Towards Zero-Time Wakeup of Line Cards in Power-Aware Routers

Tian Pan*, Ting Zhang*, Junxiao Shi†, Yang Li*, Linxiao Jin*,
Fuliang Li*, Jiahai Yang*, Beichuan Zhang† and Bin Liu*
*Tsinghua National Laboratory for Information Science and Technology
*Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China
†Computer Science Department, The University of Arizona, Tucson, AZ 85721, USA

***Abstract*—As the network infrastructure has been consuming more and more power, various schemes have been proposed to improve power efficiency of network devices. Many schemes put links to sleep when idle and wake them up when needed. A presumption in these schemes, though, is that router's line cards can be waken up quickly. However, through systematic measurement of a major vender's high-end router, we find that it takes minutes to get a line card ready under the current implementation. To address this issue, we propose a new line card design that (1) keeps the host processor in a line card always up, which only consumes a small fraction of power, and (2) downloads a slim slot of popular prefixes with higher priority, so that the line card will be ready for forwarding most of the traffic much earlier. We design algorithms that ensure fast and correct longest prefix match lookup during prioritized routing prefix download. Experiments on real hardware show that the wakeup time can be reduced to 127.27ms, which is 0.3% of the original line card wakeup time, well supporting many power-saving schemes.**

I. Introduction

Power consumption of the network infrastructure has been increasing fast, fueled by the deployment of more powerful network devices to serve exponentially increasing data traffic. A few studies [1], [2] have shown that the Internet consumes 37% of the total power of the Information and Communication Technologies (ICT) industry, where switches/routers dominate the power expense of the Internet. A number of schemes have been proposed to improve network's power efficiency. For example, GreenTE [3], an intra-domain traffic engineering solution, saves power by using fewer links to carry traffic and putting idle links to sleep. Another example is Buffer-and-burst [4], in which the upstream router deliberately buffers traffic for a little while so that the downstream router will have a little more idle time between the traffic bursts, and can put links to sleep during the idle periods. One critical assumption in these schemes and the similar ones is that the line card can be quickly waken up and ready for packet forwarding when needed. This, however, is largely based on the experience with the host NIC and the desire to simplify protocol design, and does not have the support of empirical data.

This work is supported by 863 project (2013AA013502), NSFC (61073171, 61202489), the Specialized Research Fund for the Doctoral Program of Higher Education of China (20100002110051), Tsinghua University Initiative Scientific Research Program (20121080068), NSF grant CNS-3017600.
Corresponding Author: Bin Liu (lmyujie@gmail.com).
978-1-4799-3360-0/14/$31.00 2014 IEEE

In this work, we conduct systematic measurement of a high-end commercial router to find out its line card wakeup time. The result shows that it takes about five minutes for a line card with 200K resident routes to be ready for packet forwarding. As the size of the routing table increases, the wakeup time will increase too. This will greatly limit the applicability of the power-efficient network proposals which are based on the sleep/wakeup mechanism.

To achieve the goal of zero-time line card wakeup, i.e., making a sleeping line card ready before traffic arrives, we need to understand where it spends time during the boot process. Our analysis of the line card boot log reveals the following insights. First, it takes a considerable amount of time to boot up the on-board CPU (the host processor) and download/initialize the system software, but only a small fraction of power to keep them running [5]. Second, downloading the routing table is a time-consuming process and it takes longer to download larger tables. However, not every entry in the table has the equal access opportunity by the incoming packets; a small percentage of routes are responsible for most of the traffic.

Based on these observations, we propose a new line card design with two major changes. First, a separate power supply is dedicated to the on-board CPU and related circuits to keep them running during line card sleep. This doesn't cost much power but saves quite a lot of boot time. Second, routing prefixes are prioritized during routing table download so that popular prefixes will be downloaded earlier. This will make the line card ready to forward most of the traffic earlier. There are, however, some technical issues in prioritized routing prefix download: we need to guarantee the correctness of longest prefix match when an incomplete routing table is in the line card, and we should be able to handle the resource contention when incoming traffic and high-frequency route updates are competing simultaneously for the routing table. To address these issues, we design algorithms to guarantee the correct prefix download sequence, and a cache-based architecture to address the resource contention issue during line card wakeup.

On a hardware prototype with FPGA-based network processor and traffic manager, we implement the new design and conduct experiments. The new line card can wake up from sleep mode in 127.27ms, only 0.3% of the time the original design takes. This drastic improvement makes many power-efficient network proposals supportable.

The rest of this paper is organized as follows. The measurement on line cards of a commercial router is elaborated in §II. Then we discuss the insights and opportunities learned from the measurement and propose the corresponding designs in §III. §IV conducts prioritized prefix download while §V discusses the hardware-related issues during routing table download. §VI gives the evaluation. After reviewing related work in §VII, we conclude the paper in §VIII.

II. Measurement

A. High-End Router Architecture

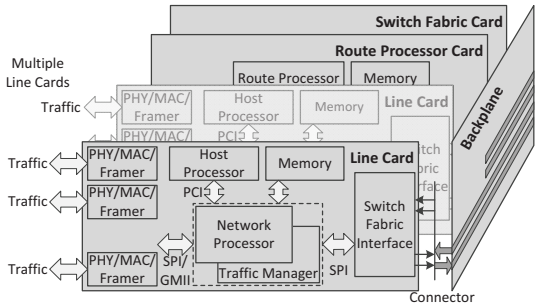


Fig. 1. Conventional High-End Router Architecture

First, we briefly introduce the conventional architecture of high-end routers. It consists of a number of cards plugged into the *backplane*, which provides the physical connectivity. Among these cards, the *route processor card* implements the control plane; it runs routing protocols and maintains the routing table. There're usually multiple *line cards* for distributed packet processing. Each line card contains several functional components, including multiple *ports*, which connect to external cables, a *host processor*, which downloads the routing table from the route processor and manages local hardware, a *network processor*, which integrates programmable packet processors and task-specific co-processors to facilitate diverse network applications, a *traffic manager*, which performs packet queuing, shaping and scheduling. The *switch fabric* is a separate card that provides high-speed interconnects among all the line cards, such that packets can go from any ingress line card to any egress line card at line speed.

B. Devices and Equipment in Measurement

The measurement is taken on Huawei's NE40E-X8 router, which is sold since 2010 and represents the best technology at that time. The measurement is based on router's minimal working configuration which contains two route processor cards (1:1 backup), two LPUF-21-A line cards and one switch fabric card. The fan speed is set to 60% manually. Each line card can handle 20Gbps traffic and has two sub-slots plugged with two 10Gbps fiber interface cards. Two traffic generators are used, each is capable of sending and receiving 10Gbps traffic. They can also act as BGP speakers to inject BGP routes into the router. The host processor runs Huawei's VRP (versatile routing platform) on VxWorks OS. The debug output from the serial console is recorded to break down the line card wakeup process. The power of the router is measured via Yokogawa digital meter on router's -48V DC power supply.

C. Line Card Wakeup Sequence

A fine-grained timeframe is interpreted from the log, providing a deep insight into what has happened during the line card wakeup (Table I). It takes 292s for the line card to be fully recovered when 200K routes are installed. The wakeup process can be roughly partitioned into 3 phases, including software preparation on the host processor (0-60s), hardware configuration in the data plane (60-134s) and "interface card initialization" (134-292s). In the 1st phase, system software is downloaded from the route processor and starts running on the host processor. In the 2nd phase, hardware integrity is verified, microcode programs are loaded into network processor and traffic manager is configured. In the 3rd phase, it is *observed* that the interface cards are initialized with a confusingly long time. When the downloaded routing table size varies, the time spent on the 1st phase and 2nd phase is almost constant, but the time for detecting the interface cards varies and we find it grows linearly with the number of installed routes. However, after the line card is fully recovered, we unplug and plug back an interface card and find its recovery time is 15s regardless of the routing table size. Hence it is evident that routing table download actually saturates the host processor and delays interface card initialization. Figure 2 shows how the line card wakeup time grows with the number of installed routes.

TABLE I
LPUF-21-A Line Card Wakeup Events (200K Routes Installed)

Time (s)	Events During LPUF-21-A Line Card Wakeup
0	"power on" command is confirmed by operator
7	board is powered on
12	firmware detects the board
24	boot loader starts
34	system software version is compared with the route processor
40	checksums of software files are verified
42	system software is downloaded from the route processor
46	VxWorks OS starts on the host processor
60	VRP starts
63	clocks in line card are set, hardware in line card is detected
84	traffic manager's memories are tested
95	network processor is initialized
98	software files for data plane processing are uncompressed
103	network processor's on-chip buffers and buses are tested
116	on-board DDR3 DRAMs are tested
120	switch fabric interface is initialized
130	data plane tasks on network processor are started
132	TCAM allocation is configured
133	line card registers itself with the route processor
134	data channel between switch fabric and line card is opened
277	interface card 0 is detected and tested (200K routes installed)
288	interface card 1 is detected and tested
292	IP line protocol goes up

D. Line Card Power Consumption

We illustrate measurement results on line card power consumption. The router under minimal working configuration consumes 657W when two line cards are idle. The power is increased by 19W when a pair of 10Gbps links is 100% utilized and the increase grows linearly with the link utilization

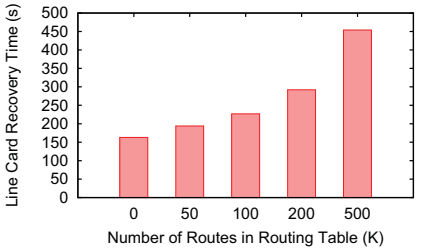


Fig. 2. Line Card Recovery Time vs. Number of Routes in the Downloaded Routing Table

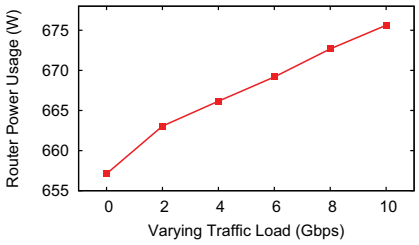


Fig. 3. Router Power Consumption Under Varying Traffic Loads

TABLE II
LINE CARD POWER BREAKDOWN

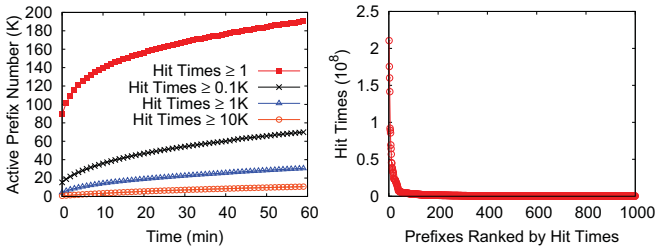
line card component	power
ASICs/NPs	53%
local power supplies	14%
memories	13%
CPU/DRAM	9%
interconnect	7%
others	4%

(Figure 3). The difference of power usage appears to be rather small under varying traffic loads (the idle power is more than 97% of its peak power). This implies commercial routers under current design would be insensitive to the traffic load change to save power adaptively (e.g., via link-rate adaptation). The power is decreased by 1.59W if one OC-192 transceiver on the interface card is unplugged, which can be regarded as the power saving from port shutdown. The power is decreased by 13W if one interface card is unplugged. Though transceivers and interface cards can be recovered quickly (in 1s and 15s, respectively), the power saving benefits from turning them off are quite trivial. The power is reduced by 380W if two line cards are unplugged. The remaining 277W is consumed by the route processor cards, fans, backplane, switch fabric card and power supply. When the router runs at full capacity, the line cards will dominate the total power usage predictably. Due to the equipment limitation, we cannot obtain the component-level power usage of a line card. Instead, we refer to Wobker's report on Cisco's core router [5]. It is measured that the data plane hardware for high-speed packet processing is the dominant power consumer while the host processor merely shares 9% of the total line card power usage (Table II).

E. Power-Law Behaviors in Route Lookup

We conduct a further study on route lookup behaviors since routing table download accounts for the most of the line card wakeup time (143s/292s). We capture traffic from a 20Gbps gateway link on Tsinghua University campus network and the traffic contains non-anonymized[1] IP addresses (it has 3370670140 IPv4 packets and 38768679 unique destination addresses) with a duration of 60 minutes. The routing table is obtained from the same router and contains 340584 routes (among them, 295970 are at the leaf nodes of the binary trie). We extract packet destination addresses from the 60-minute traffic, search them in the routing table and count the hit times of each routing prefix. Figure 4(a) shows that the number of active prefixes (hit times $\geq$ 1) accumulates relatively slow as the time elapses. Moreover, the popular prefixes (e.g., hit times $\geq$ 10K) have a much smaller but more stable number. We rank all the prefixes based on their hit times in descending order in Figure 4(b) (we only plot the first 1000 prefixes out of the 340584 due to the overlong "tail"). The distribution exactly

[1]Nearly all public traces on the Internet are prefix-preserving anonymized for privacy thus cannot be straightforwardly used in this measurement.



(a) Cumulative Active Prefix Number (b) Distribution of Prefix Hit Times

Fig. 4. Power-Law Behaviors in Routing Prefix Lookup

follows the power law that to the right is the long tail while to the left are the few prefixes that dominate. In fact, popular prefixes with hit times more than 100K only occupy 0.6% of the total prefixes, but dominate 85.2% of the traffic.

III. DESIGN PRINCIPLES

Keep host processor standby. The 1st phase is mainly about booting up the host processor and getting its OS running (see Table I). Meanwhile, the host processor together with its DRAM consumes only 9% of the line card power according to Wobker's report (Table II). Therefore, if we add a separate power supply to the host processor and keep it running during line card sleep, we can cut down 60s of the wakeup time with a small amount of power expense. If the host processor is kept running, it can communicate with the route processor to synchronize its routing table even when the other parts of the line card are turned off. When the line card wakes up, the host processor can directly install the local copy of the routing table into the search engine in data plane (e.g., TCAM) instead of having to download the table from the remote route processor. This should also save remarkable wakeup time.

Fast data plane recovery. In the 2nd phase, memory diagnosis takes a significant amount of time because from a cold start, the system needs to verify the memory behaviors correctly. However, if the line card wakes up from a sleep mode instead of a cold start, it is probably fine to skip memory diagnosis. This would require the line card to remember that it is booting from the sleep mode. It is easy to implement since the host processor is up during the sleep. This phase also spends lots of time on initializing various tasks. However, not every task is crucial or needs to be completely started before packet forwarding begins. E.g., per-flow QoS guarantee can be provided at a later time, thus we can reduce the amount

of memory to be initialized in traffic manager to get shorter wakeup time. Packets can be allowed into the line card once a minimized set of essential tasks are done, even though there may be some other initialization tasks still going on.

Speed up popular prefix download. The 3rd phase is downloading the routing table into the search engine, such as TCAM, identified as the most time-consuming task during the line card recovery (§II-C). Luckily, not every routing prefix has equal access probability by the incoming packets; actually, a slim slot of the routing table will cover the majority of the destination addresses in traffic (§II-E). In addition, according to the previous works [6], [7], the small number of popular prefixes responsible for the bulk of backbone traffic are remarkably stable. Therefore, we could download these "heavy hitters" with higher priority and allow traffic to go into the line card once these "heavy hitters" are installed in the data plane, which will reduce the "effective" prefix download time. The "heavy hitters" can be identified and counted via existing approaches [8], [9], and in this paper we assume the prefix popularity can be readily obtained. Since the routing table is partially downloaded before traffic arrives, incoming packets will have a small probability to miss the route lookup. Given the small amount, these packets can be delivered to either the default route (0.0.0.0/0) or the host processor (who forwards packets as a slow-path) where a complete copy of the routing table resides. As the prefix download sequence is reordered, a pending problem is how to decide the optimal prefix download sequence that will greedily exploit the popularity while guaranteeing the correctness (e.g., longest prefix match should be at least satisfied). We'll discuss this problem in §IV.

IV. Prioritized Routing Table Download

A. The Problem

Given prefix hit rate can be well predicted by its historical popularity, popularity-based prefix download appears to be a greedy yet effective strategy. However, straightforwardly doing this will potentially cause longest prefix match (LPM) violation and we'll show this in the following example. For a routing table in Figure 5, according to the popularity, the prefix download sequence should be "d b f g c a e". Suppose at a certain moment, prefix d has been downloaded into the search engine while the other prefixes are still pended on the host processor. Right at this moment, a packet with destination address "111..." is arriving and allowed to search the partially downloaded routing table, then it will match prefix d according to LPM. However, if the routing table were completely downloaded, it should actually match prefix g since g (111*) is longer (more specific) than d (1*). The violation of LPM is an error-prone behavior with a high risk of packet misforwarding.

The violation of LPM stems from the overlook of routing table hierarchy. Formally, the prefix download sequence can be defined as $p_0, p_1, ..., p_{n-1}$. To preserve LPM, for arbitrary i and j, if p_i is a more specific prefix of p_j, then $i < j$ should be satisfied. In other words, if p_i is the descendant of p_j in the binary prefix trie, p_i should be downloaded ahead of p_j. In popularity-based strategy, if a non-leaf prefix is more popular than some of its descendants, it will be downloaded ahead of its less popular descendants, which will trigger an LPM violation.

It is expected that the partially downloaded prefixes will cover as many destination addresses as possible for a minimized routing table miss rate. Given traffic will be allowed into the line card after the download of the first m prefixes, we define cumulative popularity $C(m)$ as $\sum_{i=0}^{m-1} p_i.pop$, thus the optimal prefix download sequence should satisfy

$$\text{maximize } C(m)$$
$$\text{s.t. } i < j \text{ , if } p_i \text{ is a more specific prefix of } p_j, \forall i, \forall j$$

Solving this problem should give us the optimal sequence of prefixes to download. Depending on the expected wakeup time and the tolerance of default route forwarding or slow-path forwarding, we may also impose an upper limit on m.

B. Reduction to an Existing Problem

The above problem can be intuitively reduced to a well-known problem in graph theory named as "topological sort generation". A topological sort of a directed acyclic graph (DAG) is a linear ordering of its vertices such that every directed edge $\{u, v\}$ from vertex u to vertex v, u comes before v in the ordering. Similarly, to satisfy LPM, the more specific prefix in the binary trie should be downloaded before its ancestors. It is motivated to construct a DAG from the binary prefix trie to reduce the original problem to the more familiar one. The reduction is shown in Figure 6. First, we remove invalid prefix nodes (e.g., the node connects prefix d and prefix e as a junction) in the binary trie, which may split the trie into multiple directed acyclic subgraphs. Then, we traverse each subgraph to add directed edges from every ancestor to its descendants. E.g., prefix d in the DAG already has two edges inherited from the binary trie to its left/right child; one more edge to the remaining descendant prefix g is required to confirm the partial order relation. Next, we generate all the topological sorts from the DAG. The output order should be reversed to satisfy LPM. Finally, we compare the cumulative popularity values to find the optimal. Though topological sort generation has been widely studied [10], [11], it is actually NP-complete [12], and we have to rely on time-consuming backtracking to approach the optimal result. Since we have *not* found a reverse reduction from the NP-complete topological sort generation to our problem, it is insufficient to claim that our problem is NP-complete as well.

C. Near-Optimal Prefix Download Sequence

Backtracking on a large routing table takes lots of time. Here, we propose a near-optimal solution to address the LPM violation issue in polynomial time. The idea is to passively adjust the popularity-based prefix download sequence once the LPM violation is detected. It works as follows. First, prefixes are ranked by popularity in descending order. Then, we do a preorder trie traversal and make sequence adjustment

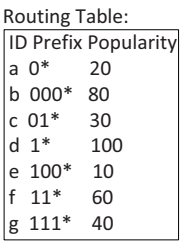


Fig. 5. Routing Table

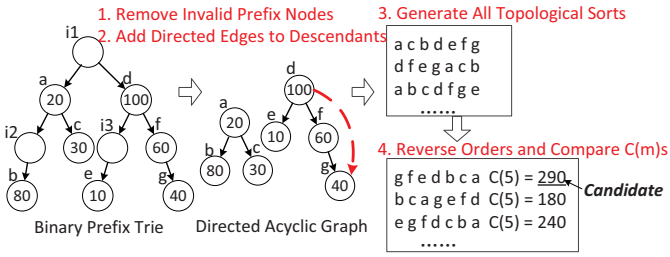


Fig. 6. Solve the Problem via Topological Sort Generation

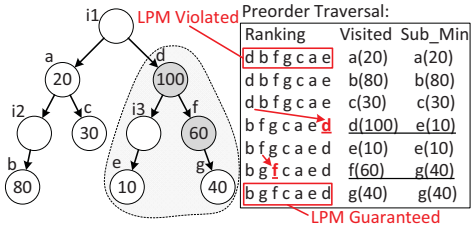


Fig. 7. Prefix Sequence Adjustment at LPM Violation

when LPM is violated. At each visited prefix during the traversal (denoted as *visited*), we search the subtrie rooted at *visited* for the prefix with the minimal popularity (denoted as *sub_min*). *sub_min* can either be the descendant of *visited* or be *visited* itself (when *visited* is at the leaf node or it has the minimal popularity in the subtrie). If *sub_min* is less popular than *visited*, a violation of LPM is detected, which will be eliminated by moving *visited* behind *sub_min* in the prefix download sequence. During the preorder traversal, we recursively move popular non-leaf prefixes behind their more specific but less popular descendants to achieve LPM guarantee. E.g., in Figure 7, when we visit prefix d during the traversal, we should adjust the download sequence to let d appear behind e, since the least popular prefix in the subtrie rooted at prefix d is prefix e, which is less popular than prefix d. The recursive function to perform the backward movement of non-leaf prefixes is listed in Algorithm 1.

In Algorithm 1, get_sub_min(v) gets the least popular prefix in the subtrie rooted at v; is_prefix(v) is required since not all prefixes in the trie are valid; move_behind(v, $v.sub\_min$) moves v from its original place to the behind of $v.sub\_min$; $v.l$ and $v.r$ represent the left and right child of prefix v. One naive implementation of get_sub_min(v) is to do a traversal from the root of each subtrie in a way of brute-force node inspections. But it will give rise to repetitive calculation since different subtries could be overlapped. Instead, we calculate the least popular prefix in each subtrie recursively and apply memoization to avoid repetitive calculation when solving overlapped subproblems. $v.sub\_min$ can be solved according to the following recursive formula

$$g(v) = \begin{cases} v & \text{if } v.l = v.r = null, \\ min(v, g(v.r)) & \text{if } v.l = null, v.r \neq null, \\ min(v, g(v.l)) & \text{if } v.l \neq null, v.r = null, \\ min(g(v.l), g(v.r)) & \text{if } v.l \neq null, v.r \neq null. \end{cases}$$

Here, g(v) represents get_sub_min(v), and min(x, y) is defined as "$x.pop < y.pop\ ?\ x : y$". The return value of each recursive function is memoized. Since not all nodes in the binary trie are valid routing prefixes, we assign $+\infty$ to the popularity field (*pop*) of these invalid prefix nodes as sentinels to simplify the boundary conditions in g(v).

In fact, Algorithm 1 only guarantees LPM without yielding the optimal result. Suppose $m = 5$, in the download sequence "b g f c a e d" generated from Algorithm 1, $C(5) = 230$. However, we could find another sequence "g f e d b c a", which is also LPM-satisfied but with $C(5) = 290$. In Algorithm 1, the prefix at the root of the subtrie is regulated to be placed behind its less popular descendants, while actually, this is not the only way of placement. E.g., the less popular descendants can be moved forward before the prefix at the root. Different placement strategies may derive different results. Practically, since most routing prefixes reside at the leaf nodes, Algorithm 1 can yield a near-optimal result (we evaluate it in §VI-B).

Algorithm 1: Prefix Sequence Adjustment at LPM Violation

```
1  Function lpm_adjust(TrieNode v)
2      if v == null then
3          return;
4      end
       /* get the least popular prefix in subtrie */
5      v.sub_min ← get_sub_min(v);
6      if is_prefix(v) && v.pop > v.sub_min.pop then
           /* move v behind v.sub_min */
7          move_behind(v, v.sub_min);
8      end
9      lpm_adjust(v.l);
10     lpm_adjust(v.r);
11 end
```

D. Optimal Solution in Polynomial Time

In this section, we solve the optimization problem via dynamic programming in polynomial time.

For each prefix v, we define $v.c[i]$ as the cumulative popularity of the first i prefixes in the *optimal* prefix download sequence from the subtrie rooted at v ($0 < i \leq v.sub\_num$, where $v.sub\_num$ is the number of valid prefixes in the subtrie rooted at v). Hence, maximizing $C(m)$ with LPM guaranteed is equal to solving $root.c[m]$. If v is at the leaf node, $v.c[1]$ is initialized to be $v.pop$, $v.c[i] (i \neq 1)$ is initialized to be 0; otherwise, $v.c[i]$ is initialized to be 0.

Then, we start to recursively define the value of the optimal solution. Assuming the optimal prefix download sequences from the subtries rooted at v's left and right child prefixes ($v.l$ and $v.r$) have already been solved, i.e., $v.l.c[i]$ and $v.r.c[i]$ have been calculated, then we can derive $v.c[i]$ according to the following recursive formulas in different cases.

If v is not included in the first i prefixes which will be downloaded with higher priority, we will have

$$v.c[1] = max(v.l.c[1], v.r.c[1])$$
$$v.c[2] = max(v.l.c[2], v.l.c[1] + v.r.c[1], v.r.c[2])$$
$$v.c[3] = max(v.l.c[3], v.l.c[2] + v.r.c[1], v.l.c[1] + v.r.c[2], v.l.r[3])$$
$$.....$$
$$v.c[i] = max(v.l.c[i], ..., v.l.c[i-j] + v.r.c[j], ..., v.r.c[i]), 0 < j < i$$

Since prefixes in the subtrie rooted at $v.l$ do not have overlap with prefixes in the subtrie rooted at $v.r$, the union of these two prefix sets will not generate LPM violation.

There are boundary conditions that v does not have left or right child prefix. In this case, $v.c[i]$ can be solved by

$$v.c[i] = \begin{cases} v.l.c[i] \text{ if } v.l \neq null,\ v.r = null, \\ v.r.c[i] \text{ if } v.l = null,\ v.r \neq null. \end{cases}$$

If v is included in the first i prefixes, *all* prefixes in the subtrie rooted at v should be included in the first i prefixes. If some descendants of v do not appear in the first i prefixes, they will be downloaded after v, which will cause LPM violation since they have longer prefixes than v. In this case, $v.c[i]$ can be solved by

$$v.c[i] = v.sub\_sum$$

Here, $v.sub\_sum$ is defined as the popularity summation of valid prefixes in the subtrie rooted at v. Both $v.sub\_sum$ and $v.sub\_num$ of each prefix can be precalculated.

For each prefix v, we define $v.s[i]$ to record the first i prefixes in the optimal prefix download sequence from the subtrie rooted at v. Note that the value of $v.c[i]$ is an integer while that of $v.s[i]$ is a set of prefixes. In the beginning, $v.s[i]$ is initialized to be $\emptyset$. During the recursive problem solving, $v.s[i]$ can be calculated along with $v.c[i]$. To prevent repetitive calculation on overlapped subproblems, we use memoization as well. The complete recursive procedure to solve the optimal prefix download sequence is shown in Algorithm 2.

Algorithm 2: Solving Optimal Download Sequence via DP

```
1  Function array_fill (int i, TrieNode v)
2      if v.c[i] is already solved then
3          return;
4      end
       /* v is included in the first i prefixes     */
5      if i == v.sub_num then
6          v.c[i] ← v.sub_sum;
7          v.s[i] ← all prefixes in the subtrie rooted at v;
       /* v in not in the first i prefixes          */
8      else if i < v.sub_num then
9          if v.l ≠ null && v.r == null then
10             array_fill(i, v.l); v.c[i] ← v.l.c[i]; v.s[i] ← v.l.s[i];
11         else if v.l == null && v.r ≠ null then
12             array_fill(i, v.r); v.c[i] ← v.r.c[i]; v.s[i] ← v.r.s[i];
13         else if v.l ≠ null && v.r ≠ null then
14             loop_begin ← min(i, v.l.sub_num);
15             loop_end ← i − min(i, v.r.sub_num);
16             for j = loop_begin; j ≥ loop_end; j−− do
17                 array_fill(j, v.l); array_fill(i − j, v.r);
18                 if max ≤ v.l.c[j] + v.r.c[i − j] then
19                     j_max ← j;
20                     max ← v.l.c[j] + v.r.c[i − j];
21                 end
22             end
23             v.c[i] ← max;
24             v.s[i] ← v.l.s[j_max] ∪ v.r.s[i − j_max];
25         end
26     end
27 end
```

The procedure will not halt until all the arrays are filled. Let's figure out the computational complexity. In Algorithm 2, all the prefixes in the binary trie will be visited. Each prefix is related to an array with i elements ($v.c[1]$, $v.c[2]$, ..., $v.c[i]$). To find out the optimal download sequence for each i, in worst case, comparisons in an inner *for* loop will be triggered (line 16). Therefore, the computational complexity of Algorithm 2 is $O(n^3)$. The solution space of $i1.c[5]$ is organized into a tree structure as shown in Figure 8. The tree node is visited in *depth-first* order during problem solving.

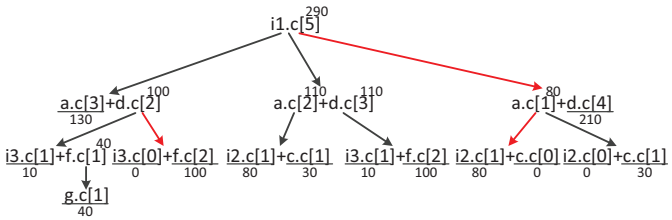


Fig. 8. Solving $i1.c[5]$ Using Algorithm 2

V. Implementation Issues

This section discusses hardware-related issues when downloading the routing table into the data plane.

Perform Batch Download of Routing Table. The route download process can be split into the following steps. (i) When the host processor is powered on while the boot procedure is finished, all the routes will be downloaded from the route processor into the host processor for consistency. (ii) The host processor invokes some computation to convert the logic representation of a routing prefix into hardware-specific instructions. E.g., a route update will be translated into a series of TCAM update instructions. We'll show in §VI-D that this step costs considerable time, especially on the less powerful embedded processor[2]. (iii) The instructions are executed by hardware to insert the route into the search engine (e.g., TCAM). In our design, the host processor can have a separate power supply from the other parts of a line card as discussed in §III, therefore it will maintain the route consistency with the route processor even when the line card is put into sleep. That is to say, step (i) can be skipped during the line card wakeup. Further, if the host processor could precompile hardware-specific instructions from the logic representation of the routing prefixes and perform a batch download of the entire routing table, step (ii) can be skipped as well, which will boost line card wakeup dramatically.

Reduce Lookup/Update Conflicts via Caching. Massive packet lookups and route updates will coexist at the search engine (e.g., TCAM) during the line card wakeup for two reasons. (i) The routing table is designed to be downloaded in a batch. (ii) Traffic will be allowed into the data plane before the complete download of a routing table. When a bank of TCAM is updated, it will freeze lookup requests to that bank until the completion of the update [13], [14]. Hence lookup/update conflicts will occur at the port of TCAM. If we assign a higher priority to update requests, the update of

[2]Most commercial routers use embedded processors as the host processor in the line card to save power and reduce heat dissipation.

the less popular prefixes will limit the lookup performance and cause buffer overflow and packet dropping. On the other hand, if lookup requests have a higher priority, massive lookups will saturate the search engine and starve the download of the remaining prefixes. The critical resource contention puts us in dilemma. Given temporal locality is widely exploited in Layer-3 traffic [15], we design a cache-based architecture to resolve lookup/update conflicts as shown in Figure 9. A cache indexed by the destination address is placed in front of the TCAM to absorb a large amount of lookup requests. If a lookup request is hit in the cache, the corresponding packet will be directly forwarded without disturbing the TCAM. Otherwise, the lookup request will be sent to a buffer waiting for the TCAM search. Therefore, plenty of time slots are reserved for TCAM update operations. The cache will be updated with the most recent search results from the TCAM. A scheduler is designed to process update requests with higher priority except that the buffer has the risk of overflow. This scheduling strategy enables efficient prefix update while adaptively preventing packets from being dropped. After all the prefixes have been downloaded, the line card will migrate to the regular running state, in which TCAM itself can well handle the incoming traffic. At this time the cache can be deallocated to the on-chip memory pool.

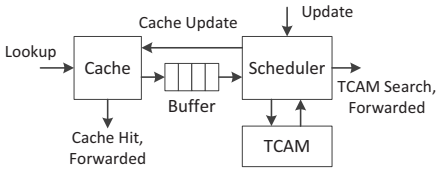


Fig. 9. Cache-Based Architecture to Resolve Lookup/Update Conflicts

Cache Coherence Issue. As the cache is searched via exact match (for speed) while the backup TCAM is searched via longest prefix match, there is a coherence issue during the cache update. If a lookup request is missed in the cache as well as in the TCAM because the prefix is not yet downloaded, according to cache replacement convention, the destination address of that packet will be updated into the cache tagged as "NO_MATCH". However, when the corresponding prefix is downloaded into the TCAM, the previous cache entry tagged as "NO_MATCH" should be outdated. To maintain cache coherence, we need to inform the cache about the invalidation of that entry. Essentially, it can be further translated into a problem of finding destination addresses in the cache using the prefix, which appears to be the reverse engineering of route lookup via longest prefix match. Solving the problem is not that straightforward. Here, we circumvent it by disallowing the cache update from failed TCAM search requests, preventing the cache from being filled with pending results.

VI. Evaluation

A. Methodology

The evaluation is divided into three parts. First, we evaluate the prefix ranking algorithms proposed in §IV. Suggestions on when to allow traffic into the line card are also provided based on the experimental results. Second, we build a discrete event simulation of the cache-based architecture proposed in §V to reveal the interaction among different system parameters. Finally, we illustrate how much time can be reduced in the line card wakeup process by applying proposed designs. In the above experiments, we use the same trace and routing table described in §II-E. For the last experiment, since the commercial router is a closed system and difficult for us to do architectural modifications on it, we do the experiment on a high-speed line card prototype built in our lab, where FPGA-based network processor and traffic manager are contained (Figure 10). We do not build extra route processor card, instead, we use a server to download the routing table to the host processor in the line card prototype via Ethernet.

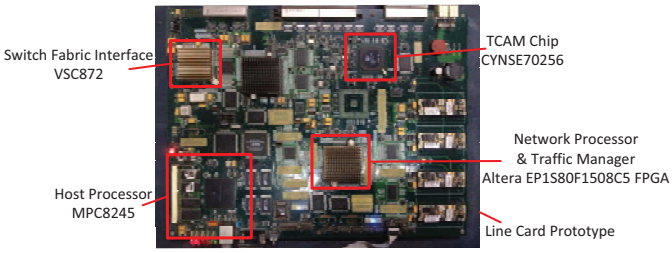


Fig. 10. FPGA-Based Line Card Prototype

B. Prioritized Prefix Download

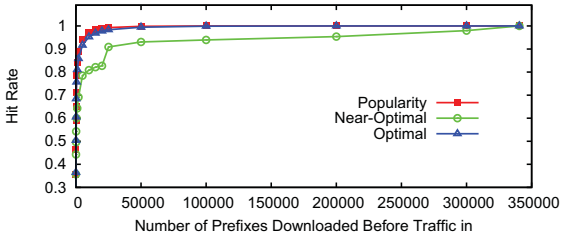


Fig. 11. Prefix Hit Rate Varies as Traffic is Allowed in at Different Time

Figure 11 illustrates the relationship between prefix hit rate and the number of downloaded prefixes before traffic is allowed into the line card using three strategies described in §IV. The more prefixes are in the data plane before traffic arrives, the higher prefix hit rate will be. The popularity-based strategy achieves the highest prefix hit rate, but will suffer from LPM violation. The optimal strategy (Algorithm 2) eliminates LPM violation and achieves a prefix hit rate extremely close to that of the popularity-based strategy. The near-optimal strategy also has a pretty good prefix hit rate, but with a lower computational complexity. The explanation lies in that leaf prefixes account for a large proportion (295970/340584) in the evaluated routing table and sequence adjustments of non-leaf prefixes occur in a relatively small number. E.g., only 11141 sequence adjustments are made to avoid LPM violation with Algorithm 1. The observation that leaf prefixes dominate the total prefix number is also mentioned in other works [16], [17]. When adopting the optimal strategy, the prefix hit rate will be over 96.8% when traffic is allowed in after the first 15000 prefixes (4.4%) are downloaded.

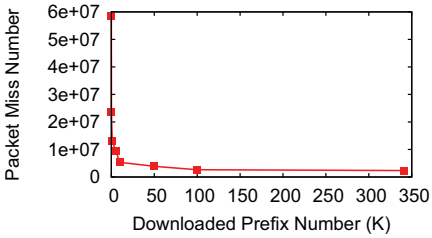


Fig. 12. Packet Miss Number vs. Downloaded Prefix Number (Affected by Traffic Arrival Time)

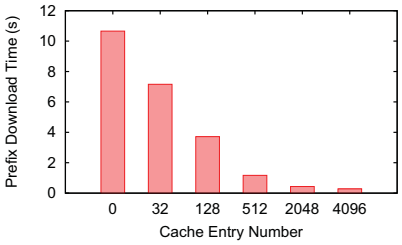


Fig. 13. Prefix Download Time vs. Cache Size

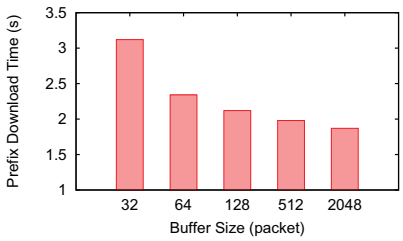


Fig. 14. Prefix Download Time vs. Buffer Size

C. Simulation Under Cache-Based Architecture

A discrete event simulation is built strictly according to the system architecture in Figure 9. The input parameters are elaborated to match with the real hardware. TCAM is set to work at 66MHz and can process one lookup request per clock cycle. The TCAM update frequency is set to be 1.65MHz as measured from a real TCAM chip CYNSE70256. The cache is assumed to work under 100MHz frequency. The packet arrival interval is set to be 32ns to simulate the worst-case OC-192 traffic. The prefix download sequence is generated from the optimal solution. The default cache entry number is set to be 512 and the default buffer size is set to hold 64 packets. We use LRU as the cache replacement policy in the simulation.

Figure 12 shows the number of packets missed in route lookup when traffic is allowed into the line card at different time (i.e., with the data plane accumulating different numbers of popular prefixes). Here, we use a 1-minute traffic. The curve trend is in accord with Figure 11 that the packet miss number drops sharply after a slim slot of popular prefixes are downloaded. When nearly all the prefixes are downloaded, the packet miss rate approaches 4% in the final, which equals to the inherent default route forwarding rate in the traffic.

Figure 13 shows the prefix download time varies significantly under different cache sizes. Fewer cache entries result in a lower cache hit rate thus more packets will be sent to TCAM to interrupt the prefix update process. Therefore the total prefix download time will be accumulated to a relatively large value. When the cache entry number increases over a large number (e.g., 512 in our experiment), the time reduction is not so remarkable. In fact, trade-off exists in the on-chip memory cost and the total prefix download time.

Figure 14 shows the prefix download time changes under different buffer sizes. The change appears to be relatively trivial compared with Figure 13. After some debugging, we find that packets will be queued in the buffer waiting to be scheduled (update has a higher priority than lookup). As a result, adding buffer size will not drastically reduce the download time. However, a small-sized buffer is still required for high-speed traffic, e.g., when the buffer size is set to be less than 16 (packets), the update process will be suspended more frequently to prevent packets from being dropped.

D. Reduction of Line Card Wakeup Time

We measure the line card wakeup time on our FPGA-based prototype to find out how much time is needed to recover a minimized set of router's functions, which include packet sending/receiving via Ethernet PHY/MAC and Layer-3 packet forwarding. To speed up the wakeup process, time-consuming memory diagnosis is skipped. When the system is powered on, the software package is downloaded from a remote server (acting as the route processor) to the host processor in the line card prototype. It takes 6s for VxWorks to be completely started which includes 1s for software package download via TFTP protocol and 5s for booting up the system. In the prototype, the configuration of Ethernet interfaces is hardwired which costs nearly no time. Then, the routing table (consisting of 340548 routes) is downloaded into the search engine, and this process can be further divided into 3 subroutines. 1) The routing table is downloaded from the remote server to the local host processor via TFTP, which costs 4718.2ms. 2) On the host processor, the routing table is parsed into hardware-specific instructions and a prefix trie is built as well for the ease of future updating. This subroutine costs 27539.6ms. 3) Finally, the hardware-specific instructions are executed to install the prefixes into TCAM and it consumes 2545.3ms.

TABLE III
LINE CARD PROTOTYPE WAKEUP TIME BREAKDOWN

Events	Time (ms)
software package downloaded via network	1000
VxWorks booted up	5000
routing table downloaded via network	4718.2
prefix parsing; trie construction	27539.6
prefix updated into TCAM	2545.3

If the host processor is kept up, the 6s for OS initialization and 4718.2ms for routing table download via network can be skipped. If we download the most popular prefixes (e.g., 5%) before traffic arrives, the time for prefix parsing, trie building and prefix downloading can be reduced to 1/20 of the original time. If we preprocess the prefixes on the host processor and download the routing table in a batch, prefix parsing and trie building can be further skipped. Finally, the total line card wakeup time can be reduced to 127.27ms for executing TCAM instructions to download prefixes in a batch (0.3% of the original line card wakeup time). In Table I, hardware initialization time of the commercial router costs 70s. If we could modify the system BIOS to enable a minimized hardware configuration before traffic arrives, the hardware initialization time could be shortened. Hence, the total line card wakeup time here could be sufficiently small to support power-saving schemes via sleeping/waking up line cards.

VII. Related Work

As the network infrastructure consumes more and more power, improving a router's power efficiency has seen increasing interests from both the academia [18] and the industry [19]. Usually, we achieve power saving at different levels. (1) Dynamic frequency scaling (DFS) and dynamic voltage scaling (DVS) at chip level are conventionally used in reducing power when adapted with the traffic loads [20]. (2) Link-rate adaptation at link level (i.e., switching from 1Gbps to 100Mbps or 10Mbps) is suitable for Ethernet links where traffic is light [21], [22]. (3) Sleep/standby at link/interface level can exploit putting the link/interface into sleep, while temporarily buffering the arriving traffic [4]. Alternatively, traffic engineering can aggregate all the traffic across a network to a subset of the links, thus enabling opportunities to put other links to sleep [3]. Sleep/standby at link/interface level can be more suitable for power saving on backbone networks, since most Internet backbone links use SDH/SONET specification thus link-rate adaptation does not apply. Further, putting an entire link to sleep is more power-efficient than chip-level frequency/voltage scaling. In sleep/standby schemes coordinated by traffic engineering, if a sleeping link cannot be ready quickly after receiving the wakeup command, incoming packets could possibly be dropped. However, many recent schemes on power saving presume the link can be recovered in zero-time to ease the protocol design [23], [3], [24]. E.g., [3] use a centralized controller to collect the link status to achieve the globally-optimized decision. The link status will be repeatedly collected and calculated every 5-15 minutes which is actually the same magnitude of the line card wakeup time in real world. In [23], the variable data center traffic loads will trigger the dynamic adjustment of the link state via centralized controlling. Line cards using existing designs could not keep pace with the periodical on/off instructions sent from the centralized controller according to our measurement. To our investigation, we've found no literature addressing the issue of fast line card wakeup. Our new line card design can well support the above sleep/wakeup power saving schemes.

VIII. Conclusion

A line card in a commercial router usually spends 5-10 minutes to be ready to correctly forward packets after powered on. This entirely subverted the previous assumption that a line card costs zero-time to wake up in recent studies, causing the inapplicability of many power-saving schemes. To support fast line card wakeup, we made penetrating measurement on the boot procedure of line cards. Based on the observations, we propose designs including separating the on-board CPU power supply from the other parts of a line card, setting up a minimized hardware configuration first and especially, prioritizing prefix download, thanks to the finding of the power law in route lookup. Experiments on a prototype show that we can reduce the wakeup time to 127.27ms, which can well support the existing sleep/wakeup power saving schemes.

References

[1] J. Chabarek, J. Sommers, P. Barford, C. Estan, D. Tsiang, and S. Wright, "Power awareness in network design and routing," in *INFOCOM, 2008 Proceedings IEEE*. IEEE, 2008, pp. 457–465.

[2] M. Gupta and S. Singh, "Greening of the internet," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM, 2003, pp. 19–26.

[3] M. Zhang, C. Yi, B. Liu, and B. Zhang, "Greente: Power-aware traffic engineering," in *International Conference on Network Protocols (ICNP), 2010*. IEEE, 2010, pp. 21–30.

[4] S. Nedevschi, L. Popa, G. Iannaccone, S. Ratnasamy, and D. Wetherall, "Reducing network energy consumption via sleeping and rate-adaptation." in *NSDI'08*, vol. 8, 2008, pp. 323–336.

[5] L. J. Wobker, *Power consumption in high-end routing systems*, http://www.nanog.org/meetings/nanog54/presentations/Wednesday/Wobker.pdf.

[6] J. Rexford, J. Wang, Z. Xiao, and Y. Zhang, "Bgp routing stability of popular destinations," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*. ACM, 2002, pp. 197–202.

[7] R. Govindan and A. Reddy, "An analysis of internet inter-domain topology and route stability," in *INFOCOM, 1997 Proceedings IEEE*, vol. 2. IEEE, 1997, pp. 850–857.

[8] S. Ramabhadran and G. Varghese, "Efficient implementation of a statistics counter architecture," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 31, no. 1. ACM, 2003, pp. 261–271.

[9] A. Kumar, J. Xu, and J. Wang, "Space-code bloom filter for efficient per-flow traffic measurement," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 12, pp. 2327–2339, 2006.

[10] Y. L. Varol and D. Rotem, "An algorithm to generate all topological sorting arrangements," *The Computer Journal*, vol. 24, no. 1, pp. 83–84, 1981.

[11] D. E. Knuth and J. L. Szwarcfiter, "A structured program to generate all topological sorting arrangements." *Inf. Process. Lett.*, vol. 2, no. 6, pp. 153–157, 1974.

[12] G. Brightwell and P. Winkler, "Counting linear extensions is #p-complete," in *STOC '91*. New York, NY, USA: ACM, 1991, pp. 175–181.

[13] H. Song and J. Turner, "Fast filter updates for packet classification using tcam," in *GLOBECOM'06. IEEE*. IEEE, 2006, pp. 1–5.

[14] D. Shah and P. Gupta, "Fast incremental updates on ternary-cams for routing lookups and packet classification," in *Proc. of Hot Interconnects-8, Stanford, CA, USA*, 2000.

[15] C. Partridge, "Locality and route caches," in *NSF Workshop on Internet Statistics Measurement and Analysis*, vol. 19, 1996.

[16] T. Yang, Z. Mi, R. Duan, X. Guo, J. Lu, S. Zhang, X. Sun, and B. Liu, "An ultra-fast universal incremental update algorithm for trie-based routing lookup," in *International Conference on Network Protocols (ICNP), 2012*. IEEE, 2012, pp. 1–10.

[17] L. Luo, G. Xie, Y. Xie, L. Mathy, and K. Salamatian, "A hybrid ip lookup architecture with fast updates," in *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2012, pp. 2435–2443.

[18] R. Bolla, R. Bruschi, F. Davoli, and F. Cucchietti, "Energy efficiency in the future internet: A survey of existing approaches and trends in energy-aware fixed network infrastructures." *IEEE Communications Surveys and Tutorials*, vol. 13, no. 2, pp. 223–244, 2011.

[19] *System Power Challenges*, http://www.cisco.com/web/about/ac50/ac207/proceedings/POWER_GEPPS_rev3.ppt.

[20] S. Herbert and D. Marculescu, "Analysis of dynamic voltage/frequency scaling in chip-multiprocessors," in *2007 ACM/IEEE International Symposium on Low Power Electronics and Design*, 2007, pp. 38–43.

[21] H. Anand, C. Reardon, R. Subramaniyan, and A. George, "Ethernet adaptive link rate (alr): Analysis of a mac handshake protocol," in *Proceedings 2006 31st IEEE Conference on Local Computer Networks*, 2006, pp. 533–534.

[22] C. Gunaratne, K. Christensen, B. Nordman, and S. Suen, "Reducing the energy consumption of ethernet with adaptive link rate (alr)," *IEEE Transactions on Computers*, vol. 57, no. 4, pp. 448–461, 2008.

[23] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "Elastictree: Saving energy in data center networks." in *NSDI*, vol. 3, 2010, pp. 19–21.

[24] N. Vasić, P. Bhurat, D. Novaković, M. Canini, S. Shekhar, and D. Kostić, "Identifying and using energy-critical paths," in *CoNEXT '11*. New York, NY, USA: ACM, 2011, pp. 18:1–18:12.