A Pipelined IP Address Lookup Module for 100 Gbps Line Rates and beyond

Domenic Teuchert and Simon Hauger

Institute of Communication Networks and Computer Engineering (IKR)
Universität Stuttgart, Pfaffenwaldring 47
70569 Stuttgart, Germany
{dteuch,hauger}@ikr.uni-stuttgart.de

Abstract. New Internet services and technologies call for higher packet switching capacities in the core network. Thus, a performance bottleneck arises at the backbone routers, as forwarding of Internet Protocol (IP) packets requires to search the most specific entry in a forwarding table that contains up to several hundred thousand address prefixes. The Tree Bitmap algorithm provides a well-balanced solution in respect of storage needs as well as of search and update complexity. In this paper, we present a pipelined lookup module based on this algorithm, which allows for an easy adaption to diverse protocol and hardware constraints. We determined the pipelining degree required to achieve the throughput for a 100 Gbps router line card by analyzing a representative sub-unit for various configured sizes. The module supports IPv4 and IPv6 configurations providing this throughput, as we determined the performance of our design to achieve a processing rate of 178 million packets per second.

1 Introduction

The ongoing increase of the Internet traffic necessitates to upgrade the capacity of the backbone network continuously. As a consequence, 100 Gbps Ethernet will be deployed in 2010 [1] requiring core routers to perform 150 million IP address lookups per second and ingress port. For each lookup operation, the router's forwarding engine (FWE) has to determine the most suitable next-hop router by using the packet's destination IP address as a key for searching a forwarding table. This table lookup is a complex task in software as well as hardware, since it requires to find the longest matching prefix (LMP) as the most specific entry. Moreover, the forwarding table holds several hundred thousand entries and grows even further [2]. The requirements of good scalability regarding 128-bit long IPv6 addresses and of efficient table update processing lead to additional difficulties.

Facing these requirements, today's high-speed routers typically use specialized hardware to implement the FWE. Ternary Content Addressable Memories (TCAMs) can perform one table lookup per clock cycle. However, TCAMs scale unfavorably with table and key sizes, and they consume significantly more power than standard memory. Therefore, algorithmic lookup methods [3,4,5,6,7] are increasingly implemented in specialized hardware modules. Among these methods,

© Springer-Verlag Berlin Heidelberg 2009

the Tree Bitmap algorithm [5] implemented in Cisco's CRS-1 core routers [8] offers a very smart and balanced trade-off between memory usage, lookup performance, and update complexity.

In this paper, we present the design and prototypical implementation of a flexible and fully pipelined hardware lookup module based on the Tree Bitmap algorithm. In order to investigate whether and with what effort algorithmic IP lookup modules can be realized in hardware for future line speeds of 100 Gbps and beyond, we clearly focused on maximum throughput and resource efficiency, as opposed to other published Tree Bitmap implementations [5,9]. By processing all packets in a pipeline, our design effectively performs one IP address lookup in each clock cycle. Additionally, it supports high-speed, non-blocking updates of the forwarding table. By adjusting several configuration parameters, one can also adapt the module to various requirements concerning performance, resource utilization, and memory parameters.

We tested the module's functionality on a hardware platform based on a Field Programmable Gate Array (FPGA). Due to the module's deterministic behavior, we can show that the processing logic of our design can be utilized for 100 Gbps line speeds using current FPGA devices. Additionally, we studied how many pipeline registers are needed per functional block of our lookup module, in order to achieve a desired throughput with different configurations.

In the following section, we describe a typical high-speed router architecture and give a short review of the IP address lookup problem as well as different solutions with a focus on the implemented Tree Bitmap algorithm. In Sect. 3, we detail on our hardware realization. Finally, we discuss the achieved simulation and synthesis results in Sect. 4, before we conclude this paper.

2 IP Routing

IP Routing and its underlying specifications determine the constraints for a qualified architecture and IP lookup method of a high-speed FWE. In this section, we introduce the primary factors that have directly affected our hardware design.

2.1 High-Speed Router Architecture

Routers generally perform two basic functions: (a) Exchanging topology information with other routers in order to build a routing table and (b) forwarding packets from ingress to egress ports based on this table. The former function is run on the control plane, whose timing requirements allow its implementation as part of a software-controlled processing entity, commonly termed the router's slow path. The latter function belongs to the data plane, which has to fulfill strict speed constraints. Thus, packet forwarding, as part of the router's fast path, is implemented using specialized hardware on core routers.

Typically, a decentralized architecture as shown in Fig. 1 is used to achieve high scalability in terms of the number of interfaces. Several line cards are connected to a switch fabric, each accommodating one or more physical interfaces

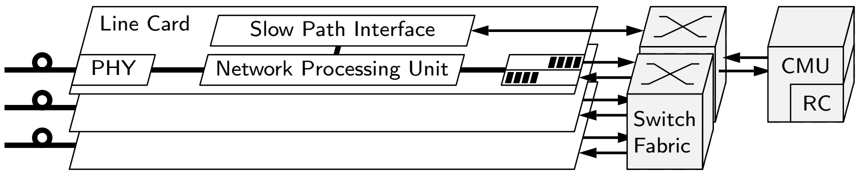


Fig. 1. Architecture of a typical core router

(PHY), packet buffers, a slow path interface, and a network processing unit (NPU). The slow path interface hands routing protocol messages over to the route controller (RC) within the control and management unit (CMU) and downloads the updated forwarding table entries from there. The NPU performs the fast path functions including packet classification, table lookup, and the associated packet modifications. The IP lookup module we present in this paper is intended to be part of such an NPU and leverages the common approach to increase the throughput by pipelining the processing tasks within the NPU.

2.2 IP Address Lookup

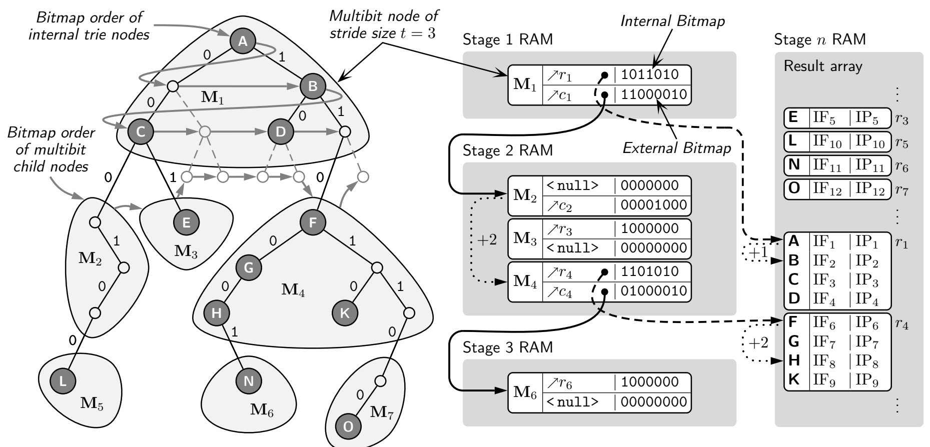
IP addresses consist of a prefix identifying a particular network and a host part indicating the destination within the respective network. To mitigate the increasing address shortage, Classless Inter-Domain Routing (CIDR) has been introduced, which allows prefixes to take any number of IP address bits in order to flexibly aggregate contiguous prefixes to a single entry in the forwarding table. As IP addresses are not strictly hierarchically distributed, the resulting address space fragmentation leads to exception entries—also denoted as more specifics—which account for 40–60% of the total forwarding table size in the Internet backbone [2]. It is thus necessary to find the longest matching prefix (LMP) for a precise forwarding decision. The large sizes of backbone forwarding tables in combination with CIDR make the LMP search a complex task that, moreover, has to be performed at a high frequency in core routers. Therefore, specialized hardware implementations are the method of choice.

Generally, two solutions exist: (a) TCAMs that accomplish a parallel LMP search effectively in only one clock cycle and (b) numerous algorithmic lookup methods. Although TCAMs are often applied in commercial routers, [10] shows that algorithmic solutions based on multibit tries allow to store larger tables on a given chip size. Furthermore, TCAMs have a high power dissipation per memory bit, which finally makes algorithmic lookup methods the best candidate to meet the future demand on fast IP address lookups.

Apart from hash-based methods, most algorithmic solutions are based on a binary search tree, which is referred to as *trie* in this context. Using a trie, the search space is significantly reduced in each step, but storing the trie structure causes a certain memory overhead. To address this issue, some solutions [7,3,5] propose to use a compression method to pack several trie nodes in one compacted multibit node that can be processed in a single search step. The LC-trie [7] and the Lulea algorithm [3] achieve a very good scalability of memory demand against table size but do not support incremental table updates. The Tree Bitmap algorithm, which we have implemented, uses a different compression scheme that results in short and deterministic update times while preserving the properties of fast prefix search and memory efficient table storage.

2.3 Tree Bitmap Algorithm

Like other trie-based methods, the Tree Bitmap algorithm utilizes multibit nodes that each cover, as illustrated in Fig. 2(a), a subset of the search trie according to the defined stride size t. During a lookup operation, the trie is hence traversed multibit node by multibit node. For each visited multibit node, a t-bit fragment of the destination IP address is used to check whether a more specific prefix (filled gray circle) than the currently known exists in this node, and whether a respective child node exists to continue the lookup. If no suitable child node exists, the last found prefix is the LMP.



(a) Trie structure (b) Memory mapping

Fig. 2. IP lookup table in Tree Bitmap representation

For the given example of stride size $t = 3$ in Fig. 2 and the IP address `195.•.•.•` ($= 11000011_2\ldots$), the most specific existing prefix in the first multibit node is **B** (`1*`), and the suitable child node is $\mathbf{M}_4$ (going right (`1`), right (`1`), left (`0`) through root node $\mathbf{M}_1$). Within multibit node $\mathbf{M}_4$, the most specific existing prefix is now **H** (= `11000*`). As no further suitable child node exists (when going left (`0`), left (`0`), left (`0`) through $\mathbf{M}_4$), **H** is the LMP.

To store the trie structure efficiently, the Tree Bitmap algorithm uses two bitmaps and two pointers for each multibit node. The Internal Bitmap (cf. Fig. 2(b)) represents all $2^t - 1$ possible prefixes associated with the internal trie nodes of the multibit node. A set bit corresponds to an existing prefix in the forwarding table. The address of the actual next-hop entry, which is saved in a dedicated memory, is the sum of the result pointer r and an offset. This offset can be easily determined by counting the number of set bits left of the bit position that corresponds to the found prefix. In our example, **B** is the most specific existing prefix in $\mathbf{M}_1$, which is associated with the third bit from left in the Internal Bitmap. As there is only one set bit in the lower-order positions, the offset is 1. The External Bitmap similarly represents all 2^t possible child nodes. Again, a set bit corresponds to an existing child node, and an offset is computed by the number of set lower-order bits.

Beyond this basic algorithm, [5] proposed several optimizations which predominantly aim for smaller memory word widths and a higher memory utilization: One option is the use of an *Initial Array* to process some of the multibit trie's relatively small top levels in a single memory access. Besides this optimization, we also implemented *End Node* handling, which eliminates a large number of almost empty nodes by using an otherwise unused External Bitmap as an extension of the Internal Bitmap. The third optimization that has been adopted in our design is called *Split Tree Bitmap* and nearly halves the memory demand of the multibit nodes.

Previously published hardware implementations based on the Tree Bitmap algorithm are the reference implementation of [5] and the low-cost oriented solution of [9]. The former implementation achieves 25 million lookups per second using a four times replicated forwarding table stored in external DRAM and an Application-Specific Integrated Circuit (ASIC) to implement the required logic. The latter takes a more economic approach utilizing a single external SRAM and a commodity FPGA, which results in a maximum of 9 million lookups per second at 100 MHz system clock rate. Implementations of other algorithmic solutions, such as [11], have achieved lookup rates up to 250 million lookups per second but either have a higher memory demand or do not score an update performance comparable to that of the Tree Bitmap algorithm.

3 Design and Implementation

In view of the future core router requirements, we designed an IP address lookup module implementing the Tree Bitmap (TBMP) algorithm to check the suitability of trie-based lookup solutions for 100 Gbps line rates. In the following, we identify our design objectives and present details of the module structure.

3.1 Design Objectives

The key objective of our design is the capability to perform lookups with a throughput sufficient for a 100 Gbps Ethernet line card. This shall be achieved by a pipeline design that allows to process effectively one datagram per clock cycle. Thus, a minimum pipeline clock frequency of 150 MHz is required.

Secondly, the module shall support several thousand forwarding table updates per second—as needed in the Internet backbone—without interrupting the fast path.

Finally, our design aims to provide high adaptability to different hardware platforms by means of comprehensive configuration options. These do also allow for statistical research of different setups and enhancements by providing the freedom to partition the trie using arbitrarily sized strides. Easy modifiability of the multibit node encoding with respect to pointer lengths and its internal layout is therefore intended.

3.2 Overall Architecture

Our design extensively utilizes pipelining to efficiently employ the on-chip resources. By contrast, the TBMP implementation introduced by [9] uses time-division multiplexing of several TBMP automata to increase the processable lookup rate. Fig. 3 depicts the basic pipeline structure on block level. On this level, we simply segmented the search trie by assigning each trie level a separate pipeline stage with dedicated memory and processing logic. Due to the flexible module configuration, the overall pipeline structure can incorporate a variable number of these basic TBMP Lookup Stages realizing the algorithmic core functionality. Besides these, an optional Initial Array Stage, an optional Internal Node (iNode) Stage—required for the Split Tree Bitmap optimization—and the final Result Stage, holding the array of next-hop IP addresses, are part of the pipeline. The last two stages in the figure map the next-hop IP address to the corresponding layer 2 address and egress port ID (EPID), and thus avoid redundancy in the result array. The Update Interface assigns accesses of the slow path to the individual memories of the pipeline stages. To achieve a high throughput, all stages themselves are internally pipelined, too.

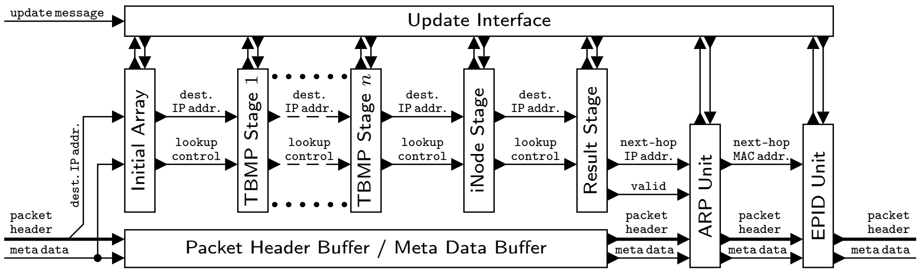


Fig. 3. Block diagram of the Tree Bitmap Lookup Module

3.3 Basic TBMP Stage

The structure of a TBMP Stage is shown in Fig. 4. Via the depicted RAM Interface Module, update and lookup operations access the on-chip memory, which holds the multibit nodes of one trie level. External memory can not be used due to the bandwidth constraints of a single RAM component and the pin count limitations of available chips. As a consequence, we accepted a limited

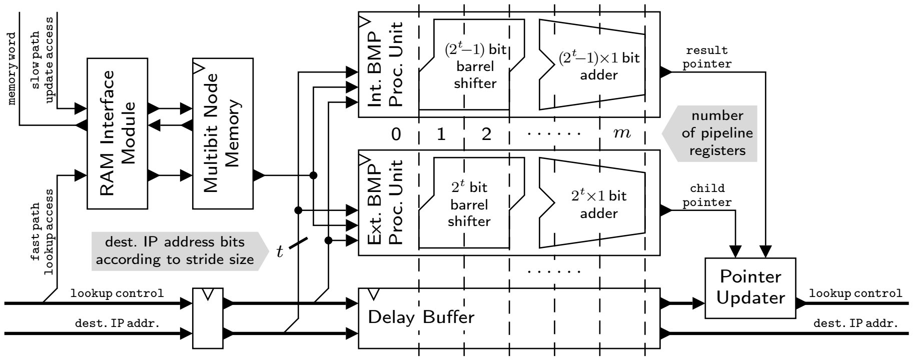


Fig. 4. Simplified block diagram of a TBMP Stage

forwarding table size of our prototype implementation, since even cutting-edge FPGAs do not offer enough memory to support on-chip table storage for core router FWEs—despite the efficient table compression of the TBMP algorithm. Besides the rather pricey option of an ASIC with larger on-chip memories, we discuss different solutions to this problem in Sect. 4.

For deterministic update access times, we have employed true dual-port RAM blocks, which allow update and lookup operations to access the node memory simultaneously. The easy realizability of dual-port RAM is a further advantage of on-chip memories. Thus, non-blocking updates are supported, making table inserts a fast operation that only depends on the slow path processing speed. Based on the formula in [5], the achievable update rate can thus be roughly estimated as $h_{\mathrm{upd}} = \frac{f_{\max}}{2^t + C}$ with $0 < C < (\frac{w}{t} + 3)$, w as the IP address length, and $f_{\max}$ as the maximum supported clock rate.

After the memory access cycle, in which a full multibit node structure is retrieved, the data is split into its node components and processed by separate units evaluating the Internal and External Bitmap. According to the TBMP algorithm, the logic of these units determines an index in the corresponding bitmap that yields the current LMP and the subsequent child node, respectively. Starting with these indexes, the units count all set bits in the lower-order positions to calculate the offsets within the memory block of the next-hop entries and accordingly within that of the multibit child nodes. The lower-order bits are extracted from the bitmap by means of a barrel shifter, and the set bits are then counted by a wide adder unit. Since the barrel shifter and the adder have to process fairly large vectors of $2^t - 1$ and 2^t bits, they take most of the combinatorics of the bitmap processing units. Therefore, we designed them in a way so that both can be mapped to an arbitrary number of pipeline registers m to adjust the length of the critical combinatorial path according to the stride size t and the desired throughput. In Sect. 4, we present the results of an empirical study investigating how many pipeline registers are required to achieve a desired performance with a given stride size.

If the optional Split Tree Bitmap optimization is employed, the Internal Bitmap unit is removed from the TBMP Stages, and a special iNode Stage is attached to the pipeline (cf. Fig. 3). With few different stride sizes, this saves combinatorics along with benefits in memory utilization and word widths. In the iNode Stage, zero, one or two simultaneous memory accesses are required for each lookup. To avoid memory duplication, such dual accesses can use both ports of the internal RAM. Possible conflicts with update accesses are resolved by an arbiter prioritizing lookups over updates. The statistical frequency of these cases, however, should not affect the update performance significantly.

4 Evaluation

The functional correctness of the TBMP Lookup Module has been validated by simulating multiple test cases. The design has been subsequently synthesized and transferred to an Altera Stratix II EP2S60 FPGA embedded in the Universal Hardware Platform (UHP) of the IKR [12]. In our test setup, a computer connected to three 1 Gbps line interfaces of the UHP has successfully shown the correct execution of lookups with different stride and optimization configurations.

For the evaluation of the potential maximum performance, we investigated a single TBMP stage. Since the throughput of the module pipeline is deterministic, a fix relationship exists between clock frequency and lookup rate. To find out the optimum pipelining degree for a given stride size t, we determined the maximum clock frequency $f_{\max}$ for different stride sizes and a varied number of registers m inside the bitmap processing unit. Fig. 5 shows the results obtained from the timing analysis of the synthesis tool used. As expected, large stride sizes require more registers to shorten the critical path so that a clock frequency of over 200 MHz is supported. Considering stride sizes in ascending order, the achievable absolute maximum performance is increasingly bounded by the growing interconnect delay between the Update Interface and the individual

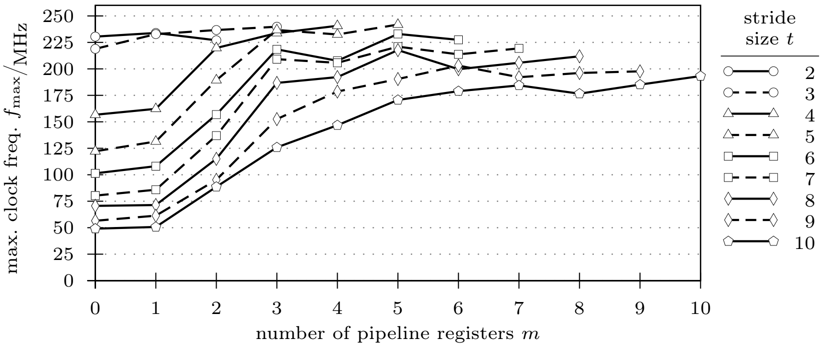


Fig. 5. TBMP Stage timing analysis results for different stride sizes with varied number of bitmap unit pipeline registers

RAM Interface Modules. Though, clock rates of more than 200 MHz have been achieved up to a stride size of 8.

If the complete lookup module is integrated on the FPGA used, performance degrades, as larger interconnect delays are unavoidable if the utilization of the on-chip resources increases. The timing analysis nevertheless resulted in 178 MHz for $f_{\max}$ in a configuration with an 8-bit wide Initial Array and three TBMP stages of stride size 8. With this frequency, the lookup module is capable to fulfill the requirements for 100 Gbps Ethernet to process more than 150 million packets per second and to handle an update rate of several ten thousand messages per second, according to the equation in Sect. 3.3.

The above mentioned implementation requires about 15,000 registers and 12,000 adaptive lookup tables, which equals to 43% logic utilization of the deployed Stratix II FPGA. Using the largest high-end FPGA of the 40-nm Stratix IV family, the processing logic of the lookup module utilizes only 4% of the available logic cells for an IPv4 implementation and 17% for an IPv6 implementation of the developed design. However, since FPGA manufacturers seek for a chip area split between logic and memory blocks that is suitable for the average application, even the above mentioned leading-edge FPGA offers not enough memory resources for a full backbone forwarding table—despite the efficient TBMP coding scheme. With 23 Mbit of embedded memory, the FPGA allows to store only about 180,000 prefixes on-chip assuming a perfectly balanced memory utilization and an average memory demand of 128 bit per prefix [5].

A solution to the memory problem could be efforts to manufacture FPGAs providing larger on-chip SRAM blocks by an adjusted logic-to-memory area split. A second approach is the use of external SRAM or DRAM components, which leads to the problem that current FPGAs offer too few IO pins to connect a dedicated memory for each trie level stage. Sharing memories between stages based on recurrent time slots does not solve the problem either, since the available memory timings do not allow to achieve the total bandwidth required for the 100 Gbps Ethernet processing performance. With today's FPGAs, only a multi-chip solution is viable. An option in a commercial scope might be an ASIC-based lookup module that can be used comparably to a TCAM device.

5 Conclusions

Increasing traffic together with the introduction of 100 Gbps Ethernet in the Internet backbone requires routers to process up to 150 million IP address lookups per second and line interface. Considering also power consumption as well as scalability with respect to growing forwarding tables and IPv6 addresses, algorithmic hardware solutions appear to be most suitable to meet these demands.

In this paper, we presented an extensively pipelined Tree Bitmap Lookup Module, which is capable to effectively process one packet per clock cycle. Additionally, it features a high-speed update interface. By offering multiple configuration parameters, one can adjust the design to different requirements. The lookup module passed several functional tests both in simulations and in a setup using our FPGA-based hardware platform. Synthesis results for an IPv4 configuration placed on an Altera Stratix II FPGA yield a maximum clock rate of 178 MHz. This allows to process up to 178 million lookups and several ten thousand updates per second being ample for 100 Gbps core router line cards. On today's high-end FPGAs, even IPv6 implementations and higher lookup rates are possible. Our prototypical lookup module utilizes the FPGA's on-chip memory, which does not suffice for large backbone forwarding tables. Thus, possible future FPGAs offering larger memories, multi-chip solutions, or ASIC-based lookup modules replacing power demanding TCAM devices are needed for commercial deployment.

References

1. D'Ambrosia, J., Law, D., Nowell, M.: 40 Gigabit Ethernet and 100 Gigabit Ethernet technology overview (November 2008)
2. Huston, G.: BGP analysis reports: IPv4 route-views statistics (March 11, 2009), `http://bgp.potaroo.net/bgprpts/rva-index.html`
3. Degermark, M., Brodnik, A., Carlsson, S., Pink, S.: Small forwarding tables for fast routing lookups. SIGCOMM Comp. Comm. Review 27(4), 3–14 (1997)
4. Lampson, B., Srinivasan, V., Varghese, G.: IP lookups using multiway and multicolumn search. IEEE/ACM Transactions on Networking 7(3), 324–334 (1999)
5. Eatherton, W., Varghese, G., Dittia, Z.: Tree Bitmap: hardware/software IP lookups with incremental updates. SIGCOMM Comp. Comm. Review 34(2), 97–122 (2004)
6. Song, H., Turner, J., Lockwood, J.: Shape shifting tries for faster IP route lookup. In: ICNP 2005: Proc. of the 13th IEEE Int'l Conf. on Network Protocols, November 2005, pp. 358–367 (2005)
7. Nilsson, S., Karlsson, G.: IP-address lookup using LC-tries. IEEE Journal on Selected Areas in Comm. 17(6), 1083–1092 (1999)
8. Tsiang, D., Ward, D.: Advances in router architecture: The CRS-1 and IOS-XR. In: Cisco Networkers 2004 (July 2004)
9. Taylor, D.E., Lockwood, J.W., Sproull, T.S., Turner, J.S., Parlour, D.B.: Scalable IP lookup for programmable routers, vol. 2, pp. 562–571 (2002)
10. Narayan, H., Govindan, R., Varghese, G.: The impact of address allocation and routing on the structure and implementation of routing tables. In: SIGCOMM 2003: Proc. of the 2003 Conf. on Appl., Tech., Arch., and Protocols for Comp. Comm., pp. 125–136 (2003)
11. Jiang, W., Prasanna, V.K.: A memory-balanced linear pipeline architecture for trie-based IP lookup. In: HOTI 2007: Proc. of the 15th Annual IEEE Symposium on High-Performance Interconnects, pp. 83–90 (2007)
12. IKR: Universal Hardware Platform (UHP) – A hardware construction kit for rapid prototyping (March 12, 2009), `http://www.ikr.uni-stuttgart.de/Content/UHP/`