16}$ data plane programming language [28] with respect to the Portable Switch Architecture (PSA) [19]. The design of PSA is fully compatible with the PFCA architecture. First, it allows the programmability of the ingress and egress pipelines. Next, PSA switch can allocate necessary registers to implement Light Traffic Hitters Detection module. The forwarding tables in the ingress pipeline can be enriched with counters or meters per each entry, which enables heavy hitter detection with PFCA. Alternatively, P4 registers can be used for detecting the heavy flows. Upon such detection, the ingress pipeline can generate a notification (via packet digest construction of PSA architecture) and send it to the control plane port. Finally, several PSA targets support parallel matching against multiple tables in the ingress pipeline [21], [29], as we design in PFCA with SRAM and DRAM forwarding tables.

In our P4 code, first we defined the packet headers and the parser that decouples the Ethernet frame and IP headers from a packet. Second, we defined the match-action tables for both caches and the FIB on DRAM. Next, we defined the registers necessary for storing prefix counters and prefix hashes of Light Traffic Hitters Detection module.

1 According to the current specification of PSA, the values of the counters cannot be read from the data plane. Implementing PFCA with PSA will be facilitated if such constraint is eliminated.
We show the snippets of the ingress pipeline code in Listing 1. Initially, we let ingress traffic pass through Level-1 cache. In case of a table hit, a packet’s metadata field \( \text{miss} \) is set to zero, so the control flow passes the packet’s header to the Light Traffic Hitters Detection module. The module is programmed in P4 using hashing functions available in the virtual P4 switch and the standard control flow commands provided by P4. In case of a Level-1 cache miss, \( \text{miss} \) is set to one and the packet is matched against Level-2 cache and the slow FIB. A Level-2 cache hit is treated similarly to a Level-1 cache hit. We show the snippets of our P4 code for the Light Traffic Hitters Detection module in Listing 2.

To program the prototype of the Route Manager in the control plane, we used API generated by P4 compiler based on our P4 code. As the works on P4 PSA compiler are still in progress, we used a different switch model to run the emulation. We tested the partial PFCA implementation in bmv2 [30] virtual environment for P4-programmed switches. In our current P4 prototype, PFCA is able to forward packets and collect the victim routes into the hash tables of the Light Traffic Hitters Detection module. As the Route Manager detects heavy flows in Level-2 cache or the slow FIB, it randomly chooses victim entries and performs prefix migrations. In addition, the Route Manager handles the BGP update requests and applies them to the forwarding tables of the data plane.

V. EVALUATION

A. Experiment Setup

For the experiment, we used the following realistic datasets: (1) A routing table with 599,298 entries from the RouteViews project [31]; (2) A one-hour traffic trace with 45,730 BGP updates from the RouteViews project; (3) A one-hour anonymized traffic trace collected by the CAIDA [32] with more than 3.5 billions of packets. All datasets were collected at 03/17/2016. By combining these three datasets we simulated a one-hour operation of an L3 switch/router with PFCA. For a full simulation, we coded PFCA in C, including the control plane with BGP updates handler, the Route Manager and the ingress pipeline with the Light Traffic Hitters Detection module. We ran the experiment on an Intel Xeon Processor E5-2603 v3 1.60GHz machine. The correctness of BGP update handling for an extended FIB table in PFCA was verified with VeriTable [26]. We evaluated our FIB caching architecture with the following metrics:

1) Cache-miss ratio per 100K packets for Level-1 and Level-2 caches.
2) The number of cache installations/evictions in Level-1 and Level-2 caches.
3) The number of BGP updates applied to Level-1 and Level-2 caches.

```c
if (hdr.ipv4.isValid()){
    level1.apply();
    if (meta.miss_bit == 1){
        level2.apply();
        if (counter > threshold){
            // Building a packet digest for the control plane
            meta.packet_digest.prefix = prefix;
            meta.packet_digest.memType = DRAM;
        }
    } else {
        if (counter > threshold){
            meta.packet_digest.prefix = prefix;
            meta.packet_digest.memType = L2;
            /* Go to LTHD for Level-2 */
        } else {
            /* Go to LTHD for Level-1 */
        }
    }
}
```

Listing 1. Ingress pipeline at PFCA.

```c
/* Defining registers and the hash function */
Register bit <32>, bit <32>[20] lthd_keys_1;
Register bit <32>, bit <32>[20] lthd_keys_2;
Register bit <8>, bit <32>[20] lthd_counts_1;
Register bit <8>, bit <32>[20] lthd_counts_2;
Hash bit <32>[PSA_hashAlgorithm_t.CRC32] h1;
Hash bit <32>[PSA_hashAlgorithm_t.IDENTITY] h2;

/* LTLD module */
/* Stage 1 for a tuple (prefix, counter) */
hashed_prfx = h1.get_hash(pfx);
prefix_key_1 = lthd_keys_1.read(hashed_prfx);
prefix_counter_1 = lthd_counts_1.read(hashed_prfx);
/* Case 1: the slot is occupied by the same prefix */
if (prefix_key_1 == meta.matched_prefix){
    /* Update the counter and leave the pipeline */
    lthd_counts_1.write(hashed_prfx, counter);
    carry = 0;
}
/* Case 2: the slot is empty */
else if (prefix_counter_1 == 0){
    /* Install the key and the counter */
    lthd_keys_1.write(hashed_prfx, prefix);
    lthd_counts_1.write(hashed_prfx, counter);
    carry = 0;
}
/* Case 3: the slot is occupied by a heavier flow */
else if (prefix_counter_1 > counter){
    /* Replace the tuple (prefix_key_1, prefix_counter_1) */
    by (prefix, counter) and move to Stage 2/*
    lthd_keys_1.write(hashed_prfx, prefix);
    lthd_counts_1.write(hashed_prfx, counter);
    carry = 1;
}
/* Case 4: the slot is occupied by a lighter flow */
else {
    /* Carry (prefix, counter) to the next stage */
    prefix_key_1 = prefix;
    prefix_counter_1 = counter;
    carry = 1;
}
if (carry == 1){
    /* Stage 2 for a tuple */
    prefix_key_1, prefix_counter_1 */
}
```

Listing 2. Light traffic hitters detection at level-1 cache.
We did not measure cache-miss latencies in this experiment since the lookup latency depends on the switch target. However, the delay for a cache-missed packet can be estimated approximately as an LPM search time on TCAM ($\approx 4\mu s$ [33]).

B. Tuning the FIB Caching Architecture

Several parameters can affect the performance of PFCA. First, the number of stages in the Light Traffic Hitters Detection module can be changed, along with the size of each hash table, similarly to HashPipe proposed in [27]. Decreasing the size of the hash tables and increasing the number of stages may increase the accuracy of the less popular routes selection. However, the tuning process is also constrained by the switch’s memory limit. Second, the popularity threshold for migrating a prefix to a faster memory may be changed in Level-2 (SRAM) cache and the FIB in DRAM. The threshold affects the frequency of installations into Level-1 (TCAM) and Level-2 caches, as well as the frequency of cache victim evictions and replacements when a faster memory is full. A low value of the threshold typically results in more frequent installations and victim evictions. Furthermore, we observed that frequent installations into a cache lead to increasing number of cache misses. On the other side, the high value of the threshold deters installations of heavy flow prefixes into a faster memory, which might increase the number of cache-misses as well.

The optimal tuning of PFCA also depends on the memory size of TCAM/SRAM chips and the patterns of the traffic on the switch. While the theoretical approach to finding the optimal tuning parameters will constitute our future work, in this paper we demonstrate the simulation results based on the following parameters:

1) For the Light Traffic Hitters Detection module, we set the number of stages to be 4, and size of each hash table to be 10. Since each entry in the module contains a 4-byte prefix and a counter, the memory overhead for Light Traffic Hitters Detection is calculated as $(4+4)\times 4 \times 10 = 320$ bytes. Because we use separate cache victim tables for Level-1 and Level-2 caches, the overall overhead for the switch memory is 640 bytes.

2) When the caches are not full, we set low thresholds in Level-2 cache and the slow FIB, 15 and 1, respectively, to quickly initialize them.

3) If Level-1 cache is full, the threshold in Level-2 cache is set to 300 matches per minute.

4) If Level-2 cache is full, the threshold in the slow DRAM FIB is set to 100 matches per minute.

In the simulation we assume the limits of TCAM and SRAM memory sizes to be 20K and 40K entries, respectively. To verify the effectiveness of the chosen parameters, we ran a second experiment with a different traffic trace and obtained similar results.

C. Results

We started our experiment with empty Level-1 and Level-2 caches. As it can be seen from Figure 12, both caches are filled quickly because of the low thresholds we set for cache initialization. During the experiment, PFCA restricts Level-1 and Level-2 caches to 20K and 40K maximum entries respectively.

1) Cache-Miss Ratio: Cache-miss ratio is one of the most important characteristics to measure the performance of an FIB caching algorithm or framework. Figure 9 shows the cache-miss ratio with popularity-based victim selection for every 100K packets passing through the ingress pipeline of the switch. The least popular prefixes were collected by the Light Hitter Detection module. At the beginning of the experiment, the miss ratios for both cache tables were relatively high, since we did not initialize them up with empirically popular entries. However, the number of misses decreases quickly as the popular prefixes are filled into Level-1 and Level-2 caches. On average, the miss ratio for Level-1 cache is slightly less than 0.175%, and the miss ratio for Level-2 cache rarely exceeds 0.1% and is 0.015% on average. In general, the larger sizes of TCAM and SRAM caches, the lower cache-miss ratios for both caches.

Figure 10 shows the cache-miss ratio for Level-1 and Level-2 caches for a similar FIB architecture with random cache victim selection. The average miss ratio for Level-1 cache is 0.248% and the highest miss ratio after cache initialization is over 1%, which is much higher than that in the popularity-based victim selection. For Level-2 cache, the average miss ratio is very close to the one for popularity-based victim selection ($\approx 0.015\%$). Such results for Level-2 cache can be explained by the observation that it consists of prefixes whose popularity are more uniformly distributed. However, note that the random eviction results are versatile and uncontrollable, thus it may lead to a large number of installations/evictions if a popular prefix was selected. On the contrary, our popularity-based victim selection approach will always choose relatively unpopular prefixes, and thus the results are more reliable and predictable.

In addition to evaluating random cache victim selection strategy, we simulated PFCA with the LRU-based cache victim selection policy, i.e., an algorithm that stores prefixes in a heap and selects the last recently matched prefix as a cache victim. Obviously, such an algorithm cannot be implemented in a switch due to its complexity. However, our goal was to compare the results of LRU-based victim selection with our Light Hitter Detection module. After running the simulation with the same data traces, we obtained the...
average cache-miss ratio for Level-1 cache slightly worse than cache-victim selection based on light traffic hitters (0.177% vs 0.175%). Similarly, for Level-2 cache, LRU-based victim selection resulted in more cache-misses than when using Light Traffic Hitters Detection module (0.017% vs 0.015%). Figure 11 shows the cache miss ratio of LRU-based victim selection for every 100K packets during the experiment. Overall, Light Traffic Hitters Detection achieves the smallest cache-miss ratio while using a line-rate algorithm.

2) Installations/Evictions in a Full Cache: Recall that the installations are triggered when a prefix in a slower memory becomes popular, i.e., its counter reaches a threshold value. If the threshold is reached for a prefix in Level-2 cache, the prefix will be removed from it and installed into Level-1 cache; if the threshold is reached for a prefix in the FIB on DRAM, the prefix is installed into Level-2 cache and its Cache flag is set. Note that if a cache is full, each installation requires an eviction of another prefix from the cache.

Figure 13 illustrates the numbers of cache installations and evictions in a full Level-1 for (a) popularity-based, (b) random and (c) LRU-based victim selection implementations. As we can observe in the graph, the popularity-based approach yields nearly half of the number of cache installations and evictions, when compared to the random victim selection approach. As expected, the LRU-based victim selection approach, that chooses the least matched prefix as a cache victim, achieves the smallest value for installations and evictions. However, as we show in V-C.1, Light Traffic Hitters Detection module achieves the minimal cache-miss ratio among those three approaches. On average, PFCA with the popularity-based approach installed only 8 entries in Level-1 cache and 20 entries in Level-2 cache for every million of data packets.

3) BGP Updates Impact: Figure 14 illustrates that 45,600 BGP updates triggered only 292 writes for Level-1 cache in TCAM and 1,120 writes for Level-2 cache in SRAM during a one-hour simulation. The results demonstrate that most of the BGP updates are not linked with popular routes in the caches, which indicates that FIB caching utilizes expensive routing memories more cost-effectively than existing practice. Intuitively, we can explain these results by the fact that the most of the BGP instability comes from misconfigurations and failures, which is peculiar to the least popular destinations [34], [35].

An important advantage of PFCA is that it can minimize the number of expensive writing operations on TCAM, since each write in TCAM triggers resorting of the entries on the chip. Meanwhile, the number of installations of popular routes is minimal once prefixes from heavy flows are installed into TCAM and prefixes from active flows are installed into SRAM (see Figure 13). Note that the number of installations into a table is equal to the number of evictions in case if the memory on a chip is full.

In general, the results show efficiency of the proposed architecture. PFCA can achieve the minimal number of misses.
with only 3.5% of the global FIB installed on TCAM and
13.2% of the global FIB installed on SRAM, significantly
reducing hardware and energy costs for these memory units.

VI. RELATED WORK

The problem of TCAM overflow and its possible solutions
were studied in [7]. Routing protocols such as Locator/ID
Separation Protocol (LISP) [36] need a global re-architecture
of the network. The aggregation of the global FIB can be
applied locally to a router and save more than 70% of TCAM
space [11]–[13]. However, this solution has two weaknesses.
First, it incurs a large number of TCAM writes when handling
BGP updates. Second, this solution does not utilize the high
skewness of the Internet traffic pattern and many aggregated
unpopular routes are still unused.

Kim et al. revisited FIB caching in [17]. The paper describes
the cache hiding problem and proposes prefix extensions to
avoid it. Gadkari et al. presented another study on efficiency
of FIB caching in [15]. The traffic analysis described in the
paper shows that an FIB cache with 10K entries achieves
99.5% cache hit ratio for an FIB with 500K entries. The paper
proposed an FIB caching architecture with the full FIB located
between the control and data planes of a router. The authors
in [17] and [15] showed a higher efficiency of LRU victim
selection policy than LFU. However, the cache replacement
policies in these studies are software-based and may not be
effective to handle the Internet traffic at a high speed. In [37],
authors present Traffic-aware Flow Offloading (TFO) strategy
for selecting heavy hitters. TFO outperforms LRU for a cache
with less than 10K routes. However, TFO requires collecting
flow statistics at a router’s controller and regular sorting of
the large number of prefixes according to their popularity.
Liu et al. in [18] proposed using OpenFlow and idle timeouts
for each cache entry to evict unused entries. This solution may
not be able to handle sudden bursts of new popular traffic,
in case if there is not enough entries being expired during
such bursts. In [38] authors solve cache hiding problem by
building dependency graphs and caching independent groups
of rules. Bienkowski et al. in [39] present an online Tree
Caching algorithm that requires no prefix extensions of the
FIB. To avoid cache hiding, both Cacheflow and Tree Caching
do not to install or evict separate routes from the cache,
but instead have to apply these operations on large sets of
overlapping routes, which is a complex operation for the global
FIBs. While evaluation of Cacheflow on backbone traffic traces
showed at least 88% cache-hit ratio, Tree Caching has not been
evaluated experimentally but instead was given with theoretical
bounds on its performance. In [40], Rottenstreich et al.
propose lossy packet classifier compressors for optimale rule
caching, that can be used for identifying the most popular
flows in an FIB and offloading TCAM from idle flows. The
approximate classification achieves the optimal cache-hit
ratio given the size of the cache. However, it allows false
classifications for cache-misses which can lead to the incorrect
forwarding behavior of a router. The cached classification
achieves the full forwarding correctness by redirecting some
of the lookups to a slow memory. Both algorithms are based
on dynamic programming with non-linear time complexity,
which may impede the deployment of the optimal rule
caching in backbone networks with constantly changing
traffic patterns and thousands of BGP updates per second.

In PFCA, all data packets are entirely handled in the data
plane. We designed a pipeline-based algorithm to quickly
identify the heavy flows and the least popular prefixes for
cache replacement. We evaluated PFCA using the real data
traces with more than 3.5 billions of packets and the routing
table with almost 600K entries. In PFCA, Level-1 cache
with only 3.34% of entries achieves at least 99.8% cache-hit
rate. For Level-2 cache with only 6.68% FIB entries PFCA
achieves 0.015% cache miss ratio on average. In addition,
PFCA preforms a reliable and controllable line-rate cache
victim selection and replacement, and efficiently handles BGP
updates.

VII. DISCUSSION

One of the properties of PFCA is that its data plane operates
with solely non-overlapping prefixes. Thus, for each destina-
tion IP address, there can be only a single match in the FIBs
of PFCA, rather than multiple matches with different prefix
lengths. This property cancels the necessity of conducting
expensive Longest Prefix Matching (LPM) lookups. While
generating non-overlapping set of prefixes increases the size
of an FIB and the churn caused by BGP updates, PFCA
cancels these drawbacks by selecting a small subset of popular
prefixes for forwarding more than 99% of traffic; as we show
in Section V, this subset consists of 5% of entries of a full
FIB rarely affected by BGP updates. With the elimination
of LPM, prefixes on TCAM no longer need to be sorted
by their prefix lengths which significantly reduces the costs
of entry insertions and deletions. Moreover, it is possible to
leverage PFCA for software switches that do not use expensive
TCAM memory and apply forwarding hash tables on a cheaper
memory instead.

VIII. CONCLUSION

In this paper we introduce a Programmable FIB Caching
Architecture (PFCA) with two levels of FIB cache in TCAM
and SRAM respectively, and the slow FIB in a DRAM mem-
ory. A great advantage of this architecture is that operators
can flexibly configure their TCAM and SRAM for efficient
memory utilization. In addition, we proposed the Light Traffic
Hitters Detection module that performs pipeline-based cache
victim selection for cache replacement. We designed PFCA
based on Portable Switch Architecture (PSA) and partially
implemented its prototype using P4 programming language.
We demonstrated the effectiveness and efficiency of our archi-
tecture in terms of cache misses, route installations and evic-
tion, and BGP updates handling. We employed two-hour traffic
traces to evaluate and verify PFCA’s performance. According
to the evaluation, PFCA achieves 99.825% cache-hit ratio for
Level-1 cache with 3.34% entries of the full FIB. Moreover,
only 0.015% of the data packets were forwarded by the FIB
on the slow DRAM memory, thanks to the Level-2 cache
with 6.68% entries of the full FIB. Finally, 45,600 BGP updates included in the data trace resulted in merely 152 FIB changes in the Level-1 cache, showing that PFCA significantly minimizes BGP churn in the TCAM memory.

REFERENCES


[8] (2019). The 768k or Another Internet Doomsday? Here’s How to Deal With the TCAM Overflow at the 768k Boundary. [Online]. Available: https://www.nccn.org/blog/768k-day-512k-tcam


Garegin Grigoryan (Graduate Student Member, IEEE) received the Diploma degree in automated data processing and management systems from the Department of Cybernetics, Moscow Engineering Physics Institute, in 2012, and the M.S. degree in computer science. He is currently pursuing the Ph.D. degree with the Rochester Institute of Technology. In 2012, he joined Diasoft, a Moscow-based company that provides a wide range of financial software solutions. In 2016, he entered the Graduate Program of Clarkson University, Potsdam, NY, USA. His current research interests include routing scalability, aggregation and caching of forwarding tables, network security, software-defined networking, software switching, and programmable data planes.
Yaoqing Liu received the master’s and Ph.D. degrees in computer science from the University of Memphis. He is currently the Program Coordinator of M.S. in Cybersecurity and Information Assurance Program (MSCSIA) and an Assistant Professor with the Gildart Haase School of Computer Sciences and Engineering, Fairleigh Dickinson University (FDU). His research interests include ethical hacking, system vulnerability analysis, networked systems, and blockchain applications. His publications appear in highly reputed conference proceedings and journals, such as the IEEE INFOCOM, ACM SIGCOMM CCR, and ACM/IEEE ANCS. He has been a TPC member and a technical paper reviewer for IEEE conferences, journal magazines, and transactions. His research work has been selected as one of the 2018 TechConnect Defense Innovation Awards recognizing the potential positive impact for the warfighter and national security.

Minseok Kwon received the Ph.D. degree in computer science from Purdue University in 2004. He is currently a Professor with the Department of Computer Science, Rochester Institute of Technology (RIT). His main research interests include computer networks with focus on development of network system architectures and their evaluation. His work involves developing network algorithms and protocols in packet processing at routers, programmable routers, software-defined networking (SDN), container networking, network security, and data center networks. He has also been developing solutions for high performance file systems and replication in large-scale distributed computing. He has coauthored dozens of articles published in international journals and conferences, including the IEEE/ACM TRANSACTIONS ON NETWORKING, Computer Networks journal, the IEEE ICNP, the IEEE Globecom, and the IEEE IWQoS.