A HIGH-PERFORMANCE IPV6 LOOKUP ENGINE ON FPGA

Thilan Ganegedara, Viktor Prasanna

Ming Hsieh Dept. of Electrical Engineering
University of Southern California
Los Angeles, CA 90089
email: {ganegeda, prasanna}@usc.edu

ABSTRACT

We present a routing table partitioning based solution for a high-performance IPv6 packet lookup engine on Field Programmable Gate Arrays (FPGAs). Based on the statistics collected from real-life backbone IPv6 routing tables, we propose a partitioning algorithm that creates both disjoint and balanced prefix groups. For each partition a *range tree* is built to perform IPv6 lookup. These range trees are mapped onto independent pipelines on FPGA such that for a single IPv6 lookup, only one partition is active. This yields high dynamic power efficiency via selective stage memory enabling. The balanced partitioning enables us to exploit the memory layout of the FPGA to align the pipeline with the on-chip memory blocks for enhanced performance and resource usage. Post place-and-route results on a state-of-the-art FPGA platform shows that a throughput of 200+ Gbps can be achieved for a 1 million entry IPv6 routing table.

1. INTRODUCTION

With the rapid growth of the Internet, the addressing capabilities of IPv4 has become inadequate [1]. This has given rise to the use of the next version of IP addressing, IPv6. Even though IPv6 has the addressing capability, performing packet forwarding can be challenging due to 1) memory increasements and 2) supporting high lookup rates operation at high-speed, especially in core routers. Employing the existing solutions proposed for IPv4 and tweaking them for IPv6 may not be straightforward [2]. In addition, such methods may cause the memory footprint of the IPv6 routing tables to dramatically increase which can potentially exceed the memory available on current networking platforms.

Recently, there have been many efforts to devise memory and/or resource efficient IPv6 lookup engines that deliver high-performance. In [3], the IPv6 engine was realized using a combination of Content Addressable Memory (CAM) and Programmable Logic Device (PLD), in which the prefix search was performed by the CAMs while the PLD aggregated the results from the CAMs (priority encoder) to yield Longest Prefix Match (LPM) [4]. Although the routing table was distributed across multiple CAMs based on the subnet mask length, all the CAMs were required to perform the search, thus increasing the amount of computations performed per lookup. The architecture yields lower throughput due to the lack of pipelining and the delays introduced by the priority encoder. In [5,6], tree based solutions using routing table partitioning were introduced. The lack of disjointness in prefix groups require these solutions to search all the partitions to perform LPM, which results in higher power consumption.

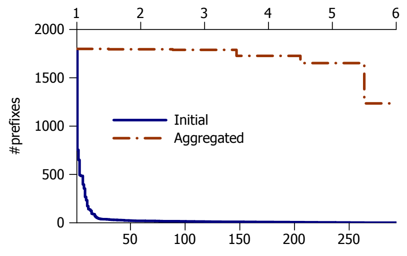
In this work, we propose a solution to perform *wire-speed* and large scale IPv6 packet forwarding on FPGA. We devise an algorithm that partitions a routing table into a set of disjoint, yet balanced, prefix groups, which enables us to search only one partition to find the matching prefix for a given packet header. This feature can be used to disable the inactive partitions in order to reduce the total power consumption. Even though routing table partitioning has been studied [2,5,6], partitioning and early identification of the partitions has not been exploited to enhance power efficiency of packet forwarding engines for IPv6. Based on this partitioning, we propose a range tree [6] based solution to achieve high packet forwarding rates. We evaluate our solution on a Xilinx Virtex 7 FPGA. Post place-and-route results show that a throughput of 200+ Gbps can be achieved for a 1 million entry IPv6 routing table. Further, compared with state-of-the-art Ternary Content Addressable Memory (TCAM) the proposed lookup engine is $50\times$ power efficient.

2. PARTITIONING AND RANGE TREE BASED IPV6 LOOKUP

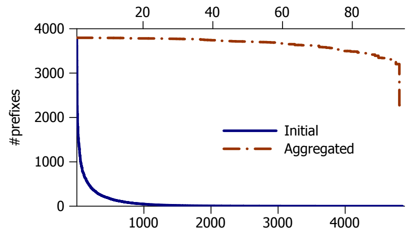
2.1. Datasets

The real-life IPv6 routing tables obtained from RIPE Routing Information Service (RIS) project [7] shows that the

This work is supported by the United States National Science Foundation under grant No. CCF-1116781. Equipment grant from Xilinx Inc. is gratefully acknowledged.



(a) $p = 20, n_i = 294, n_a = 6$



(b) $p = 20, n_i = 4854, n_a = 94$

Fig. 1. The effects of partitioning and aggregation of partitions. Figure 1(a) show the partitioning of real routing table and Figures 1(b) shows the partitioning of the synthetic routing table. Note that the upper and lower X-axis corresponds to the aggregated and initial numbers of partitions.

real-life IPv6 routing tables are small (~10K prefixes), hence, the full effect of a backbone routing table cannot be observed using them. For this reason and to evaluate the scalability of the proposed solution, we generated large IPv6 routing tables using an extended version of FRuG [8]. Using FRuG, the prefix distribution and the structure of the seed routing tables are preserved when generating the synthetic routing tables.

2.2. Disjoint Grouping of Prefixes

In this work, we consider partitioning using the initial bits of the prefixes. The initial bits based partitioning is essentially dividing the address space into multiple disjoint sections and considering subsets of prefixes belonging to each section as a smaller routing table. For example, if p bits are used there can be as many as $O(2^p)$ partitions. Depending on the prefix distribution of a given routing table, there may not be 2^p partitions and some partitions may be very small while some are very large. Also, when partitioning this way, the prefixes with length less than p needs to be expanded. Our analysis of real routing tables indicate that prefixes of length 25 and shorter constitute only 1% of routing table. Hence, the effect of shorter prefixes is negligible.

2.3. Balanced Partitioning

In order to demonstrate the results of the considered partitioning scheme, we use the real [7] and synthetic routing tables. Moreover, we consider partitioning of both real and synthetic routing tables to show that synthetic routing tables are not biased in any way when it comes to prefix distribution.

Algorithm 1 Aggregated partitioning

```
1: procedure AGGREGATE($initial, aggregate, maxsize$)
2:     $initial.sort(descending\ count)$;
3:     $bool\ done \leftarrow false$;
4:     while $true$ do
5:         $done \leftarrow true$;
6:         for $ipart$ in $inital.begin$ to $initial.end$ do
7:             if $!(ipart.taken)$ then
8:                 $partition\ newpart$;
9:                 $newpart.members \leftarrow ipart.members$;
10:                $newpart.size = ipart.members.size$;
11:                $ipart.taken \leftarrow true$
12:                $done \leftarrow false$;
13:                $break$;
14:        if $done$ then
15:            $break$;
16:        for $epart$ in $inital.end$ to $initial.begin$ do
17:            if $!epart.taken$ then
18:                if $newpart.size + epart.size \leq maxsize$ then
19:                    $newpart.members \leftarrow epart.members$;
20:                    $newpart.size += epart.members.size$;
21:                    $epart.taken \leftarrow true$;
22:                else
23:                    $break$;
24:            $aggregate.push\_back(newpart)$;
     return $aggregate$
```

In order to address the variable partition size issue, we devised Algorithm 1. The algorithm essentially combines larger partitions with smaller partitions in order to create partitions of similar sizes. This step preserves the disjoint property of the prefix groups, which enables us to retain the benefits of the same. Figure 1 shows the effect of using Algorithm 1. The number of bits used for partitioning is denoted by p, and n_i and n_a stand for initial and aggregated number of partitions, respectively.

3. HARDWARE ARCHITECTURE

The basic building block of the hardware architecture is a linear pipeline. Each stage consists of a Processing Element (PE), a stage memory and stage registers. We use the implicit range tree approach (storing the lower bound value of each range) for the hardware engine since a packet needs to traverse the entire pipeline for the search to complete, regardless of whether the search terminated in an intermediate stage. For high performance, we further pipelined a PE such that the key comparison stage and the memory access stage are pipelined internally. This significantly improves the performance while adding little logic overhead.

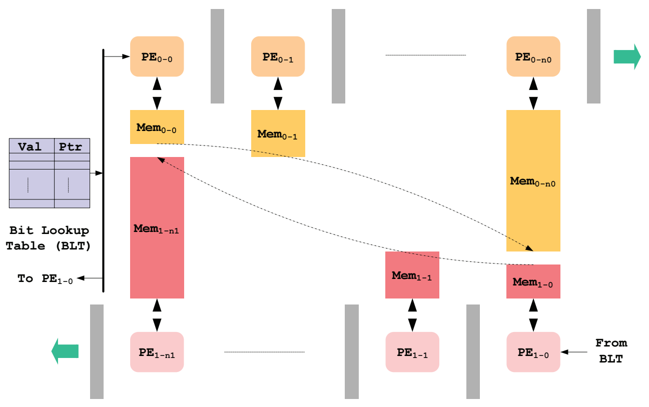


Fig. 2. Pipelined IPv6 lookup architecture on FPGA. The two pipelines shown are aligned in such a way that the stage memory of the range trees are aligned along BRAM columns on FPGA for improved resource usage.

The architecture is illustrated in Figure 2. The figure shows how the sub-pipelines for different partitions are arranged on the FPGA physically. Each sub-tree has an exponential memory distribution since the range tree discussed here is complete [6]. Therefore, each level k has exactly 2^k nodes, with exception for the last level. The block memory on FPGA is arranged as vertical columns along the chip and the pipeline stages can be aligned along these vertical columns to take advantage of the resource distribution of the chip. The Xilinx PlanAhead [9] tool was used to draw the pipeline layout on the chip.

4. IPV6 LOOKUP ENGINE PERFORMANCE

4.1. Throughput

We report the performance of the architecture presented in Figure 2. The architecture was implemented on a Xilinx Virtex 7 X1140V FPGA and the post place-and-route results are reported here. We used both distributed and block RAM available on FPGA to facilitate higher scalability. The considered chip has 84 Mbit total memory available. The proposed solution scaled up to a 1 million entry IPv6 routing tables while sustaining high lookup rates. We exploited the dual-ported feature of both distributed and block RAM on FPGA to enhance the throughput by processing two packet headers at a time by a single pipeline stage (via stage memory sharing). From Figure 3, it can be seen that the proposed architecture can operate at high lookup rates lead ing to throughputs of 200+ Gbps per pipeline for minimum size (64 Byte) packets.

The decrease in performance is due to larger stage memory when using larger routing tables. With increasing pipeline depth, stage memory size increases exponentially. In order to generate large stage memory blocks, on FPGA, several BRAM blocks need to be combined. Due to the column-wise arrangement of BRAM on FPGA, creating larger stage

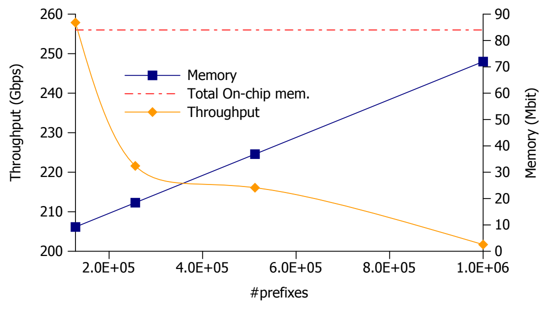


Fig. 3. Throughput and memory footprint of the hardware lookup engine for increasing routing table size. The dotted line denotes the maximum on-chip memory available on the Virtex 7 X1140T FPGA

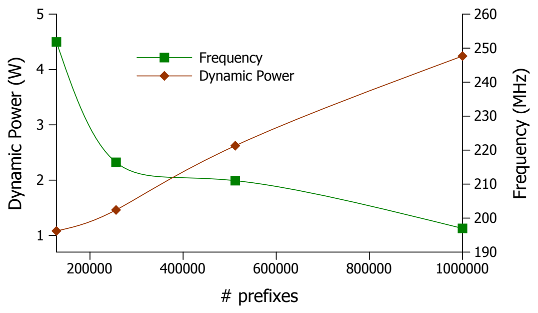


Fig. 4. Power consumption of the IPv6 lookup engine for increasing routing table size

memory introduces penalties in access time. The effect of that can be seen in Figure 3.

4.2. Power

Further, due to partitioning of the routing table, we were able to limit the power consumption of the architecture significantly. For the hardware approach, we limited the number of partitions used as the gains achieved by increasing the partitions has diminishing returns and when using larger number of bits for the initial partitioning of the routing table, the BLT becomes significantly larger causing excessive memory consumption. Since only one partition needs to be searched, the dynamic power consumption of the architecture is significantly reduced as opposed to searching using the full tree [5]. In Figure 4, we report the dynamic power consumption which includes the power consumed by logic, memory and routing measured using the Xilinx XPower Analyzer tool.

The lookup rate of the proposed IPv6 lookup engine is on par with that of [5,10], however, the power savings achieved in this work via partitioning is not possible with those solutions. Further, the scalability of the proposed solution is higher than that of both [5,10]. For example, our architecture is able to support up to 1 million entry IPv6

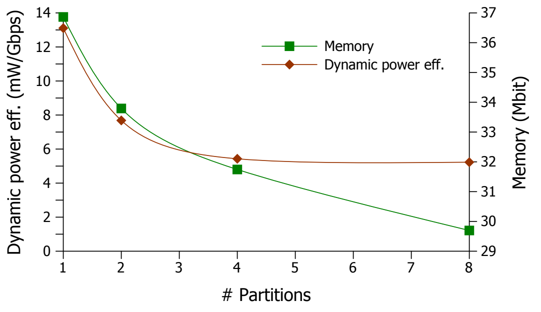


Fig. 5. Power and memory requirement variation with increasing number of partitions

routing tables, which is not possible with the two solutions in comparison, without the use of large external memory.

4.3. Benefits of Partitioning

In order to assess the benefits of the disjoint and balanced prefix grouping, we used a 512K entry routing table and evaluated performance for different number of partitions. Figure 5 summarizes the results of our experiments. As it can be seen, the power efficiency, measured in power per unit throughput, improves with increasing number of partitions. This is due to selective enabling of lookup pipelines which is facilitated by partitioning. Compared with a state-of-the-art TCAM [11] operating at 360 MHz and consuming 2W/Mb, our calculations indicate that the proposed solution is $50\times$ power efficient, on the average.

The tapering off of the power savings is due to the increased routing and logic power with the increasing number of partitions. The tradeoff is between the increased logic and routing power, and power savings from selective memory enabling. When more partitions are instantiated, the amount of non-memory type resources (logic, interconnect, registers, etc.) that needs to be "clocked" increases.

Another aspect of partitioning is the memory savings. Since the first p bits of the prefixes are already processed by the initial bit lookup table (BLT), those bits need not be stored in the range tree nodes. In our experiments, the value of p ranged from $0, 6, 10, 14$ for the number of partitions $1, 2, 4, 8$, respectively, and the memory requirement decrease is highlighted in Figure 5.

5. CONCLUSION

In this work, we proposed a range tree based solution for IPv6 lookup that is suited for both software and hardware platforms. We devised a tunable partitioning algorithm that forms a set of disjoint and balanced size subsets of prefixes, given a routing table. This enabled us to improve power and memory efficiency on hardware platforms. The solution was tested on a state-of-the-art Field Programmable Gate Array (FPGA) platform and the post place-and-route results show that the proposed solution is able to operate at $200+$ Gbps throughput, for a 1 million entry IPv6 backbone routing table. Further, the experimental results showed that the proposed solution with 8-way partitioning is able to deliver $65\times$ and $2.5\times$ better power efficiency compared with Ternary Content Addressable Memory and baseline architecture (i.e., no partitioning), respectively.

6. REFERENCES

[1] A. R. for Internet Numbers (ARIN), "IPv4 address exhaustion," https://www.arin.net/announcements/2011/20110203.html.

[2] M. Ruiz-Sanchez, E. Biersack, and W. Dabbous, "Survey and taxonomy of IP address lookup algorithms," *Network, IEEE*, vol. 15, no. 2, pp. 8 –23, 2001.

[3] T. Hayashi and T. Miyazaki, "High-speed table lookup engine for IPv6 longest prefix match," in *Global Telecommunications Conference, 1999. GLOBECOM '99*, vol. 2, 1999, pp. 1576 –1581.

[4] Wikipedia, "Longest prefix match," http://en.wikipedia.org/wiki/Longest_prefix_match.

[5] H. Le and V. Prasanna, "Scalable tree-based architectures for IPv4/v6 lookup using prefix partitioning," *Computers, IEEE Transactions on*, vol. 61, no. 7, pp. 1026 –1039, July 2012.

[6] P. Zhong, "An IPv6 address lookup algorithm based on recursive balanced multi-way range trees with efficient search and update," in *Computer Science and Service System (CSSS), 2011 International Conference on*, June 2011.

[7] RIPE, "Ripe routing information service (ris)," http://www.ris.ripe.net/.

[8] T. Ganegedara, W. Jiang, and V. Prasanna, "Frug: A benchmark for packet forwarding in future networks," in *Performance Computing and Communications Conference (IPCCC), 2010 IEEE 29th International*, Dec. 2010, pp. 231 –238.

[9] Xilinx, "Planahead design and analysis tool," http://www.xilinx.com/tools/planahead.htm.

[10] M. Bando, Y.-L. Lin, and H. J. Chao, "Flashtrie: Beyond 100-gb/s IP route lookup using hash-based prefix-compressed trie," *Networking, IEEE/ACM Transactions on*, vol. 20, no. 4, pp. 1262 –1275, Aug. 2012.

[11] H. Yu, "A memory- and time-efficient on-chip tcam minimizer for IP lookup," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pp. 926–931.