Performance Analysis of IP Routing Lookup Algorithms : Patricia Tree based vs. Hashing based

Author: Packer Ali, Dhamim
Department of Mathematics & Computer Science, Kent State University
April 2000

Abstract

Internet address lookup is a challenging problem due to increasing routing table sizes, increased traffic, higher speed links and migration to 128 bit IPv6 addresses. IP routing lookup requires computing the Best-Matching Prefix. Several algorithms have been proposed to speed up this critical component of IP Routing. The widely used algorithm, as of today, is based on Patricia (Practical Algorithm to Retrieve Information Coded in Alphanumeric) trees. This project aims at designing and comparing the performance of a basic Patricia-tree based approach and the recently proposed variants of Hashing-based approach (linear search, binary search and binary search with backtracking), that appeared in the paper - *"Scalable High Speed IP Routing Lookups"* **(in ACM SIGCOMM 1997 journal)**

Table of Contents

1. **Introduction : Problem Specification**
2. **Plan / Architecture / Approach**
3. **Alternatives to your approach**
4. **Justification for selected plan**
5. **Theoretical Space & Time Complexity**
6. **Empirical Results**
7. **Relative advantages/limitations/innovations**
8. **Possible future work improvements**

Appendix A: How to install the code

Appendix B: How to run the code

1. Introduction : Problem Specification

A major step in IP packet forwarding is to lookup the destination address (of an incoming packet) in the routing database. More specifically, to search the routing table for the longest prefix matching the destination IP address.

This search becomes very critical in terms of time consumed, especially in border or backbone routers that need to service more than thousands of incoming packets very frequently. Also, a typical Ipv5 backbone router has over 30,000 entries in its routing table. Hence the problem of lookup becomes very crucial to the IP Router's performance.

Standard techniques for exact matching such as perfect hashing, binary search, etc. cannot directly be used for IP lookup.

The **Best-Matching Prefix (BMP)** problem specification:

- When an IP router receives a packet, it must compute which of the prefixes in its database has the longest match when compared to the destination address in the packet.
- The packet is then forwarded to the output link associated with that prefix.

For Example:-

Consider the routing table has entries for the prefixes,

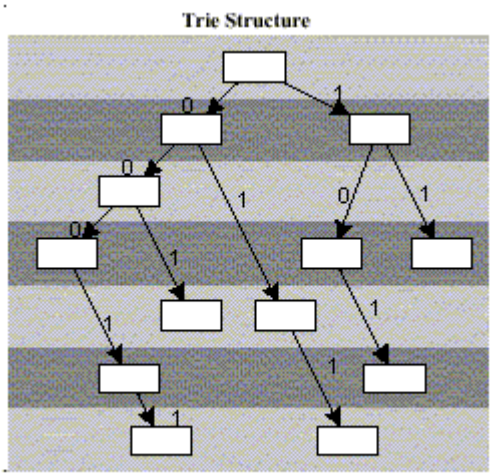
P1 = 0101
P2 = 0101101
P3 = 01011010101 1

The BMP (Best Matching Prefix) for 010101101011 is P1
The BMP (Best Matching Prefix) for 010110101101 is P2

2. Plan / Architecture / Approach

Algorithms designed & compared to solve BMP problem

- **Patricia Tree algorithm**



- This is the most commonly available IP lookup implementation found in the BSD kernels. Current implementations have made a number of improvements on Sklower's original implementation.
- The basic implementation is store the prefix entries as nodes in a Trie data structure that is optimized for storage and retrieval. Since the IP addresses are binary, the trie data structure has a alphabet size of 2 with the alphabet set as { 0, 1 }
- The implementation in this project is a very minimal and basic version of the actual patricia tree algorithm.
- Please refer this web-site for more information on this algorithm :

- **Hashing based IP lookup - (with linear & binary search)**

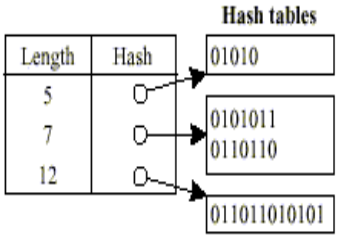


Figure 1: Hash Tables for each possible prefix length

- Professors in Washington University, St. Louis, proposed this algorithm.

- It basically maps the different levels/bit_lengths in a Patricia tree into separate hash tables as shown in the previous diagram.
- It then does a binary search on these hash tables to cut down the search cost to logarithmic order. As a result, of this improvement, it needs to insert what are called markers to help in the binary search process.
- And to avoid backtracking, pre-computations are done when inserting entries in to the routing table.
- For more information, please, refer their on-line paper "Scalable High Speed IP Routing Lookups" at http://www.acm.org/sigcomm/sigcomm97/papers/p182.html

- The implementation in this project experiments on 2 possible designs for hashing, namely,
  - Open Hashing - Simple Table Lookup
  - Closed Hashing - linked list

3. Alternatives to my approach

Some alternatives to my implementation could be:-

- The initial basic IP Routing schemes for IP Routing in the first TCP/IP implementations. These were either based on a simple linear table, or a simple hash-bucket scheme that computes a hash value based on the destination address and searches the corresponding hash table.
- *Mutating Search Trees* proposed in the same paper which is primarily referred to for this report/implementation – **"Scalable High-Speed IP Routing Lookups"**
- As proposed in a paper – **"Deterministic IP Table Lookup at Wire Speed"**
  - http://www.isoc.org/inet99/proceedings/4j/4j_2.htm
- As proposed in the paper – **"Small Forwarding Tables for Fast Routing Lookups"**
  - http://www.acm.org/sigcomm/sigcomm97/papers/p192.pdf
- As proposed in the paper – **"IP Lookups using multiway and multicolumn search"**
  - http://www.ccrc.wustl.edu/~cheenu/research.html
  - http://www.tik.ee.ethz.ch/~mwa/HPPC/

4. Justification for selected plan

All the alternatives suggested in the previous section were not chosen, either because

- they didn't show much significant improvement from the current implementations in terms of space & time complexity (as the case with the first alternative) or
- they would require more time for analysis & implementation (as is the case with the proposals in the last four alternatives)

Also, with my own implementations of Patricia tree algorithm and the proposed hashing based scheme, a very minimal, basic & unoptimized implementation has been chosen due to the very same reason of lack of time for analysis & implementation.

5. Theoretical Space & Time Complexity

W - the length of an address and
N - the number of routing table entries

Complexities/ Algorithms	**Buildup Time Complexity**	**Search Time Complexity**	**Space/Memory Complexity**
Patricia Tree Algorithm	O(N*W)	O(W)	O(N)
Hashing-based algorithm with Binary Search	O(N*log(W))	O(log(W))	O(N*log(W))

- **Patricia Tree algorithm :-**

  Time Complexity
  - the worst-cast time in the basic implementation was shown to be $O(W^2)$
  - Current implementations have made a number of improvements on Sklower's original implementation with improvement of worst-case to O(W)

  Note:-
  - Knuth - The recurrence relation of this algorithm is one of the most complex in analysis.

  Build time complexity - O(N*W)
  Search time complexity - O(W)

  Space Complexity
  - Worst case if the binary tree is completely full = 2 W - 1 nodes
  - But usually less, as many entries share the nodes and the tree is seldom complete.

  Memory Usage complexity> - O(N)

- **Hashing-based algorithm (with Binary Search):-**

  Time Complexity
  - No. of Hashes to be made $O(\log(W_{dist}))$ where $W_{dist} <= W$ - the number of distinct lengths in the database.
  - Hash-lookup cost - Table lookup vs closed Hashing

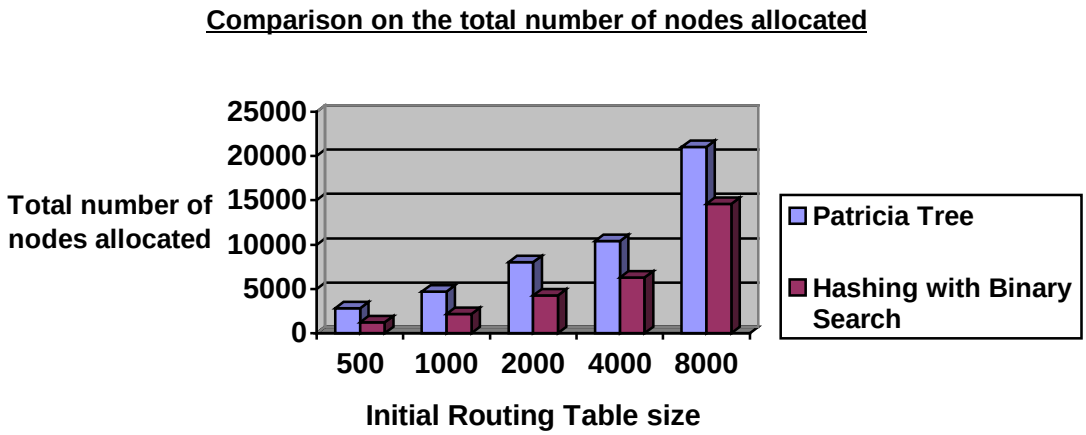
  Build complexity - O(N*log(W))
  Search complexity - O(log(W))

  Space Complexity
  - additional marker entries need to be inserted to avoid backtracking
  - No. of entries + No. of markers

  Memory Usage complexity - O(N*log(W))

6. Empirical Results

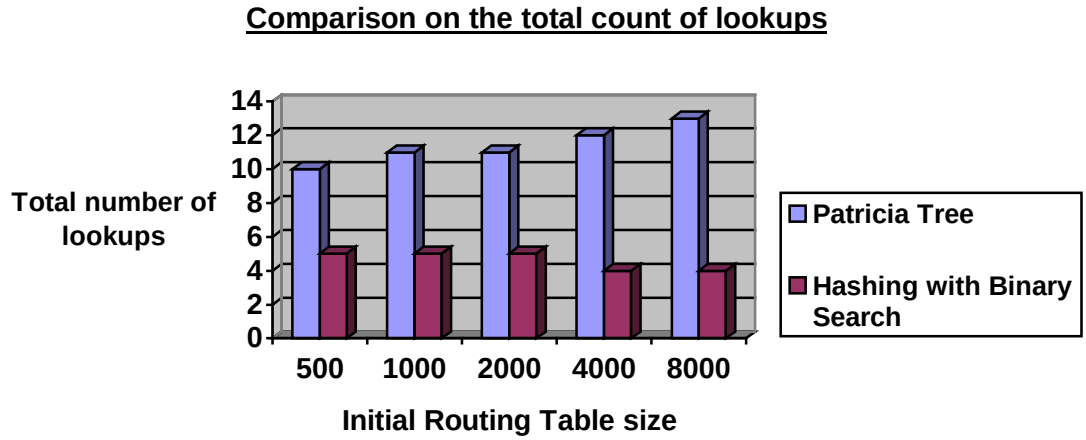
- **Comparison on the total number of nodes allocated**



Observation:

- The total number of nodes allocated during build-up gives us an approximate estimate as to the total amount of memory consumed by the two algorithms in general.
- As the routing table size increases, which is the case in border routers, we see that the Hashing based scheme actually consumes less number of total nodes than the basic Patricia tree scheme. It is to be noted than in the Hashing-based scheme many prefixes end up sharing the same set of markers and so the overall cost is reduced.
- The empirical result is hence found to be in conformance with the theoretical estimate.

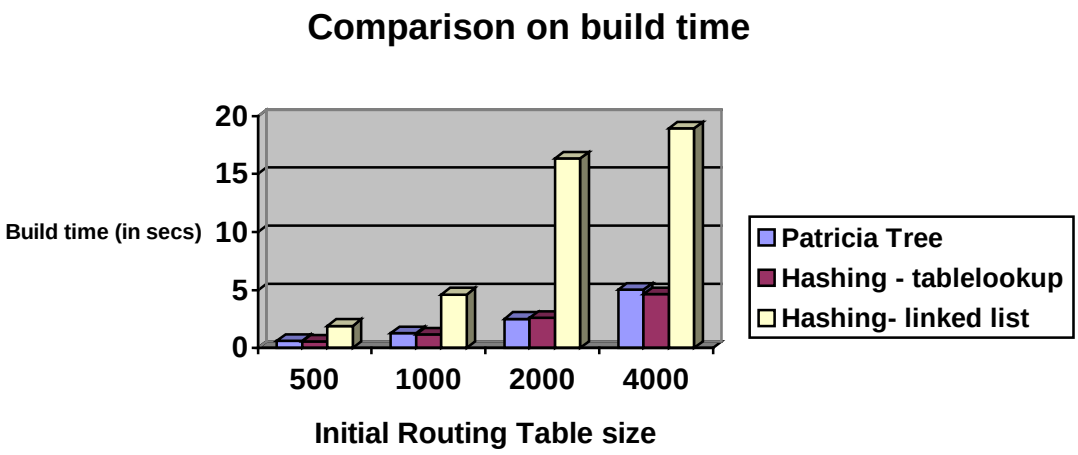
- **Comparison on the total count of lookups**



Observation:

- The total number of average lookups made during search gives us an approximate estimate as to the total time for searching on the average.
- As the routing table size increases, which is the case in border routers, we see that the Hashing based scheme actually makes only a logarithmic order of lookups compared to the almost linear (equal to maximum address length) lookups than the basic Patricia tree scheme makes. This logarithmic order is attributed to the binary searching we do in the hashing based scheme with the additional benefit of pre-computations to avoid backtracking.
- The empirical result is hence found to be in conformance with the theoretical estimate.

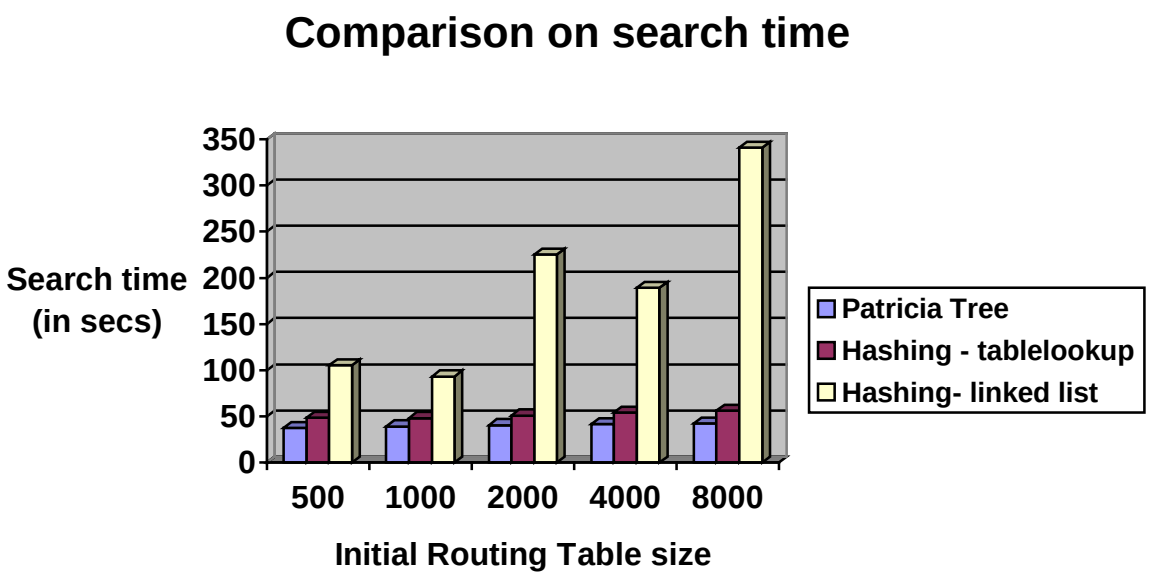
- **<u>Comparison on the total routing table buildup time</u>**



<u>Observation:</u>

- The total time taken to build the initial routing table gives us an approximate estimate as to the time complexity of inserting a new prefix into the routing table on the long run.
- As the routing table size increases, which is the case in border routers, we see that the Hashing based scheme actually takes lesser to time to insert an entry in to its already increasing routing table entries compared to the basic Patricia tree scheme. This can be attributed to the fewer lookups made and the sharing of already present markers which reduces the number of entries to be made into the routing table.
  - Within the hashing-based variants, it is found that the closed hashing variant performs poorly as the routing table increases. This is as expected since our basic un-optimized scheme just does a sequential search on the hash buckets to insert a new entry.
- The empirical result is hence found to be in conformance with the theoretical estimate.

- **Comparison on the search time**



Observation:

- The total time taken to search the initial routing table gives us an approximate estimate as to the time complexity of searching the best matching prefix into the routing table on the long run.
- As the routing table size increases, which is the case in border routers, we see that the Hashing based scheme actually takes more time to search an compared to the basic Patricia tree scheme. There is always a constant time difference between the basic Patricia tree scheme and the hashing based scheme based on tablelookup. Actually this is a contradiction to our observation on the total number of lookups made. Truly, the hashing based scheme should perform better than the Patricia tree in this crucial search time complexity. I attribute this costly difference to my implementation wherein I read in the prefixes and destination addresses as character strings rather than integers in binary. That latter case, which is how it actually gets done in routers, would no doubt improve the hashing-based lookup since its lookup will no longer require the frequent character to integer conversion.
  - Within the hashing-based variants, it is found that the closed hashing variant performs poorly as the routing table increases. This is as expected since our basic un-optimized scheme just does a sequential search on the hash buckets to search the BMP.
- The empirical result is hence found to be in contradiction with the theoretical estimate due to some of the costly operations in my implementation. Due to lack of time and due to the increased complexity of the implementation, the current simpler and hence less efficient implementation was chosen.

NOTE: Since both the algorithms considered are common algorithms (and hence not my own original design) already implemented & tested, no proof of correctness is provided. For proof of correctness in the basic hashing based scheme for build-up and searching, please refer the paper – “Scalable High Speed IP Routing Lookups”.

7. Relative advantages/limitations/innovations

Advantages:

- Simplicity of implementation
- Verification that the hashing-based scheme cannot work with a naïve binary search scheme without either backtracking or pre-computation can be made satisfactorily. (This is pointed out in the research paper. Please refer the research paper for more details.)
- Verification of some of the basic theoretical estimates on the number of nodes allocated and average number of lookups made.
- Affirms the importance of a good hashing function, apart from other concerns, as to the choice for a closed-hashing based implementation for the hashing-based scheme.

Limitations:

- The basic implementation is NOT optimized. In reality, both the Patricia tree algorithm and the hashing-based schemes can be very much optimized for performance.
- The actual Patricia tree implementation avoids branching at all bit-levels of the prefix. This will cut down the total memory cost as well as the searching cost significantly
- The hashing based scheme also can be optimized significantly both in terms of total memory and time complexity by taking advantage of the prefix characteristics.
- Another critical limitation of this implementation & analysis is that the cost of deleting a prefix entry is not considered, for example when a prefix entry times out. The IP lookup algorithm should be also efficient in this operation.

Innovations:

- Certain innovative mechanisms have been incorporated in our hashing-based scheme for the table-lookup to make the implementation easier.
- The prefix itself is not stored in the hash table to conserve space. Instead two integers are stored to allow recreating any prefix that is found in the hash tables.
- Also our basic Patricia tree implementation uses a very simple mechanism to store the prefixes instead of the very complicated (but a lot more optimized) BSD implementation.

8. Possible future work improvements

Some suggestions for future work could be:-

- Improve hashing function to take account of no. of entries in each hash table
- Each time a search reference is made in the hash table, move the prefix to the beginning in anticipation of another search immediately
- Increase the number of pointers in each hash table to point to appropriate lengths – cut-down cost of searching entire table always
- Incorporating an efficient mechanism to allow prefix entries to timeout and be updated.

Appendix A: How to install the code

1. Download Instructions:

- File - "project.tar.gz" contains all the necessary program and data files for this project.
- All files within "project.tar.gz" are further gzipped to save space
- "project.tar.gz" also contains a program that helps in automating the process of creating data files, necessary for input to all the programs

2. Installation Instructions:

- To unzip “project.tar.gz”, please execute
  gunzip project.tar.gz

- To untar the contents of "project.tar", please execute the
  tar -xvf project.tar

- Now, gunzip all the files in the project directory by executing,
  gunzip *.gz

- After unzipping the files, please check that you have the following files with their corresponding file sizes as:

File sizes	File names
1107	generate.C
14824	ips_chashing.C
6075	ips_chashing.h
13095	ips_tlookup.C
4781	ips_tlookup.h
6710	patricia.c
55879	prefix_list
255602	search_list

Appendix B: How to run the code

1. Compilation Instructions:

- Compiler needed for Hashing-based algorithm

  You can use either
  - the HP-UX C++ compiler - (CC file_name.C -lm at the shell prompt) or
  - the GNU project C++ Compiler - (g++ file_name.C -lm at the shell prompt) or
  - any other complaint C++ compiler

- Compiler needed for Patricia algorithm

  You can use either
  - the HP-UX C compiler - (cc file_name.c at the shell prompt) or
  - any other complaint C compiler

The shell prompt commands for the corresponding algorithms are as follows:-

- Patricia Tree algorithm

  $ cc patricia.cc -lm

- Hashing Based algorithm

  $ CC ips_tlookup.C -lm /* for table-lookup hashing algorithm */
  $ CC ips_chashing.C -lm /* for closed hashing based algorithm */

2. Generating Data Inputs for both algorithms

- By default, both algorithms read two files
  - "prefix_list" : to initialize one's routing table
  - "search_list" : to search the BMPS in the initialized routing table

- You can use the program "generate.C" to generate both "prefix_list" & "search_list"
  - "generate.C" reads the table-size required and a random number initializer from the user.
  - Since it outputs the routing table entries to the standard output, you will need to redirect its output to the corresponding files as:

```
// create a "prefix_list" file with 1000 entries
$ CC generate.C -lm > prefix_list
  1000                          /* waits for routing table size input  */
  786                       /* waits for random number initializer */
$
// create a "search_list" file with 16000 entries
$ CC generate.C -lm > search_list
  16000
```

7
$