IP Lookup as a Critical Functionality of Packet Processors

Zoran G. Čiča, *Member, IEEE*

***Abstract* — Packet processing represents the most significant part of a router's data plane and has a large impact on router's scalability. Packet processing consists of many functions and some of them can become critical for the future router's scalability. In this paper, the IP lookup as one of the most critical packet processing functions is analyzed to determine its impact on the Internet router's scalability. Also, in this paper we propose a new modification of our previously proposed IP lookup algorithm BPFL.**

***Keywords* — FPGA, Internet router, IP lookup, packet processing.**

I. Introduction

THE Internet is still rapidly expanding as the number of Internet hosts and Internet traffic keep growing continuously. So-called 'killer' applications that have high throughput demands are becoming more and more popular. Multicast traffic, nowadays, represents a significant portion of the Internet traffic, and it puts an additional burden on the Internet routers. The Internet routers represent the most important devices on the Internet as they enable the connectivity between Internet hosts. For all the aforementioned reasons, Internet routers must continuously improve their capacity in order to enable Internet's further growth.

Routers consist of a control plane and a data plane. The control plane implements routing protocols (BGP, OSPF, RIP), management protocols (SNMP), reservation protocols (RSVP), etc. The control plane is software implemented to enable high flexibility of the router configuration and management. Since today's processors are very powerful, the control plane is not the critical part of the router. The data plane represents the path of the user IP datagrams through the router. The data plane contains packet processors at each port and a packet switch (crossbar). To enable router's high capacity and high speed port support, the data plane must be hardware implemented. As the link speeds are continuously increasing, faster ports must be implemented. Packet processing must support processing at wire speed, as otherwise the packets would be lost, which would have negative consequences on network performances. As some of the packet processing functions like IP lookup are very complex, data plane represents the critical part of the router.

This work is funded by the Serbian Ministry of Education, Science and Technological Development (TR32022), Telekom Srbije and Informatika.

Zoran G. Čiča is with the School of Electrical Engineering, University of Belgrade, Serbia (phone: 381-11-3218377; e-mail: zoran.cica@etf.rs).

Packet processing is the most critical part of the data plane. It contains multiple functions such as IP header processing, IP lookup, packet segmentation into fixed-sized cells, cell scheduling, cell buffering, etc. Some of the packet processing functions like IP header processing and IP lookup are independent of a router's architecture. But, some packet processing functions like cell scheduling depend on a router's architecture. In this paper, we analyze IP lookup that does not depend on a router's architecture.

In [1], packet processing functions have been analyzed. It has been shown that IP lookup belongs to the most critical packet processing functions.

IP lookup, for the destination IP address of the incoming packet, performs the forwarding table search to determine to which output port the incoming packet should be forwarded. The IP lookup early became one of the most critical packet processing functions because the size of the forwarding tables was continuously growing, so todays forwarding tables can contain up to 400K entries [2]. Also, due to the classless addresses, multiple solutions can be found during the lookup and the best (longest) match should be selected as the final solution. The LPM (Longest Prefix Matching) rule makes the lookup even more complex. Many lookup algorithms were proposed that were suited well for IPv4 forwarding tables [3]-[6]. However, as the transition to longer IPv6 addresses is inevitable, IP lookup attracted again a lot of attention as the previously proposed lookup algorithms were not suited for longer IPv6 addresses. To enable large IPv6 forwarding table support, new lookup algorithms have been proposed recently [7]-[12].

In this paper, IP lookup as the critical packet processing function will be described and analyzed. Also, a new lookup algorithm TPFL (TCAM Parallelized Frugal Lookup) is proposed in this paper. TPFL algorithm represents a modification of our previously proposed BPFL (Balanced Parallelized Frugal Lookup) and BPFL-SM (BPFL with Shared Memory) algorithms [7]-[9], where the goal of the introduced modification is to improve update performances. The paper is organized as follows. In the next section, the packet processor architecture is presented and explained, as it represents the background for the IP lookup as the packet processing critical function. In the third section IP lookup is analyzed and a short description of our BPFL and BPFL-SM algorithms is given. In the fourth section, TPFL is presented and explained. TPFL is also compared to our previously proposed BPFL and BPFL-SM algorithms. The fifth section concludes the paper.

II. Packet Processor Architecture

User IP datagrams traverse the data plane of the Internet router. The data plane consists of packet processors at input/output ports and a packet switch (crossbar). The first three layers of OSI model (physical layer, data link layer and network layer) are implemented in packet processors. Packet processors also schedule cells for transmission from input ports to corresponding output ports via the crossbar.

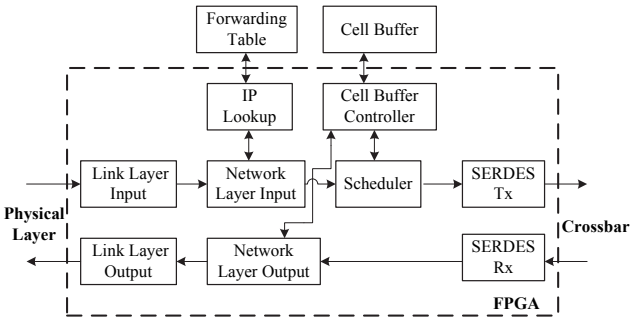

Fig. 1. Packet processor architecture on FPGA.

The packet processor architecture on FPGA chip is shown in Fig. 1. Specialized chips are used to implement the physical layer, so the physical layer processing is not shown in Fig. 1. Packet processor contains several modules, and each module performs a set of packet processing functions. Link layer input module performs processing of the received frames. This module contains functions such as CRC check, link layer address filtering, encapsulated protocol filtering, IP datagram extraction. IP datagrams, extracted from the valid frames, are sent to network layer input module. There, IP header is processed. If some errors are detected in the IP header (checksum error, incorrect version, TTL expired), the IP datagram is discarded. Otherwise, IP header is modified (TTL field is decremented, IP header checksum is updated), and such an IP datagram is segmented into fixed-sized cells. At the same time, destination IP address is sent to IP lookup module to determine to which output port the datagram should be forwarded. Cells are temporarily buffered while they wait the result of the IP lookup module. IP lookup module performs the forwarding table search based on the destination IP address of the IP datagram. The forwarding table stores the information about all networks known to the router. The forwarding table is usually placed in the external fast SRAM memory. The IP lookup module returns the result as the ID of the output port to which the datagram should be forwarded. After the network layer input module receives the information from the IP lookup module, cells are passed to a scheduler, along with the information received from the IP lookup module. The scheduler performs the scheduling of cells for transmission to corresponding output ports via the crossbar. While the cells wait for their scheduled transmissions, they are buffered into a cell buffer. The cell buffer must be capable of receiving large quantities of cells to avoid packet losses in the case of temporary congestions, and it is usually placed in external DRAM memories that have large capacities. The cell buffer controller enables reading and writing the cells to the cell buffer. Scheduled cells are transmitted to the crossbar via fast multigigabit serial links using the SERDES (SERializer/ DESerializer) module. SERDES Tx module performs parallel to serial conversion when sending cells to the crossbar, while SERDES Rx module performs serial to parallel conversion when receiving cells from the crossbar. A network layer output module stores the received cells into the cell buffer. When the datagram is completed (all cells of the datagram are received), the datagram is reconstructed. A reconstructed IP datagram is sent to the link layer output module that adds a data link layer header. Then, the created frame is sent to a physical layer for transmission to the network.

All packet processor modules can be implemented on a FPGA chip, as shown in Fig. 1. The forwarding table and cell buffer represent the storage units that are placed in external memories. The forwarding table stores the information about network topology collected via routing protocols from the router's control plane. The IP lookup module searches the forwarding table during the IP lookup. The cell buffer stores the cells that wait their scheduled time for transmission at the input port and also stores the cells that wait for the reconstruction of their corresponding IP datagrams at the output port.

One FPGA chip can contain multiple packet processors [1]. In such cases, all packet processors on the same FPGA chip would share the IP lookup module and the forwarding table, as well as the cell buffer and the corresponding cell buffer controller. All other packet processor modules are not shared, so the number of each of these modules is equal to the number of packet processors on the FPGA chip.

III. IP Lookup

The main function of the Internet routers is forwarding of the IP datagrams to their corresponding destinations. To be able to perform this task, routers implement routing protocols (BGP, OSPF, RIP) in the control plane. Routing protocols enable the routers to exchange information about all destination networks known to them, and then to learn the network topology and to create a forwarding table based on the exchanged information. A forwarding table in each router contains reachability information about all known networks in the form of forwarding entries that consist of a network address (prefix) and an output port ID to which all datagrams destined to a corresponding network should be forwarded. When a router receives a datagram, the IP lookup module performs the search of the forwarding table to determine to which output port the received datagram should be forwarded. This search process is called the IP lookup.

IP lookup is a very complex function for the following reasons. Multiple matches to one destination IP address can be found in the forwarding table due to classless addressing and prefix aggregation. In such cases, the longest match must be selected as the final result according to the LPM rule [9]. This makes the IP lookup a two-dimensional problem, as the value and length of the prefix must be used to determine the final result of the IP lookup. Thus, it is not enough to find the match in the

forwarding table but one must also be sure that the found match is the longest match. This two-dimensional aspect makes the IP lookup very complex.

Another important aspect is the IP lookup duration [9]. The worst-case scenario is when a router continuously receives the shortest frames. IP lookup must be completed in the time that is equal to the shortest frame duration, as otherwise the frames would pile up in the router and eventually some frames would be discarded. As the link speed increases, the IP lookup must become faster to avoid the aforementioned problem. The maximum lookup time $T_{L\max}$ can be calculated as:

$$T_{L\max} = L_{\min} /(C \cdot N_{PP}), \quad (1)$$

where $L_{\min}$ is the shortest frame size, C is port/link speed, and N_{PP} is the number of packet processors that share the same IP lookup module. Table 1 shows the IP lookup maximum time for various link i.e. port speeds, in the case when the shortest frame is 64B long (i.e. ethernet shortest frame) and only one packet processor uses the IP lookup module.

TABLE 1: MAXIMUM IP LOOKUP TIME FOR VARIOUS PORT SPEEDS.

Port speed	*Max. IP lookup time*
100Mbps	5120ns
1Gbps	512ns
10Gbps	51.2ns
40Gbps	12.8ns
100Gbps	5.12ns

As we can see from Table 1, the maximum IP lookup time is very short in the case of very fast ports/links, only a few ns. In the case when multiple packet processors share one IP lookup module, the maximum IP lookup time is even shorter.

IPv4 address space is exhausted, so the transition to longer IPv6 addresses is inevitable. However, this makes the IP lookup even harder as the search space is now much larger. Also, older lookup algorithms were adjusted to IPv4 prefix distribution, so usually they are not suited well for IPv6 forwarding tables. New lookup algorithms must be able to support large IPv6 forwarding tables.

The network topology constantly changes for various reasons like a link or router failure, addition of new routers and links to the network, congestion conditions, etc. Routing protocols exchange, immediately upon noticed topology changes, new topology information so the routers can update their forwarding table content in order to keep the correct forwarding of datagrams. The update of the forwarding table content must be fast enough to avoid incorrect forwarding in the modified topology environment. Usually, the forwarding table content is compressed to lower its memory requirements. Thus, there must be a tradeoff between the speed of the forwarding table update, and the level of compression of the forwarding table content.

It is obvious that the IP lookup is a very complex function and can become critical in the case of very high speed ports (>40Gbps) as the maximum lookup time becomes very short. The IP lookup can limit router's scalability in the aspect of supported port/link speeds. So there is a need for very efficient lookup algorithms that can achieve high lookup speeds, while supporting large IPv4 and IPv6 forwarding tables.

We have previously proposed a new lookup algorithm BPFL that achieves a high lookup speed of one lookup per clock cycle while frugally consuming hardware resources [7], [8]. The proposed algorithm supports both IPv4 and IPv6 large forwarding tables. Also, BPFL-SM algorithm has been previously proposed and it further optimizes the original BPFL algorithm [9].

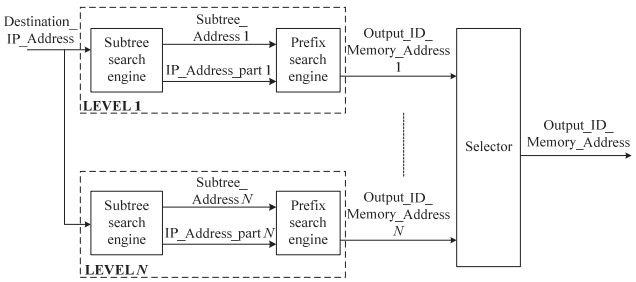

Fig. 2. BPFL architecture.

BPFL is a tree-based lookup algorithm. A binary tree that represents the forwarding table is split into equally sized levels. Only non-empty subtrees that contain at least one existing prefix (i.e. network address) are stored in each level. A subtree prefix represents the path through the original binary tree from the root of the binary tree to the root of the corresponding subtree. All levels are searched in parallel and a selector, according to the LPM rule, selects the best result. The result represents the address in the output ID memory where the corresponding output port ID, to which the datagram should be forwarded, is stored. Fig. 2 shows the BPFL architecture.

Fig. 3. Subtree search engine.

Each level comprises a subtree search engine and a prefix search engine. A subtree search engine is shown in Fig. 3. A subtree search engine stores subtree prefixes in balanced trees whose ranges do not overlap. The selected balanced tree is traversed to determine does the subtree of interest exist in the corresponding level. If the subtree exists then the prefix search engine examines the subtree to find the longest match if such exists. The result of the prefix search engine is sent to a selector. A prefix search engine is shown in Fig.4. A subtree memory contains the indices of existing prefixes and every location in this memory is associated to a corresponding node of the balanced trees in a subtree search engine of the same level. In the case of densely populated subtrees, an overflow memory contains the complete bitmap of such subtrees. Search in a prefix search engine starts in a subtree memory and eventually continues in an overflow memory if the subtree is densely populated.

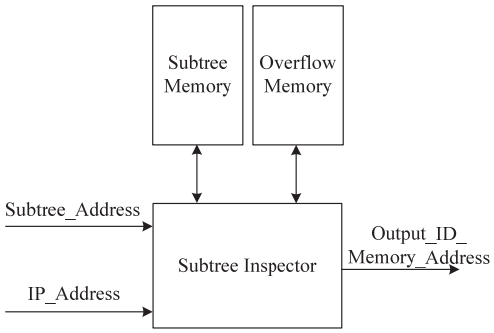

Fig. 4. Prefix search engine.

The BPFL-SM represents the BPFL modification, that is better suited for hardware implementation on FPGA [9]. In the BPFL-SM, there is only one prefix search engine that is used by all levels (Fig. 5). A selector is placed between subtree search engines and joined prefix search engine. Only one found non-empty subtree is examined in the prefix search engine. When multiple subtree search engines find a non-empty subtree of interest, the subtree from the deepest level is selected by the selector. As the selected subtree might not contain a matching node, the output port ID of the deepest existing prefix on the subtree prefix path is added to each subtree in a subtree memory. This additional information slightly increases overall memory requirements, however, it also enables a much more efficient hardware implementation since the overall number of memories is decreased.

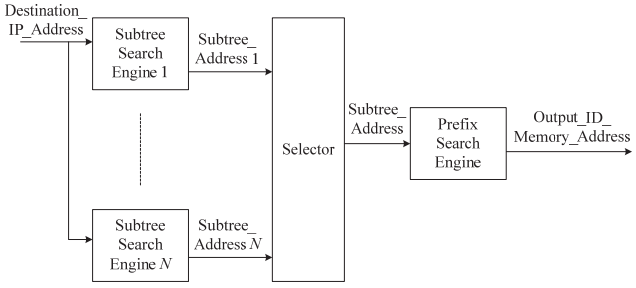

Fig. 5. BPFL-SM architecture.

Both the BPFL and BPFL-SM use a pipeline, thus in every clock cycle a new lookup can be started. Therefore, the BPFL and BPFL-SM achieve a high lookup throughput of one lookup per clock cycle. Both algorithms use static addressing wherever possible, so they have low memory requirements. This property enables support for very large forwarding tables with more than 350K entries even in IPv6 case.

To show that the proposed algorithms have very low memory requirements, they were compared to other existing lookup algorithms, such as tree base algorithms like FIPL [5], POLP [10] and Priority Tree [11]; TCAM based algorithms 1-1 [12] and M12-Wb [12]; and hashed based algorithm PFHT [6]. A comparison of lookup algorithms for a large forwarding table of 365K entries is given in Table 2. Since IPv6 forwarding tables are still small, they were simulated based on the method proposed in [13]. It is clear that in all cases on-chip memory requirements are the lowest in the case of the proposed BPFL and BPFL-SM algorithms. The proposed algorithms have the lowest total memory requirements in the case of IPv6 addresses, but TCAM based algorithms have the lowest total memory requirements in the case of IPv4 addresses. However, TCAM based algorithms use expensive TCAM memories.

TABLE 2: COMPARISON OF LOOKUP ALGORITHMS.

No of entries	*Vers.*	*Algorithm*	*Total mem. [MB]*	*On-chip mem. [MB]*
365K	IPv4	FIPL	6.38	3.22
		POLP	4.11	3.23
		PT	3.79	3.22
		1-1	2.10	1.40
		M12-Wb	3.17	1.23
		PFHT	10.94	6.44
		BPFL	3.84	0.52
		BPFL-SM	3.88	0.56
	IPv6	FIPL	74.67	68.72
		POLP	11.72	9.25
		PT	5.18	4.83
		1-1	3.49	2.80
		M12-Wb	4.74	2.42
		PFHT	12.29	7.79
		BPFL	2.38	1.01
		BPFL-SM	2.49	1.13

The low on-chip memory requirements enable the implementation of BPFL and BPFL-SM on only one FPGA chip even in the case of the largest IPv6 forwarding tables [1].

IV. TPFL ALGORITHM

BPFL and BPFL-SM have very small total and on-chip memory requirements which make them suitable for hardware implementation [1]. Both of these algorithms achieve a high lookup throughput of one lookup per clock cycle. However, the only downside is the update complexity in the worst case. In the worst case, a movement of all subtree prefixes in all balanced trees of one level might be needed to enable the placement of a new subtree prefix or to keep the balancing when one subtree prefix is removed. It was shown in [14] that BPFL has lower update performances than the POLP algorithm.

In this paper we propose the TPFL algorithm that improves the update performances of BPFL and BPFL-SM algorithms while keeping the low total and on-chip memory requirements and a high lookup throughput. The TPFL algorithm introduces a modification in the subtree search engine, while the prefix search engine is the same as in BPFL-SM case, as shown in Fig. 5.

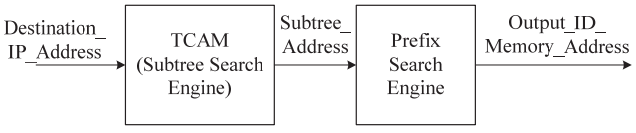

Fig. 6. TPFL architecture.

The TPFL architecture is shown in Fig. 6. In TPFL instead of balanced trees, a TCAM (Ternary Content Addressable Memory) memory is used in the subtree search engine. Now, only the prefix search engine is placed inside the FPGA chip, so a smaller amount of FPGA resources is used, and a cheaper FPGA chip can be selected for implementation. However, one additional chip (TCAM memory) is needed for TPFL. TCAM stores

subtree prefixes of all levels. Based on the destination IP address value, TCAM finds the longest match and returns the location address of the longest match that corresponds to a subtree address in the prefix search engine. The prefix search engine performs the same tasks as in the BPFL-SM case. TCAM memory in TPFL performs the tasks of the subtree search engine and the selector of BPFL-SM. Obviously, TPFL also uses a pipeline technique, thus the lookup throughput of one lookup per one clock cycle is maintained as in BPFL and BPFL-SM cases.

The size of TCAM memory location must be set to the length of a subtree prefix from the deepest level, so the overall memory requirements of the subtree search engine are slightly higher than in the BPFL and BPFL-SM case. This can be avoided if multiple TCAMs are used where one TCAM would be associated to each level. In this case, the size of TCAM location would correspond to the length of the subtree prefix of the corresponding level. However, this is not recommended as each TCAM is in fact one chip, thus the printed board layout would be more complex. Also, the TCAM that is shared by all levels is more adjustable to potential future prefix distribution variations.

It was shown in [8] that the worst update case in BPFL involves a movement of all subtree prefixes in the subtree search engine of the level that contains the largest number of non-empty subtrees. The number of moved nodes (subtree prefixes) can be even larger than 60K in the case of the largest forwarding tables. The same holds for BPFL-SM because it contains the same subtree search engines as BPFL. This worst update case can significantly slower down the lookup process, and packet losses might occur. It is important to mention that the probability for the worst case to happen is very small.

In the TPFL case, the number of movements in the subtree search engine is N-1 in the worst case, where N is the number of levels. The worst case is shown in Fig. 7. In the worst case, a new subtree prefix, that corresponds to the deepest level, is added and only the lowest place in TCAM memory is free. The TCAM memory requires that the subtree prefixes are sorted according to their lengths [9]. Therefore, we move one subtree prefix of the highest level to the free location. Next, we move the subtree prefix of the next level to the location of the previously moved subtree prefix. The process is repeated, until one subtree prefix from each level (except for the deepest level) is moved. In the free location of the last moved subtree prefix, a new subtree prefix is added. From the given example, it is clear that each added subtree prefix requires in the worst case the movement of only one subtree prefix from each higher level. Therefore, the update that requires movement of subtree prefixes is not critical anymore. In the TPFL algorithm, the worst update case is when a subtree becomes densely populated so the overflow memory must be used for the corresponding subtree. In this case, output port IDs in the output ID memory that correspond to non-empty nodes in the corresponding subtree are also moved. The number of these nodes depends on the density threshold that determines whether the subtree is densely populated or not. However, the number of movements is of the order of several tens of nodes, which is insignificant when compared to BPFL and BPFL-SM worst update case.

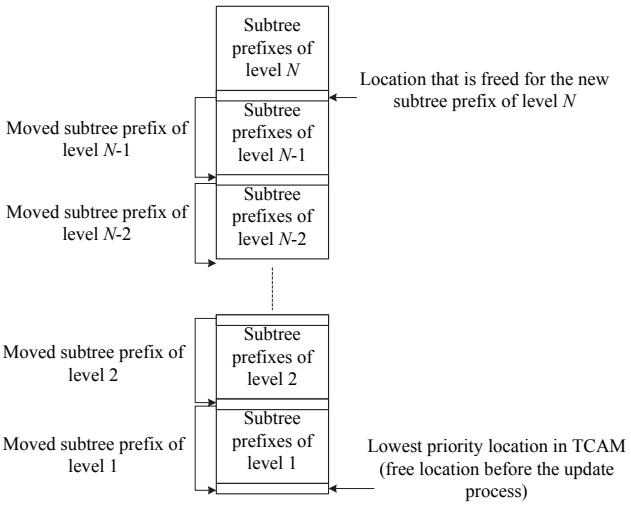

Fig. 7. TPFL worst case update in subtree search engine.

TABLE 3: COMPARISON OF TPFL, BPFL AND BPFL-SM.

No of entries	*Vers.*	*Algorithm*	*Total mem. [MB]*	*On-chip mem. [MB]*
105K	IPv4	BPFL	0.94	0.16
		BPFL-SM	0.96	0.17
		TPFL	0.96	0.18
	IPv6	BPFL	0.77	0.34
		BPFL-SM	0.81	0.38
		TPFL	0.85	0.42
210K	IPv4	BPFL	2.07	0.32
		BPFL-SM	2.10	0.35
		TPFL	2.11	0.36
	IPv6	BPFL	1.46	0.65
		BPFL-SM	1.54	0.72
		TPFL	1.62	0.81
365K	IPv4	BPFL	3.84	0.52
		BPFL-SM	3.88	0.56
		TPFL	3.90	0.58
	IPv6	BPFL	2.38	1.01
		BPFL-SM	2.49	1.13
		TPFL	2.59	1.23

Table 3 shows a comparison of total and on-chip memory requirements of our previously proposed BPFL and BPFL-SM, and the TPFL that is proposed in this paper. TPFL has slightly larger total and on-chip memory requirements than BPFL and BPFL-SM, because the size of the TCAM memory location must be adjusted to the longest subtree prefix. This problem could be overcome by using multiple TCAM memories, where each memory would be associated to one level. However, since the TPFL memory requirements are still very low when compared to other existing lookup algorithms analyzed in Table 2, there is no need for the usage of multiple TCAM memories.

As we could see, TPFL has all the good properties of the previously proposed BPFL and BPFL-SM, that include low total and on-chip memory requirements and a high lookup throughput. Also, TPFL has much better update performances and is more flexible to potential future prefix distribution changes than the BPFL and BPFL-SM.

V. Conclusion

A critical packet processing function, thr IP lookup has been analyzed in this paper. The IP lookup depends on port speed and network size that reflects on the number of forwarding table entries. As it was shown in this paper, high quality solutions for the IP lookup exist and provide support to high capacity routers. Also, a new lookup algorithm is proposed in this paper, the TPFL algorithm. The TPFL algorithm further improves our previously proposed BPFL and BPFL-SM algorithms. TPFL is more flexible and has much better update performances while keeping all the good properties of BPFL and BPFL-SM reflected in the low memory requirements and a high lookup throughput.

References

[1] Z. Čiča, „Analysis and Implementation of Packet Processing Functions in Internet Routers," *Proc. of TELFOR 2012*, Belgrade, Serbia, November 2012.

[2] http://bgp.potaroo.net.

[3] S. Nilsson, G. Karlsson, „IP-address lookup using LC-tries," *IEEE Journal on Selected Areas in Communications*, vol. 17(6), pp. 1083–1092, June 1999.

[4] S. Sahni, K.S. Kim, „Efficient construction of multibit tries for IP lookup," *IEEE/ACM Transactions on Networking*, vol. 11(4), pp. 650-662, August 2003.

[5] D. Taylor, J. Lockwood, T. Sproull, J. Turner, D. Parlour, „Scalable IP lookup for programmable routers," *Proceedings of IEEE INFOCOM 2002*, New York, USA, June 2002.

[6] H. Song, S. Dharmapurikar, J. Turner, J. Lockwood, „Fast hash table lookup using extended Bloom filter: An aid to network processing," *Proceedings of SIGCOMM '05*, Philadelphia, USA, August 2005.

[7] Z. Čiča, A. Smiljanić, „Balanced Parallelised Frugal IPv6 Lookup Algorithm," *IET Electronics Letters*, vol.47(17), pp. 963-965, August 2011.

[8] Z. Čiča, L.Milinković, A. Smiljanić, „FPGA Implementation of Lookup Algorithms," *Proc. of IEEE Workshop on High Performance Switching and Routing 2011*, Cartagena, Spain, July 2011.

[9] A. Smiljanić, Z. Čiča, „A Comparative Review of Scalable Lookup Algorithms for IPv6," *Computer Networks*, vol.56(13), pp. 3040-3054, September 2012.

[10] W. Jiang, Q. Wang, and V. K. Prasanna, "Beyond TCAMs: An SRAM-based parallel multi-pipeline architecture for terabit IP lookup," *Proc. of IEEE INFOCOM*, April, 2008.

[11] H. Lim, C. Yim, E. Swartzlander, „Priority tries for IP address lookup," *IEEE Transactions on Computers*, vol. 59(6), pp. 784-794, June 2010.

[12] W. Lu, S. Sahni, „Low-power TCAMs for very large forwarding tables," *IEEE/ACM Transactions on Networking*, vol. 18(3), pp. 948-959, June 2010.

[13] M. Wang, S. Deering, T. Hain, L. Dunn, "Non-random Generator for IPv6 Tables," *Proc. of IEEE. Symposium on High-Performance Interconnects*, pp. 35-40, August 2004.

[14] N. Maksić, Z. Čiča, A. Smiljanić, „Updating Designed for Fast IP Lookup," *Proc. of IEEE Workshop on High Performance Switching and Routing 2012*, Belgrade, Serbia, June 2012.