Towards Hardware Architecture for Memory Efficient IPv4/IPv6 Lookup in 100 Gbps Networks

Jiří Matoušek, Martin Skačan, Jan Kořenek
IT4Innovations Centre of Excellence
Faculty of Information Technology
Brno University of Technology
Božetěchova 2, Brno, 612 66, Czech Republic
imatousek@fit.vutbr.cz, xskaca00@stud.fit.vutbr.cz, korenek@fit.vutbr.cz

***Abstract*—With growing speed of computer networks, core routers have to increase performance of longest prefix match (LPM) operation on IP addresses. While existing LPM algorithms are able to achieve high throughput for IPv4 addresses, an IPv6 processing speed is limited. To achieve 100 Gbps throughput, LPM operation has to be processed in dedicated hardware and a forwarding table has to fit into an on-chip memory. Current LPM algorithms need large memory to store IPv6 forwarding tables or use compression with dynamic data structres, which can not be simply implemented in hardware. Therefore, we provide analysis of available forwarding tables of core routers and propose a new representation of prefix sets. The proposed representation has very low memory demands and is suitable for high-speed pipelined processing, which is shown on a new highly pipelined hardware architecture with 100 Gbps throughput.**

***Keywords*—IP address, Longest Prefix Match, Memory**

I. Introduction

Internet speed and network link capacities are growing very fast, which brings new challenges for design and architectures of all active network elements, especially core routers. Current backbone networks widely use 10 Gbps links and will be upgraded to support 40 Gbps or 100 Gbps throughput [1] in the near future. Moreover, the growing number of devices connected to the Internet has a direct impact on the increasing number of entries in forwarding tables, as is shown for example in [2]. These trends push still increasing requirements on speed and size of memory for storing forwarding tables.

The most time critical operation of IP packet forwarding is Longest Prefix Match (LPM). This operation looks up an entry in a forwarding table, which contains the longest prefix equal to a destination IP address of an incoming packet.

To achieve wire-speed 100 Gbps throughput, it is necessary to perform more than 160 million LPM operations per second. In such a situation, a new matching result has to be provided every 6 ns, which is possible to achieve only with dedicated hardware [3]. However, such architectures usually suffer from slow and energy intensive accesses to an external memory. To achieve high throughput and low power consumption, it is necessary to store the forwarding table in an on-chip memory.

Therefore, we provide analysis of available IPv4 and IPv6 forwarding tables and propose a new representation of prefix sets, which can be stored in a small on-chip memory. The proposed representation of prefix sets provides a good memory efficiency ratio not only for IPv6, but also for IPv4 addresses. Moreover, it is suitable for high-speed pipelined processing, which is shown on a new highly pipelined hardware architecture with 100 Gbps throughput. The proposed architecture consists of pipeline stages with exactly the same processing elements. Therefore, IPv4 addresses can be processed by the first part of the IPv6 pipeline to reduce utilization of hardware resources.

The rest of the paper is organized as follows. Section II contains a brief summary of related LPM algorithms. Next section (III) describes performed analysis. Details of the proposed novel representation are provided in section IV, while the hardware architecture able to work with a forwarding table in such representation is shown in section V. Experimental results are summarized in section VI. The paper is concluded in section VII, which also contains remarks about our future work in this area.

II. Related Work

Many commercial devices utilize *TCAM* devices to perform LPM operation. TCAM is able to provide a matching result in just one clock cycle. However, it is expensive, power-hungry, and slow in updating its content. Therefore, many algorithmic solutions to LPM have been proposed [4], [5], [6], [7].

The majority of LPM algorithms is based on a *trie* data structure [4]. It encodes a set of prefixes from a forwarding table into a binary tree. Each node of the tree has up to two pointers to child nodes where left and right child nodes represent prefixes created from the parent's prefix by appending 0 and 1, respectively. LPM is then performed by traversing the trie from the root to leaves according to bit values of a packet's destination address taken from the most significant bit to the least significant bit. The last prefix node visited during such traversal represents the longest matching prefix.

The trie data structure is well designed to implement adding, removing and prefix matching operations. However, because of the high number of pointers, the trie is not a memory efficient representation of a prefix set. Moreover, it does not scale well with the length of the destination IP address because it allows processing of only one input bit in each step.

In order to allow processing of multiple input bits per step, multibit tries have been designed. One of the best known implementation of the multibit trie approach is called *Tree Bitmap* (TBM) [5]. Prefixes from a forwarding table are within

978-1-4673-6136-1/13/$31.00

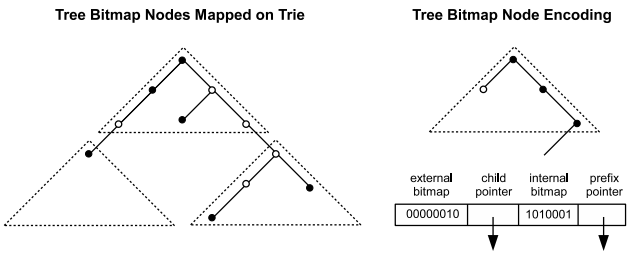

Fig. 1. Tree Bitmap Mapping and Encoding

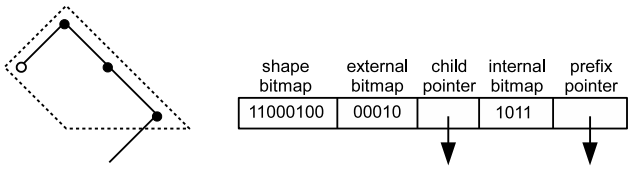

Fig. 2. Shape Shifting Trie Node Encoding

TBM stored in a 2^{SL}-tree, where each TBM node can contain up to $2^{SL} - 1$ trie nodes. The parameter SL is called stride length and it specifies the number of input bits processed in each step. Mapping TBM nodes with $SL = 3$ to the trie is shown in Figure 1.

Structure of the TBM node is also shown in Figure 1. The node contains two pairs consisting of a bitmap and a pointer, which allow to access ordinary child- (external) or prefix- (internal) related information. Such a compact representation allows the node to be read from a memory in just one clock cycle and use of bitmaps makes TBM easy to implement in hardware. The fixed structure of the node also simplifies performing incremental updates of the forwarding table. However, it may introduce high memory overhead, especially in a sparse prefix tree.

Shape Shifting Trie (SST) [6] is another multibit trie algorithm. It is based on TBM but it reduces memory overhead introduced by TBM when representing a sparse prefix tree. Instead of having nodes with a fixed structure, SST allows nodes to adapt to a structure of an underlying trie. This adaptivity is allowed by another bitmap (shape bitmap) introduced in a node's representation (see Figure 2) and is constrained only by the parameter K, which specifies the maximum number of trie nodes represented by the SST node. Even though SST shows very low memory demands, its computational complexity is usually unacceptable. Moreover, to the best of our knowledge, there is no hardware architecture for SST.

An LPM architecture for 100 Gbps networks with currently the lowest memory demands has been described in [7]. This algorithm, which will be further referred to as Prefix Partitioning Lookup Algorithm (or PPLA), also uses the trie data structure. However, the trie is utilized only for partitioning a set of prefixes into several disjoint subsets, which are stored in separate binary search trees or 2–3 trees, each of them processed in a separate processing pipeline. PPLA has good memory efficiency (1 B of memory for storing 1 B of IPv4 or IPv6 prefix) [7], but building this internal representation is connected with very high pre-processing overhead. Moreover, memory demands of PPLA grows linearly with the number

TABLE I. DETAILS OF PREFIX SETS

Prefix Set	Prefixes	Source	Date
	IPv4		
rrc00	332 118	http://data.ris.ripe.net/	2010-06-03
AS2.0	386 653	http://bgp.potaroo.net/	2011-12-13
IPv4-space	220 779	http://bgp.potaroo.net/	2011-12-21
route-views	442 748	http://archive.routeviews.org/	2012-09-20
	IPv6		
AS1221	10 518	http://bgp.potaroo.net/	2012-09-21
AS6447	10 814	http://bgp.potaroo.net/	2012-09-21

TABLE II. MEMORY DEMANDS OF DIFFERENT LPM ALGORITHMS

Prefix Set	Prefixes	Memory Demands [Kb] Trie	TBM (*SL=5*)	SST (*K=32*)
		IPv4		
rrc00	332 118	47 639.677	9 689.432	6 930.441
AS2.0	386 653	104 596.712	30 714.061	15 001.143
IPv4-space	220 779	24 252.430	5 702.065	4 081.008
route-views	442 748	62 650.455	11 942.068	8 774.961
		IPv6		
AS1221	10 518	3 518.297	1 275.422	588.516
AS6447	10 814	3 673.781	1 326.521	617.124

of stored prefixes, which is more than memory demands of trie-based LPM algorithms. When prefixes are stored in a data structure based on the trie or in the trie itself, nodes close to the root are shared by several prefixes and less memory is used to store the forwarding table. Therefore, we focus our analysis on Trie, TBM, and SST algorithms, which all utilize this property to compress the prefix set.

III. ANALYSIS

Our analysis is based on real IPv4/IPv6 forwarding tables acquired from different sources on different days. Details of all prefix sets extracted from forwarding tables are summarized in Table I. Experiments with these data were performed using Netbench tool [8].

First of all, we have performed analysis of memory demands of selected LPM algorithms. The value of the parameter SL was chosen with respect to minimal memory demands of TBM. TBM algorithm with $SL = 5$ and SST algorithm with $K = 32$ have nodes, which can represent almost the same number of underlying trie nodes.

As can be seen in Table II, the lowest memory demands can be achieved with SST. However, as stated in section II, there is no hardware architecture for this algorithm. On the other hand, TBM is easy to implement in hardware but shows higher memory demands than SST. In order to design a hardware architecture with low memory demnads, it will be necessary to combine positive aspects of both TBM (easy to implement in hardware) and SST (low memory demands).

Therefore, we have focused on analysis of structural characteristics of a TBM data structure. In this analysis we have performed a classification of TBM nodes according to the number of prefixes within the node and the number of its child nodes. Such measurement has been done for representations of AS2.0 (IPv4) and AS1221 (IPv6) prefix sets with results presented in Tables III and IV, respectively.

Results for both prefix sets show two significant sets of nodes with high memory overhead when encoded in a standard TBM format. The first set consists of leaf nodes, most of which contain no more than 4 prefixes. The second significant set

TABLE III. CLASSIFICATION OF NODES REPRESENTING AS2.0 (386 653 PREFIXES, 595 219 NODES)

Prefixes	Child Nodes								
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
0	0	**230 142**	**13 072**	**6 882**	**5 245**	**3 590**	**3 322**	**3 802**	**12 156**
1	**249 277**	5 894	3 803	2 467	3 739	1 538	2 011	795	1 731
2	**17 185**	3 117	3 600	1 412	2 282	642	650	303	678
3	**4 685**	1 005	1 740	537	685	242	247	128	256
4	**3 341**	451	478	200	238	127	106	65	148
5	**569**	67	48	22	47	20	14	13	42
6	**163**	25	17	11	17	7	8	1	33
7	**31**	6	1	0	3	4	2	7	27

TABLE IV. CLASSIFICATION OF NODES REPRESENTING AS1221 (10 518 PREFIXES, 25 063 NODES)

Prefixes	Child Nodes								
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
0	0	**11 303**	**1 666**	**812**	**538**	**184**	**145**	**131**	**249**
1	**8 965**	547	142	19	17	3	2	1	1
2	**193**	21	14	4	3	0	1	0	0
3	**50**	3	3	3	1	0	1	0	0
4	**29**	3	1	1	3	1	1	0	0
5	**0**	1	0	1	0	0	0	0	0

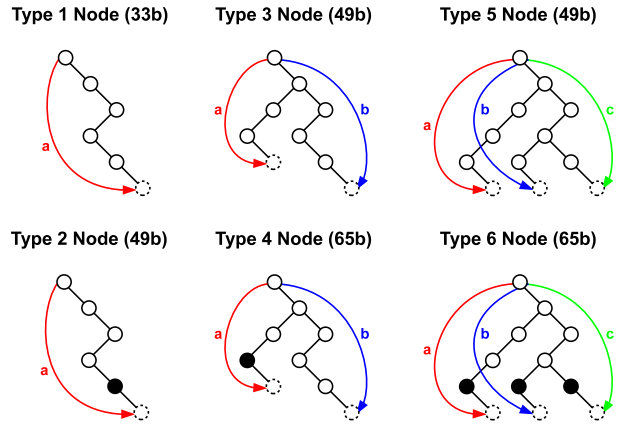

Fig. 3. Newly Proposed Node Types

contains internal nodes without prefixes, where the majority of such nodes have at most 4 or exactly 8 child nodes. We will try to optimize encoding for the most common type of nodes from these two significant sets.

IV. PREFIX SET REPRESENTATION

Performed analysis has shown possibilities for further improvements of TBM's memory demands. Based on these results, we propose novel, optimized encoding for nodes with up to three child nodes and containing up to three prefixes. Such nodes are illustrated in Figure 3. Other situations in the trie (higher amount of branches or prefixes close to each other) will be covered by the standard TBM node with $SL = 5$. Since this TBM node can also more effectively encode nodes with 4 prefixes and/or 4 child nodes, which were also identified as nodes significant from optimization point of view, new encoding for such nodes is not proposed.

All newly proposed node types can be divided into three groups according to the number of trie branches, which can be represented by nodes of that type. A segment of the trie with only one branch can be represented either by a node of the type 1 or 2. Nodes of the type 3 and 4 are utilized when representing two branches of the trie and three branches are covered by nodes of the type 5 or 6. In these pairs, when the type of the node is odd, such node does not allow presence of a prefix. Similarly, when the type of the node is even, the node allows presence of a prefix but only at the lowest level of the underlying trie. The number of bits required for encoding of each new node type is shown in Figure 3. Encoding of the TBM node with $SL = 5$ (including 2 b for node type information) utilizes 95 b.

Mapping new nodes onto the trie is done from the root to leaves by repeated applying of a procedure, which chooses the best node for next unmapped position in the trie. The procedure performs trial mapping of all types of node under current circumstances and determines the best type using equation (2). In equation (1) p_i is the number of prefixes covered by the node of the type i, n_i is the number of trie nodes covered by the node of the type i, and $size_i$ is the number of bits utilized by the representation of the node type i.

$$price(i) = \begin{cases} \frac{p_i}{size_i} & \text{if } \frac{p_i}{size_i} > 0 \\ \frac{n_i}{size_i} & \text{otherwise} \end{cases} \quad (1)$$

$$\max_{i=1}^{7}\{price(i)\} \quad (2)$$

V. HARDWARE ARCHITECTURE

The proposed prefix set representation can be classified as the multibit trie approach. Multibit tries provide a matching result in n steps, where n is the maximum height of a tree representation of a prefix set. If we want to design a hardware architecture for our representation, which permits 100 Gbps throughput, we have to employ a processing pipeline with a separate pipeline stage for each level of the tree. Thanks to distributed nature of FPGA on-chip memory, each pipeline stage can have its own block of memory.

The pipelined hardware architecture for the proposed prefix set representation is shown in Figure 4. Except the pipeline itself, there is also shown an architecture of a single processing element (PE). Its operation is similar to processing of instructions in a standard CPU. At the beginning, PE *fetches* a node from a given address in the mem. Then, a node type is *decoded* in the Type dec and finally, based on the result of decoding, the node is "*executed*", i.e. processed by the corresponding Type proc block.

Because of the utilization of the processing pipeline, we can obtain a new matching result in each clock cycle. In order to achieve 100 Gbps throughput, a clock cycle can not be longer than 6 ns. This is ensured by a possibility to insert intra-stage registers between successive PE operations (fetch, decode, execute), which all separately meet this timing.

VI. EXPERIMENTAL RESULTS

In order to examine our novel prefix set representation, we have implemented mapping of new node types onto the trie. Therefore, we have been able to perform similar experiments as those performed during analysis of current LPM algorithms (see section III). First of all, we have measured memory demands of our mapping scheme when representing IPv4 and IPv6 prefix sets from Table I. The results of this measurement are presented in Table V together with information about

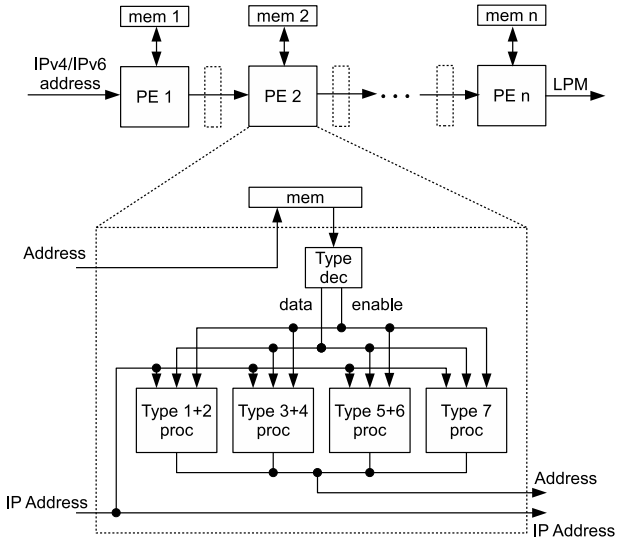

Fig. 4. Processing Pipeline With Detail of One Processing Element (PE)

TABLE V. MEMORY DEMANDS OF PROPOSED MAPPING SCHEME AND ITS COMPARISON TO TBM ($SL = 5$) AND SST ($K = 32$)

Prefix Set	Prefixes	New Nodes [Kb]	TBM Savings	SST Savings
		IPv4		
rrc00	332 118	6 979.760	27.965 %	- 0.712 %
AS2.0	386 653	12 681.440	58.711 %	15.464 %
IPv4-space	220 779	4 215.680	26.067 %	- 3.300 %
route-views	442 748	8 825.680	26.096 %	- 0.578 %
		IPv6		
AS1221	10 518	462.160	63.764 %	21.470 %
AS6447	10 814	486.720	63.309 %	21.131 %

savings of our approach compared to TBM ($SL = 5$) and SST ($K = 32$). As can be seen, our mapping scheme reduces memory demands for representing IPv4 prefix sets by more than 25 % when compared to TBM and almost reaches results of SST. IPv6 prefix sets can be represented with even higher savings — more than 20 % when compared to SST. It is also clear that the proposed mapping scheme allows to represent all used prefix sets in an on-chip memory of current FPGAs (most of Xilinx Virtex-6 LXT FPGAs [9], all Xilinx Virtex-6 SXT and HXT FPGAs [9], and all Xilinx Virtex-7 FPGAs [10]).

To compare memory efficiency of our algorithm with PPLA, we have computed bytes of memory to bytes of prefixes ratio, which is for PPLA reported in [7]. We have also determined this ratio for our measurements of TBM and SST memory demands. Computed values can be found in Table VI. Since average memory efficiency of PPLA is 1.00 for IPv4 prefix sets and 0.90 for IP6 prefix sets, presented results clearly show that our mapping sheme is better than PPLA in terms of memory efficiency.

VII. CONCLUSION AND FUTURE WORK

The paper has proposed the new memory efficient hardware architecture for longest prefix matching in 100 Gbps networks. We have introduced the new representation of prefix sets, which has very low memory demands and is suitable for high-speed pipelined processing. As can be seen in experimental

TABLE VI. MEMORY EFFCIENCY RATIO (BYTES OF MEMORY / BYTES OF PREFIXES) OF PROPOSED MAPPING SCHEME, TBM, AND SST

Prefix Set	Prefixes	New Nodes	TBM (*SL=5*)	SST (*K=32*)
		IPv4		
rrc00	332 118	0.673	0.934	0.668
AS2.0	386 653	1.050	2.542	1.242
IPv4-space	220 779	0.611	0.826	0.592
route-views	442 748	0.640	0.863	0.634
		IPv6		
AS1221	10 518	0.703	1.940	0.895
AS6447	10 814	0.720	1.963	0.913

results, the new representation of prefix sets has better memory efficiecy ratio than TBM and is comparable to SST algorithm. It is important to note that SST algorithm uses a dynamic data structure to store a forwarding table and, therefore, it is not suitable for hardware implementation. The proposed architecture utilize the new representation of prefix sets and pipeline processing to achieve 100 Gbps throughput. As all pipeline stages consist of the same processing elements, the architecture can be easily implemented in hardware and IPv4 and IPv6 addresses can be processed within the single pipeline.

As future work, we want to optimize node encoding and utilize partial dynamic reconfiguration to change distribution of memory blocks among pipeline stages according to the actual prefix set.

ACKNOWLEDGMENT

This work was supported by the research programme MSM 0021630528, the IT4Innovations Centre of Excellence CZ.1.05/1.1.00/02.0070, and the grant BUT FIT-S-11-1.

REFERENCES

[1] *Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications; Amendment 4: Media Access Control Parameters, Physical Layers, and Management Parameters for 40 Gb/s and 100 Gb/s Operation*, IEEE Std. 802.3ba-2010, Jun. 2010.

[2] (2013, Jan.) IPv6 / IPv4 Comparative Statistics. [Online]. Available: http://bgp.potaroo.net/v6/v6rpt.html

[3] M. Á. Ruiz-Sánchez, E. W. Biersack, and W. Dabbous, "Survey and Taxonomy of IP Address Lookup Algorithms," *IEEE Netw.*, vol. 15, no. 2, pp. 8–23, Mar. 2001, ISSN 0890-8044.

[4] E. Fredkin, "Trie Memory," *Communications of the ACM*, vol. 3, no. 9, pp. 490–499, Sep. 1960, ISSN 0001-0782.

[5] W. Eatherton, G. Varghese, and Z. Dittia, "Tree Bitmap: Hardware/Software IP Lookups with Incremental Updates," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 2, pp. 97–122, Apr. 2004, ISSN 0146-4833.

[6] H. Song, J. Turner, and J. Lockwood, "Shape Shifting Tries for Faster IP Route Lookup," in *Proc. of the 13th IEEE International Conference on Network Protocols (ICNP'05)*. IEEE Computer Society, 2005, pp. 358–367, ISBN 0-7695-2437-0.

[7] H. Le and V. K. Prasanna, "Scalable Tree-based Architectures for IPv4/v6 Lookup Using Prefix Partitioning," *IEEE Trans. Comput.*, vol. 61, no. 7, pp. 1026–1039, Jul. 2012, ISSN 0018-9340.

[8] V. Pus, J. Tobola, V. Kosar, J. Kastil, and J. Korenek, "Netbench: Framework for Evaluation of Packet Processing Algorithms," in *Seventh ACM/IEEE Symposium on Architecture for Networking and Communications Systems (ANCS'11)*. IEEE Computer Society, Oct. 2011, pp. 95–96, ISBN 978-0-7695-4521-9.

[9] "Virtex-6 Family Overview. DS150(v2.4)," Xilinx, Inc., Jan. 2012.

[10] "7 Series FPGAs Overview. DS180(v1.13)," Xilinx, Inc., Nov. 2012.