Pragmatic Router FIB Caching

Kaustubh Gadkari*, M. Lawrence Weikum†, Dan Massey* and Christos Papadopoulos*
*Department of Computer Science
Colorado State University
Email: $\{kaustubh, massey, christos\}$@cs.colostate.edu
†Email: lawrencemq@gmail.com

***Abstract*—Several recent studies have shown that router FIB caching offers excellent hit rates with cache sizes that are an order of magnitude smaller than the original forwarding table. However, hit rate alone is not sufficient - other performance metrics such as memory accesses, robustness to cache attacks, queuing delays from cache misses etc., should be considered before declaring FIB caching viable.**

In this paper, we tackle several pragmatic questions about FIB caching. We characterize cache performance in terms of memory accesses and delay due to cache misses. We study cache robustness to pollution attacks and show that an attacker must sustain packet rates higher than the link capacity to evict the most popular prefixes. We show that caching was robust, even during a recent flare of NTP attacks. We carry out a longitudinal study of cache hit rates over four years and show the hit rate is unchanged over that duration. We characterize cache misses to determine which services are impacted by FIB caching. We conclude that FIB caching is viable by several metrics, not just impressive hit rates.

I. Introduction

The growth of forwarding table (FIB) sizes, fueled by factors such as multi-homing, traffic engineering, deaggregation and the adoption of IPv6, has led to a renewed interest in FIB caching methods. Past work [9] has shown repeatedly that there is significant traffic locality in the Internet that makes FIB caching beneficial. However, FIB caching has not transitioned to practice. Part of the reason might be that past work has focused on demonstrating that FIB caching is beneficial, leaving several practical and engineering questions unanswered. These include, how should the cache be implemented in a modern line card? Who suffers most from cache misses and how? How long does it take for a miss to be serviced? What are the memory bandwidth requirements for cache updates? How easily can one attack the cache? In this paper we address such practical questions and show that FIB caching is not only highly beneficial, but also very practical. We hope that our work answers important engineering questions and leads to renewed interest in building routers with caches.

Is FIB caching still relevant? Can't Cisco already support a million entries in their line cards? Opinions range from "it's not an issue" to "the sky is falling". We do not attempt to take a position in this debate but seek only to inform. There are recent trends, however, that make the matter worth revisiting. One is the slow but steady growth of IPv6, which steadily adds more prefixes in the FIB. Another is the quest to build a Tb/s forwarding chip, which for packaging reasons will have limited on-chip memory, a perfect candidate for a cache.

ISBN 978-3-901882-68-5 © 2015 IFIP

In summary, we make the following contributions:

- We classify packets that cause cache misses according to their type and protocol.
- We evaluate the effect of cache misses on delay and buffer utilization.
- We evaluate the effect of caching on memory bandwidth requirements.
- We examine the behavior of the system under a cache pollution attack by someone who wants to replace popular with unpopular prefixes.

To achieve fast cache updates we propose an architecture that includes a *cacheable FIB* i.e. a FIB that does not suffer from the cache hiding problem (explained later). The cacheable FIB is derived from the standard FIB.

Briefly, our results are as follows: first, we confirm past observations that FIB caching is effective: with a cache size of 10K entries hit rates are in excess of 99.5%. Second, our classification of cache misses shows that NTP and DNS queries are the main culprits of cache misses. Not surprisingly, TCP control packets (SYNs, SYNACKs, FINs, FINACKs and RSTs) account for 70% of the TCP misses. Third, we show that very few packets need to be queued due to cache misses and they suffer insignificant queuing delays. Finally, we show that a cache recovers quickly when subjected to cache pollution attacks aiming to replace cache entries with unpopular prefixes. In our datasets, an attacker must send 1.2 billion packets/sec over a 10G link to effectively disrupt the cache.

Our data comes from a regional ISP and thus our observations are mainly from the edge of the network. However, our study can easily be repeated for the network core by someone with access to the appropriate packet traces.

The rest of the paper is organized as follows. Section II introduces previous work that has looked into FIB scaling methods. In section III, we introduce our FIB caching solution. In section IV, we introduce the cache hiding problem. Next, we introduce our hole filling algorithm and evaluate it in section IV-A. In section V, we describe the datasets used in our evaluation. We evaluate LRU caches in section VI and the effect of cache misses in section VII. In section VIII, we evaluate the robustness of our caching system when it is being attacked. Finally, we conclude in section IX.

II. Related Work

Approaches to reduce the FIB size have been studied for more than a decade ([3], [6], [8], [11], [14], [18], [19], [25],

[27], [31]). These approaches fall into two broad categories - caching-based and aggregation-based approaches. Our work is independent of FIB aggregation techniques and hence we do not discuss these approaches in this work. There are other approaches that achieve FIB size reduction by reducing the RIB size [7], [21], [30]. We believe that our work is complementary to this work and the reduction of the RIB size will result in a more dramatic decrease in the cache size.

The idea of using route caching was first proposed by Feldmeier in 1988 [8], which was further extended by Kim *et al.* in [14]. Kim *et al.* introduce the cache hiding problem, where a less specific prefix in the cache hides a more specific entry from the FIB, causing an incorrect forwarding decision. We discuss this further in section IV. To solve the cache hiding problem, the approach in [14] splits the IPv4 address space into the constituent /24 prefixes. Other approaches to handling the cache hiding problem include treating related prefixes as an atomic block so that all cache operations (insertion, lookup and deletion) are carried out on the block as a whole, using a complicated data structure with on-the-fly computation to remove inter-dependencies between prefixes [16] or using genetic algorithms to allow the cache policy to evolve while tracking the heavy hitter flows [29].

In this paper, we propose a *hole filling* algorithm (Section IV-A) to address the cache hiding problem, similar to the method proposed in [17]. While the algorithm presented in [17] generates the needed most-prefix on a per-packet basis, thus incurring a per packet cost, our algorithm pre-computes the prefix to add. By adding these extra prefixes, we ensure that there are no overlapping prefixes in the FIB. Consequently, a packet hitting a prefix not in the cache will cause the correct route to be fetched from the full FIB, instead of an incorrect longest prefix match with a covering prefix already in the cache. We discuss this algorithm in section IV-A. We show that the number of extra prefixes added to the FIB is only a small fraction of the total number of FIB entries. Further, by using this *cacheable FIB*, we show that we can forward 99% or more packets along the fast path with a cache size of the order of 10,000 entries.

In [26], the authors propose a traffic offloading strategy to leverage the Zipf-like properties of Internet traffic. The proposed system, Traffic-aware Flow Offloading (TFO) is a heavy hitter selection strategy that leverages the Zipf-like properties and the stability of the heavy hitters across various time scales. TFO maintains the set of current heavy hitter flows in fast memory and sends packets from flows not in the heavy hitter set to a slower path. Results show that TFO sends a few thousand packets/second to the slow path.

In [24], Rétvári *et al.* introduce the concept of *FIB compression*. They introduce two FIB compression techniques - XBW-l and trie-folding. Results show that the XBW-l compression technique can reduce the size of the FIB to 100-300KB, while FIBs compressed with trie-folding require 150-500KB of memory.

Previous work [10], [20] has investigated the impact of various cache poisoning attacks on caches and has proposed methods of mitigating such attacks. However, the systems investigated were software systems (web proxies) that have a different set of constraints than the hardware-based route caching system proposed in this work. Further, unlike previous work, we consider the impact of cache misses on the system performance not only in terms of hit rates achieved but also the types of requests resulting in cache misses and their implications for operators.

III. Cache System Design

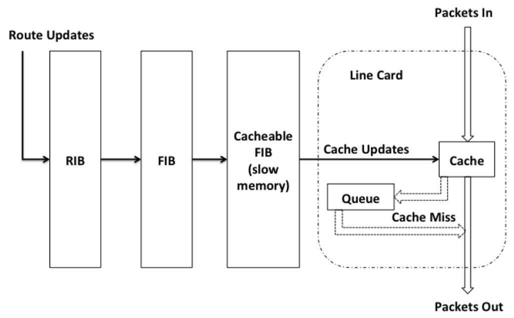


Fig. 1: Proposed Caching System Architecture

Figure 1 shows our proposed FIB caching architecture. The RIB and FIB on the left are part of the standard hardware architecture. We add (a) a *cacheable FIB*, which resides in main memory, (b) a cache, which can take the place previously occupied by the FIB, (c) a queue to hold packets experiencing a cache miss, and (d) the appropriate logic to handle cache updates and misses. The router processor handles incoming route information as usual and computes the local RIB (routing table) and FIB as before. In a traditional architecture the router would load the entire FIB in the line card; in our architecture, the router derives the cacheable FIB from the standard FIB and pushes just the cache in the line card, which is one or two orders of magnitude smaller than the original FIB.

Our caching architecture may seem to keep two copies of the FIB, the original and cacheable FIB, which look like a waste of memory. However, the two FIBs are very similar as we will show later and an implementer can easily use appropriate data structures to avoid duplication.

Each incoming packet incurs a lookup in the cache using the standard longest prefix match algorithm. If there is a hit, the packet is immediately forwarded along the cached route. If there is a miss, the packet is queued in the line card until the cache is updated with the appropriate prefix from the cacheable FIB.

Next, we elaborate on the need and derivation of a cacheable FIB.

IV. The Need for a Cacheable FIB

Entries in the original FIB cannot be cached due to the *cache hiding problem*, which occurs when a less specific prefix in the cache hides a more specific prefix in the FIB. This is a result of the fact that a route-caching system looks for the longest-matching prefix only in the cache, thus missing possible longer matches in the full FIB. This can lead to incorrect forwarding, loops and packet losses.

To further understand the problem, consider an empty cache and a full routing table that only has two prefixes:

10.13.0.0/16 associated with an output interface O1, and 10.13.14.0/24 associated with an output interface O2. Suppose a packet arrives with a destination IP address of 10.13.2.3. The router will find the longest-matching prefix (LMP) for the destination IP address, which is 10.13.0.0/16, and will install the route [10.13.0.0/16 $\rightarrow$ O1] in the cache. Assume that the next packet that arrives at the router has a destination IP address of 10.13.14.5. The router will find the previously installed route [10.13.0.0/16 $\rightarrow$ O1] in the cache and will forward the packet along O1. However, this is incorrect, since the correct LMP for 10.13.14.5 is 10.13.14.0/24 and the packet should have been forwarded on interface O2.

Past work addressed the cache hiding problem by either caching only /24 prefixes [14], by using a complex data structure with on-the-fly computation to eliminate inter-dependencies between prefixes [16], by adding on-the-fly to the FIB the appropriate leaf node corresponding to the packet's destination address [17] or by treating a set of related prefixes as an atomic block and performing cache actions (insertion, lookup and delete) on the atomic block instead of an individual prefix. Our approach is similar to the approach presented in [17], except that we pre-calculate the entire cacheable FIB table.

A. Generating a Cacheable FIB

We have previously seen that simply caching existing FIB entries would lead to incorrect forwarding due to the cache hiding problem. In this section we describe an algorithm to generate a *cacheable FIB*, i.e. a FIB that is free from the cache hiding problem.

We call our algorithm the *hole filling* algorithm because it starts with the original FIB and fills in holes between related entries to produce a new FIB whose entries are cacheable. While the new FIB is larger because the algorithm adds more entries, the increase is small, as we will show later, and caching makes it irrelevant.

The intuition behind the algorithm is as follows - if a prefix covers other prefixes in the table, we delete that prefix and add its children to the FIB. We repeat this process until there are no covering prefixes, at which point we are left with a cacheable FIB.

To better understand the algorithm, consider the FIB shown in Table Ia. This FIB is non-cacheable, since the 10.13.0.0/16 prefix, if present in the cache, will hide the 10.13.14.0/24 prefix. This FIB needs to be transformed into a cacheable FIB.

We first choose the prefix from the FIB with the smallest mask length, which is 10.13.0.0/16 in this case. Then we check if this prefix has any children. If the selected prefix has a child, we split the prefix into its two children, otherwise we add the prefix to the cacheable FIB. In this example, since 10.13.0.0/16 has a child, 10.13.14.0/24, we split the /16 into its two constituent /17s and remove the original 10.13.0.0/16, as shown in Table Ib.

The process continues, until the non-cacheable FIB is empty, and the cacheable FIB contains the leaf nodes necessary to forward all packets correctly. Table Ic shows the final, cacheable FIB.

Prefix	IFF
10.13.0.0/16	1
10.13.14.0/24	2

(a) Non-cacheable FIB

Prefix	IFF
10.13.0.0/17	1
10.13.128.0/17	1
10.13.14.0/24	2

(b) Non-cacheable FIB after one iteration of hole filling algorithm

Prefix	IFF
10.13.128.0/17	1
10.13.64.0/18	1
10.13.32.0/19	1
10.13.16.0/20	1
10.13.0.0/21	1
10.13.8.0/22	1
10.13.12.0/23	1
10.13.14.0/24	2
10.13.15.0/24	1

(c) Cacheable FIB

TABLE I: Original non-cacheable FIB, after one iteration of hole filling algorithm and final cacheable FIB

1) FIB Inflation Due To Hole Filling: The hole filling algorithm produces a cacheable FIB that is larger than the original. For example, the original FIB in Table Ia has only two entries, while the cacheable FIB shown in Table Ic has nine entries, an increase of 350%. We next investigate the effect of the algorithm on real FIBs.

We measured inflation on a FIB from our regional ISP, which contains next hop information, and on FIBs from RouteViews [23] that do not contain next hop information, which we approximate using the next hop AS. Table II shows at worst inflation is under 9%. Since the cacheable FIB resides in main memory the impact is very small.

Table	Original	cacheable	Increase
Regional ISP	441,778	468,095	5.96%
Hurricane	444,977	470,911	8.51%
Telstra	440,156	466,416	8.43%
Level-3	436,670	462,966	8.41%
AOL	438,719	464,988	8.40%
NTT-A	439,078	465,536	8.38%
ATT	438,265	464,662	8.37%
SAVVIS	438,582	465,045	8.37%
Sprint	438,634	465,079	8.37%
VZWBIZX	436,821	463,220	8.35%
SWISS-COM	169,861	173,636	8.27%
KPNE	439,288	465,790	6.03%
Tiscali	438,993	465,343	6.00%
IIJ	440,196	466,505	5.98%

TABLE II: FIB size increase due to hole filling algorithm. The increase in FIB size due to hole filling is minimal.

V. Data Sets and Trace Statistics

We used nine 12H and 24H packet traces taken at links between our regional ISP and one of its tier-1 providers. We captured traffic using specialized hardware [1] that ensured no packets were dropped during capture and packet timestamps were accurate to a nanosecond resolution. We then used a trie-based longest prefix matching algorithm to determine the matching prefix for each packet. The cacheable FIB was derived from the actual FIB obtained from the router where the packet trace was captured. Table III shows the trace statistics.

No.	Date	Time	Link	No. packets
T1	3/31/09	27H	1G	7,620,972,889
T2	8/17/09	24H	1G	3,821,714,756
T3	8/3/10	24H	1G	2,084,398,007
T4	8/3/10	24H	1G	2,050,990,835
T5	12/14/11	12H	1G	625,547,727
T6	04/13/12	12H	1G	3,729,282,487
T7	2/14/13	12H	10G	22,622,946,036
T8	3/20/13	12H	10G	21,998,786,996
T9	2/21/14	12H	10G	18,435,172,172

TABLE III: Trace statistics. We use trace T8 in our caching performance analysis and trace T9 in our cache robustness analysis.

In an interesting twist, there was a sustained NTP reflection attack in trace T9. In this attack, several comprised machines belonging to our ISP's customers were used as amplifiers to attack other hosts on the Internet using a NTPD vulnerability [2]. Figure 2 shows the overall average bit rate per hour as well as the average NTP bit rate per hour. NTP traffic accounted for 1.3% of all traffic during the duration of the trace. In comparison, trace T8 has approximately the same byte rates as trace T9. However, NTP accounted for only 0.1% of all traffic in trace T8.

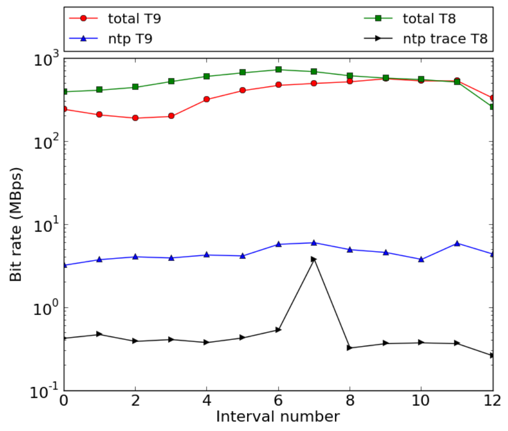


Fig. 2: Comparison of average overall and NTP bit rates per hour between "normal" and attack traffic traces. Trace T9 was captured during an NTP attack.

For brevity, we show statistics from trace T8 only. Results from the other traces (except T9) are similar. We use trace T9 in our cache robustness analysis in section VIII-B.

VI. Results

In this section, we present results using FIB caching emulation, using real traces to drive the emulator.

A. *Cache System Performance*

We begin by evaluating cache performance using a standard cache replacement policy - least recently used (LRU). Note that we have repeated all the experiments in this paper with the least frequently used (LFU) cache replacement strategy, but do not show the results here due to space constraints. In general, we observed that LFU performs worse than LRU. Qualitatively, our results confirm previous observations, but it is important to show them to establish a baseline. Figure 3 shows the performance of an LRU cache with varying cache sizes for trace T8 (Table III). We plot average cache hit rates at 5-minute intervals.

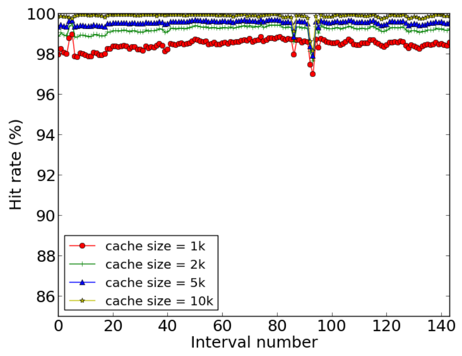


Fig. 3: LRU Hit Rates for cache sizes varying from 1k to 10k

Figure 3 shows that LRU consistently achieves very high hit rates. With a cache size of only 1K entries the hit rate is 98%. The hit rate increases with the cache size, reaching almost 99.9% with a cache of 10K entries. LRU achieves maximum hit rates of 99.26%, 99.74%, 99.91% and 99.96% with cache sizes of 1K, 2.5K, 5K and 10K respectively. Note that the hit rate penalty is very small, even with a cold cache. For the rest of the paper, unless otherwise noted, we will use a cache size of 10K. The reason is that even with 10K entries, this is a still a very small cache compared with the original FIB size (currently around 500K).

B. *Impact of Route Updates*

Routers periodically receive route updates from their peers. These updates may add new prefixes (announcements) to the table or withdraw existing prefixes from the table (withdrawals). Each such update may cause one or more entries in the cache to be invalidated.

To evaluate the effect of route updates we took a full RIB table from a RouteViews [23] peer and generated a cacheable FIB. We again approximate next hop information using the next hop AS from the ASPATH attribute. Then, we applied updates for the same peer to the cacheable FIB and generated a new cacheable FIB. Finally, we measured the difference between the original cacheable FIB and the newly generated cacheable FIB.

The size of the cacheable FIB generated from the original FIB increased from 458,494 to 464,512, a growth of only 1.29%. 82.5% of the prefixes in the cacheable FIB were the same as those in the original FIB. After subsequent updates were applied, the size increase as well the number of prefixes that changed was negligible - the average change in the size of the cacheable FIB was only 0.0018%.

Next, we count the number of prefixes that had next hop changes, since only these prefixes will have to be invalidated if present in the cache. Our results show that, on average, only 144 prefixes in each 15 minute set of updates had a next hop change. The insignificant change in the size of the cacheable FIB after applying updates, coupled with the fact that only a few hundred prefixes will have to be invalidated from the cache due to change in forwarding state, suggest that routing updates will have very little impact on the cache state and forwarding performance.

It should be noted that while this study is limited in scope to tables and updates from only one RouteViews peer, we believe that a more comprehensive study will yield similar results. Research shows that routing tables from different RouteViews peers have very few differences [28] and hence we believe that our results are applicable to other peers as well.

C. *Trends in Cache Size*

The global routing table has been growing steadily for the past several years. Measurements show that the routing table grew from 150,000 entries in 2003 to 517,802 entries at the time of this writing, representing an increase of 245% in the last decade alone [12]. Routers thus need increasing amounts of memory and processing power [13]. The conventional wisdom that FIB memory will scale at rates surpassing the rate of growth in the routing information handled by core router hardware is not true for the low-volume, customized hardware used in core routing [22].

Given the fast rate of increase of the global routing table over time, the natural question to ask is does a FIB cache face a similar rate of growth? To get some insight into the answer we measure the hit rates achieved with a cache size of 10K entries on packet traces gathered from 2009 through 2014. These traces, except trace T9, were collected at the same point in our ISP, so they truly capture the cache evolution over the last four years. Trace T9 was captured at another monitoring location in our ISP, but results show that the hit rates for trace T9 are similar to the hit rates for the other traces.

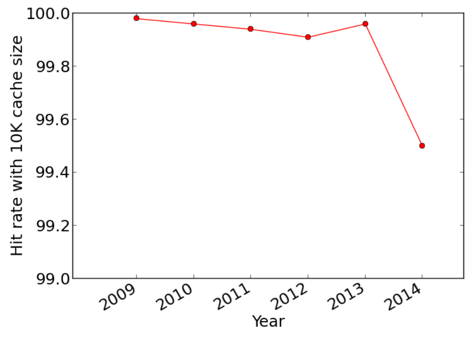


Fig. 4: Hit rates achieved by a 10k entry cache remain almost constant from 2009-2014. The 2014 trace contains the NTP attack traffic.

Figure 4 shows the results. While there is some minor variation, the hit rates achieved with a cache size of 10K are consistently higher than 99.95%, meaning that over the past four-year period the cache size did not need to change to maintain the same cache hit rate. Thus, while the global routing table has certainly changed, traffic locality has not changed sufficiently to affect the cache size.

VII. Analysis of Cache Misses

In our FIB caching system when a packet arrives and the appropriate prefix is not in the cache, the packet is queued while a cache update takes place. This introduces some delay before the packet is forwarded. Delay may affect packets in different ways. For example, delaying video packets may result in dropped frames and choppy video. On the other hand, queuing DNS packets may result in only a slightly longer resolution delay. In this section we characterize the type of packets queued due to misses and the delay they experience.

First, we classify the types of packets that cause cache misses to determine which applications are affected. Second, we determine buffer requirements and queuing delays due to cache misses. Finally, we analyze the impact of cache misses on memory bandwidth requirements in routers. We show results only for trace T8 from Table III. Results for other traces are similar and have been omitted due to space constraints. Since trace T9 was captured during an ongoing NTP reflection attack, the cache misses in this trace were heavily influenced by the attack traffic. We present the analysis for trace T9 later in section VIII-B.

A. *Classification of Cache Misses*

We classify packets causing cache misses as follows. First, we separate TCP and UDP packets. For TCP, we classified them as SYNs, SYNACKs, FINs, FINACKs, RSTs and DATA packets. Any TCP packet that was not a SYN, SYNACK, FIN, FINACK or RST is classified as a DATA packet. Preliminary analysis of UDP packets shows that NTP and DNS packets suffered most misses. We therefore plot NTP and DNS packets separately, with the remaining packets classified as "UDP Other". Figure 5 shows the classification of packets causing cache misses with an LRU cache.

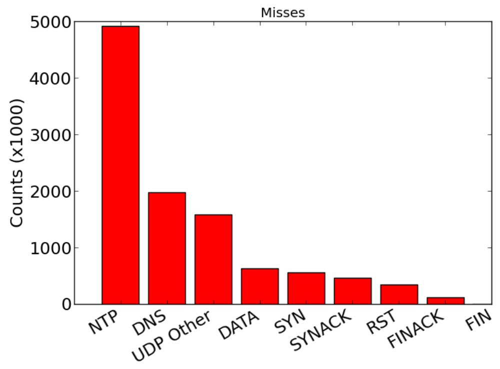


Fig. 5: Classification of packets causing cache misses

We also see that the largest number of cache misses is caused by NTP packets. The reason is that while NTP requests are regular, they are also infrequent. Thus, the prefixes required to forward the NTP packets are regularly evicted from the cache, only to be re-inserted when the next NTP request occurs. In our dataset, NTP packets were destined to 69,097 unique prefixes.

DNS packets are the second largest cause of cache misses. This occurs due to a DNS query which needs to be forwarded to the authoritative nameserver of the zone that owns the prefix. If the prefix is not in the cache, the DNS query will result in a cache miss. In our dataset, DNS packets hit 58,435 unique prefixes.

B. *Effects of Cache Misses on Queuing*

There are several ways of dealing with packets causing cache misses. One possible strategy is to forward the packets along a slower path until the appropriate route can be inserted into the FIB/cache [4]. If this strategy is employed, then there is no need for queues/buffers at the routers to store packets. However, the packets causing a cache miss have to travel a longer path, thus experiencing longer delay and possible reordering. Another strategy is to queue the packets at the router until the route is inserted into the cache. While the packets do not incur any longer paths (and hence stretch), line cards must now have enough buffer space and the appropriate logic to queue packets. In our analysis we assume the latter strategy for several reasons: (a) it prevents packet reordering, which router vendors try hard to avoid, (b) to the best of our knowledge it has not been evaluated before, and (c) as our results show, the buffer requirements are modest and the queues can easily fit in existing buffers.

1) Packet Queuing Emulator: We built an emulator in order to measure queuing at the router during cache misses. Figure 6 shows the block diagram of our emulator. We assume that it takes about 60ns to update the cache (the lookup time for DRAM) and that a packet needs to be queued while the update is taking place.

When a packet arrives we do a longest prefix match against the cache. If there is a match, the packet is forwarded and does not incur any additional delays. If there is no match, against the cache the packet is queued. When it gets to the head of the queue, we do a prefix fetch from the cacheable FIB, update the cache and forward the packet.

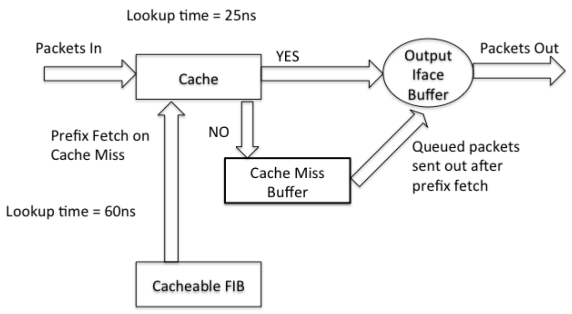


Fig. 6: Queue Utilization Simulator

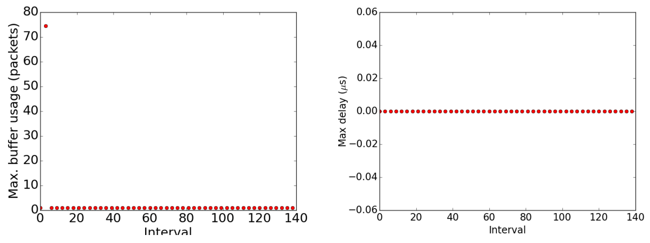
We measure the maximum queue utilization during a given 5 minute interval. We also look at the maximum queuing delay in our emulator i.e. the time elapsed between when a packet entered the queue and when it exited the queue. Let us assume that the time required to perform a cache lookup is T_C, the time required to perform a slow-memory lookup is T_S and the current buffer size is B. If a packet arrives at the queue at time T_A the time the packet departs the queue T_D is given by:

$$T_D = T_A + T_C + (B * T_S)$$

The queuing delay D is then

$$D = T_D - T_A$$

2) Evaluation: Figure 7a shows the maximum queue utilization during 48 5-minute intervals. In the first interval we see a startup effect due to the cold cache. Since there are no entries in the cache, arriving packets cause many cache misses and have to be queued, resulting in a maximum queue size of 73 packets. After this initial startup period the queue drains and queue utilization drops. The maximum queue utilization is only 1 packet, which means that assuming 1500 byte packets, we need only 1500 bytes of buffer space to buffer LRU cache misses. Even when the queue utilization peaks during the cache warm-up period, only 73 packets are queued. To store these packets, we will need only 110KB of buffer space.



(a) Max. Queue Utilization (b) Max. Queueing Delay

Fig. 7: Max. Queue Utilization and Queuing Delays

Figure 7b shows the maximum queuing delays for a LRU cache during the same intervals as above. Packets queued after a LRU cache miss suffer virtually no queuing delay. This is due to the small average queue size (1 packet per 5 minute interval) and the relatively fast cache updates (in the order of ns) which keep the backlog small.

While these numbers are specific to our dataset, we believe that they generalize well in similar environments. Moreover, the buffer requirements are in line with what is found in routers today, where the rule of thumb is that buffers should hold an RTT's worth of data.

C. *Memory Bandwidth Requirements*

Another important benefit of FIB caching is a reduction in the number of lookups that need to be performed on the full FIB, which is often kept in slow (DRAM) memory. Without caching one lookup typically needs to be performed per packet. Moreover, the next hop information is often only part of the lookup information, which may also include filtering, QoS, and other state, increasing the demand on memory bandwidth. With increasing FIB sizes and line speeds these lookups are nearing the bandwidth limit of router memory.

Caching has the potential to drastically reduce the number of lookups to external memory. For example, one may envision a system where the cache is kept on-chip and the full RIB remains on external slow DRAM. The latter needs to be accessed only when a cache miss occurs. The question then is, what are the savings in terms of memory bandwidth if one uses an on-chip cache?

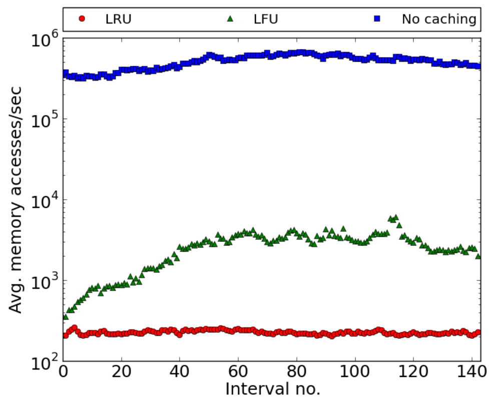


Fig. 8: Memory bandwidth requirements reduce by orders of magnitude when caching is employed.

Figure 8 shows the average number of memory lookups required per second in each 5 minute interval in the trace. Without caching, the number of memory lookups is equal to the packet rate, since each packet requires a memory lookup. With caching, a memory lookup is required only in case of a cache miss. We see that the number of memory accesses required with caching is several orders of magnitude lower than the number of accesses required when no caching is deployed. If the cache uses a LRU replacement policy the memory accesses are on the order of 10^2 accesses per second and the rate stays almost constant. This is to be expected, since we see that the hit rates achieved with LRU stay constant throughout the duration of the trace.

Caching offers a order of magnitude improvement in memory bandwidth when compared to no caching. Coupled with the fact that the required cache can easily fit on a chip and that the cache size appears to remain constant over time, caching virtually eliminates any memory bandwidth issues for current and potentially future routing table sizes. This is significant for scaling future routers.

VIII. Cache Robustness

The use of FIB caching, especially with LRU, exposes routers to a cache pollution attack, where an attacker attempts to disrupt packet caching and increase the cache miss rate. An attacker can pollute the cache by continuously sending packets to numerous unpopular prefixes that would not be otherwise cached, in an attempt to evict legitimate prefixes. Depending on the attack rate, the attacker can either cause the cache to thrash, thus increasing the miss rate, or reduce the effectiveness of the cache by ensuring that at least part of it is always polluted with bogus entries. This attack will adversely affect the packet forwarding performance as well, by substantially increasing the number of packets that need to be queued while the new prefix is fetched, potentially leading to packet loss.

In this section we investigate cache pollution attacks and their feasibility. We describe a generalized threat model where the attacker tries to replace a subset of cache entries, and then estimate the attack rate required to adversely affect the cache. Next, we evaluate cache performance when subjected to a real NTP attack that happened to be present in trace T9.

A. *Generalized Threat Model*

In this section we describe a generalized threat model to attack the cache, and determine the rate at which an attacker must send packets to disrupt packet delivery. We assume that the attacker knows or can determine in advance the set of popular and unpopular prefixes at a given vantage point. We also assume that the attacker has the capability to send packets at high rate, up to the capacity of the link, either from a single host or a botnet. We also assume that the goal of the attacker is to replace legitimate cache entries with unpopular entries for the sole purpose of disrupting cache performance. In other words, we assume a deliberate attack on the cache and not a general attack on the infrastructure (e.g., DDoS). The latter will trigger other network defenses.

To achieve the above goal, the attacker still needs to send packets at a high enough rate to evict legitimate prefixes quickly from the cache. Thus, the attack cannot be stealthy since its packet rate must compete head-to-head with legitimate traffic. Next, we estimate the packet rate that would make such attack viable.

1) Intensity of an Effective Attack: To determine the rate at which the attacker must send packets to affect cache performance we first rank prefixes according with their popularity during a low- and high-traffic interval. Recall that our measurement interval is 5 minutes. We chose to investigate the per-interval behavior rather than average traffic over the entire 24H period in order to determine both high and low required attack rates.

We estimate the required attack intensity. If an attacker wants to hit the i^{th} entry in the cache (and all the prefixes below it) it must send packets to enough prefixes to evict the i^{th} entry and all the other entries below it. So for example, if the attacker wants to blow the entire cache, then the attacker must attack at P_{attack} rate, which must be greater than the cache size N multiplied by P_{top} which is the packet rate of the most popular prefix. To generalize, the attack rate to evict the i bottom prefixes from the cache must be:

$$P_{attack} >= P_i * i$$

In the low traffic interval, the most popular prefix received 33,049,971 packets in five minutes for an average packet rate of 110,166 packets per second, whereas in the high traffic interval the most popular prefix received 37,079,737 packets for an average of 123,599 packets per second. Thus, to replace just the most popular prefix from the cache, the attacker needs to send packets at a rate between 1,101,660,000 packets/sec and 1,235,990,000 packets/sec to an idle prefix. For a 10Gb/s link and 64 byte packets the maximum packet rate possible is 19,531,250 packets/sec, thus the required attack rate is not feasible as it far exceeds the capacity of the link.

Note that such an attack can be easily detected by looking for spikes in the cache miss rate. Once detected, one can imagine several defenses against this type of attack, such as pinning down at least part the historical prefix working set until the attack subsides. This poses little danger of false positives, since when prefixes suddenly become popular (as in the case of a flash crowd), they are unlikely to do it in numbers in the order of the cache size. Thus, we believe that practical attacks of this kind can only affect part of the cache and are easily weakened by reasonably over-engineering the cache.

B. Cache Performance Under a Real DDoS Attack

As described in section V, trace T9 was captured during an ongoing NTP reflection attack. Figure 2 shows that overall, NTP traffic accounted for approximately 1.3% of all traffic in the trace. 5659 unique addresses from our regional ISP were the target of this attack [5]. Even though this was not an attack on the cache, we take advantage of this unfortunate incident to investigate the performance of a FIB cache under attack conditions and compare performance with a version of the same trace that has NTP traffic removed.

Figure 9 shows the hit rates achieved by a 10k entry LRU cache for all traffic in trace T9 as well as the hit rates for all non-NTP traffic.

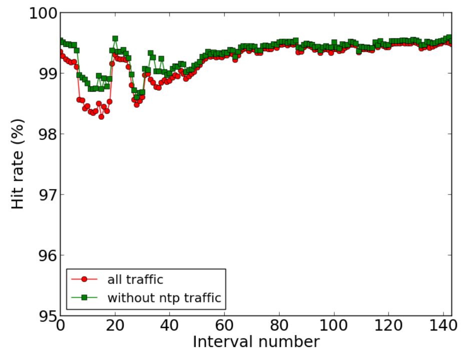
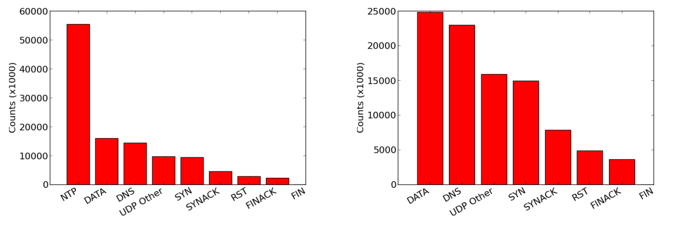


Fig. 9: Comparison of LRU hit rates for all traffic and non-NTP traffic. The cache size was set to 10k entries.

As Figure 9 shows, the cache performs well under this incident. The difference in the hit rates achieved by the cache do not differ by more than 0.5% during the trace. Thus, caching remains robust. However, comparing Figures 3 and 9, it is clear that the hit rates for trace T9 are lower than of trace T8, although the difference is less than 1%.

Next, we look at the breakdown of packets resulting in cache misses as shown in Figure 10a.

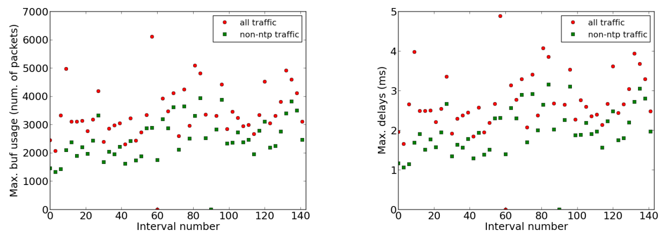


(a) Breakdown of misses (b) Breakdown of non-NTP misses

Fig. 10: Comparison of cache misses for all traffic and non-NTP traffic. The cache size was set to 10k entries.

As expected, the NTP traffic accounts for a majority of the packet misses. Out of a total of $114 * 10^6$ cache misses observed during the trace, NTP packets caused $55*10^6$ misses or 48%. The next highest number of misses were caused by TCP DATA packets, with $25*10^6$ misses (22%). NTP therefore caused 2.2 times more misses than the TCP DATA packets. Figure 10b shows a zoomed-in version of Figure 10a, with the NTP traffic filtered out of the trace. We see that TCP DATA and DNS packets account for the bulk of the misses, similar to what we see in Figure 5.

To further quantify the performance of the cache during the NTP incident, we measured the maximum buffer usage and queuing delays incurred by packets during the ongoing NTP attacks. We compare these with the buffer usage and queuing delays with the NTP traffic filtered out. Figures 11a and 11b show the results.



(a) Buffer usage (b) Buffer delays

Fig. 11: Comparison of buffer usage and delays for all traffic and all non-NTP traffic. The cache size was set to 10k entries.

As Figures 11a and 11b show, the buffer usage and queuing delays with the attack traffic are not drastically different to those with the attack traffic filtered out. The maximum buffer usage with the attack traffic included is 6104 packets, compared to 3944 packets without the attack traffic. The corresponding maximum queuing delays are 4.8ms and 3.2ms respectively. Thus we conclude that the NTP incident did not put undue strain on the caching system.

The difference in hit rates observed for traces T8 and T9 also reflects in the buffer usage and buffering delays, as seen from Figures 7 and 11. Trace T9 and T8 were captured on different links and trace T9 had more diversity in terms of the number of prefixes carrying packets than T8. In trace T8, the average number of unique prefixes carrying packets in each 5 minute interval we investigated was 15,310, compared to 24,965 prefixes for trace T9. In future work, we plan to investigate traces T8 and T9 further to quantify the differences observed in the results.

IX. Conclusions

In this paper we take a look at some practical considerations in the implementation of FIB caching. We extend previous work in significant ways by looking at practical issues, such as characterization of cache misses, queuing issues (buffer occupancy and delay), memory bandwidth requirements and robustness against cache pollution attacks. We used traces collected at links from our regional ISP to a Tier-1 ISP for our analysis. Our design uses a cacheable FIB to avoid the cache hiding problem, and we presented an algorithm to convert a regular FIB to a cacheable FIB.

Our work has some limitations. First, we only look at packet traces from a single regional ISP. We therefore cannot evaluate cache performance at core routers, where traffic may be more diverse causing hit rates to drop. While we do not have access to data from core routers to answer this question (we need a packet trace and a simultaneous snapshot of the FIB at a core router), the tools and methodology we developed are applicable to a core environment and we plan to repeat the study once we have an appropriate dataset.

Are there trends that may invalidate the benefits of FIB caching? On the contrary, recent trends such as traffic concentration to major social networks, search engines that reside in datacenters [15] and CDNs make caching even more likely to provide benefits. In these environments, traffic becomes more focused and likely to hit a smaller set of prefixes, resulting into a more stable working set.

Other potential benefits of employing FIB caching include manufacturing of cheaper router hardware and routers with longer service cycles. Future terabit/sec forwarding chips may employ FIB caching for less on-chip memory or to allow the non-cached portion of the FIB to be more aggressively compressed. Finally, current architectures with the entire FIB on DRAM may also benefit by backing away from the looming memory bandwidth wall.

REFERENCES

[1] Endace. http://www.endace.com.

[2] Ntp reflection attack - internet security—sans isc. SANS ISC. https://isc.sans.edu/forums/diary/NTP+reflection+attack/17300.

[3] How to choose the best router switching path for your network. http://www.cisco.com/warp/public/105/20.pdf, August 2005.

[4] H. Ballani, P. Francis, T. Cao, and J. Wang. Making routers last longer with viaggre. In *Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, NSDI'09, pages 453–466, Berkeley, CA, USA, 2009. USENIX Association.

[5] J. Czyz, M. Kallitsis, M. Gharaibeh, C. Papadopoulos, M. Bailey, and M. Karir. Taming the 800 pound gorilla: The rise and decline of ntp ddos attacks. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, IMC '14, pages 435–448, New York, NY, USA, 2014. ACM.

[6] R. Draves, C. King, S. Venkatachary, and B. Zill. Constructing optimal ip routing tables. In *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, pages 88 –97 vol.1, mar. 1999.

[7] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis. Locator/ID Separation Protocol (LISP). draft-ietf-lisp-08.txt, 2010.

[8] D. Feldmeier. Improving gateway performance with a routing-table cache. In *INFOCOM '88. Networks: Evolution or Revolution, Proceedings. Seventh Annual Joint Conference of the IEEE Computer and Communcations Societies, IEEE*, pages 298–307, 1988.

[9] K. Gadkari, D. Massey, and C. Papadopoulos. Dynamics of prefix usage at an edge router. In *PAM '11: Proceedings of the 12th International Conference on Passive and Active Network Measurement*, pages 11–20, 2011.

[10] Y. Gao, L. Deng, A. Kuzmanovic, and Y. Chen. Internet cache pollution attacks and countermeasures. In *Proceedings of the Proceedings of the 2006 IEEE International Conference on Network Protocols*, ICNP '06, pages 54–64, Washington, DC, USA, 2006. IEEE Computer Society.

[11] W. Herrin. Opportunistic topological aggregation in the rib-fib calculation? http://www.ops.ietf.org/lists/rrg/2008/msg01880.html, 2008.

[12] G. Huston. Bgp analysis reports. http://bgp.potaroo.net/index-bgp.html.

[13] G. Huston and G. Armitage. Projecting future ipv4 router requirements from trends in dynamic bgp behaviour. In *Australian Telecommunication Networks and Applications Conference (ATNAC)*, 2006.

[14] C. Kim, M. Caesar, A. Gerber, and J. Rexford. Revisiting route caching: The world should be flat. In *PAM '09: Proceedings of the 10th International Conference on Passive and Active Network Measurement*, 2009.

[15] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian. Internet inter-domain traffic. In *Proceedings of the ACM SIGCOMM 2010 conference*, 2010.

[16] H. Liu. Routing prefix caching in network processor design. In *Computer Communications and Networks, 2001. Proceedings. Tenth International Conference on*, pages 18 –23, 2001.

[17] Y. Liu, S. O. Amin, and L. Wang. Efficient fib caching using minimal non-overlapping prefixes. *SIGCOMM Comput. Commun. Rev.*, 43(1):14–21, Jan. 2012.

[18] Y. Liu, L. Wang, and B. Zhang. Fifa: Fast incremental fib aggregations. In *INFOCOM 2013: In Proceedings of the 32nd IEEE International Conference on Computer Communications*, April 2013.

[19] Y. Liu, X. Zhao, K. Nam, L. Wang, and B. Zhang. Incremental forwarding table aggregation. In *GlobeCom 2010*, Dec. 2010.

[20] V. Manivel, M. Ahamad, and H. Venkateswaran. Attack resistant cache replacement for survivable services. In *Proceedings of the 2003 ACM Workshop on Survivable and Self-regenerative Systems: In Association with 10th ACM Conference on Computer and Communications Security*, SSRS '03, pages 64–71, New York, NY, USA, 2003. ACM.

[21] D. Massey, L. Wang, B. Zhang, and L. Zhang. Proposal for scalable internet routing and addressing. draft-wang-ietf-efit-00.txt, 2007.

[22] D. Meyer, L. Zhang, and K. Fall. Report from the IAB workshop on routing and addressing, 2007.

[23] U. of Oregon. The routeviews project. Advanced Network Topology Center and University of Oregon. http://www.routeviews.org/.

[24] G. Rétvári, J. Tapolcai, A. Kőrösi, A. Majdán, and Z. Heszberger. Compressing ip forwarding tables: Towards entropy bounds and beyond. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 111–122, New York, NY, USA, 2013. ACM.

[25] S. Richardson. Vertical aggregation: A strategy for fib reduction, 1996.

[26] N. Sarrar, T. U. B. T-labs, S. Uhlig, and R. Sherwood. Leveraging Zipf s Law for Traffic Offloading. 42(1):17–22, 2012.

[27] Z. A. Uzmi, M. Nebel, A. Tariq, S. Jawad, R. Chen, A. Shaikh, J. Wang, and P. Francis. Smalta: practical and near-optimal fib aggregation. In *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies*, CoNEXT '11, pages 29:1–29:12, 2011.

[28] H. Yan, B. Say, B. Sheridan, D. Oko, C. Papadopoulos, D. Pei, and D. Massey. Ip reachability differences: Myths and realities. In *Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on*, pages 834 –839, April 2011.

[29] M. Zadnik and M. Canini. Evolution of cache replacement policies to track heavy-hitter flows. In *PAM '11: Proceedings of the 12th International Conference on Passive and Active Network Measurement*, pages 21–30, 2011.

[30] X. Zhang, P. Francis, J. Wang, and K. Yoshida. Scaling ip routing with the core router-integrated overlay. In *ICNP '06: Proceedings of the Proceedings of the 2006 IEEE International Conference on Network Protocols*, pages 147–156, 2006.

[31] X. Zhao, Y. Liu, L. Wang, and B. Zhang. On the aggregatability of router forwarding tables. In *Proceedings of the IEEE INFOCOM 2010*