ITC19/ Performance Challenges for Efficient Next Generation Networks 31-38
LIANG X.J. and XIN Z.H.(Editors)
V.B. IVERSEN and KUO G.S.(Editors)
Beijing University of Posts and Telecommunications Press

A High Performance IP Lookup for IPv6 using Parallel Computation Models

D.Diwakar and Srinivasan.T

D.Diwakar. M.E Final Year,
Department of Computer Science and Engineering,
Sri Venkateswara College of Engineering,
Sriperumbudur, India.
divakar.dayalan@gmail.com

Srinivasan.T,
Assistant Professor,
Department of Computer Science and Engineering.
Sri Venkateswara College of Engineering,
Sriperumbudur, India.

Abstract: One of the key design issues for the next generation routers is the IP Lookup mechanism. IP address lookup is the challenging because it requires a longest matching prefix lookup. It is compounded by increasing routing table size, increased traffic and migration to IPv6 addresses. Existing solutions like BSD radix Tries, scale poorly when traffic in router increases or when employed for the IPv6 addresses. In our paper we describe a parallel computation models based lookup for the routing table. This mechanism provides lookup for a maximum of n*16 IPv6 addresses simultaneously, where n represents the number of processors. We propose a parallel binary trie search technique for the lookup to facilitate concurrent insertions and searching on the routing table. Using the proposed technique a router can achieve a much higher packet forwarding rate and throughput.
Keywords: IPv6, Parallel Computation, IP Lookup

1. INTRODUCTION

The Internet is becoming omnipresent and the emergence of multimedia networking applications result in the increase in the network traffic. As a consequence the new version of Internet Protocol 6 (IPv6) is being standardized will replace the current 32 bit addresses with a virtually inexhaustible 128-bit address space.

For every incoming packet, a router must perform a IP lookup in the routing table to determine the next hop of the packet by its destination address. The IP lookup in a route may decompose on to two steps. Firstly the router finds a set of routing entries that match the beginning IP destination address of the incoming packet. Secondly, among this set of matched routes the router selects the one with the longest prefix. This is the route to forward the

packets. This process of finding the longest prefix match is one of the bottlenecks in packet forwarding process.

Most of the IP address lookup schemes developed so far have dealt with IPv4 routing mechanism efficiently to a large extent. However, when the IPv6 routing protocol is introduced the problem of routing millions of communication packets every second becomes labyrinth.

In this paper we propose a new way to the IP route lookups based on the parallel computation model (n-PCM). This model is capable of providing lookups for the maximum of n*16 IPv6 addresses at a time (n representing the number of processors). This is achieved my employing n processor at a time. Using the parallel binary trie search there has been a significantly reduction in the average number of memory accesses and increase in the packet forwarding rate.

The rest of the paper is organized as follows. Section 2 describes the drawbacks in the existing approaches to IP lookups. Section 3 details more on the proposed methodology. Subsection 3.1details on the parallel computation model (n-PCM) and subsection 3.2 details on parallel binary search over the routing table. Finally we conclude in Section 4.

2. RELATED WORK

In this section, we study some existing approaches to IP lookups and their problems. We discuss approaches based on Trie based schemes and Hashing.

2.1 Trie Based Schemes

The basic scheme, which inspired proposing some new approaches, is radix trie or binary trie[2]. Binary Trie is a simple data structure, which represents strings with paths from root to the leaf or any node in the middle. Binary Trie allow the representation of arbitrary length prefixes, they have the characteristic that long sequences or one-child nodes may exist. We are forced to inspect the bits where no actual branching decision is made. If W is the length of the address, the worst-case time in the basic implementation can be shown to be $O(W^2)$.

Current implementation to improve the time and space performance resulted in Patricia Trie (or BSD Trie). It modifies the trie by compressing the path and eliminating the unnecessary nodes. Patricia Trie makes a lot of sense when the binary trie is sparsely populated; but when the number of prefixes increases the prefixes increases and the trie gets denser, using the path compression has got little benefit. Despite this the above implementations requires up to 32 or 128 costly memory accesses (for IPv4 or IPv6 respectively). The above Trie structures also need large storage structure [2].

Multibit tries to improve the lookup speed (IPv4 addresses), by inspecting several bits at a time with respect to binary tries. It is only a constant factor in length dimension [5]. Hence multibit tries scale badly to the longer Ipv6 addresses.

2.2 Hashing

The alternative approach for IP routing table search is Hashing. Unlike the tree structure, hashing stores whole addresses and the assumption is that the number of addresses is limited and can be kept in cache. This method does not take advantage of the hierarchical address structure and is not well suited for internet applications.

3. PROPOSED SCHEMES

3.1 Parallel Computation Model (n-PCM)

In our parallel computation model (n-PCM) we try to increase the number of processors to increase the packet-forwarding rate. Increase in the number of processors is not the only dimensional growth; it clearly shows the basic building process of the next generation routers to be more complex and powerful as well.

The multiprocessor architecture can be logically viewed as a hierarchical structure with two levels of control. At the top level the multiprocessor is controlled by the front-end systems, which serves as an interface to the incoming packets and co-ordinates the activity of the group.

At the lower level the processor receiving the packets from the front-end system does the lookup operation.

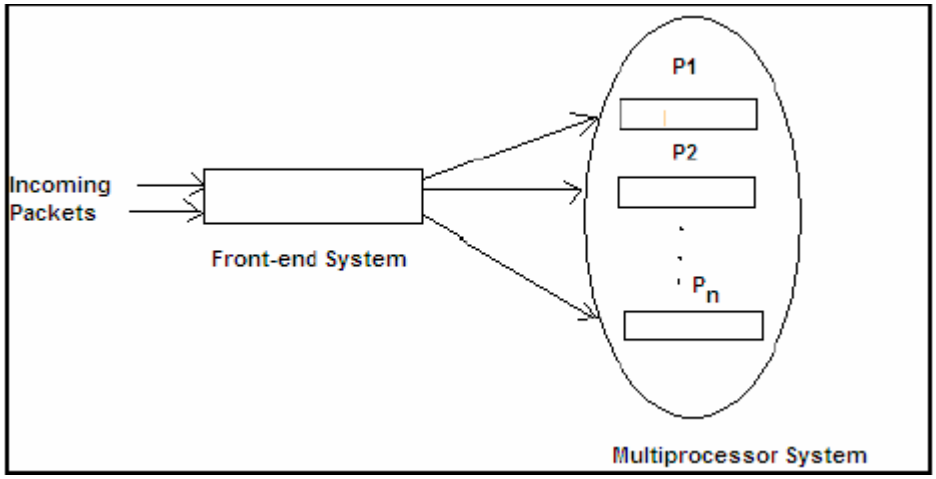

Figure 1. n-PCM Model

In this n-PCM model the front-end system distributes the incoming packets to the lower level processors for the lookup operation. The n-PCM model abstracts out the interconnection network in terms of few parameters like overhead involved in transmitting / receiving packets. Hence this model does not suffer from performance degradation when ported from one network to another.

n-PCM model does not make any unrealistic assumptions.

The parameters involved in this model are:

Delay: This is the upper bound value incurred in communicating the message containing the word from one processor to another.

Interval: It is the minimum gap between consecutive message transmission or message receptions at the processor.

Overhead: It is the length of time the processor is engaged in the transmission or reception of message. During this process the processor is not allowed to perform any other operation.

The front-end system takes care of maintaining the routing table. Low-level processors

perform uniform memory access over the routing table in the front-end system for the lookup operation. In the n-PCM model the front-end system distributes the packets to the lower level processor and takes care of insertions in the routing table.

The parameters mentioned above are statically defined based on the processing capacity of the processor. The algorithm for the n-PCM model is given below.

Function n-PCM (IP Packets)
Initialize delay, interval, and overhead.
Set the starting time to zero.
For each packet **do**
Push the packet into the FIFO of one of the lower level processor
Time= 2*Overhead + Delay
Wait for the Interval period.
End Loop
End Function

Table.1
Memory module allocation

Lookup Unit #	Bits 63,64,65 and 66
1	0001
2	0010
3	0011
---	---
15	1110
16	1111

The complete parallel lookup mechanism is shown in Figure.2.

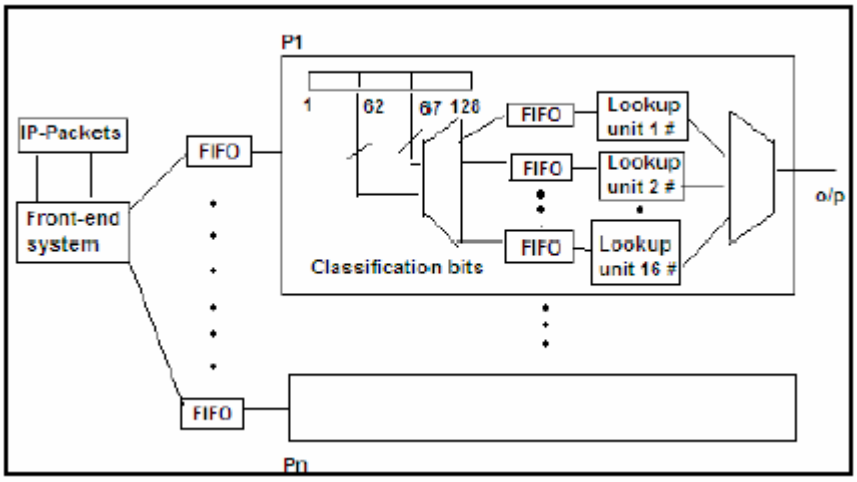

Figure 2. n-PCM Lookup Strategy

3.2 Parallel Binary Trie Search

Based on the IPMA survey reports the prefixes stored in a routing table can be classified into several flows averagely depending on certain bits of them [1]. We use bits 63,64,65 and 66 (called ID bits) to classify the packets in the routing table into 16 categories as shown in Table 1. Then the search for the longest matching prefix for this incoming prefix is performed using parallel binary trie technique. The algorithm for parallel lookup is given below.

```
Function Lookup ( )
    For each Processor do in parallel
    While (FIFO not empty) do
    Pop the packet from the local FIFO.
    Use the ID bits of Destination Address to classify them.
    Push IP Address into the FIFO of the corresponding
    lookup unit.
    For each Lookup unit do in parallel
        While (FIFO not empty) do
            Pop an address from
            the local FIFO.
            Use parallel binary trie search to find
            BMP.
            Push the Next-hop Address to output
            cache.
        End While
      End Loop
    End While
  End Loop
End Function
```

The idea behind the lookup system is to perform a binary search on the hash tables organized by the prefix length [7]. The algorithm for the parallel binary search is given below.

```
    Function ParallelBinarysearch (A)
    Initialize the search range S as the whole array L
    Initialize the number of processors.
    Initialize the search sequence for each processor.
    Do in Parallel
    Let i correspond to the middle level in S
    Extract the first L[i].length bits of A to A'
    Search (A', L[i].hash) // search for A'
    If found then
        set S= lower half of S
    Else
        set S= upper half of S
    While S is not a single entry
    End Function
```

In this parallel binary search technique the binary search is modified into an (N+1) array search. At each stage of the algorithm the sequence is split into N+1 subsequences of equal length. The N processors simultaneously probe the elements at the boundary between successive subsequences. For instance, we have prefixes P1=0, P2=00, P3=111.Suppose the prefix we search for is 111, the search starts from the prefix P2. Since the P2 does not match, the search algorithm should proceed further till a match is found. But since there is no indication of the path to be taken, this approach fails to arrive at the best matching prefix.

To solve this problem, we store markers, which are the first N bits of the prefix to be inserted, where N is the length of the prefix at that level. These markers are stored at the levels, which would be reached while searching for a matching prefix, with prefix length shorter than the prefix being inserted. So in the above example, we add a marker entry 11, to indicate P3, at the level containing P2 to direct the binary search to the lower half of the routing table for a better match. Also the number of markers to be stored for each prefix is an important issue to be handled. Inefficient usage of markers will degrade the performance of the binary search and also increase the memory consumption. So an efficient approach to store markers is proposed in the next section. This approach enables efficient storage of markers when compared to the marker requirement stated in [7].

3.2.1.Marker Storage Algorithm.

Our approach to store markers for a prefix is based on the bit pattern of the prefix. As already explained it suffices to store markers in those levels that would be visited by the binary search and whose length is shorter than that of the prefix to be that is inserted. The algorithm for our marker storage is given below.

Function MarkerStore (prefix)
Initialize count = 0, Level = 0
Initialize bin = 0000
// Scans the prefix to find the length of prefix
count = Length(prefix)
// Find the binary of the length ‘count’
bin = Binary (prefix)
// Scan this ‘bin’ for number of 1’s
count = Scan (bin)
For I = count -1 loop till I > 0 **do**
    Level = Level + (2 ^ I)
    Add a marker entry for the prefix in the
    level indicated by “Level”
    Search for BMP of marker and store it in the BMP field of
    the marker.
Next I
End Function

For instance, if the prefix is P1=11001, then it should be inserted in the level 5 (0101 in binary). The number of 1s in the binary format of 5 is 2. Based on the above algorithm, the Level would become 4, which is the only level that would be reached during the search for a

prefix of length 5 and whose length is smaller than 5. Hence a marker is added to level 4 for the prefix whose length is 5.

This marker storage algorithm is efficient as the number of potential parents for storing the markers is optimized in comparison to the existing approach stated in [7]. Also, since the number of markers stored for the prefix to be inserted is reduced, the overhead of marker insertion during the prefix insertion process is also reduced.

Consider the prefixes P1=1, P2=00, P3= 111. Now according to the marker storage logic explained above, marker for P3 will be stored at the level containing P2.Now when a prefix 110 is to be searched, the search starts at P2 and proceeds to the lower half of the table since a matching marker is available. But the best prefix match is available in the upper half of the hash table. Such a marker misleads the searching algorithm. A solution for this misleading marker problem has been proposed in [7].

A new field called BMP is stored for each marker. This field contains the best matching prefix of that marker. When we use the misleading marker and fail to arrive at the best matching prefix, the value in the BMP field of the latest marker arrived at is the longest matching prefix for the destination address.

3.2.2. Insertion in Binary Search

As the marker storage is optimized, the insertion algorithm is very efficient. Moreover the amount of memory needed for storing the marker is also reduced. Even in the worst case the number of markers needed for a particular prefix is less than $\log_2 128$. The insertion algorithm is given below.

Function Insertion
Use **binary search** to search for the level
where prefixes of length equal to the length
of the prefix to be inserted is available.
Then insert prefix into the hash table.
Call the marker store function to insert the
marker in the potential parents along with
it’s BMP.
End function

4. CONCLUSION

We have proposed an algorithm for the next-generation routers using the parallel computation model (n-PCM) and parallel binary trie search for the best matching prefix in the routing table. This approach is extremely efficient that scales with the logarithm of address size.

By classifying the prefixes in the routing table and introducing multiprocessor concept into the next-generation routers, we are able to provide a maximum of n*16 simultaneous lookups. This strategy of parallel computation in lookup drastically increases the packet-forwarding rate of a router even for Ipv6 address formats. Additionally parallel binary trie search over the routing table facilitates searching in parallel. Hence the proposed method performs better than the existing algorithms for the lookup of Ipv6 addresses.

5. REFERENCES

[1]. Srinivasan.T.et al, “A High Performance Parallel IP Lookup Technique Using Distributed Memory Organization”, Proceedings of the IEEE International Conference on Information Technology (ITCC'04), Lasvegas, USA, April 2004.

[2]. Jonathan Chao, H., “ Next Generation Routers”, Proceedings of the IEEE, Vol.90, No.9, September 2002.

[3]. Bulter Lampson, Venkatachary Srinivasan, George Varghese, “ IP Lookups Using Multiway and Multicolumn Search”, IEEE/ACM Transactions on Networking, Vol.7, No.3, June 1999.

[4].RuizSancgez,M.A.,Biersack,E.W.,Dabbous,W.,“ Survey and Taxonomy of IP Address Lookup Algorithms” IEEENetwork,Vol.15,Issue 2,pp.8-23,March-April 2001.

[5]. Suri,S.,Varghese,G.,Warkhede,P.R., “Multiway Range Trees:Scalable IP Lookup with fast updates”,Tech.Rep.99-28,Washington Univ,1999.

[6]. Srinivasan,V.,Varghese,G.,“Fast IP Lookups using Controlled Prefix Expansion”,ACM TOCS,vol.17,pp,1-40,Feb.1999.

[7]. Waldvogel, M.m Varghese, G., Turner, J., Plattner, B.,“Scalable High Speed IP Routing Lookups”, SIGCOMM Proceedings, Cannes, France, 1997.