Efficient IP Lookup Algorithm

K.J. Poornaselvan, S. Suresh, C. Divya Preya, and C. G. Gayathri

The rapid growth of traffic in the Internet, backbone links of several gigabits per second are commonly deployed. To handle gigabit-per-second traffic rates, the backbone routers must be able to forward millions of datagrams per second on each of their ports. Fast IP address lookup in the routers, which uses the datagram’s destination address to determine for each datagram the next hop, is therefore crucial to achieve the datagram forwarding rates required. Also the packet may encounter many routers before it reaches its destination. Hence decrease in delay by micro seconds results in immense cut down in the time to reach the destination. IP address lookup is difficult because it requires a Longest Matching Prefix search. Many lookup algorithms are available to find the Longest Prefix Matching; one such is the Elevator-Stairs Algorithm. It provides a total search time of $O\left(\frac{w}{k} + k\right)$ by indexing hash table to Practical Algorithm to Retrieve Information Coded in Alphanumeric (PATRICIA), where $w$ is the length of the IP address and $k$ is the level of Trie. Elevator Stairs Algorithm uses linear search at the $k$-level is modified to binary search at the $k$-level of Trie. At the $k$th-level, non-branching nodes are added to jump $k$ levels of Trie which reduces the time for searching in the Trie. It provides a better search time over the existing Elevator-Stairs Algorithm, by accomplishing a two-way search in the trie.

1.0 Introduction

1.1 Address Lookup

The primary role of routers is to forward datagram toward their final destinations. For this purpose, a router must decide for each incoming datagram where to send it next. More exactly, the forwarding decision consists of finding the address of the next-hop router as well as the egress port through which the datagram should be sent. This forwarding information is stored in a forwarding table that the router computes based on the information gathered by routing protocols. To consult the forwarding table, the router uses the datagram’s destination address as a key; this operation is called address lookup.

Once the forwarding information is retrieved, the router can transfer the datagram from the incoming link to the appropriate outgoing link, in a process called switching. The key issue in router performance is the IP routing lookup mechanism used for the ferrying of the large number of datagrams which are transmitted from source to intermediate/destination router (see [5]). A serious time delay in datagram transmission occurs over the internet as a delayed search process occurs when finding the best matching IP address in the routing table.
These limitations probed to implement an efficient IP lookup Algorithm which reduces the searching time of an IP Address in Routing Table. The feature of insertion of a new IP address and deletion of an existing IP address is encapsulated within this project.

2.0 Elevator Stairs Algorithm

Elevator-Stairs Algorithm ameliorates the IP lookup efficiency in the large IP Routing tables (see [4]). It uses a path compressed trie which is called as Trie (refer [2]). In this trie, traversed path of any two nodes are compressed through the removal of non-branching nodes between them. IP lookup is initialized by searching the longest matching prefix in the k-level tree, which is build from Trie by adding non-branching nodes. As a next step, multiple bit key corresponding to the matching prefix is determined. Then a search of the hash table at the levels k, 2k, 3k,.. is performed, where the value of k is a constant between zero and maximum length of the IP address. If it matches with the level it searches for the destination node in the unmodified Trie using linear search.

Fig. 1.1 Searching in elevator-stairs algorithm

Lookup time in the k-level of the PATRICIA is of the order \( O \left( \frac{w}{k} \right) \).
- Total search time of the algorithm is of the order \( O \left( \frac{w}{k+k} \right) \)
- Some of the problems in the Elevator-Stairs Algorithm are
  - Increased Lookup time due to Linear search at the kth level
    - Lookup time in the k-level of the PATRICIA is of the order \( O \left( \frac{w}{k} \right) \).
    - Total search time of the algorithm is of the order \( O \left( \frac{w}{k+k} \right) \)
3.0 Modified Elevator Stairs Algorithm

General Description

Traditional technique involves the usage of Elevator-Stairs Algorithm for searching the IP Address in the IP Lookup Table. The proposed system is a modified version of Elevator-Stairs Algorithm by altering the searching technique. Binary search is replaced by the bi-directional linear search at the kth-level Trie instead of linear search in the Elevator-Stairs Algorithm. So it drastically reduces the lookup time of an IP address in the IP Lookup table. The searching is initiated by matching the longest matching prefix in the kth-level Trie, which is build from the Trie by adding non-branching nodes.

In Elevator-Stairs Algorithm Searching in the kth-level Trie is done using Linear Search. So the Lookup time in the K-level of the PATRICIA is in the order $O \left( \frac{w}{k} \right)$ and the total search time of the Algorithm is of the order $O \left( \frac{w}{k+k} \right)$, where $w$ is the maximum length of the IP Address and $k$ is a constant value between 1 and $w$. In the Case of Modified Elevator-Stairs Algorithm searching for the Best Matching prefix is done using Binary Search at the kth-level of the Trie. So, this proposed technique algorithm provides a better lookup time when compared to the existing algorithm of order $O \left( \frac{w}{k+k} \right)$.

Two cases of LPM search is shown in the Fig. 1.2, one at level 3k and the other at level 2k+2 from the root node.

In the former case, the search finds successive matches in the hash tables at levels $k, 2k$, and $3k$, but fails to find a match at level $4k$ and also fails to traverse the Trie. Thus, an inference is drawn that the LPM for the IP address is in level 3k and subsequently the NHP information is retrieved from the corresponding node in the hash table.

In the latter case, the search finds a match in hash tables at levels $k$ and $2k$, but fails to find a match in level $3k$. Subsequently the search traverses the Trie starting from the corresponding node in level $2k$, and finds a match at a node in level $2k+2$. Thus, in the worst case, the search goes through $W/k$ levels of the $k$th-level-tree and $k-1$ node traversals in the Trie.

This System consists of three modules namely Search, Insertion and Deletion. The search module finds exact match for the IP address. Insertion module, searches the location where to be inserted and IP address is inserted. Similarly, Deletion calls the Search modules and then sweeps out the IP address.

Search

The search module uses the Trie and kth-level Trie to accomplish the search operation

• Trie
  a) Each edge is labeled with exactly one bit.
  b) For any node, edges connected to its child nodes have distinct labels.
c) Every string is mapped to some node in the tree such that the concatenation of the bits on the path form toot to exactly spell out, and every leaf of the tree are mapped to by some string.

- **$K^{th}$-Level Trie**
  
  Procedure Build $k$th-level-tree ($v$)
  
  Create an empty hash table $H(v)$
  
  For each edge that crosses level $k$ form root
  
  Add a non-branching node at level $k$
  
  For each node $u$ at level $k$
  
  Path = path label between $v$ and $u$
  
  $p$ = build $k$th-level tree ($u$)
  
  Insert $p$ into $H(v)$ with key = path

**Fig. 1.2 Searching in Modified Elevator Stairs Algorithm**

In this function, a non-branching node is added to each level that crosses from the root node in Trie. The non-branching node at various levels indirectly represents the length of the IP address at each level. These non-branching nodes are recorded in a hash table at each of the root node. Non-branching node acts index for jumping various levels of the Trie. A data structure is created so that the search algorithm can jump $k$ levels of Trie, where $k$ is an integer between 1 and $w$. The algorithm is initially called with the root node as input. Let $k$ level of a Trie denotes a level (string depth) $i_k$ for some integer $i$ such that $0 \leq i \leq w/k$.

- **Searching in Kth Level Trie**
  
  Procedure FindIP(node, $p$, pos, port)
  
  port = copy of NHP at node
  
  if node represents a leaf in Trie
  
  return (port)
  
  key = $p[pos+1....pos+k]$
if key is in H(node)
v = node corresponding to key in H(node)
if v is no more than k level away from node
return(FindIP(v,p,pos+k,port))
else
e = edge between node and v
if p matches the edge label of e
child = node at the end of e
l = length of e
return(FindIP(v,p,pos+1,k,port))
else
return(port)
else
pnode = node in PATRICIA tree corresponding to node
return(FindIP_PAT(pnode,p,pos,port))

The search for longest matching prefix on the IP address p using kth-level-tree
starts at root and set the variable current_port_number to the default port number.
The variable current_port_number stores the port number assigned to the longest
matching prefix of length less than the string depth of the current node. At string
depth of ik in the kth-level-tree, the search algorithm updates the current_port_number
to the port number stored in the current node in the kth-level-tree, if
the node has a copy of the port number. The search algorithm checks for the
node with key \( p[\lfloor ik+1...(i+1)k \rfloor] \) exits. If such a node exists, the lookup mechanism
follows the pointer to the node at level \( (i+1)k \) and continues the search. If such
a node does not exits, it depicts the search for the LPM must end in the Trie
between levels ik and \( (i+1)k \).

**Searching in Trie**

Procedure FindIP_PAT(node,p,pos,port,length)
if a NHP is assigned to the node
port = NHP assigned to the node
if the node is a leaf
return(port)
e = edge of node that starts with p[pos+1]
mid = length/2
while length !=0
if pos+1 > mid
root = p[mid+1]
else
root = p[pos+1]
end while
if no such node exists
return(port)
if edge label of e matches p
child = node at the end of e
l = length of e return(FindIP_PAT(child,IP,pos+l.port))
else
return(port)

The search operation in the Trie starts with the root nodes with some arguments which describe the node and its position. This recursive algorithm searches for the required IP address with the length of the address. It searches for the edge label of the trie with IP address, after finding the match it assigns it as the child node and calls again the same function still it finds the leaf node. the linear search used in the traditional algorithm is replaced by the binary search here. This decreases the search time drastically.

Insertion

procedure Insert(p)
Search prefix p in the same way as FindIP
If the search finishes inside an edge
Create a node there
pnode = node where the search ends
If p = path-label (pnode)
pn = NHP assigned to p
Copy pn to appropriate nodes in
kth-level-tree
Else
Attach a leaf for p at pnode
E = new edge between p and pnode
If e crosses a k-level
Create anode at the k-level on e
parent =pnode’s parent in
kth-level-tree
knode = node in kth-level-tree
to represent pnode
Insert a pointer to knode in
H (parent)
If H (parent) becomes full
Double the size of H (parent)
Rehash the elements in H (parent)

Insertion of an IP address in the routing table starts with the search for the longest match to the IP address being inserted. It is akin to searching the longest matching prefix of IP address except that the prefix may be shorter than bits. When the search is completed, the nodes of kth-level tree and PATRICIA tree that need to be modified are known.
There are two cases of insertion:

a) The prefix that needs to be inserted forms a new leaf in the Trie, or
b) The inserted prefix does not form a new leaf and in that case the structures of the PATRICIA tree and the kth-level-tree do not change.

Tree Representation

Fig. 1.3: Tree Before Insertion

The tree shown in Fig. 1.3 is created from the Root position with null string as the key value. The node is inserted into the tree based upon the length of the IP address. Insertion of node is performed based on the binary tree criteria. Fig. 3.3 shows tree updation after insertion.

Fig. 1.4: Tree Updation After Insertion
In case (a), a new leaf node is attached at the point where the search ends. If the edge to the leaf crosses a k-level, a new node is created on this edge at the level in the Trie. A pointer to the new node is added to the hash table at $k(i-1)$ level with key $p[ik+1...(i+1)k]$.

In case (b), there are no changes in the structure of the Trie, but the new port number must be copied to the nearest kth-level below the updated node in kth-level-tree.

**Deletion**

procedure Delete(p)
Search prefix p in the same way as FindIP
pnode = node matching p
if p = path-label (pnode)
pn = NHP assigned to p’s parent
  copy pn to appropriate nodes in kth-level-tree
else
  e = the edge to p
  delete the leaf for p
  if e crosses a k-level
    knode = the node on e at the k-level
    klnode = node in kth-level-tree
      representing knode
    parent = pnode’s parent in kth-leveltree
    Delete knode
    Delete the pointer to kpnode in H(parent)
    Delete klnode
    Delete pnode
    If H(parent) has few entries
      Half the size of H(parent)
      Rehash the elements in H(parent)
Deletion of an IP address in the routing table starts with the search for the longest match to the IP address being inserted. It is the same as searching the longest matching prefix of IP address except that the prefix may be shorter than bits. When the search is completed, the nodes of kth-level-tree and PATRICIA tree that need to be modified are removed.
Database File

Fig 1.5 Database File – input.txt

The Fig. 1.5 displays the database window for the system, which contains the IP address and port address.

Fig 1.6 Input Insertions from Database

The window is displayed after the construction of Trie with data form input.txt.

Fig. 1.7: Main Interface

This is the main interface of the system which contains the input panel and the memory monitor panel.
Input

Fig. 1.8: Search Module – Input Panel

This shows the input window for the search module, the IP address is entered in the appropriate textbox.

Output

Fig. 1.9 Search Module - Lookup Time

The output of the search module is shown with IP address entered in the apposite textbox. It displays the Port number and the Lookup time respectively.
Memory Monitor

Fig. 5.7: Search Module - Memory

Monitor

This shows the Memory used by the system to search for the IP address specified to the system.

Conclusion and Future Work

The rapid growth of Internet Traffic causes increase in size of routing table. The current day routers are expected to perform longest prefix matching algorithm to forward millions of datagram each second, and this demand on router is increasing even while the prefix search database is expanding in both the dimensions, i.e., IP address length (128 bits for IPv6) and number of prefixes. When there is migration from IPv4 to IPv6, the routing table size increases exponentially. So this system has modified the Elevator-stairs algorithm to reduce lookup latency. This provides an efficient way of searching through Trie and klevel- tree.

The modified Elevator-Stairs algorithm currently applicable for IPv4 address scheme. IPv4 address scheme consists of address space of 32 bits. So, the size of routing table is up to the maximum of 232 entries. This is itself constituted to increase in size of the routing table. The algorithm can be further modified to support to IPv6, which is 128 bit size address space. This Algorithm can be used to enhance the searching technique in Mobile IP Lookups, which builds an efficient and effective Wireless LAN.
References


