Experimental Evaluations of Algorithms for IP Table Minimization

Angelo Fanelli, Michele Flammini, Domenico Mango,
Giovanna Melideo, and Luca Moscardelli

Dipartimento di Informatica, Università di L'Aquila,
Via Vetoio Loc. Coppito, 67010 L'aquila, Italy
{angelo.fanelli,flammini,mango,melideo,moscardelli}@di.univaq.it

Abstract. The continuous growth of the routing tables sizes in backbone routers is one of the most compelling scaling problems affecting the Internet and has originated considerable research in the design of compacting techniques. Various algorithms have been proposed in the literature both for a single and for multiple tables, also with the possibility of performing address reassignments [1,5].

In this paper we first present two new heuristics, the BFM and its evolution called BFM-Cluster, that exploit address reassignments for the minimization of $n > 1$ routing tables, and their performances are experimentally evaluated together with the already existing techniques. Since a main problem posed by the growth of the routing tables sizes is the consequent general increase of the table lookup time during the routing of the IP packets, the aim is twofold: (i) to measure and compare the compression ratios of the different techniques and (ii) to estimate the effects of the compression on the lookup times by measuring the induced improvement on the time of the main algorithms and data structures for the fast IP address lookup from the original tables to the compressed ones. Our point is that the existing methods are efficient in different situations, with BFM-Cluster heuristic outperforming all other ones.

Keywords: Routing, IP protocol, compression, lookup times, optimal and approximation algorithms.

1 Introduction and Motivations

In the Internet Protocol (IP) communications between hosts are possible by means of interconnected forwarding elements called routers. Each router consists of input interfaces, output interfaces, a forwarding engine and a routing table. Exchanged messages are arranged into packets or datagrams. Unlike the circuit-switched networks, every packet travels across the network independently of all others. This implies that each packet is labelled with both a globally unique source and a destination address, which it must carry along. In a router, the bottleneck in packet forwarding is to lookup the destination address of an incoming

C. Demetrescu (Ed.): WEA 2007, LNCS 4525, pp. 324–337, 2007.

packet in the routing database, that is to determine the output interface through which the packet must be forwarded.

Unfortunately, the huge and disorganized growth of the Internet during the last years has caused an excessive increase in the number of entries of the routing tables. Beside their memory requirements, the main problem posed by this phenomena is the consequent general increase of the table lookup time during the routing of IP packets. Thus, a considerable research effort has been devoted in the design of techniques for reducing the size of IP routing tables.

In [5] the authors presented an optimal polynomial time algorithm (*Optimal Routing Table Constructor* - ORTC) for constructing a routing table that has the least possible number of entries, while still providing the same routing information. Moreover, they experimentally evaluted ORTC by showing that it reduces the number of prefixes by around 40%.

The envisaged close enhancement of the IP protocol to version 6 urgently requires the solution of the IP routing tables minimization problem in a new and more effective way, that is by performing address reassignments. Such a facility can actually be exploited also inside the current version of the protocol, thanks to the introduction of the so called *Network Address Translators*, NATs for short, by which independent address reassignments are possible inside subnetworks [6]. In this scenario, efficient tables minimization algorithms exploiting addresses reassignments are of crucial importance. Motivated by the above discussion and by exploiting address reassignments, in [1] the authors provided a new polynomial time optimal algorithm, called BF, that minimizes the size of single routing tables and a $3h$-approximation algorithm (h is the length of the IP addresses), called BF-Multi, that minimizes the sum of the sizes of $n > 1$ routing tables.

Starting from the above results, in this paper we first introduce two new heuristics, the BFM heuristic and its evolution called BFM-Cluster, that exploit address reassignments for the minimization of $n > 1$ routing tables, and then we focus on the experimental evaluation of the above mentioned techniques for IP tables minimization (i.e., ORTC, BF, BF-Multi, BFM and BFM-Cluster). Since a main problem posed by the growth of the routing tables sizes is the consequent general increase of the table lookup time during the routing of the IP packets, the aim is twofold: (i) to measure and compare the compression ratios of these techniques and (ii) to estimate the effects of the compression on the lookup times by measuring the induced improvement of the time of the main algorithms and data structures for the IP address lookup (e.g., Binary Search on prefix Length [18,19], Multi-ary Tries [15,16] and LC-Trie [14]).

The paper is organized as follows. In the next section we present the stat-of-art in IP address tables minimization; in Section 3 we introduce our new heuristics. Section 4 is devoted to the implementation details such as software, hardware, requirements and metrics. Section 5 illustrates our contributions on the comparison of the compression ratios of the table minimization algorithms and on the effect of tables minimization on the IP lookup data structures. Section 6 concludes by analyzing experimental results and discussing directions for

future work. We apologize for the many omissions and missing details, but space constraints imposed several limitations.

2 Preliminaries

In the IP protocol each host is assigned a unique address of $h = 32$ bits, with h presumably higher in future versions. Routers consist of input interfaces, output interfaces, a forwarding engine and a routing table. Each routing table is a list of entries, where each entry e consists of three different fields: a network mask $mask_e$, a destination network address $dest_e$ and a next-hop address $next_e$. Field $mask_e$ consists of a h bit binary string of the form $1^{l_e}0^{h-l_e}$, where l_e is called the length of the mask. The network address $dest_e$ is a h bit binary string representing the IP address associated to the entry, and the next-hop address $next_e$ identifies the index of the output interface corresponding to the entry.

A given IP address matches entry e if its leading l_e bits are coincident with the respective l_e ones of $dest_e$. Consider two entries e and f such that $l_e < l_f$ and the leading l_e bits of $dest_f$ exactly matching the ones of $dest_e$. It is easy to see that any IP address matching entry f matches also entry e. We then say that there is an *inclusion* of entry f in entry e.

Since an IP address can match different entries, routing is performed on a longest mask matching entry base, that is the output interface chosen to forward a packet is taken from the matching entry having the longest mask. The efficient solution of such a problem, called *Longest Matching Prefix*, has given rise to different algorithms, some of which are briefly presented in the following subsections, together with an overview of the minimization ones implemented for the experimentation. Due to space limitations their presentation is necessarily incomplete, with many details just mentioned and/or left unspecified. However, the interested reader can refer to the given literature for a more detailed description.

2.1 IP Lookup Algorithms

Many fast route lookup algorithms have been proposed in the last few years [2,3,4,8,9,11,14,16,19]. Based on the data structure used, these algorithms can be classified into one of the following three categories: trie-based, comparison-based and hash-based. The trie-based algorithms use the traditional key search idea and organize the routing table into a tree-like data structure [7]. Each node in the trie has zero or more child nodes. Each lookup of a key starts at the root of the trie and then walks down to find the longest match. The idea underlying comparison-based algorithms is mainly related to the binary search scheme. Modifications have been devised to accommodate the prefix into sorted arrays. In hash-based algorithms, the results of hashing functions are used as indexes into memory. The perfect hash function completes the lookups in only one memory access that can achieve the highest searching speed.

The particular algorithms used in this paper to evaluate the effect of table minimization on their lookup times are among the ones with the best performances:

Binary Search on prefix Length [18,19], which combines comparison-based and hash-based techniques, and Multi-ary Tries [15,16] and Level Compressed (LC) Trie [14], both belonging to trie-based category.

2.2 Table Minimization Without Address Reassignment

The Optimal Routing Table Constructor (ORTC) is an algorithm given in [5] which reduces the number of entries in a routing table.

Given a routing table that provides forwarding information for IP addresses using longest prefix matching, ORTC produces a new routing table that has the *same forwarding behavior* and *the least possible number of entries*.

A binary tree representation is used to graphically depict a set of prefixes. Each successive bit in a prefix corresponds to a link to a child node in the tree, with a 0 corresponding to the left child and a 1 corresponding to the right child. Note that the binary tree generally contains more nodes than prefixes, since every successive bit in the prefix produces a node. The nodes are labeled with next-hop information, typically a small integer or a set of small integers. Roughly speaking, ORTC optimizes a routing table using three steps over the binary tree representation. The first step propagates routing information down to the trees leaves. The second step finds the most prevalent next hops, by propagating information (sets of next hops) from the leaves back towards the root. In fact, shorter prefixes close to the root of the tree should route to the most popular or prevalent next hops. Finally, a third step moves down the tree, choosing a next hop from the set of possibilities for a prefix and eliminating redundant routes.

The space and time complexity of ORTC algorithm are $O(hN)$, where N is the number of entries in the input routing table.

2.3 Table Minimization with Address Reassignment

In [1] the authors propose algorithms for the minimization of routing tables using address reassignments. This approach differs substantially from the ORTC's one, where the hosts must maintain their original addresses, and allows to improve the effect of minimization by assigning IP addresses so as to obtain the maximum possible compression.

An optimal polynomial time algorithm (called *BF*) was presented in [1] for the case of single routing tables.

Let us briefly describe the underlying idea.

Denoted as a_i the number of hosts reached through the i-th output interface, if inclusions between entries are not allowed, then the set of the addresses matching any entry e in the table has cardinality 2^{h-l_e} and consequently the minimum number of entries corresponding to the i-th output interface is at least equal to the minimum number of powers of 2 whose sum is equal a_i. This clearly corresponds to the number of bits equal to 1 in the $h+1$ bit binary string that encodes a_i. Let $one(a_i)$ denote such a number. Then the minimum size of the table is at least $\sum_{i=1}^{\delta} one(a_i)$, where δ is the number of output interfaces. Moreover, it is not difficult to show that such a number of entries is always achievable.

Allowing the inclusion of one entry f in one entry e with $next_e \neq next_f$ and $l_e < l_f$, modifies the number of addresses matching the entries corresponding to the output interface $next_e$ from a_{next_e} to $a_{next_e} + 2^{h-l_f}$. As a consequence, the number of entries corresponding to the output interface $next_e$ becomes $one(a_{next_e} + 2^{h-l_f})$. Concerning the effect of such an inclusion on the entries of the output interface $next_f$, it is possible to charge a cost of one to the inclusion to keep track of the fact that such an entry will be effectively realized inside e, while the number of addresses matching the remaining entries of $next_f$ becomes $a_{next_f} - 2^{h-l_f}$. Exploiting such ideas, algorithm BF constructs a table of minimum size in time $O(h\delta)$.

Unfortunately, in the general case in which we are interested in minimizing the sum of the sizes of $n > 1$ tables[1], as shown in [1] this problem is *NP-hard*, but there exists a $3h$-approximation algorithm, called BF-Multi, that exploits a matrix representation of the instances of the problem. In fact, the routing behavior of any router r_j can be represented by means of a boolean matrix A^j in which each row is associated to a destination host and each column to one output interface of r_j. Let A be the *global matrix* given by the horizontal concatenation of all the matrices A^j. Let us define a *segment* in a given column of A as a maximal vertical sequence of consecutive entries equal to 1 in the column. The approximation algorithm is based on the idea that any permutation π of the rows of A corresponds in a natural way to an assignment of addresses to the hosts. Namely, the host corresponding to row i after the permutation receives the IP address given by the h bit binary string encoding $i - 1$. Since each segment corresponds to a limited number of entries in the IP routing table, a permutation causing a low number of segments in A yields also a solution with a low overall number of table entries. One of such permutations is than determined by reducing the problem to *minimum metrical TSP*.

3 The New Heuristics

In this section we introduce new heuristics for the problem of minimizing the sum of the sizes of $n > 1$ tables with address reassignment.

We emphasize that in both heuristics we focus on the address reassignment, i.e. we care about assigning addresses to host such that the total size of the routing tables can be minimized. On the other hand, we discard optimization issues relative to the minimization of each of the final routing tables obtained after the address reassignment; this is due to the fact that, after the addresses have been reassigned, such a problem is equivalent to the one of minimizing a routing table without address reassignment, which is optimally solved by the *ORTC* algorithm.

All the details of both heuristics will be shown in the full version of the paper.

[1] Notice that without address reassignment the problem can be trivially solved by independently applying ORTC to each single table.

3.1 BFM Heuristic

The main idea of the *BFM* heuristic is to construct a new virtual router $\bar{r}$ starting from the n routers as follows. For each host x we compute a n-tuple $\mathcal{U}_x = (out_x^1, \ldots, out_x^n)$ containing the n next hop interfaces associated to the host in each router, i.e. out_x^i is the next hop interface associated to x in router r_i. Let $\mathcal{U}$ be the set of all the obtained tuples. For each $u \in \mathcal{U}$, let H_u be the set of hosts corresponding to the n-tuple u. We run the BF algorithm on the new virtual router $\bar{r}$, in which each n-tuple u represents a virtual output interface with $|H_u|$ associated hosts. Finally, starting from the address assignment determined by BF, we construct each of the n final tables by selecting the corresponding next hop interfaces from the virtual next hops, i.e. from the n-tuples.

3.2 BFM-Cluster(k) Heuristic

This heuristic is an evolution of the BFM one just described and as an input parameter k, whose tuning is discussed in subsection 5.2. Since the size of the tables compressed by BF is usually very close to the number of output interfaces, in the BFM-Cluster heuristic we try to reduce the number of virtual output interfaces by clustering the tuples with a low number of not coinciding coordinates. In particular, when constructing the virtual router $\bar{r}$, we partition the set of tuples in clusters $c_1, c_2, \ldots$ such that tuples with a low fixed number of different components are in the same cluster, and finally, for each cluster c_i, we add to $\bar{r}$ a virtual output interface with a number of associated hosts equal to the sum of the numbers of hosts associated to each tuple in c_i. More specifically, the partition process is as described in the following. First of all, let us define tuples *d-close* if the number of their different components is at most d. We maintain an (initially empty) set $\bar{\mathcal{U}}$ of *leader* tuples, and analyze one by one all the tuples in $\mathcal{U}$: for each $u \in \mathcal{U}$, if u is k-close to a leader tuple $\bar{u} \in \bar{\mathcal{U}}$, we add u to the cluster whose leader is $\bar{u}$, otherwise we create a new cluster having u as leader. Finally, as in the BFM heuristic, we have to construct each of the n final tables. To this aim we first partition the set of addresses assigned to each cluster c_i between the tuples belonging to c_i, and then we proceed by selecting from the tuples the next hop interfaces relative to each table.

4 Methods Testing

We implemented the techniques described in the Section 2 in the C language. Overall, the developed C code consists of about 3000 lines. Since we are interested in the compression ratio and in the relative improvement of the lookup time, we performed the experimentation on a personal computer with a 2-GHz Pentium 4 processor and 256 MB of RAM running Windows XP.

4.1 Test Data

The techniques are tested on 15 existing routing tables, downloaded from the routing table snapshots provided by the IPMA Project [13,17] and from the

route server lists provided in [10,12], as well as on 56 artificially generated tables by independently modifying the next hop of each entry of a starting original table with a fixed probability between 10% and 30%.

For the minimization techniques, we considered 13 input instances composed with existing routing tables, and 7 instances composed with artificially generated ones. More precisely, for every set of tables we have run the implemented compression techniques and then constructed the auxiliary data structures both for the initial and compressed tables. In order to test the improvement times of the IP address lookup algorithms, each technique has been tested against three different traffic files, each containing $15 \cdot 10^6$ IP addresses on the original tables and on the compressed ones. Overall, we performed about 3000 different tests. More precisely, one traffic file is obtained as a permutation and repetition of the IP addresses originating (i.e., matching at least one entry) from the considered routing tables, and the other two traffic files contain random IP addresses.

Table 1 describes the existing routing tables features (i.e., the number of the entries in the routing table, the number of distinct next-hops found in the table and the date of tables snapshots), and the sets of the tables[2] considered in our tests. In order to have a more compact and readable presentation, we omit the description of the artificially generated tables and the one of the sets of these tables.

Table 1. Existing routing tables and their sets used as input instances

Tables					Sets												
ID	Table	Date	# Entries	# Next hop	1	2	3	4	5	6	7	8	9	10	11	12	13
1	utah.rep.net	02-11-04	153	6		x			x		x	x	x				
2	mae-west	24-08-97	15050	57	x			x				x	x	x	x	x	x
3	aasd	24-08-97	20328	19	x							x	x				
4	pb	24-08-97	20637	3	x							x	x				
5	mae-east	24-08-97	38470	59	x			x				x	x	x	x	x	x
6	funet	19-11-97	41709	20	x							x	x	x	x	x	x
7	as5388	02-11-04	62531	112		x		x	x		x	x	x	x	x	x	x
8	wcg.net	02-11-04	121800	898		x			x		x	x	x	x	x	x	x
9	ip.att	02-11-04	145682	23		x			x	x	x	x	x				
10	he	02-11-04	140112	196		x			x	x	x	x	x				x
11	ip.tiscali	02-11-04	145648	1			x		x	x	x	x					
12	opentransit	02-11-04	153317	15			x	x	x	x	x	x	x		x	x	
13	gbls	02-11-04	154740	303			x	x		x	x	x	x	x	x	x	x
14	oregon-ix	02-11-04	161635	53			x			x	x	x	x			x	
15	colt.net	02-11-04	162008	1			x			x	x	x					

[2] Unfortunately, we haven't current snapshots for mae-east, aads, pb, funet and mae-west. Anyway, we refer to these tables of the 1997 because they were taken as input in the main experimentation works about IP address lookup and IP tables minimization [5,18].

4.2 Measurement Principles

The main metric used to evaluate the IP tables minimization algorithms is the compression or reduction ratio, defined as $\frac{d-d_c}{d}$, where d is the initial size of the table and d_c is the size after the compression. More precisely, we have based our experimental analysis on the comparison between the total number of the table entries produced by the algorithms and the initial one. Moreover, referring to the IP lookup algorithms implemented, we have evaluated in percent terms the improvement of the lookup time (defined as $\frac{t-t_c}{t}$, where t is the lookup time on the initial table and t_c is the lookup time on the compressed one) achieved starting from the compressed tables with respect to the one yielded by the original ones.

5 Experimental Results

In order to have a more compact and readable presentation giving a direct indication of our experimental outcome, we present the results in global way. Moreover, since the tests of the compression algorithms and of the IP address lookup techniques on the artificially generated tables lead to almost the same results and considerations, we describe only the experimental results concerning the real world tables. Finally, concerning the IP address lookup techniques, we show both the global results relative to all the three traffic files, and the ones relative only to the "first" traffic file, i.e., the one containing addresses obtained from the existing routing tables.

5.1 The Case of a Single IP Routing Table

Table 2 shows the compression ratios obtained by running both the ORTC and the BF algorithms on the original tables in the case of a single routing table.

Table 2. Compression ratios of the original tables in the case of a single routing table

	ORTC	BF
Compression ratio	57.66%	99.80%

We notice the excellent compression results provided by the BF algorithm which can in fact exploit the address reassignment. As an example, after the compression obtained by BF, the routing table "mae-east" presents only 133 entries versus 38470 entries of the original table. Concerning the ORTC algorithm, the results pointed out by the experimental study reflect the theoretical analysis and the experimental evaluation presented in [5].

The effects of the compression on the lookup times are shown in tables 3 and 4 where we provide the average reduction of the lookup times of the implemented algorithms.

Table 3. Lookup times improvement of the IP address lookup algorithms executed on all the traffic files with respect to the original times on uncompressed routing tables

	Multibit	Binary search	LC-Trie
ORTC	8.56%	12.31%	0.65%
BF	77.88%	64.90%	32.22%

Table 4. Lookup times improvement of the IP address lookup algorithms on the first traffic file with respect to the original times on uncompressed routing tables

	Multibit	Binary search	LC-Trie
ORTC	9.53%	1.59%	12.21%
BF	87.22%	69.60%	43.19%

5.2 The Case of Multiple IP Routing Tables

Tables 5, 6, 7 and 8 show the experimental results in the case of compression of multiple IP routing tables. More precisely, Table 5 shows the reduction ratios on the sets of routing tables described in Table 1, Table 6 groups such results by set cardinality, whereas table 7 and 8 provide a measure of the induced improvement of the time of the main algorithms and data structures for the fast IP address lookup from the original tables to the compressed ones. Moreover, tables 9, 10 and 11 show the experimental results of the IP lookup algorithms on the single sets of routing tables described in Table 1.

Table 5. Compression ratios in the case of multiple routing tables

Set	ORTC	BF-Multi	BF-Multi + ORTC	BFM	BFM + ORTC	BFM-C.(1)	BFM-C.(1) + ORTC
1	33.53%	84.30%	94.44%	94.19%	95.92%	80.52%	96.77%
2	48.86%	55.53%	83.77%	85.29%	90.11%	35.62%	92.11%
3	67.20%	95.04%	98.07%	97.85%	98.78%	89.24%	98.79%
4	55.55%	76.74%	91.82%	92.70%	94.52%	66.73%	95.28%
5	56.48%	56.46%	83.67%	82.46%	89.88%	33.22%	92.72%
6	63.16%	80.62%	93.01%	90.99%	94.97%	64.03%	96.00%
7	60.29%	48.42%	81.96%	74.14%	86.94%	24.69%	91.09%
8	57.66%	12.18%	69.24%	45.08%	75.97%	−39.10%	83.74%
9	51.29%	−12.97%	60.78%	38.86%	69.61%	−55.60%	79.32%
10	51.97%	50.60%	82.80%	81.23%	87.60%	16.79%	90.74%
11	53.72%	44.76%	80.28%	78.62%	86.26%	15.91%	90.17%
12	52.37%	25.01%	73.60%	68.55%	80.30%	−4.28%	86.27%
13	53.28%	43.10%	79.91%	78.24%	86.13%	15.42%	90.00%

We evaluated the performance of the BF-Multi, BFM and BFM-Cluster(k) algorithms also with an additional compression step obtained by running ORTC

Table 6. Compression ratios in the case of multiple routing tables, grouped by set cardinality

#tables	ORTC	BF-Multi	BF-Multi + ORTC	BFM	BFM + ORTC	BFM-C.(1)	BFM-C.(1) + ORTC
5	57.16%	79.66%	92.61%	93.10%	95.31%	69.35%	96.08%
6	51.97%	50.60%	82.80%	81.23%	87.60%	16.79%	90.64%
7	57.70%	60.18%	85.60%	83.92%	90.26%	37.35%	92.86%
8	52.37%	25.01%	73.60%	68.55%	80.30%	−4.28%	86.27%
10	60.29%	48.42%	81.96%	74.14%	86.94%	24.69%	91.09%
13	51.29%	−12.97%	60.78%	38.86%	69.61%	−55.60%	79.32%
15	57.66%	12.18%	69.24%	45.08%	75.97%	−39.10%	83.74%

Table 7. Average lookup times improvement of the IP address lookup algorithms executed on all the traffic files with respect to the original times on uncompressed routing tables

	Multibit	Binary search	LC-Trie
BF-Multi	46.68%	38.16%	29.54%
BF-Multi + ORTC	59.35%	49.06%	−4.34%
BFM	61.63%	51.34%	−19.93%
BFM + ORTC	62.34%	46.77%	8.40%
BFM-Cluster(1)	37.41%	−11.73%	−17.67%
BFM-Cluster(1) + ORTC	65.44%	51.98%	16.51%

on their output tables. In fact, while on one hand BF-Multi does not exploit inclusions of entries (optimized by ORTC), that is each IP address matches at most one entry, on the other hand the BFM and BFM-Cluster(k) heuristics do not guarantee the minimality of the output tables, since the same output interface of a table may be associated to many virtual output interfaces (n-tuples).

In order to tune the parameter k of BFM-Cluster(k), we have executed the heuristic for different values of k. Since the number of tables to be minimized simultaneously is never greater than 15, we have obtained better results for small values of k, and the best ones (presented in the tables) for $k = 1$.

6 Analysis of the Results and Future Work

First of all, we point out how the reduction in the lookup times is higher when executing the lookup algorithms on the traffic file containing addresses obtained from the considered routing tables (i.e. the first traffic file). Thus, it results that the lookup time for addresses not matching any entry of a table is poorly affected by the size of the routing table.

The experimentation shown the effectiveness of the BF algorithm for the compression of a single table. It is worth noticing that BF could obtain a higher

Table 8. Average lookup times improvement of the IP address lookup algorithms executed on the first traffic file with respect to the original times on uncompressed routing tables

	Multibit	Binary search	LC-Trie
BF-Multi	57.57%	48.13%	47.87%
BF-Multi + ORTC	65.94%	55.50%	28.19%
BFM	82.62%	65.64%	15.31%
BFM + ORTC	82.99%	59.50%	37.83%
BFM-Cluster(1)	66.69%	35.87%	7.27%
BFM-Cluster(1) + ORTC	83.66%	60.72%	39.06%

Table 9. Average lookup times improvement on the algorithms executed on all the traffic files with respect to the original times on uncompressed routing tables (BF-Multi algorithm)

	LC Trie		Multibit		Binary search on IP length	
Set	BF-Multi	BF-Multi+ORTC	BF-Multi	BF-Multi+ORTC	BF-Multi	BF-Multi+ORTC
1	21.80%	−3.90%	57.44%	36.86%	37.54%	40.67%
2	30.88%	−6.14%	42.86%	48.45%	32.68%	42.95%
3	55.13%	24.80%	68.45%	66.46%	72.29%	73.59%
4	36.90%	14.32%	53.19%	43.04%	50.85%	58.98%
5	28.01%	−9.19%	44.89%	46.39%	34.01%	46.51%
6	44.69%	10.08%	58.57%	55.59%	59.03%	66.54%
7	33.56%	−9.43%	47.38%	48.11%	36.41%	50.05%
8	23.50%	−7.24%	40.16%	32.56%	25.48%	42.00%
9	18.61%	−16.38%	29.45%	27.71%	14.14%	31.55%
10	22.72%	−16.19%	42.75%	32.83%	32.62%	47.03%
11	19.87%	−13.83%	39.67%	27.59%	35.12%	43.96%
12	23.18%	−11.87%	39.44%	29.67%	29.07%	47.41%
13	25.28%	−11.51%	42.72%	34.60%	36.87%	46.61%

compression thanks to the address reassignment facility, which was not allowed to ORTC. Concerning the IP lookup times, the multibit and binary search on IP length algorithms present a better performance on the compressed tables with respect to LC-Trie. As foreseen, the lookup times are lower when the tables are compressed by the BF algorithm, with times ranging from 32.22% to 77.88%, depending on the lookup algorithm.

For the case of multiple tables, we can observe two factors that negatively affect the performances: (i) the number of tables in the sets and (ii) the "*homogeneity*" of the tables. In fact, both the BF-Multi algorithm and the BFM and BFM-Cluster ones degrade when the number of tables in the sets increases (see Tables 6 and 12). However, we have shown that it is possible to obtain a good average compression by further running the ORTC algorithm on the arising compressed tables. For all the algorithms, we have noticed that the compression

Table 10. Average lookup times improvement on the algorithms executed on all the traffic files with respect to the original times on uncompressed routing tables (BFM algorithm)

	LC Trie		Multibit		Binary search on IP length	
Set	BFM	BFM+ORTC	BFM	BFM+ORTC	BFM	BFM+ORTC
1	10.49%	10.68%	41.07%	68.91%	67.86%	38.09%
2	−30.42%	−0.80%	57.50%	62.50%	59.31%	45.16%
3	31.41%	32.13%	67.11%	77.55%	74.70%	73.72%
4	31.25%	19.24%	73.39%	62.10%	69.80%	51.00%
5	−16.40%	8.16%	58.75%	62.59%	67.62%	46.23%
6	2.59%	27.18%	69.18%	73.63%	70.67%	68.80%
7	−28.52%	9.21%	61.12%	66.03%	64.45%	50.34%
8	−111.07%	−3.39%	0.80%	53.57%	48.69%	34.38%
9	−105.17%	−11.56%	2.68%	51.52%	47.68%	28.14%
10	−17.35%	5.84%	56.40%	59.61%	65.50%	44.80%
11	−42.02%	6.71%	41.51%	57.96%	41.66%	44.93%
12	−0.10%	0.82%	66.24%	55.73%	60.27%	40.51%
13	16.25%	4.94%	71.68%	58.67%	62.98%	41.85%

Table 11. Average lookup times improvement on the algorithms executed on all the traffic files with respect to the original times on uncompressed routing tables (BFM-Cluster(1) algorithm)

	LC Trie		Multibit		Binary search on IP length	
Set	BFM-C.(1)	BFM-C.(1) + ORTC	BFM-C.(1)	BFM-C.(1) + ORTC	BFM-C.(1)	BFM-C.(1) + ORTC
1	20.38%	5.36%	59.40%	74.32%	37.56%	39.08%
2	27.36%	12.12%	45.00%	64.66%	37.31%	44.83%
3	29.45%	34.94%	67.60%	79.03%	56.60%	74.06%
4	49.08%	14.60%	59.44%	69.34%	41.25%	57.29%
5	40.27%	18.78%	53.02%	66.96%	37.11%	54.37%
6	45.90%	36.62%	58.97%	75.20%	56.64%	69.06%
7	41.35%	21.18%	50.25%	66.66%	40.75%	55.91%
8	−16.16%	16.19%	27.77%	53.76%	−57.74%	41.98%
9	−125.06%	9.44%	27.84%	52.71%	−53.52%	37.55%
10	−91.00%	6.55%	−41.00%	63.88%	−107.27%	53.31%
11	−141.89%	15.20%	−4.62%	62.54%	−182.52%	44.54%
12	−83.34%	9.81%	37.01%	59.18%	−29.64%	51.25%
13	−26.01%	13.90%	45.58%	62.53%	−29.64%	52.49%

ratio worsen as the tables become less "*homogeneous*". We tried to formalize such an intuition by running the algorithms on set of tables independently obtained by perturbing the entries of a real world one with a fixed probability. Table 12 provides the reduction ratios on the sets of routing tables artificially generated, grouped by cardinality and perturbing probability. We can notice that the

Table 12. Compression ratio on sets of routing tables obtained by perturbing the entries of real world one with fixed probability

#tables	Perturbating Prob.	BF-Multi	BF-Multi + ORTC	BFM	BFM + ORTC	BFM-C.(1)	BFM-C.(1) + ORTC
7	10%	33.71%	72.24%	75.23%	78.65%	25.58%	88.41%
15	10%	−50.15%	47.90%	35.00%	44.94%	−16.77%	62.09%
15	30%	−324.16%	−46.91%	3.14%	11.33%	−12.82%	13.68%

algorithms increase considerably the number of table entries when the homogeneity degree of the tables decreases, i.e. when the perturbing probability is higher.

Overall, the BFM and BFM-Cluster(1) heuristics present better performances than BF-Multi, with BFM-Cluster(1) being the best one.

Finally, concerning the IP lookup times, we have shown that in the auxiliary data structures the compression generally induces a proportional improvement. In fact, we have observed an evident lowering of the lookups times, as shown in Tables 7, 9 and 10. Again, the multibit and the binary search on IP length algorithms present a performance on the compressed tables better than the one obtained by the LC-Trie algorithm.

Many question are left open. First of all, an interesting issue is the extension of the experimentation work to the new IP release with addresses of 128 bits (IPv6). Moreover, it would be nice to determine other effective algorithms and heuristics, also guaranteeing better approximation ratios.

Finally, our work was meant as a first attempt toward the investigation of the effectiveness of the existing and newly proposed methods for IP tables compression. Starting from this basis, new heuristics should be devised also taking more into account other intrinsic features of the IP protocol, like for instance the hierarchical structure of the network.

Acknowledgement. The authors would like to thank Prof. Venkatachary Srinivasan for his help during the retrieval of the real world IP routing tables used in the tests and Prof. Marcel Waldvogel for his suggestions on the implementation of some auxiliary data structures.

References

1. Bilò, V., Flammini, M.: On the ip routing tables minimization with addresses reassignments. In: Proc. of the 18th International Parallel and Distributed Processing Symposium (IPDPS) (2004)
2. Buchsbaum, A.L., Fowler, G.S., Krishnamurthy, B., Vo, K.-P., Wang, J.: Fast prefix matching of bounded strings. ACM Journal of Experimental Algorithms, vol. 8 (2003)
3. Crescenzi, P., Dardini, L., Grossi, R.: IP address lookup made fast and simple. In: Nešetřil, J. (ed.) ESA 1999. LNCS, vol. 1643, Springer, Heidelberg (1999)
4. Degermark, M., Brodnik, A., Carlsson, S., Pink, S.: Small forwarding tables for fast routing lookups. In: Proceedings of ACM Sigcomm, pp. 3–14 (1997)

5. Draves, R., King, C., Srinivasan, V., Zill, B.: Constructing optimal IP routing tables. In: Proceedings of The Conference on Computer Communications, 18th joint conference of the IEEE Computer and Communications Societies (INFOCOM) (1999)
6. Egevang, K., Francis, P.: The ip network address translator (NAT). Internet RFC 1631 (May 1994)
7. Ford, W., Topp, W.: Data Structures with C++. Prentice-Hall, Englewood Cliffs (1996)
8. Gupta, P., Lin, S., McKeown, N.: Routing lookups in hardware at memory access speeds. In: Proceedings of The Conference on Computer Communications, 17th joint conference of the IEEE Computer and Communications Societies (INFOCOM), pp. 1240–1247 (1998)
9. Gupta, P., Prabhakar, B., Boyd, S.P.: Near optimal routing lookups with bounded worst case performance. In: Proceedings of The Conference on Computer Communications, 19th joint conference of the IEEE Computer and Communications Societies (INFOCOM), pp. 1184–1192 (2000)
10. Kernen, T.: Traceroute org. http://www.traceroute.org/#Route Servers (2005)
11. Lampson, B., Srinivasan, V., Varghese, G.: IP lookups using multiway and multicolumn search. IEEE/ACM Transactions on Networking 7(3), 324–334 (1999)
12. NANOG. The north american network operators' group. http://www.nanog.org/lookingglass.html (2005)
13. Nilsson, S.: Home page. http://www.nada.kth.se/s̃nilsson
14. Nilsson, S., Karlsson, G.: Ip-address lookup using lc-tries. IEEE Journal of Selected Areas in Communications 17(6), 1083–1092 (1999)
15. Srinivasan, V.: Fast and efficient Internet lookups. PhD thesis, Washington University (1999)
16. Srinivasan, V., Varghese, G.: Faster ip lookups using controlled prefix expansion. ACM Transactions on Computer Systems 17(1), 1–40 (1999)
17. Michigan University and Merit Network. Internet performance and analysis (ipma) project. http://www.merit.edu
18. Waldvogel, M.: Fast Longest Prefix Matching: Algorithms, Analysis, and Applications. PhD thesis, Swiss Federal Institute of Technology - Zurich (2000)
19. Waldvogel, M., Varghese, G., Turner, J., Plattner, B.: Scalable high-speed ip routing lookups. In: Proceedings of ACM Sigcomm, pp. 25–36, (October 1997)