Distilling Router Data Analysis for Faster and Simpler Dynamic IP Lookup Algorithms

Filippo Geraci[1] and Roberto Grossi[2]

[1] IIT-CNR, Pisa, and Dipartimento di Ingegneria dell'Informazione, U. Siena, Italy
filippo.geraci@iit.cnr.it
[2] Dipartimento di Informatica, Università di Pisa, 56127 Pisa, Italy
grossi@di.unipi.it

Abstract. We consider the problem of fast IP address lookup in the forwarding engines of Internet routers. We analyze over 2400 public snapshots of routing tables collected over five years, discovering what we call the *middle-class effect*. We exploit this effect for tailoring a simple solution to the IP lookup scheme, taking advantage of the skewed distribution of Internet addresses in routing tables. Our algorithmic solution is easy to implement as it is tantamount to performing an indirect memory access. Its performance can be bounded tightly and has very low memory dependence (e.g. just one memory access to off-chip memory in the hardware implementation). It can quickly handle route announcements and withdrawals on the fly, with a small cost which scales well with the number of routes. Concurrent access is permitted during these updates.

1 Introduction

The IP lookup problem is a recurrent problem in the literature for packet forwarding in the Internet [16]. Routers have to forward lots of packets from input interfaces to output interfaces (*next hops*) based on packet's destination Internet address, called an *IP address*. Forwarding a packet requires an IP address *lookup* at the routing table to select the next hop corresponding to the packet. (We will use the term "routing table" to denote what is more properly called a "forwarding table.") As routers have to deal with links whose speed constantly improves, the address lookup is considered one of the major bottlenecks [6, 16]. For networks with a link speed of 10 gigabits per second (OC-192), they need to forward up to 33 million packets per second, assuming that each packet is 40 bytes long. Other bottlenecks, such as those involved by fair queuing policy and IP switching technology, are well understood and handled [11].

In IPv4 [13] the prefixes are binary strings of variable length using the syntax $X.Y.W.Z/L$ to represent the first L bits of the 4-byte word $X.Y.W.Z$, where $8 \leq L \leq 32$. Prefixes can be up to 128 bits in IPv6 [5] (but then have a different syntax). The use of prefixes increases the complexity of the IP address lookup problem. For each packet, more than one prefix in the routing table can match the packet's IP address. In this case, the adopted rule is to take the *longest matching prefix*. Given prefixes $p_1, p_2, \ldots, p_n$, for any binary string x we want

prefix	hop
65.10.10.0/24	1
192.168.0.0/17	2
192.168.0.0/18	3
192.168.64.0/18	2
192.168.0.0/32	4
192.168.0.0/29	5

Table 1. A routing table.

layer 1		layer 2	
65.10.10.0/24	1	192.168.0.0/24	3
192.168.0.0/17	2	192.168.0.0/32	4
192.168.0.0/18	3	192.168.0.0/29	5
192.168.64.0/18	2		
192.168.0.0/24	255		

Table 2. Its two-layer organization.

to identify the longest p_i that equals the first bits of x, where $1 \leq i \leq n$. For example, let us consider the prefixes in Table 1. Both prefixes 192.168.0.0/17 and 192.168.0.0/18 match the IP address 192.168.32.125; hence, the packet is forwarded to next hop 3. We will only consider situations arising with single hops, since dealing with multi-hops is very similar. No-route-to-host is the special next hop 0 associated with the empty prefix ϵ.

In this paper we stress the importance of data analysis on real routing tables *before* designing IP lookup algorithms. We begin with the experimental analysis performed on public databases of nearly 2400 snapshots of routing tables collected over five years. We identify some new parameters characterizing the (skewed) distribution of prefixes in routing tables. Based upon our findings, we provide a new and simple solution to the IP address lookup problem.

Our starting point is the result based on full expansion and compression of routing tables by Crescenzi, Dardini and Grossi [4]. (It was later referred to as CDG in [3].) Let us illustrate CDG conceptually with Table 1 for IPv4, considering all possible 2^{32} IP addresses that can be queried. For each such address, we associate with it the corresponding next hop according to its longest prefix match in Table 1. Now, let us organize the 2^{32} next hops thus computed in a $2^{16} \times 2^{16}$ matrix. Lookup time is now a direct access to this matrix; however, its size does not fit current capacity of L2 caches. Observing that many rows and columns contain repeated values, CDG considers only the distinct rows (as individual sequences of 2^{16} next hops each); they are further compressed using run length encoding (RLE) on their values. The lookup requires three accesses but the size reduces to very few megabytes.

To our knowledge CDG is the first to describe a lookup scheme whose design is fully driven by data analysis. A frequently cited survey [16] published in 2001 shows that CDG is almost an order of magnitude faster than its state-of-the-art competitors at that time (see Table 3 in [16]). Even in the worst case, the frequency of lookups with small response time is impressively high and does not depend on the traffic through the router (see Fig. 22 in [16]). Unfortunately, CDG has some drawbacks. The survey reports that "Schemes using multibit tries and compression give very fast search times. However compression and the leaf pushing technique used do not allow incremental updates. Rebuilding the whole structure is the only solution." Moreover, some authors [3, 7, 15] pointed out some cases in which the space requirement of CDG is too high, possibly causing its performance to suffer in the worst case.

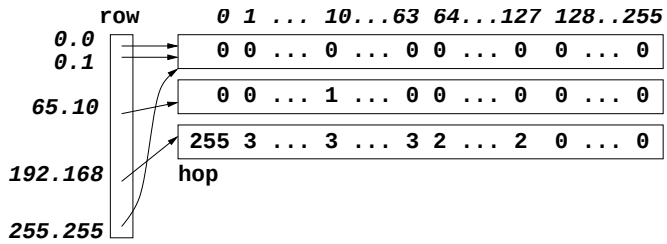

Fig. 1. The arrays `row` and `hop` for layer 1 in Table 2.

Our scheme. We present a lookup scheme that exploits the original idea of CDG in a novel and even simpler way. Going on in our illustrative Table 1, let us truncate the prefixes in the table that are longer than 24, thus retaining just 24 bits and associating with them a dummy next hop (e.g. 255). We obtain *layer 1*, as shown in the left column of Table 2. The prefixes longer than 24 constitute *layer 2*, in the right column of Table 2, which is scarcely populated according to our data analysis. (Note that `192.168.0.0/24` in layer 2 is pushed from layer 1 to deal with IP addresses matching `192.168.0.0/`L, where $24 \leq L < 29$.) We can now revisit the approach described for CDG and apply it to layer 1. With each 24-bit address, we associate its corresponding next hop according to its longest prefix match in layer 1. Organizing the resulting next hops into a $2^{16} \times 2^8$ matrix, we keep only the distinct rows (and do not compress them with RLE) as shown in Fig. 1. It suffices to perform a lookup in two accesses in layer 1 by looking at the first 24 bits in the given IP address. For example, the next hop for IP address 192.168.32.125 can be retrieved by following the pointer in `row`[192.168] to a row of 256 entries, in which entry 32 contains the result, next hop 3. Sometimes we get the dummy next hop in layer 1 and so we also need to perform the lookup in layer 2 (this happens very rarely according to our data analysis). Access time and space occupancy are definitively improved over CDG in this way.

Our method exploits some properties that allow us to avoid the drawbacks of CDG. The main discovery is what we call the *middle-class effect* in real routing tables: even though the majority of prefixes have lengths ranging from 16 to 24, they tend to follow regular patterns. In other words, we have a good chance to store the mapping from all the 2^{32} IP addresses to the next hops into a compact table, so that lookup and update are able to access the table very quickly using indirection. Our contributions can be summarized as follows. First, we save space significantly over CDG since we have a much more stable space occupancy that scales linearly with the table size (Fig. 3, left). We no longer need the run-length encoding (RLE) adopted in CDG, because we organize suitably the prefixes. Second, we improve lookup time by nearly 30% (Fig. 3, right). Third, we can dynamize the table, performing updates quickly without rebuilding the whole structure as previously required. Our update algorithm is robust since we can efficiently bound the worst case, which is important for unauthenticated announcements [9]. Concurrent access is also permitted while updating. Our method compares favorably with previous work [16]. We plan to extend our experimental investigation to the interesting method recently proposed in [3].

router	#tables	from	to
aads	538	10-01-00	05-15-02
mae-east	230	10-01-00	06-01-01
mae-west	618	10-01-99	04-12-02
paix	78	10-01-01	03-10-02
pacbell	576	12-09-98	05-15-02
ripe-ncc	365	01-01-03	12-01-03
ripe-ncc	19	10-10-99	04-01-04

router	date	router	date
aads	05-30-01	oregon-03	07-10-03
att	07-10-03	pacbell	05-30-03
east.attcanada	07-10-03	paix	05-30-01
funet	10-30-97	telstra	03-31-01
mae-west	05-30-01	telus	07-10-03
west.attcanada	07-10-03	oregon-01	03-31-01

Table 3. Dataset description.

2 Data Analysis of Routing Tables

In this section, we describe our data analysis on routing tables to highlight a useful property for prefixes of length from 16 to 24, called the *middle-class effect*. We first describe the large data set of public routing tables for IPv4 in Section 2.1. We illustrate the middle-class effect in Section 2.2, showing how to exploit it for a two-layer organization of IP lookup tables in Section 2.3.

2.1 Databases and experimental platforms

We base our analysis on an extensive data set of more than 2400 snapshots of routing tables available from public databases, collected over a period ranging from 1998 to 2004. The major source is the IPMA project [10]. We also collected all daily data for year 2003, plus some monthly snapshots, from the RIPE NCC [14]. Some authors singled out individual snapshots that cause the worst-case behavior of CDG in terms of space occupancy; hence, they are good benchmarks for our method as well. Most of these tables have been employed in the experiments [3, 12, 15]. We report the figures in Table 3.

As for the updates, we collected *all* the announcements and withdrawals available for the entire year 2003 on RIPE NCC. In Fig. 2(left), we plot their number in millions on a daily basis. As we can see, the number of withdrawals is an order of magnitude smaller than the number of announcements. On the average, there is approximately one announcement per second; clearly, they arrive in bursts. For example, note the peak of more than 20 million updates on Oct. 25–26, 2003 (around 300 on the x-axis). We will use this particular peak for intense benchmarking in Section 4. As for the lookups, we could not find publicly available traffic traces (for privacy reasons). We instead use random data from previous work [3], as well as synthetic data. We obtain the latter by extending the approach in [2], adopted in the network community, to generate traffic data according to the distribution of the prefixes of any given routing table T (details given in the technical report [8]).

For our experiments we employed an AMD Athlon XP 1900+ (1.6GHz), 256Mb RAM DDR at 133Mhz, 256Kb L2 cache, 128Kb L1 cache (64 Kb data and 64Kb instructions) running Linux kernel 2.4.22. We used `gettimeofday` for timings. We plan to extend the experimentation to more platforms (e.g. those based on the PowerPC).

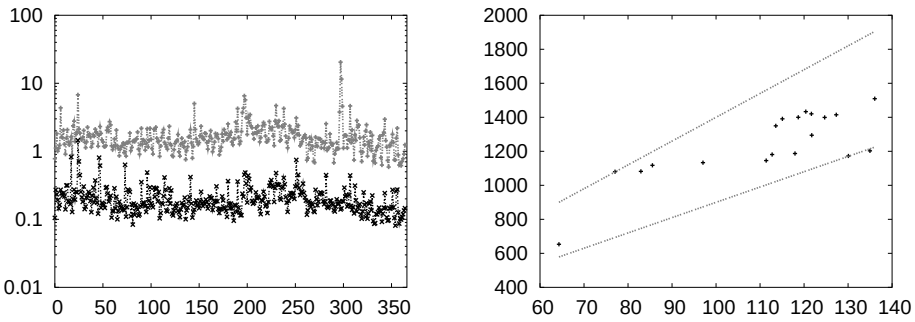

Fig. 2. *Left picture:* Millions of daily announcements (top) and of daily withdrawals (bottom) for RIPE NCC, in logarithmic scale on the y-axis. The x-axis reports the 365 days in year 2003 as numbers in the interval $0 \ldots 364$. *Right picture:* Space occupancy of our scheme scales linearly with table size. The x-axis reports the thousands of prefixes and the y-axis the number of kilobytes taken.

2.2 Distilling the middle-class effect in routing tables

In order to illustrate our ideas, let us consider any routing table T; in our case, the snapshot of the RIPE NCC router taken on April 1st, 2004, containing 138201 prefixes. (Note that analogous properties hold for the other router snapshots mentioned in Section 2.1.) What is widely known is the skewed distribution of prefixes from length 1 to 32 in T. Indeed, 98% of the prefix lengths are in the interval $16 \ldots 24$, which we call middle-class prefixes. This skew is typically a good sign for compressing data.

We can get further insight on T by examining the trie storing all the prefixes in T since an IP lookup is a traversal of a path from the root of the trie. The nodes of the tries are labeled with the next hops according to prefixes in T. Some nodes u are also marked to record the fact that the path from the root to u stores a prefix of the table. We can draw two cutlines on the trie, on levels 16 and 24. We obtain a set of at most 2^{16} sub-tries of height no more than $h = 8$. (The height is the numbering of levels, starting from 0 for the root.) For random data, we do not expect to find isomorphic sub-tries. Indeed, the probability that *no* two sub-tries are isomorphic is very near to one, i.e., $(1-p)^{2^{16}} \approx 1$ (see [1]). We instead consider a weaker notion which is more relevant in our case. Given a trie of height h, let us expand it to its complete form (also called prefix expansion) so that all the leaves are on the same level. Nodes are still labeled and marked according to the prefixes in T, except that they are now part of a complete trie (with expanded leaves that explicitly represent all possible 2^h binary strings of length h). Note that each string is associated with its correct next hop when seen as part of an IP address.

We say that two tries are *equivalent*, if the sequence of 2^h next hops in the expanded leaves on level h of the former is identical to that of the latter, when scanned in left-to-right order. In other words, when a lookup with h bits is

performed on two equivalent tries, the next hops thus returned make them indistinguishable. Note that two isomorphic tries are equivalent while the reverse is not necessarily true, since different combinations of shapes and labels/marks can yield the same sequence of next hops. We are therefore interested in selecting one representative for each class of equivalent tries. In our case, we apply this selection to the sub-tries of height at most 8 obtained from the cutlines on levels 16 and 24 (corresponding to the middle-class prefixes). How many of them are equivalent? For random data, we still expect that there are very few equivalent sub-tries. Fortunately, we observe what we call the **middle-class effect** in real routing tables T when we build the trie on the prefixes in T:

> *Many sub-tries of height* ≤ 8 *on level* 16 *are equivalent with lots of repetitions, and their nodes store the great majority of prefixes in* T.

So there is a good chance to store fewer than 2^{16} sub-tries by keeping just one representative for each equivalence class. Even though the majority of prefixes are middle-class (98% in our T), they do follow regular patterns in the routing table. In our example, table T gives 13834 nonempty sub-tries of height at most 8 on level 16. Among these, we are left with 3241 representatives of equivalence classes. These are not random data at all!

2.3 Two-layer lookup tables exploiting the middle-class effect

We now present a simple, but powerful, lookup scheme based on the middle-class effect described in Section 2.2. To be more concrete, we illustrate our approach by referring to T, shown in Table 1. Following what claimed in the middle-class effect, we can conceptually cut the trie built on the router prefixes, on level 24. We transform the resulting trie into a direct acyclic graph (DAG), in which equivalent sub-tries (of height at most 8) on level 16 are collapsed. This DAG can be represented by the data structure in Fig. 1, consisting of two components:

`hop`: This is the two-dimensional array of $\hat{\alpha} \times 256$ next hops, where $\hat{\alpha}$ is the number of non-equivalent sub-tries on level 16 of the DAG ($\hat{\alpha} = 3$ in our example); each such sub-trie is represented by its sequence of $2^8 = 256$ next hops *without* RLE compression.

`row`: This is the array of 2^{16} entries mapping the first 16 bits of IP addresses to the suitable row of `hop`. (Equivalently, they represent the children pointers of DAG nodes on level 16.)

In other words, we expand the upper part of the DAG that corresponds to the first 16 levels into `row`, and store in each row of `hop` the sequence of 256 next hops derived from each class of equivalent (collapsed) sub-tries. The pointers in `row` keep track of the corresponding sub-tries thus represented in `hop`. For any IPv4 address $x = x_1.x_2.x_3.x_4$, the next hop obtained by searching for x into the trie (compactly represented by the DAG) is that stored in $\texttt{hop}\big[\,\texttt{row}[x_1.x_2],\, x_3\,\big]$.

The above data structure forms what we call *layer 1*, which allows us to answer IP lookups by examining the first 24 bits (which is mostly the case in our collected data). The set of remaining prefixes (longer than 24 bits) in T

form *layer 2*, which is augmented by taking their first 24 bits. (Recall that we associate with these bits the dummy hop, e.g. 255, in layer 1.) Table 2 shows an example. Dummy prefixes of length 24 in layer 1 correspond to prefixes of length 24 with the correct next hop in layer 2. The number of such dummy prefixes cannot exceed that of prefixes longer than 24. At this stage, we do not need to choose any specific implementation of the lookup table for layer 2.

Before discussing the experimental analysis on the lookup in Section 3, we first assess the space occupancy of our scheme.

Fact 1 *Layer 1 occupies* $\hat{\alpha} \times 256 + 2^{16} \cdot size(ptr)$ *bytes, where* $\hat{\alpha} \leq 2^{16}$ *is the number of non-equivalent sub-tries of height at most 8 on level 16, and* $size(ptr) \geq (\log_2 \hat{\alpha})/8$ *is the number of bytes encoding a pointer to* `hop`*'s rows.*

In the worst case, `hop` occupies no more than 16 Mb and `row` needs 256 Kb (using 4-byte pointers) by Fact 1. This is actually a pessimistic estimate, since we only keep the sub-tries that are *not* equivalent each other. In order to have a fair comparison of our scheme with CDG, we must add the space taken by the lookup table adopted for layer 2:

	lookup	space (Kb)				lookup	space (Kb)		
	time	total	layer 1	layer 2		time	total	layer 1	layer 2
CDG	7012	2022	1521	501	N-Way Srch	5211	1608	1521	87
Binary Search	5221	1556	1521	35	Binary Trie	5758	1649	1521	128
K-Way Search	5274	1556	1521	35	Hybrid Trie	5297	1649	1521	128

In the above table, we report the figures for several choices with router west.attcanada, where we compare several methods for storing the prefixes in layer 2: CDG, array with binary search, k-way search (with $k = 8$ and $k = 2n$ where n is the number of prefixes), binary tries, and hybrid tries in which the first three levels are indexed by individual bytes and the next 8 levels (at most) are indexed by individual bits. Indeed, a lookup in layer 2 surely matches at least the first 24 bits by construction. Lookup times measure the number of microseconds for running 100,000 lookups.

We computed similar tables for the other snapshots: it turns out that hybrid tries are the best trade-off between space and lookup time. Choosing hybrid tries for storing prefixes in layer 2, we report in Table 4 the space improvement with respect to CDG for the 13 benchmark tables listed in Section 2.1. As we can see, the column corresponding to our scheme gives a quite stable occupancy in space with respect to the routing table size (#prefixes). This is better highlighted if we consider the entire year 2003 of RIPE NCC, with the results for our scheme being plotted on the bottom part of Fig. 3(left).

The net result for our scheme is a lookup table whose space occupancy scales linearly with the number of prefixes. (Clearly, layer 1 alone scales as well; moreover, its maximum size is 16Mb.) Fig. 2(right) illustrates this behavior for the available monthly snapshots of RIPE NNC, from October 1999 to April 2004, with a number of prefixes ranging from 65841 (yielding $\hat{\alpha} = 1404$) to 138201 (yielding $\hat{\alpha} = 3241$). Here, layer 1 has a size ranging in $9n \ldots 14n$ bytes for n

router	#prefixes	space (Kb)		random (ms)			synthetic (ms)		
		CDG	ours	CDG	ours	hit-2	CDG	ours	hit-2
aads	32505	3706	1084	*7276*	*5903*	5463	7452	4775	4820
att	121711	2188	1822	12605	7351	15	7872	4941	16
east.attcanada	127561	16418	1661	15096	8429	3220	9164	5450	3116
funet	41328	666	540	*3130*	*2461*	88	5036	2783	67
mae-west	71319	4643	1290	*7217*	*5916*	2385	7425	4565	2401
oregon-01	118190	9897	1596	*7740*	*9933*	11693	7265	6654	10651
oregon-03	142883	9026	2164	14262	9529	3565	8790	6023	3525
pacbell	45184	3170	982	6126	5078	3899	6584	4233	3458
paix	17766	2745	875	*6306*	*5522*	9683	6934	4682	8703
telstra	104096	8896	1490	*8468*	*7544*	3899	7966	5317	3690
telus	126687	11390	1724	14011	8177	2095	8630	5279	2228
west.attcanada	127576	16749	1664	15071	8353	3277	9167	5350	3050
RIPE NCC	138201	5132	1202	10936	5922	1136	6970	4106	1074

Table 4. Space occupancy and time performance of our method versus CDG for the 13 benchmarks described in Section 2.1. Space is measured in kilobytes; time is measured in microseconds. The data for the columns should be read as follows. Columns 1–2 are the benchmark names and their numbers of prefixes. Columns 3–4 report the space occupancy of CDG and our method. Columns 5–6 measure their running time for 100,000 lookups on random traffic. Column 7 (hit-2) counts how many hits our lookups made in layer 2. Columns 8–10 are similar to 5–7 but refer to synthetic traffic. The figures in italic correspond to random data for the experiments in [3, 12].

prefixes. For the sake of comparison, a straightforward storage of these prefixes alone in a routing table would require $6n$ bytes, assuming that each prefix requires a 4-byte word of memory, and its prefix length and its next hop need one further byte each. We also computed statistics for all daily snapshots of 2003 of RIPE NCC (see Section 2.1). The total size of our lookup table (using a hybrid trie for layer 2) is in the range $7n \ldots 16n$, thus confirming the linearity of space even in this case.

3 Performing Lookups

The improved space bounds described in Section 2 makes our scheme more stable to use with respect to CDG. What about lookup time in IPv4? For any given IP address $x = x_1.x_2.x_3.x_4$, we keep the variable $\texttt{lx} = x_1.x_2$ storing the first 16 bits of x and $\texttt{rx} = x_3.x_4$ storing the last 16 bits, so that $x = \texttt{lx.rx}$. We use the right shift operator on `rx` to get byte x_3 and to perform a lookup on $\texttt{lx}.x_3$. If we get the dummy value 255 in layer 1, we also need to perform a lookup in layer 2:

```
#define DUMMY  255
if ( (h1 = hop[ row[lx], rx>>8 ]) != DUMMY )
   return h1;
return lookup_layer2( lx.rx );
```

For example, an IP lookup for $x =$ `192.168.32.125` successfully stops at layer 1 by returning the next hop 3, which is located at $\texttt{hop}\big[\,\texttt{row}[\texttt{192.168}],\ \texttt{32}\,\big]$.

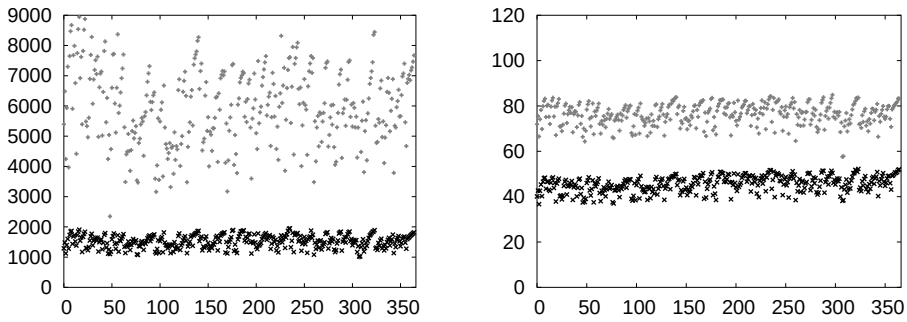

Fig. 3. *Left picture:* Space occupancy of our scheme vs. CDG for RIPE NCC. The x-axis reports the 365 daily snapshots of year 2003 numbered as 0...364; the y-axis the occupied space in kilobytes. *Right picture:* Number of milliseconds (on the y-axis) required by 1 million lookups in CDG (top) and in our scheme (bottom) using synthetic traffic. The x-axis reports the 365 daily snapshots of year 2003 numbered as 0...364.

Instead, x = `192.168.0.27` requires a lookup in layer 2, since it returns the dummy value 255 stored in `hop[ row[192.168], 0 ]`. We measured the running time of our method and of CDG on the daily snapshots of RIPE NCC for the year 2003. We employed the synthetic traffic data for each individual snapshot as explained in Section 2.1. As can be noted in Fig. 3(right), our lookups are definitively faster than those in CDG by 30%. This is consistent with the fact that we reduce CDG's number of memory accesses from 3 to 2. It turns out that the role played by the data structures in layer 2 is rather limited in our data set. We refer the reader to columns hit-2 in the experimental data reported in Table 4, for the number of hits to layer 2 out of 100,000 lookups.

4 Performing Updates

We now describe one of the main effects of our simplification of the lookup scheme. We show how to efficiently handle the updates of the lookup table when announcements (i.e. insertions) and withdrawals (i.e. deletions) of routes arrive on the fly. We do not need to rebuild the lookup table from scratch. We describe how to use our method by assuming that some reasonably efficient method has been adopted for layer 2 (e.g. tries, multi-level hashing, TCAMs, etc.). Again, we base our method on real data analysis to show that the great majority of updates involves layer 1, consistent with what was observed in the middle-class effect. We also make our scheme more robust by providing a good, exact upper bound on the number of entries changed in the lookup table in the worst case.

As described in Section 2.3, we employ `hop` and `row` for layer 1. It is crucial to observe that `hop` is stored in *row-major* order. Since we adopt the maximum number of columns, 256, the only admissible size change in `hop` is to add or remove rows. Performing this change on the columns would result in a disaster,

as the whole `hop` would need to be re-allocated dynamically, which can have a cost analogous to that of rebuilding. Here is why we opt for keeping all the 256 columns. This also guarantees a high level of concurrent access to our lookup table during its lifetime.

We assume (realistically speaking) that the prefixes in route announcements and withdrawals are of length at least 8. (They can be shorter in case of heavy network failures, but then updating the routing table is a minor problem.) We also assume that there are at most 127 distinct next hop values in layer 1. We reserve the most significant bit in each entry of `hop` to mark it as a dummy. (Note that we do not use the dummy value of 255 anymore as in Section 2.3.) Masking this bit yields the correct next hop value.

We performed data analysis on the update traces for RIPE NCC. We collected all the announcements and withdrawals available for year 2003 (see Section 2.1). We discovered that for 336 days the percentage of daily updates involving layer 2 is less then 0.1%, for 360 days that percentage does not exceed the threshold of 0.2% and that the maximum percentage is less then 0.7% This confirms once again the middle-class effect, motivating our choice to build layer 1 on the first 24 bits. We suggest to use a well-tuned trie in layer 2, so that its update cost does not significantly influence the overall performance of announcements and withdrawals in a router.

4.1 Handling announcements and withdrawals

We show how to efficiently process announcements and withdrawals that are produced during the execution of the border gateway protocol (BGP). When an announcements arrives, we have to insert a certain prefix p with its associated prefix length l_p and next hop h_p, into layer 1. Recall that $8 \leq l_p \leq 32$ by our assumptions. We distinguish among three main cases for describing the worst-case effect of this insertion on `row` and `hop`:

1) Case $l_p < 16$: Since $l_p \geq 8$, we have to change no more than 256 entries in `row`. However, each of them could change up to 256 entries in `hop`. The worst case is therefore that of changing $256 + 2^{16}$ entries. In practice, the number of entries is much smaller.

2) Case $16 \leq l_p \leq 24$: This is the most frequent case according to the middle-class effect. We can change one entry of `row` to point from one row of `hop` to another, since the insertion of p needs to change some entries of the row previously pointed in that entry of `row`. We may need to add a new row when none of the existing ones match this change. In the worst case, we change no more than $1 + 256$ entries.

3) Case $l_p > 24$: We can change one entry in `row` and one in `hop`; the latter change may cause the creation of a new row in `hop` as discussed in case 2.

We use the most significant bit to mark the next hop of a truncated prefix so as to avoid that an update falling into cases 1–2 does not propagate to 24-bit prefixes in layer 2 as a side effect. Consequently, we need to make a slight change to lookup, as shown next.

```
#define BITMASK 0x80
#define NO_ROUTE_TO_HOST  0
if ( ! ((h1 = hop[ row[lx], rx>>8 ]) & BITMASK) )
   return h1;
if ( (h2 = lookup_layer2( lx.rx )) != NO_ROUTE_TO_HOST )
   return h2;
return h1 & ~BITMASK;
```

If a lookup in layer 2 returns no-route-to-host, then we must return the next hop value (with its most significant bit cleaned) previously computed in layer 1. Although it may appear that we are harming the performance of the original lookup algorithm in Section 3, we observe that the hit ratio for the first if-statement is very high and determines the real lookup cost, which stays unchanged according to the experimental evaluation discussed in Section 3.

Withdrawals have an effect on `row` and `hop` similar to announcements, except that we have to handle "hidden" prefixes. When we delete a prefix, we should find the "parent" of that prefix and propagate its next hop downward to replace that of the deleted prefix. For example, the withdrawal of route `192.168.0.0/18` from layer 1 in Table 2 causes the propagation of the next hop 2 (associated with `192.168.0.0/17`) to `192.168.0.0/24` (replacing its next hop 3) in layer 2.

As a result we add or remove one row at most in `hop`. Removed rows are linked in a free list that can be reused for adding rows. This does not change the lookup procedure and its cost.

Since the main cost is given by the number of entries changed in `row` and `hop`, we computed statistics to account for this cost, classifying it according to cases 1–3 (both for announcements and withdrawals). We processed the peak of Oct. 25–26, 2003, in router RIPE NCC:

date	#announcements	#withdrawals	case 1	case 2	case 3
10-25-04	20459780	139787	0.68%	99.31%	0.01%
10-26-04	11538757	144937	0.67%	99.30%	0.03%

The above table shows that nearly 99.3% of the updates fall into case 2. Roughly half of them involve a prefix length $l_p = 24$, so they change just one entry in `hop`. Actually, the average number of changed entries in `row` and `hop` is nearly 1. For case 1, the most expensive one, the variance is high for a small number of updates while the rest of updates does not change any row of `hop`. On Oct. 25, 1495 updates changed entries `row` and `hop`; on Oct. 26, there were 1889. These few updates changed between 100 and 1000 entries; we found a single example in which there were 20,985 changed entries, approaching the worst case.

The net result of the case analysis discussed so far is that updates are of bounded cost in layer 1, even in the worst case. This cost scales well with the number of updates and prefixes stored in layer 1.

Fact 2 *In the worst case, the announcement or withdrawal of an IPv4 route changes at most 256 entries in* `row` *and at most* 2^{16} *entries in* `hop` *in case 1. The number of changed entries in* `hop` *becomes* 256 *in cases 2 and 3. In all cases, the empirical average number of changed entries is nearly 1.*

5 Construction of the lookup table

The construction of our table consists in building a trie and then obtaining its DAG. It is worth noting that we insert the prefixes (truncated at 24 bits) into the trie in order of *nondecreasing prefix length*. If we do not follow this pattern, we have to propagate the next hop of the currently inserted prefix downward. In other words, we change the next hop to an already created set of nodes. If we follow the above pattern instead, we have to assign the next hop only to newly created nodes and this can happen once per node. This pattern gives a better performance in the worst case. For our tables, the most time-consuming construction was for oregon-03, in 365 milliseconds. Note that, since we can quickly handle updates, the construction time is less important than in static lookup tables.

References

1. A.V. Aho and N.J.A. Sloane. Some doubly exponential sequences. *Fibonacci Quarterly*, 429–437, 1973.
2. M. Aida and T. Abe. Pseudo-address generation algorithm of packet destinations for internet performance simulation. In *IEEE INFOCOM*, 1425–1433, 2001.
3. A. L. Buchsbaum, G. S. Fowler, B. Kirishnamurthy, K.-P. Vo, and J. Wang. Fast prefix matching of bounded strings. *J. Exp. Algorithmics*, 8:1–3, 2003.
4. P. Crescenzi, L. Dardini, and R. Grossi. IP address lookup made fast and simple. *Proce. 7th Annual European Symposium on Algorithms*, 65–76, 1999.
5. S. Deering and R. Hinden. Internet protocol, version 6 (IPv6). RFC 1883, 1995.
6. S. Dharmapurikar, P. Krishnamurthy, and D. E. Taylor. Longest prefix matching using bloom filters. In *IEEE INFOCOM*, 201–212, 2003.
7. W. Eatherton, G. Varghese, and Z. Dittia. Tree bitmap: hardware/software IP lookups with incremental updates. *SIGCOMM Comput. Commun. Rev.*, 34(2):97–122, 2004.
8. F. Geraci and R. Grossi. Distilling router data analysis for faster and simpler dynamic IP lookup algorithms. Tech. Report TR-05-01, Università di Pisa, January 2005.
9. G. Goodell, W. Aiello, T. Griffin, J. Ioannidis, P. McDaniel, and A. Rubin. Working around BGP: An incremental approach to improving security and accuracy of interdomain routing. *Network and Distributed System Security Symposium*, 2003.
10. C. Labovitz, F. Jahanian, S. Johnson, R. Malan, S.R. Harris, J. Wan, M. Agrawal, D. Zhu, A. Ahuja, and J. Poland. Internet Performance Measurement and Analysis (IPMA) statistics. `http://www.merit.edu/ipma`, 1999.
11. B. Lampson, V. Srinivasan, and G. Varghese. IP lookups using multiway and multicolumn search. *IEEE/ACM Transactions on Networking*, 7(3):324–334, 1999.
12. M. Pellegrini and G. Fusco. Efficient IP table lookup via adaptive stratified trees with selective reconstructions. *12th European Symp. on Algorithms*, 24–35, 2004.
13. J. Postel. Internet protocol. RFC 791, 1981.
14. Network Coordination Centre of the Réseaux IP Européens (RIPE NCC). Routing information service project (Amsterdam router). `http://www.ripe.net/ris/index.html`, 2003.
15. L. Rizzo. Personal communication, 2003.
16. M. A. Ruiz-Sanchez, E. W. Biersack, and W. Dabbous. Survey and taxonomy of IP address lookup algorithms. In *IEEE Network*, 8–23, 2001.