Università di Pisa

Dipartimento di Informatica

Technical Report : TR-99-01

IP Address Lookup Made Fast and Simple

Pierluigi Crescenzi, Leandro Dardini, Roberto Grossi

January 17, 1999

ADDR: Corso Italia 40,56125 Pisa,Italy. TEL: +39-50-887111. FAX: +39-50-887226

IP Address Lookup Made Fast and Simple

Pierluigi Crescenzi and Leandro Dardini
Università degli Studi di Firenze
and
Roberto Grossi
Università degli Studi di Pisa

The IP address lookup problem is one of the major bottlenecks in high performance routers. Previous solutions to this problem first describe it in the general terms of longest prefix matching and, then, are experimented on real routing tables T. In this paper, we follow the opposite direction. We start out from the experimental analysis of real data and, based upon our findings, we provide a new and simple solution to the IP address lookup problem. More precisely, our solution for m-bit IP addresses is a reasonable trade-off between performing a binary search on T with $O(\log |T|)$ accesses, where $|T|$ is the number of entries in T, and executing a single access on a table of 2^m entries obtained by fully expanding T. While the previous results start out from space-efficient data structures and aim at lowering the $O(\log |T|)$ access cost, we start out from the expanded table with 2^m entries and aim at compressing it without an excessive increase in the number of accesses. Our algorithm takes *exactly three* memory accesses and occupies $O(2^{m/2} + |T|^2)$ space in the worst case. Experiments on real routing tables for $m = 32$ show that the space bound is overly pessimistic. Our solution occupies approximately one megabyte for the MaeEast routing table (which has $|T| \approx 44,000$ and requires approximately 250 KB) and, thus, takes three *cache* accesses on any processor with 1 MB of L2 cache. According to the measurement obtained by the VTune tool on a Pentium II processor, each lookup requires 3 additional clock cycles besides the ones needed for the memory accesses. Assuming a clock cycle of 3.33 nanoseconds and an L2 cache latency of 15 nanoseconds, search of MaeEast can be estimated in 55 nanoseconds or, equivalently, our method performs 18 millions of lookups per second.

1. INTRODUCTION

Computer networks are expected to exhibit very high performance in delivering data because of the explosive growth of Internet nodes (from 100,000 computers in 1989 to over 30 millions as of today's). The network bandwidth, which measures the number of bits that can be transmitted in a certain period of time, is thus continuously improved by adding new links and/or by improving the performance of the existing ones. Routers are at the heart of the networks in that they forward packets from input interfaces to output interfaces on the ground of the packets' destination Internet address, which we simply call IP address. They choose which output interface corresponds to a given packet by performing an *IP address lookup* at their routing table. As routers have to deal with an ever increasing number of links whose performance constantly improves, the address lookup is now becoming one of the major bottlenecks in high performance forwarding engines.

The IP address lookup problem was just considered a simple table lookup problem at the beginning of Internet. Now, it is unconceivable to store all existing IP addresses explicitly because, in this case, routing tables would contain millions of entries. In the early 1990s people realized that the amount of routing information would grow enormously, and introduced a simple use of prefixes to reduce space [3]. Specifically, IP protocols use hierarchical addressing, so that a network contains several subnets which in turn contain several host computers. Suppose that all subnets of the network with IP address 128.96.*.* have the same routing information apart from the subnet whose IP address is 128.96.34.*. We can succinctly describe this situation by just two entries (128.96 and 128.96.34) instead of many entries for all possible IP addresses of the network. However, the use of prefixes introduces a new dimension in the IP address lookup problem: For

Pierluigi Crescenzi, Dipartimento di Sistemi e Informatica, Università degli Studi di Firenze, Via C. Lombroso 6/17, 50134 Firenze, Italy. E-mail: piluc@dsi.unifi.it
Leandro Dardini, Dipartimento di Sistemi e Informatica, Università degli Studi di Firenze, Via C. Lombroso 6/17, 50134 Firenze, Italy. E-mail: ldardini@usl4.toscana.it
Roberto Grossi, Dipartimento di Informatica, Università degli Studi di Pisa, Corso Italia 40, 56125 Pisa, Italy. E-mail: grossi@di.unipi.it

Technical Report TR-99-01, Jan. 17, 1999, Università di Pisa

Prefix	Output interface
ϵ	A
10011111	B
10011111 11010101	C
10011111 11010101 00	D
10011111 11110	E

Table 1. An example of routing table

each packet, more than one table entry can match the packet's IP address. In this case, the applied rule consists of choosing the *longest prefix match*, where each prefix is a binary string that has a variable length from 8 to 32 in IPv4 [14]. For example, let us consider Table 1 where ϵ denotes the empty sequence corresponding to the default output interface, and assume that the IP address of the packet to be forwarded is 159.213.37.2, that is, 10011111 11010101 00100101 00000010 in binary. Then, the longest prefix match is obtained with the fourth entry of the table and the packet is forwarded to output interface D. Instead, a packet whose IP address is 159.213.65.15, that is, 10011111 11010101 01000001 00001111 is forwarded to output interface C.

Looking for the longest matching prefix in IP routing tables represents a challenging algorithmic problem since lookups must be answered very quickly. In order to get a bandwidth of, say, 10 gigabits per second with an average packet length equal to 2,000, a router should forward 5 millions of packets per second. It means that each forwarding has to be performed in approximately 200 nanoseconds and, consequently, each lookup must be realized much faster.

1.1 Previous results

Several approaches have been proposed in the last few years in order to solve the IP address lookup problem. Hardware solutions, though very efficient, are expensive and some of them may become outdated quite quickly [6; 10]. In this section, we will briefly review some of the most recent software solutions.

A traditional implementation of routing tables [16] use a version of *Patricia tries*, a very well-known data structure [8]. In this case, it is possible to show that, for tables with n elements, the average number of examined bits is $1.44 \log n$. Thus, for $n = 32000$, this value is bigger than 21. When compared to millions of address lookup requests served in one second, 21 table accesses are too many. Another variation of Patricia tries has been proposed in order to examine k bit each time [13]. However, this variation deals with either the exact matching problem or with the longest prefix matching problem restricted to prefixes whose lengths are multiples of k.

A more recent approach [11] inspired by the three-level data structure of [2], which uses a clever scheme to compress multibit trie nodes using a bitmap, is based on the compression of the routing tables by means of *level compressed tries*, which are a powerful and space efficient representation of binary tries. This approach seems to be very efficient from a memory size point of view but it requires many bit operations which, on the current technology, are time consuming.

Another approach is based on *binary search on hash tables* organized by prefix lengths [17]. This technique is more memory consuming than the previous one but, according to the experimental evaluation presented by the authors, it seems to be very efficient from the time point of view.

A completely different way of using binary search is described in [4]. It is based on multi-way search on the number of possible prefixes rather than the number of possible prefix lengths and exploits the locality inherent in processor caches.

The most recent software solution to the IP address lookup problem is the one based on *controlled prefix expansion* [15]. This approach, together with optimization techniques such as dynamic programming, can be used to improve the speed of most IP lookup algorithms. When applied to trie search, it results into a range of algorithms whose performance can be tuned and that, according to the authors, provide faster search and faster insert/delete times than earlier lookup algorithms.

While the code of the solution based on level compressed tries is public available along with the data used for the experimental evaluation, we could not find an analogous situation for the other approaches (this is probably due to the fact that the works related to some of these techniques have been patented). Thus, the only available experimental data for these approaches are those given by the authors.

1.2 Our results

Traditionally, the proposed solutions to the IP address lookup problem aim at solving it in an efficient way, whichever is the table to be analyzed. In other words, these solutions do not solve the specific IP address lookup problem but the general longest prefix match problem. Subsequently, their practical behavior is experimented on real routing tables. In this paper we go the other way around. We start out from the real data and, as a consequence of the experimental analysis of these data, we provide a simple method whose performance depends on the statistical properties of routing tables T in a way different from the previous approaches. Our solution can also be used to solve the longest prefix match problem but its performance when applied to this more general problem is not guaranteed to be as good as in the case of the original problem.

Any solution for m-bit IP addresses can be seen as a trade-off between performing a binary search on T with $O(\log|T|)$ accesses (where $|T|$ is the number of prefixes in T) and executing a single access on a table of 2^m entries obtained by fully expanding T. The results described in the previous section propose space-efficient data structures and aim at lowering the $O(\log|T|)$ bound on the number of accesses. Our method, instead, starts out from the fully expanded table with 2^m entries and aim at compressing it without an excessive increase in the (constant) number of accesses. For this reason, we call it *an expansion/compression approach*.

We exploit the fact that the relation between the 2^{32} IP addresses and the very few output interfaces of a router is highly redundant for $m = 32$. By expressing this relation by means of strings (thus *expanding* the original routing table), these strings can be *compressed* (using the run-length encoding scheme) in order to provide an implicit representation of the expanded routing table which is memory efficient. More important, this representation allows us to perform an address lookup in exactly *three memory accesses* independently of the IP address. Intuitively, the first two accesses depend on the first and second half of the IP address, respectively, and provide an indirect access to a table whose elements specify the output interfaces corresponding to groups of IP addresses (see Fig. 1).

In our opinion, the approach proposed in this paper is valuable for several reasons.

- *It should work for all routers belonging to Internet.* The analysis of the data has been performed on five databases which are made available by the IPMA project [7] and contain daily snapshots of the routing tables used at some major network access points. The largest database, called MaeEast, is useful to model a large backbone router while the smallest one, called Paix, can be considered a model for an enterprise router (see Table 7).
- *It should be useful for a long period of time.* The data have been collected over a period longer than six months and no significant change in the statistical properties we analyze has ever been encountered (see tables of Sect. 3).
- *Its dominant cost is really given by the number of memory accesses.* As already stated, the method requires *always* three memory accesses per lookup. It does not require anything else but addressing three tables (see Theorem 2.6). For example, on a Pentium II processor, each lookup requires only 3 additional clock cycles besides the ones needed for the memory accesses.
- *It can be easily implemented in hardware.* Our IP address lookup algorithm is very simple and can be implemented by few simple assembly instructions.

The counterpart of all the above advantages is that the theoretical upper bound on the worst-case memory size is $O(2^{m/2} + |T|^2)$ which may become infeasible. However, we experimentally show that in this case theory is quite far from practice and we believe that the characteristics of the routing tables will vary much slower than the rate at which cache memory size will increase.

In order to compare our method against the ones described in the previous section, we refer to data appeared in [15]. In particular, the first five rows of Table 2 are taken from [15][Table 2]. The last two rows reports the experimental data of our method that turns to be the fastest.

As for the methods proposed in [15] based on prefix controlled expansion, we performed the measurements by using the VTune tool [5] to compute the dynamic clock cycle counts. According to these measurements, on a Pentium II processor each lookup requires $3 \cdot clk + 3 \cdot M_D$ nanoseconds where *clk* denotes the clock cycle time and M_D is the memory access delay. If our data structure is small enough to fit into the L2 cache, then $M_D = 15$ nanoseconds, otherwise $M_D = 75$ nanoseconds. The comparison between our method and the best method of [15] is summarized in Table 3 where the first row is taken from Tables 9 and 10 in [15].

Method	Average ns	Worst-case ns	Memory in KB
Patricia trie [16]	1500	2500	3262
6-way search [4]	490	490	950
Search on levels [17]	250	650	1600
Lulea [2]	349	409	160
LC trie [11]	1000	—	700
Ours (experimented)	172	—	960
Ours (estimated)	—	235	960

Table 2. Lookup times for various methods on a 300 MHz Pentium II with 512 KB of L2 cache (values refer to the MaeEast prefix database as on Sept. 12, 1997)

Method	512 KB L2 Cache	1 MB L2 Cache
Controlled prefix expansion [15]	196	181
Ours	235	55

Table 3. MaeEast lookup estimated times depending on cache size ($clk = 3.33$ nanoseconds)

Due to lack of space, we focus on the lookup problem and we do not discuss the insert/delete operations. We only mention here that their realization may assume that the cache is divided into two banks. At any time, one bank is used for the lookup and the other is being updated by the network processor via a personal computer interface bus [12]. A preliminary version of our data structure construction on a 233 MHz Pentium II with 512 KB of L2 cache requires approximately 960 microseconds: We are confident that the code can be substantially improved thus decreasing this estimate.

We now give the details of our solution. The reader must keep in mind that the order of presentation of our ideas follows the traditional one (algorithms + experiments) but the methodology adopted for our study followed the opposite one (experiments + algorithms). Moreover, due to lack of space, we will not give the details of the implementation of our procedures. However, the C code of the implementation is available via anonymous `ftp` (see Sect. 3).

2. THE EXPANSION/COMPRESSION APPROACH

In this section, we describe our approach to solve the IP address lookup problem in terms of m-bit addresses. It runs in two phases, *expansion* and *compression*. In the expansion phase, we implicitly derive the output interfaces for all the 2^m possible addresses. In the compression phase, we fix a value $1 \leq k \leq m$ and find two statistical parameters α_k and β_k related to some combinatorial properties of the items in the routing table at hand. We then show that these two parameters characterize the space occupancy of our solution.

2.1 Preliminaries and notations

Given the binary alphabet $\Sigma = \{0,1\}$, we denote the set of all binary strings of length k by Σ_k, and the set of all binary string of length at most m by $\Sigma_{\leq m} = \cup_{k=0}^{m} \Sigma_k$. Given two strings $\alpha, \beta \in \Sigma_{\leq m}$ of length $k_\alpha = |\alpha|$ and $k_\beta = |\beta|$, respectively, we say that α is a *prefix* of β (of length k_α) if the first $k_\alpha \leq k_\beta$ bits of β are equal to α (e.g., `101110` is a prefix of `1011101101110`). Moreover, we denote by $\alpha \cdot \beta$ the *concatenation* of α and β, that is, the string whose first k_α bits are equal to α and whose last k_β bits are equal to β (e.g., the concatenation of `101110` and `1101110` is `1011101101110`). Finally, given a string α and a subset S of $\Sigma_{\leq m}$, we define $\alpha \cdot S = \{x \mid x = \alpha \cdot \beta \text{ with } \beta \in S\}$.

A *routing table* T relative to m-bit addresses is a sequence of pairs (p, h) where the *route* p is a string in $\Sigma_{\leq m}$ and the *next-hop interface* h is an integer in $[1 \ldots H]$, with H denoting the number of next-hop interfaces[1]. In the following we will denote by $|T|$ the size of T, that is, the number of pairs in the sequence. Moreover, we will assume that T always contains the pair (ϵ, h_ϵ) where ϵ denotes the empty string and h_ϵ corresponds to the *default* next-hop interface.

The *IP address lookup problem* can now be stated as follows. Given a routing table T and $x \in \Sigma_m$, compute the next-hop interface h_x corresponding to address x. That interface is uniquely identified by pair $(p_x, h_x) \in T$ for which (a) p_x is a prefix of x and (b) $|p_x| > |p|$ for any

[1] We are actually using the term "routing table" to denote what is more properly called "forwarding table." Indeed, a routing table contains some additional information.

other pair $(p, h) \in T$, such that p is a prefix of x. In other words, p_x is the longest prefix of x appearing in T. The IP address lookup problem is well defined since T contains the default pair (ϵ, h_ϵ) and so h_x always exists.

2.2 The expansion phase

We now describe formally the intuitive process of extending the routes of T that are shorter than m in all possible ways by preserving the information regarding their corresponding next-hop interfaces. We say that T is in decreasing (respectively, increasing) order if the routes in its pairs are lexicographically sorted in that order. We take T in decreasing order and number its pairs according to their ranks, obtaining a sequence of pairs $T_1, T_2, \dots, T_{|T|}$ such that T_i precedes T_j if and only if $i < j$. As a result, if p_j is a prefix of p_i then T_i precedes T_j. We use this property to suitably expand the pairs.

With each pair $T_i = (p_i, h_i)$ we associate its expansion set, denoted $EXP(T_i)$, to collect all m-bit strings that have p_i as a prefix. Formally, $EXP(T_i) = (p_i \cdot \Sigma_{m-|p_i|}) \times \{h_i\}$, for $1 \le i \le |T|$. We then define the expansion of T on m bits, denoted T', as the union $T' = \cup_{i=1}^{|T|} T'_i$ where the sets T'_i are inductively defined as follows: $T'_1 = EXP(T_1)$, and $T'_i = EXP(T_i) \ominus \cup_{1 \le j < i} T'_j$, where the operator $\ominus$ removes from $EXP(T_i)$ all pairs whose routes already appear in the pairs of $\cup_{1 \le j < i} T'_j$. In this way, we fill the entries of the expanded table T' consistently with the pairs in the routing table T, as stated by the following result.

FACT 2.1. *If $(p_x, h_x) \in T$ is the result of the IP address lookup for any m-bit string x, then $(x, h_x) \in T'$.*

It goes without saying that T' is made up of 2^m pairs and that, if we had enough space, we could solve the IP address lookup problem with a single access to T'. We therefore need to compress T' somehow.

2.3 The compression phase

This phase heavily relies on a parameter k to be fixed later on, where $1 \le k \le m$. We are given the expanded table T' and wish to build three tables `row_index`, `col_index` and `interface` to represent the same information as T' in less space by a simple *run length encoding* (RLE) scheme [9].

We begin by clustering the pairs in T' according to the first k bits of their strings. The cluster corresponding to a string $x \in \Sigma_k$ is $T'_{(x)} = \{(y, h_{xy}) \mid y \in \Sigma_{m-k} \text{ and } (x \cdot y, h_{xy}) \in T'\}$. Note that $|T'_{(x)}| = 2^{m-k}$. We can define our first statistical parameter to denote the number of *distinct* clusters.

DEFINITION 2.2. Given a routing table T with m-bit addresses, the *row k-size* of T for $1 \le k \le m$ is

$$\alpha_k = \left| \left\{ T'_{(x)} \mid x \in \Sigma_k \right\} \right|.$$

Parameter α_k is the first measure of a simple form of compression. Although we expand the prefixes shorter than k, we do not increase the number of *distinct* clusters as the following fact shows.

FACT 2.3. *For a routing table T, let r_k be the number of distinct next-hop interfaces in all the pairs with routes of length at most k, and let n_k be the number of pairs whose routes are* longer *than k, where $1 \le k \le m$. We have $\alpha_k \le r_k + n_k \le |T|$.*

We now describe the compression based upon the RLE scheme. It takes a cluster $T'_{(x)}$ and returns an *RLE sequence* $s_{(x)}$ in two logical steps:

(1) Sort $T'_{(x)}$ in ascending order (so that the strings are lexicographically sorted) and number its pairs according to their ranks, obtaining $T'_{(x)} = \{(y_i, h_i)\}_{1 \le i \le 2^{m-k}}$.

(2) Transform $T'_{(x)}$ into $s_{(x)}$ by replacing each maximal run $(y_i, h_i), (y_{i+1}, h_{i+1}), \dots, (y_j, h_{i+l})$, such that $h_i = h_{i+1} = \cdots = h_{i+l}$, by a pair $\langle h_i, l+1 \rangle$, where $l+1$ is called the *run length* of h_i.

The previous steps encode the 2^{m-k} pairs of strings and interfaces of each cluster $T'_{(x)}$ into a single and (usually) shorter sequence $s_{(x)}$. Note that, by Definition 2.2, α_k is the number of distinct RLE sequences $s_{(x)}$ so produced. We further process them to obtain an equal number of equivalent RLE sequences $s'_{(x)}$. The main goal of this step is to obtain sequences such that, for any i, the i-th pair of any two such sequences have the same run length value. We show how to do it by means of an auxiliary function $\varphi(s,t)$ defined on two nonempty RLE sequences $s = \langle a, f\rangle \cdot s_1$ and $t = \langle b, g\rangle \cdot t_1$ as follows:

$$\varphi(s,t) = \begin{cases} t & \text{if } s_1 = t_1 = \epsilon, \\ \langle b, f\rangle \cdot \varphi(s_1, t_1) & \text{if } f = g \text{ and } s_1, t_1 \neq \epsilon, \\ \langle b, f\rangle \cdot \varphi\big(s_1, \langle b, g - f\rangle \cdot t_1\big) & \text{if } f < g \text{ and } s_1, t_1 \neq \epsilon, \\ \langle b, g\rangle \cdot \varphi\big(\langle a, f - g\rangle \cdot s_1, t_1\big) & \text{if } f > g \text{ and } s_1, t_1 \neq \epsilon. \end{cases}$$

The purpose of φ is to "unify" the run lengths of two RLE sequences by splitting some pairs $\langle b, f\rangle$ into $\langle b, f_1\rangle, \ldots, \langle b, f_r\rangle$, such that $f = f_1 + \cdots + f_r$. The unification defined by φ is a variant of standard merge, except that it is not commutative as it only returns the (split) pairs in the second RLE sequence.

In order to apply unification to a set of RLE sequences $s_1, s_2, \ldots, s_q$, we define function $\Phi(s_1, \ldots, s_q)$ that returns RLE sequences $s'_1, s'_2, \ldots, s'_q$ as follows. First,

$$s'_q = \varphi(\varphi(\ldots\varphi(\varphi(\varphi(s_1, s_2), s_3), s_4)\ldots, s_{q-1}), s_q).$$

As a result of this step, we obtain that the run lengths in s'_q are those common to all the input sequences. Then, $s'_i = \varphi(s'_q, s_i)$ for $i < q$: in this way, the pairs of the set of RLE sequences $s_1, s_2, \ldots, s_q$ are *equally* split.

We are now ready to define the second statistical parameter. Given an RLE sequence s, let its *length* $|s|$ be the number of pairs in it. Regarding the routing table T, let us take the α_k RLE sequences $s_1, s_2, \ldots, s_{\alpha_k}$ obtained by the distinct clusters $T'_{(x)}$ with $x \in \Sigma_k$, and apply the unification $\Phi(s_1, s_2, \ldots, s_{\alpha_k})$ defined above.

DEFINITION 2.4. Given a routing table T with m-bit addresses, the *column k-size* β_k of T, for $1 \leq k \leq m$, is the (equal) length of the RLE sequences resulting from $\Phi(s_1, s_2, \ldots, s_{\alpha_k})$. That is,

$$\beta_k = |s'_{\alpha_k}|.$$

Although we increase the length of the original RLE sequences, we have that β_k is still linear in $|T|$.

FACT 2.5. *For $1 \leq k \leq m$, $\beta_k \leq 3|T|$.*

2.4 Putting all together

We can, finally, prove our result on storing a routing table T in a sufficiently compact way to guarantee always a constant number of accesses. We let $\#bytes(n)$ denote the number of bytes necessary to store a positive integer n, and a *word* be sufficiently large to hold $\max\{\log|T| + 1, \log H\}$ bits.

THEOREM 2.6. *Given a routing table T with m-bit addresses and H next-hop interfaces, we can store T into three tables* `row_index`, `col_index` *and* `interface` *of total size $2^k \cdot \#bytes(\alpha_k) + 2^{m-k} \cdot \#bytes(\beta_k) + \alpha_k \cdot \beta_k \cdot \#bytes(H)$ bytes or $O\big(2^k + 2^{m-k} + |T|^2\big)$ words, for $1 \leq k \leq m$, so that an IP address lookup for an m-string x takes* exactly three *accesses given by*

$$h_x = \texttt{interface}\big[\texttt{row\_index}\big[x[1\ldots k]\big], \texttt{col\_index}\big[x[k+1\ldots m]\big]\big]$$

where $x[i\ldots j]$ denotes the substring of x starting from the i-th bit and ending at the j-th bit.

PROOF. We first find the α_k distinct clusters (without their explicit construction) and produce the corresponding RLE sequences, which we unify by applying Φ. The resulting RLE sequences, of length β_k, are numbered from 1 to α_k. At this point, we store the β_k next-hop values of the j-th sequence in row j of table `interface`, which hence has α_k rows and β_k columns. We then set $\texttt{row\_index}[x[1\ldots k]] = j$ for each string x such that cluster $T_{(x[1\ldots k])}$ has been encoded by the j-th RLE sequence. Finally, let $f_1, \ldots, f_{\beta_k}$ be the run lengths in any such sequence. We set $\texttt{col\_index}[x[k+1\ldots m]] = \ell$ for each string x, such that $x[k+1\ldots m]$ has rank q in Σ_{m-k} and $\sum_{l=1}^{\ell-1} f_l < q \leq \sum_{l=1}^{\ell} f_l$. The space and memory access bounds immediately follow. $\square$

We wish to point out that the previous theorem represents a reasonable trade-off between performing a binary search on T with $O(\log |T|)$ accesses and executing a single access on a table of 2^m entries obtained by T.

2.5 A small example

In this section we describe a simple example of the application of our expansion/compression procedure to a routing table. In particular, let us consider the routing table T shown in Table 4 where $m = 4$ and $H = 3$.

Route	Next-hop
ϵ	A
10	B
001	B
01	C
0010	C
00	A

Table 4. An example of routing table T

The first step of the expansion phase consists of sorting T in decreasing order. The result of this step is shown in Table 5.

Route	Next-hop
10	B
01	C
0010	C
001	B
00	A
ϵ	C

Table 5. The sorted routing table T

This table is now expanded as shown in Table 6: the third column of this table shows the corresponding sub-tables T'_i according to the definition of the expanded table T'.

Let us now enter the compression phase: to this aim, we choose $k = 2$. We then have the following four clusters and the corresponding RLE sequences (shown between square brackets):

$$\begin{aligned}
T_{(00)} &= \{(00, A), (01, A), (10, C), (11, B)\} && [s_1 = \langle A, 2\rangle\langle C, 1\rangle\langle B, 1\rangle] \\
T_{(01)} &= \{(00, C), (01, C), (10, C), (11, C)\} && [s_2 = \langle C, 4\rangle] \\
T_{(10)} &= \{(00, B), (01, B), (10, B), (11, B)\} && [s_3 = \langle B, 4\rangle] \\
T_{(11)} &= \{(00, C), (01, C), (10, C), (11, C)\} && [s_4 = s_2]
\end{aligned}$$

Route	Next-hop	
1000	B	T'_1
1001	B	
1010	B	
1011	B	
0100	C	T'_2
0101	C	
0110	C	
0111	C	
0010	C	T'_3
0011	B	T'_4
0000	A	T'_5
0001	A	
1100	C	T'_6
1101	C	
1110	C	
1111	C	

Table 6. The expanded routing table T'

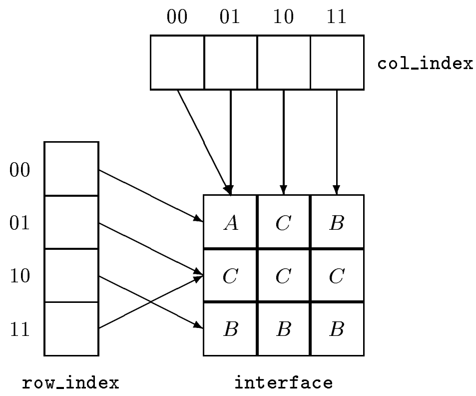

Fig. 1. The overall IP address lookup data structures

In this case $\alpha_k = 3$ (since $T_{(01)} = T_{(11)}$), $r_k = 3$ and $n_k = 2$. We now unify the RLE sequences by applying function φ. In particular, we have that

$$s'_3 = \varphi(\varphi(s_1, s_2), s_3) = \varphi(\langle C, 2\rangle\langle C, 1\rangle\langle C, 1\rangle, s_3) = \langle B, 2\rangle\langle B, 1\rangle\langle B, 1\rangle$$

and

$$\begin{aligned} s'_2 &= \varphi(s'_3, s_2) = \langle C, 2\rangle\langle C, 1\rangle\langle C, 1\rangle \\ s'_1 &= \varphi(s'_3, s_1) = \langle A, 2\rangle\langle C, 1\rangle\langle B, 1\rangle = s_1. \end{aligned}$$

Clearly, $\beta_k = 3$. Thus, $\#bytes(\alpha_k) = \#bytes(\beta_k) = \#bytes(H) = 1$ so that the memory occupancy is equal to 4+4+9=17 bytes (observe that the original table T occupies 6+6=12 bytes). In summary, the overall data structures is shown in Fig. 1.

3. DATA ANALYSIS AND EXPERIMENTAL RESULTS

In this section, we show that the space bound stated by Theorem 2.6 is practically feasible for real routing tables for Internet. In particular, we have analyzed the data of five prefix databases which are made available by the IPMA project [7]: these data are daily snapshots of the routing tables used at some major network access points. The largest database, called MaeEast, is useful to model a large backbone router while the smallest one, called Paix, can be considered a model for an enterprise router. Table 7 shows the sizes of these routing tables on Jul. 7, 1998 and on Jan. 11, 1999; moreover, the third column indicates the minimum/maximum size registered between the previous two dates while the fourth column indicates the number H of next-hop interfaces.

Router	**Jul. 7, 1998**	**Jan. 11, 1999**	**Min/Max**	H
MaeEast	41231	43524	37134/44024	62
MaeWest	18995	23411	17906/23489	62
Aads	23755	24050	18354/24952	34
PacBell	22416	22849	21074/23273	2
Paix	3106	5935	1519/5935	21

Table 7. The sizes of the routing tables

Besides suggesting the approach described in the previous section, the data of these databases have also been used to choose the appropriate value of k (that is, the way of splitting an IP address). Table 8 shows the percentages of routes of length 16 and 24 on Jul. 7, 1998 (the third column denotes the most frequent among the remaining prefix lengths): as it can be seen from the table, these two lengths are the two most frequent in all observed routing tables[2]. If we also

[2]In other words, even though it is now allowed to use prefixes of any length, it seems that network managers are still using the original approach of using multiples of 8 according to the class categorization of IP addresses.

consider that choosing k multiple of 8 is more adequate to the current technology in which bit operations are still time consuming, the table's data suggests to use either $k = 16$ or $k = 24$. *We choose* $k = 16$ to balance the size of the tables `row_index` and `col_index` (see Theorem 2.6).

Router	**Length 16**	**Length 24**	**Next**
MaeEast	13%	56%	8% (23 bits)
MaeWest	14%	53%	7% (23 bits)
Aads	25%	54%	8% (23 bits)
PacBell	13%	56%	7% (23 bits)
Paix	12%	53%	8% (23 bits)

Table 8. The percentages of prefixes of length 16 and 24 on Jul. 7, 1998

It now remains to estimate the value of the two statistical parameters α_k and β_k. Table 9 shows these values measured on Jul. 7, 1998 and on Jan. 11, 1999; moreover, the third column shows the maximum values registered between the previous two dates. From the table's data and from the fourth column of Table 7 it is then possible to argue that the $\#bytes(\alpha_k) = \#bytes(\beta_k) = 2$ and $\#bytes(H) = 1$, so that, according to the bound of Theorem 2.6, the memory occupancy of our algorithm is equal to $M_1 = 2^{16} \cdot 2 + 2^{16} \cdot 2 + \alpha_k \cdot \beta_k$ bytes. Actually, the above memory size can be slightly increased in order to make faster the access to the table `interface`: to this aim, the elements of table `row_index` (respectively, `column_index`) can be seen as memory pointers (respectively, offsets) instead of as indexes. In this way, the actual value of $\#bytes(\alpha_k)$ becomes 4 and the corresponding memory occupancy increases to $M_2 = 2^{16} \cdot 4 + 2^{16} \cdot 2 + \alpha_k \cdot \beta_k$ bytes.

Router	**Jul. 7, 1998** (α_k/β_k)	**Jan. 11, 1999** (α_k/β_k)	**Maximum** (α_k/β_k)
MaeEast	2577/277	2745/299	2821/299
MaeWest	2017/263	2335/268	2335/285
Aads	1903/259	2100/269	2140/273
PacBell	1399/256	1500/256	1500/260
Paix	722/256	984/261	989/261

Table 9. The values of α_k and β_k

According to the above two formulae, we have then the real memory occupancy of our algorithm. These values are shown in Table 10. The data relative to MaeEast seem to show that we need a little bit more than 1 Megabyte of cache memory. However, this is not true in practice: indeed, only a small fraction (approximately, 1/5) of table `row_index` is filled with used values from actual IP addresses. Since only these values are loaded in the cache memory, this implies that the real cache memory occupancy is much smaller than the one shown in Table 10.

Router	**Jul. 7, 1998** (M_1/M_2)	**Jan. 11, 1999** (M_1/M_2)	**Maximum** (M_1/M_2)
MaeEast	975973/1107045	1082899/1213971	1082899/1213971
MaeWest	792615/923687	887924/1018996	887924/1018996
Aads	755021/886093	827044/958116	837804/968876
PacBell	620288/751360	646144/777216	646424/777496
Paix	446976/578048	518968/650040	518968/650040

Table 10. The memory occupancies according to the two estimates

We have implemented our algorithm in ANSI C and we have compiled it by using the GCC compiler under the Linux operating system and by using the Microsoft Visual C compiler under the Windows 95 operating system. The C source and the complete data-tables of our analysis are available at the URL `http://www.dsi.unifi.it/~piluc/IPLookup/index.html`, where we show that the generated code is higly tuned and very few can be done to improve the performance of our lookup process.

4. CONCLUSIONS

In this paper, we have proposed a new method for solving the IP address lookup problem which originated from the analysis of the real data. This analysis may lead to different solutions. For example, we have observed that a well-known binary search approach (based on secondary memory access techniques) requires on the average a little bit more than one memory access for address lookup. However, this approach does not always perform better than the expansion/compression technique presented in this paper: the reason for this is that the implementation of the binary search requires too many arithmetic and branch operations which become the real complexity of the algorithm. Nevertheless, there exist computer architectures for which this new solution performs better: actually, it is our intention to realize a serious comparison of the two approaches (comparison that fits into the recently emerging area of algorithm engineering).

Theorem 2.6 can be generalized in order to obtain, for any integer $r > 1$, a space bound $O(r2^{m/r} + |T|^r)$ and $r + 1$ memory accesses. This could be the right way to extend our approach to the coming IPv6 protocol family [1]. It would be interesting to find a method for unifying the RLE sequences, so that the final space is provably less than the pessimistic $O(|T|^r)$ one.

Finally, we believe that our approach arises several interesting theoretical questions. For example, is it possible to derive combinatorial properties of the routing tables in order to give better bounds on the values of α_k and β_k? What is the lower bound on space occupancy in order to achieve a (small) constant number of memory accesses? What about compression schemes other than RLE?

REFERENCES

[1] Deering, S., and Hinden, R. Internet protocol, version 6 (IPv6). RFC 1883 (`http://www.merit.edu/internet/documents/rfc/rfc1883.txt`), 1995.

[2] Degermark, M., Brodnik, A., Carlsson, S., and Pink, S. Small forwarding tables for fast routing lookups. *ACM Computer Communication Review 27*, 4 (October 1997), 3–14.

[3] Fuller, V., Li, T., Yu, J., and Varadhan, K. Classless inter-domain routing (CIDR): an address assignment and aggregation strategy. RFC 1519 (`http://www.merit.edu/internet/documents/rfc/rfc1519.txt`), 1993.

[4] Lampson, B., Srinivasan, V., and Varghese, G. IP lookups using multi-way and multicolumn search. In *ACM INFOCOM Conference* (April 1998).

[5] Intel Corporation. Vtune. `http://developer.intel.com/design/perftool/vtune/`, 1997.

[6] McAuley, A., Tsuchiya, P., and Wilson, D. Fast multilevel hierarchical routing table using content-addressable memory. U.S. Patent Serial Number 034444, 1995.

[7] Merit. IPMA statistics. `http://nic.merit.edu/ipma`, 1998.

[8] Morrison, D. PATRICIA – Practical Algorithm To Retrieve Information Coded In Alfanumeric. *Journal of ACM 15*, 4 (October 1968), 514–534.

[9] Nelson, M. *The Data Compression Book*. M&T Books, San Mateo, California, 1992.

[10] Newman, P., Minshall, G., Lyon, T., and Huston, L. IP switching and gigabit routers. *IEEE Communications Magazine* (January 1997).

[11] Nilsson, S., and Karlsson, G. Fast address look-up for internet routers. In *ALEX* (February 1998), Università di Trento, pp. 9–18.

[12] Partridge, C. A 50-Gb/s IP router. *IEEE/ACM Transactions on Networking 6*, 3 (June 1998), 237–247.

[13] Pei, T.-B., and Zukowski, C. Putting routing tables into silicon. *IEEE Network* (January 1992), 42–50.

[14] Postel, J. Internet protocol. RFC 791 (`http://www.merit.edu/internet/documents/rfc/rfc0791.txt`), 1981.

[15] Srinivasan, V., and Varghese, G. Fast address lookups using controlled prefix expansion. In *ACM SIGMETRICS Conference* (September 1998). Full version to appear in *ACM TOCS*.

[16] Stevens, W., and Wright, G. *TCP/IP Illustrated, Volume 2 The Implementation*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1995.

[17] Waldvogel, M., Varghese, G., Turner, J., and Plattner, B. Scalable high speed IP routing lookups. *ACM Computer Communication Review 27*, 4 (October 1997), 25–36.