IEIE Transactions on Smart Processing and Computing, vol. 5, no. 5, October 2016
http://dx.doi.org/10.5573/IEIESPC.2016.5.5.331

Multi-Stride Decision Trie for IP Address Lookup

Jungwon Lee and Hyesook Lim*

Department of Electronic and Electrical Engineering, Ewha Womans University / Seoul, Korea
jungwon0736@gmail.com, hlim@ewha.ac.kr

* Corresponding Author: Hyesook Lim

Received September 23, 2016; Revised October 17, 2016; Accepted October 18, 2016; Published October 30, 2016

* *Regular Paper*

***Abstract*:** Multi-bit tries have been proposed to improve the search performance of a binary trie by providing flexibility in stride values, which identify the number of bits examined at a time. However, constructing a variable-stride multi-bit trie is challenging since it is not easy to determine a proper stride value that satisfies the required performance at each node. The aim of this paper is to identify several desired characteristics of a trie for IP address lookup problems, and to propose a multi-stride decision trie that has these characteristics. Simulation results using actual routing sets with about 30,000 to 220,000 prefixes show that the proposed multi-stride decision trie has the desired characteristics and achieves IP address lookup using 33% to 47% of the 2-bit trie in the average number of node accesses, while requiring a smaller amount of memory.

***Keywords*:** IP address lookup; Binary trie; Multi-stride trie

1. Introduction

The IP address lookup is a basic operation for packet forwarding by Internet routers, and it should be performed at wire speed for packets arriving at several tens of millions per second. Hence, search performance becomes a critical issue in designing IP address lookup algorithms. As a well-known basic IP address lookup structure [1], a binary trie is simple and intuitive. However, binary tries have low flexibility, in which the shape of a trie is uniquely determined when a prefix set is given. Neither the depth of the trie nor the number of prefixes compared with each input address is controlled. Moreover, binary tries do not provide high-speed search performance, because each input is examined one bit at a time. Shorter-length prefixes are compared with inputs earlier than longer-length prefixes in a binary trie, while the IP address lookup should return the longest matching prefix; therefore, the search should always continue until there is no edge to follow, even though a matching prefix is encountered. To overcome these issues of the binary trie, hash table–based algorithms using a Bloom filter were recently proposed [2-5].

Many variant structures of the binary trie have been proposed [1]. A leaf-pushing trie has prefix nodes only in leaves by pushing down all the internal node prefixes into leaves [6]. The advantage of a leaf-pushing trie is that each input is compared with a single prefix. However, it also does not provide flexibility in the shape of the trie. A multi-bit trie solves the search performance issue of the binary trie by allowing multiple bits to be examined simultaneously [7, 8]. However, since there is no known rule for determining stride values, it is not easy to construct efficient multi-bit tries with variable strides.

Several desired characteristics of a trie can be identified as follows. First is controllability. If controllability in the shape of a trie, such as trie depth or stride value, is provided, the memory requirement for routing tables, or the average and the worst-case search performances, can be controlled. Second is separability, in which internal nodes are separated from leaf nodes. If separability is satisfied, search performance can be improved by compressing internal nodes and/or by storing internal nodes in on-chip memory for fast processing, while storing prefix nodes in off-chip memory [9, 10]. Third is longest-first comparison, in which longer-length prefixes are compared earlier than shorter-length prefixes. If longest-first comparison is provided, the search can be finished as soon as a matching prefix is found [11, 12].

This paper proposes a method of constructing an efficient multi-stride decision trie that satisfies the desired characteristics. The remainder of the paper is organized as follows. In Section 2, related works are described. Section 3 describes the proposed trie. Section 4 evaluates the

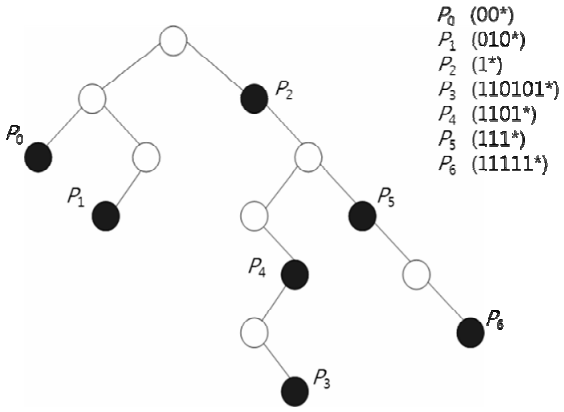

Fig. 1. A binary trie.

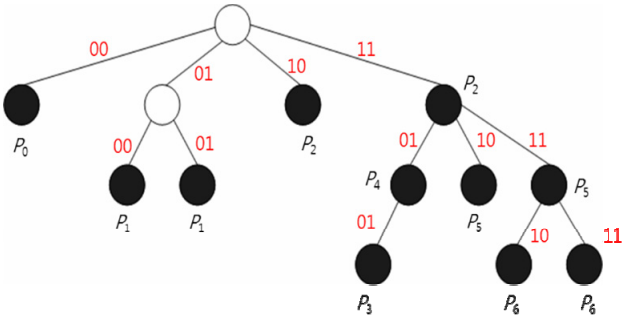

Fig. 2. A 2-bit trie.

performance of the proposed algorithm, and Section 5 concludes the paper.

2. Related Works

2.1 Binary Trie

A binary trie has a bit-wise data structure, representing one bit of each prefix using an edge of the trie. A prefix is stored in a node of the trie, where the level and the path of the node relative to the root node correspond to the length and the value of the prefix, respectively. Fig. 1 shows an example binary trie for an arbitrary set of prefixes. For the given set of prefixes, the shape of the trie is uniquely determined, which means that controllability in terms of trie depth or stride value is not provided. Prefixes can be stored in internal nodes, and hence, the binary trie does not satisfy separability. The search procedure starts from the root node, and the longer-length prefixes are compared later than shorter-length prefixes.

2.2 Multi-bit Trie

Multi-bit tries improve the search performance of a binary trie by providing flexibility in stride values [7]. Fig. 2 shows a 2-bit trie constructed for the example set. Each node can have a maximum of four children. Since two bits are examined simultaneously, some prefixes (such as one-bit prefix P_2) should be replicated into multiple nodes. The search procedure in a multi-bit trie is the same as that of a

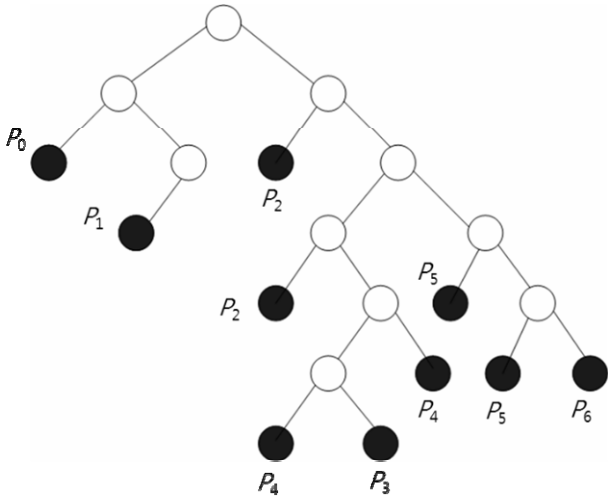

Fig. 3. A leaf-pushing trie.

binary trie, except that multiple bits are examined at the same time. Once a stride value is determined, the shape of a multi-bit trie is also uniquely determined. Hence, multi-bit tries do not provide controllability in terms of trie depth or the amount of prefix replication. It is possible to construct a multi-bit trie having different stride values at each node, but no guideline is provided to determine the proper stride value. Similar to binary tries, multi-bit tries do not provide separability, and longer-length prefixes are compared after shorter-length prefixes.

2.3 Leaf-pushing Trie

Leaf-pushing tries push down every internal node prefix into leaves so that prefixes are stored only at leaves [6]. Fig. 3 shows the leaf-pushing trie corresponding to the binary trie shown in Fig. 1. An internal node prefix can be replicated into multiple leaf nodes. The leaf-pushing trie provides separability and longest-first comparison. However, once a prefix set is given, the shape of the leaf-pushing trie is uniquely determined, and hence, it does not provide controllability.

3. Proposed Algorithm

We propose to utilize the principle of constructing a decision tree in a hierarchical intelligent cutting algorithm (HiCuts) [13] in order to construct a multi-bit decision trie with variable strides. As a decision tree–based packet classification algorithm, the HiCuts algorithm approaches the packet classification problem by partitioning a geometric search space and tuning a few parameters in a trade-off between search speed and storage requirements. In order to provide the controllability of a decision trie, we use parameters like *binth* and *space factor*, similar to HiCuts.

The IP address lookup problem can be represented using a range decomposition problem [14]. Fig. 4 shows the range [000000, 111111] covered by each prefix, assuming for simplicity that an IP address is six bits. The entire range corresponds to the root node of the binary trie,

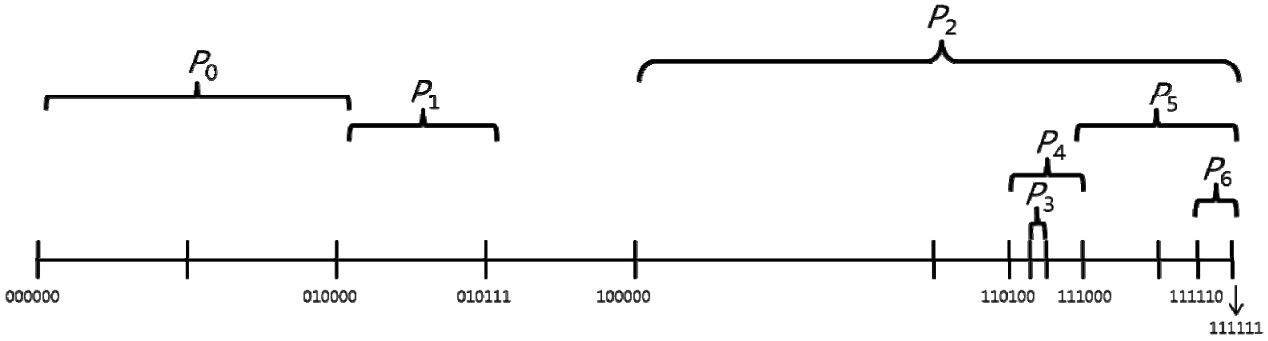

Fig. 4. Ranges Covered by Each Prefix.

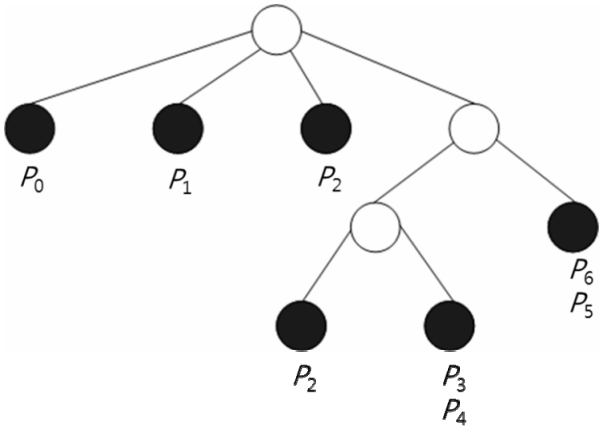

Fig. 5. Proposed decision trie.

```
int searchDecisionTrie(inAddr) {
BMP = defaultPrefix;
W = 32 //initial values
nextNode = root;
for (i = W - 1 to 0) {
node = nextNode;
if (node.type == Leaf) {
BMP = linearSearch(inAddr);
return BMP;
}
else { //node is internal
strideVal = node.strideVal;
nextNode
= getChildPtr (inAddr[i : i-strideVal+1]);
}
i = i-strideVal+1;
}
return -1;
}
```

Fig. 6. Search procedure of the decision trie.

while prefix P_0, which is 00*, covers the first quarter of the entire range, and so on. The construction procedure of a decision trie involves recursively splitting the range into smaller ranges so that the number of prefixes included in each range becomes less than or equal to a pre-specified number.

As a pre-specified number, *binth* determines the maximum number of prefixes compared with each input. If the number of prefixes included in a range is not more than *binth*, the range becomes a leaf node of a decision trie, and the prefixes included in the range are stored in the node. A smaller *binth* makes the trie deeper. The amount of prefix replication caused in splitting a range into smaller ranges is determined by the *space factor*. A larger *space factor* allows more prefix replication, and hence, a larger stride value. Starting from stride 1, the stride value can be increased, provided the total number of prefixes is smaller than an allowable number of prefixes, which is the original number of prefixes times the *space factor*. For the given example set, the proposed decision trie is shown in Fig. 5. Here, *binth* and *space factor* are 2 and 1.5, respectively. At the root node, the number of prefixes is 7, and hence, the allowable number of prefixes is 10.5. If the entire range in Fig. 5 is split into four ranges, the total number of prefixes is 8. If it is split into eight, the total number is 11, which is larger than 10.5. Hence, it was determined to split it into four ranges, and this corresponds to a stride value of 2. This process is recursively repeated. In storing prefixes in a leaf node, if a specific prefix covers the entire range corresponding to the node, prefixes shorter than the prefix do not need to be stored, since the IP address lookup problem involves determining the longest prefix matching each input. For example, in the last one-eighth range shown in Fig. 4, prefix P_2 is included, but it is not stored in node 111* in Fig. 5, since a longer prefix (P_5) covers the entire range. Prefixes are stored in decreasing order of prefix length, so that longer-length prefixes can be compared earlier than shorter-length prefixes. For example, prefix P_6 is stored first, since it is longer than prefix P_5.

The search procedure of the proposed structure is shown in the pseudo-code in Fig. 6. For each given input, starting from the most significant bit, the number of bits specified by the stride value is examined and follows the corresponding edge until a leaf node is encountered. At a leaf node, the input is compared with the stored prefix until a match is found.

For an example of an input with 110101, the search procedure is as follows. At the root node, the first two bits of the input are 11, and hence, the search follows the last edge. At the internal node of level 1, the next bit is 0, and hence, the search follows the first edge. At the internal node of level 2, the search follows the last edge, since the next bit is 1. At a leaf node, prefix P_3 is compared with the input. Since it matches, prefix P_3 is returned without comparing prefix P_4.

The proposed decision trie, which is a multi-bit trie

Table 1. Characteristics of the multi-stride decision trie for various space factors.

Prefix set	N	Proposed multi-stride trie											
		Space factor = 1				*Space factor* = 1.5				*Space factor* = 2			
		f	D_f	N_i	N_l	f	D_f	N_i	N_l	f	D_f	N_i	N_l
MAE-WEST	14553	1.06	29	8727	7272	1.09	11	2553	10783	1.13	11	2352	11538
MAE-EAST	39464	1.12	24	22797	20352	1.21	10	6616	32823	1.33	10	5558	38865
PORT80	112310	1.04	31	57999	56893	1.14	15	28279	73969	1.29	10	17304	101142
Grouptlcom	170601	1.02	25	25187	83346	1.11	13	40583	107160	1.24	10	84029	148686
Telstra	227223	1.15	32	118517	117834	1.25	16	54586	149984	1.35	10	33163	211357

Table 2. Characteristics from a comparison of various tries with the multi-stride decision trie (*binth*=3, *space factor*=1.5).

Prefix set	N	Binary trie			2-bit trie			Leaf-pushing			Proposed		
		f	D_t	N_t	f	D_t	N_t	f	D_t	N_t	f	D_t	N_t
MAE-WEST	14553	1	32	76989	1.16	15	45826	1.37	32	82669	1.09	11	13336
MAE-EAST	39464	1	30	172678	1.16	15	106460	1.50	30	193739	1.21	10	39439
PORT80	112310	1	32	225217	1.43	15	181028	1.29	32	300066	1.14	15	102248
Grouptlcom	170601	1	28	315161	1.44	13	262012	1.19	28	411295	1.11	13	147743
Telstra	227223	1	32	452905	1.37	15	360737	1.26	32	576543	1.25	16	204570

with variable strides, satisfies the desired characteristics. It provides controllability in terms of stride values, the amount of prefix replication, and the number of prefix comparisons. Since prefixes are stored only in leaf nodes, separability is also satisfied. Longer-length prefixes are compared earlier than shorter-length prefixes in a leaf node, and hence, the search can immediately finish as soon as a matching prefix is identified.

If we set both *binth* and *space factor* to 1, the proposed decision trie will be the same as the leaf-pushing trie, except that unnecessary empty nodes are not created, since the length of a prefix does not need to correspond to the level of the node storing the prefix in the proposed decision trie. For example, while prefix P_1 is stored in level 3 of the leaf-pushing trie, it is stored as a direct child of the root node, which is the 01* node, in our proposed decision trie. To the best of our knowledge, the proposed multi-stride decision trie is the first algorithm that provides controllability, separability, and longest-first comparison properties in multi-bit tries.

4. Performance Evaluation

Simulation is performed using the C language for five different prefix sets downloaded from actual routers. Each set had prefixes from about 15,000 to 227,000. We constructed the proposed trie for a fixed *binth* as 3, and variable *space factors* as 1, 1.5, and 3. Table 1 shows the characteristics of each case, where N is the number of prefixes, f is the prefix replication factor, D_t is the depth, N_i is the number of internal nodes, and N_l is the number of leaf nodes. The prefix replication factor is calculated by the total number of stored prefixes divided by N. As the *space factor* is increased, the trie depth is decreased, and the number of leaf nodes is increased. The number of input traces injected to evaluate the search performance is three

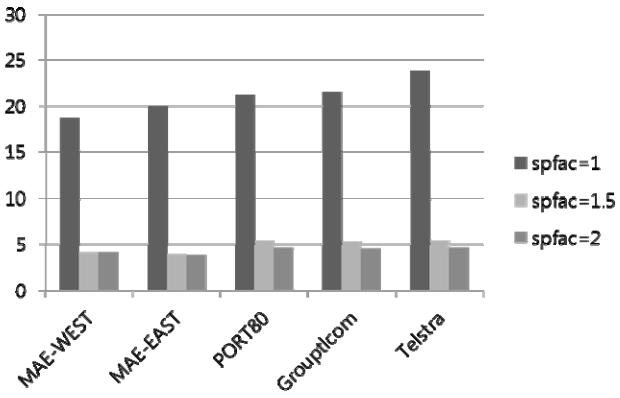

Fig. 7. Average number of internal node accesses for different *space factors* (*spfac*).

times the number of prefixes. The worst-case number of internal node accesses is equal to D_t-1 considering a leaf node in each trie. Fig. 7 shows the average number of internal node accesses. Comparing *space factors* 1 and 1.5, the average number of internal node accesses is decreased considerably by allowing further prefix replication. The number of prefix comparisons depends on *binth*. The worst number of prefix comparisons is the same as *binth*, and the average number of prefix comparisons does not significantly change for the various values of *space factor*.

We constructed a binary trie, a 2-bit trie, and a leaf-pushing trie for the same set of prefixes in a performance comparison with our proposed trie. Table 2 shows the characteristics of each trie, where N_t is the total number of nodes, including internal and leaf nodes. The prefix replication factor f of the binary trie is always 1. The proposed decision trie has a better replication factor than the leaf-pushing trie or the 2-bit trie, in most cases. It is shown that the proposed decision trie has a much smaller number of total nodes, and hence, a much smaller memory

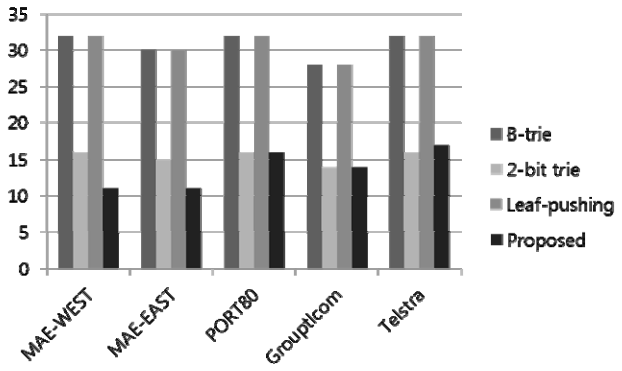

Fig. 8. Worst-case number of node accesses.

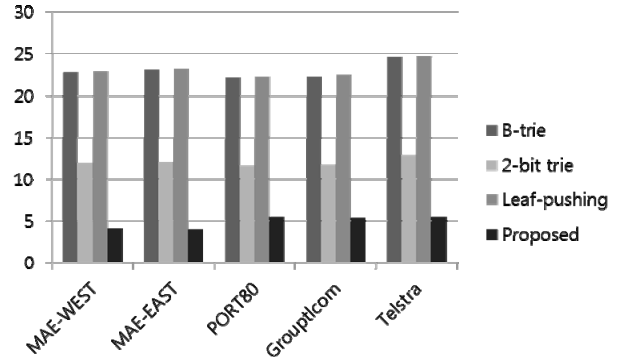

Fig. 9. Average number of node accesses.

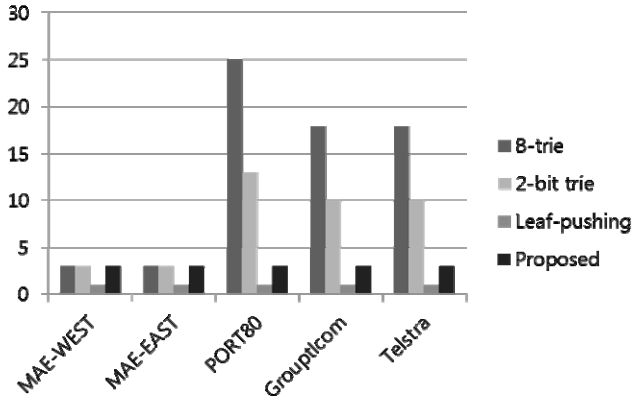

Fig. 10. Worst-case number of prefix comparisons.

requirement than other tries. Figs. 8 and 9 compare the number of memory accesses in the worst-case and for the average, respectively, including internal node accesses and leaf node accesses. It is shown that the proposed multi-stride trie can achieve IP address lookup using 33% to 47% of the 2-bit trie in an average number of node accesses.

Figs. 10 and 11 compare the number of prefix comparisons in the worst-case and average-case, respectively, assuming that internal nodes are stored in on-chip memory, and prefix nodes are stored in off-chip memory. As the number of prefixes is increased, the number of prefix comparisons is increased rapidly in the binary trie and in the 2-bit trie, while the number is

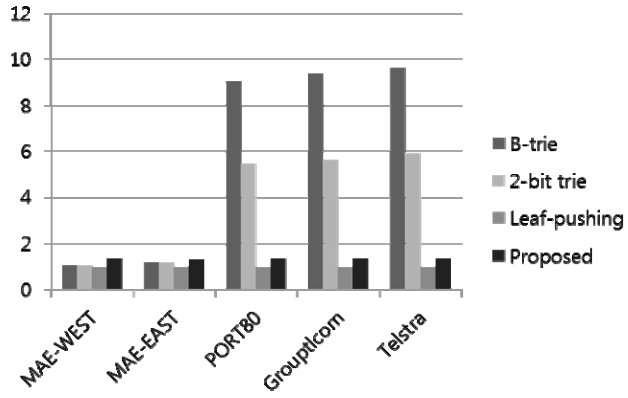

Fig. 11. Average number of prefix comparisons.

controlled in our proposed trie. We see that the proposed trie can achieve IP address lookup using 25% of the two-bit trie for the average number of prefix comparisons of large routing sets, such as PORT80, Grouptlcom, and Telstra.

5. Conclusion

This paper proposed a new approach of constructing an efficient multi-stride decision trie for the IP address lookup problem. Our proposed decision trie has the desired characteristics of controllability, separability and longest-first comparison. Compared with a 2-bit trie, the proposed trie shows better performance in both the number of node accesses and the number of prefix comparisons, while requiring a smaller amount of memory.

Acknowledgement

This research was supported by the National Research Foundation of Korea (NRF), NRF2014R1A2A1A1 1051762 and NRF2015R1A2A1A15054081. This research was also supported by the Ministry of Science, ICT and Future Planning (MSIP), Korea, under the Information Technology Research Center (ITRC) support program (IITP-2015-H8501-15-1007) supervised by the Institute for Information & communications Technology Promotion (IITP).

References

[1] H. Lim and N. Lee, “Survey and Proposal on Binary Search Algorithms for Longest Prefix Match,” *IEEE Communications Surveys and Tutorials*, vol. 14, no. 3, pp.681-697, July 2012. Article (CrossRef Link)

[2] J. Mun, and H. Lim, “New Approach for Efficient IP Address Lookup Using a Bloom Filter in Trie-Based Algorithms,” ” *IEEE Trans. on Computers*, vol. 65, no.5, pp.1558-1565, May 2016 Article (CrossRef Link)

[3] J. Mun, and H. Lim, "On Reducing False Positives of a Bloom Filter in Trie-Based Algorithms," *IEIE*

Transactions on Smart Processing & computing, vol. 4, no. 3, pp.163-168 Article (CrossRef Link)
[4] J. Lee, and H. Lim,"Binary Search on Trie Levels with a Bloom Filter for Longest Prefix Match," *IEEE HPSR*, Jul. 2014, pp. 38-43. Article (CrossRef Link)
[5] M. Kwon, P. Reviriego; S. Pontarelli "A length-aware cuckoo filter for faster IP lookup," *2016 IEEE Conference on Computer Communications Workshops*, 2016 Article (CrossRef Link)
[6] H. Lim, K. Lim, N. Lee, and K. Park, "On Adding Bloom Filters to Longest Prefix Matching Algorithms," *IEEE Trans. on Computers*, vol. 63, no. 2, pp.411-423, Feb. 2014. Article (CrossRef Link)
[7] K. Kim and S. Sahni, "Efficient Construction of Pipelined Multibit-Trie Router-Tables," *IEEE Trans. on Computers*, vol. 56, no. 1, pp.32-43, Jan. 2007. Article (CrossRef Link)
[8] C. Lin, C. Hsu, and S. Hsieh, "A Multi-index Hybrid Trie for IP Lookup and Updates," *IEEE Trans. On Parallel and Distributed Systems*, 2014. Article (CrossRef Link)
[9] D. E. Taylor, J. S. Turner, J. W. Lockwood, T. S. Sproull, D. B. Parlour, "Scalable IP Lookup for Internet Routers," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 4, pp.522-534, 2003. Article (CrossRef Link)
[10] O. Erdem and C. F. Bazlamacci, "Array Design for Trie-based IP Lookup," *IEEE Communications Letters*, vol.14, no.8, pp.773–775, 2010. Article (CrossRef Link)
[11] Q. Sun, Z. Li, and Y. Ma, "Overlapping Hash Trie: A Longest Prefix First Search Scheme for IPv4/IPv6 Lookup," *ICCT*, pp.1-4, 2006. Article (CrossRef Link)
[12] L. Wuu, K. Chen, T. Liu, "A Longest Prefix First Search Tree for IP Lookup," *ICC*, pp. 989-993, 2005. Article (CrossRef Link)
[13] P. Gupta and N. McKeown, "Classifying packets with hierarchical intelligent cuttings," *IEEE Micro*, vol. 20, no. 1, pp.34-41, Jan./Feb. 2000. Article (CrossRef Link)
[14] H. Lim, C. Yim, and Earl E. Swartzlander, Jr., "Priority Tries for IP Address Lookup," *IEEE Trans. on Computers*, vol. 59, no. 6, pp. 784- 794, Jun. 2010. Article (CrossRef Link)

Jungwon Lee received the B.S. degree in Mechatronics Engineering from Korea Polytechnic University, Gyeonggi-do, Korea, in 2011 and M.S. degree at the Department of Electronics Engineering at Ewha Womans University, Seoul, Korea, in 2013. She is currently pursuing a Ph.D. degree from the Department of Electronics Engineering at Ewha Womans University. Her research interests include address lookup, packet classification algorithms, and packet forwarding using Bloom filters at Content-Centric Networks.

Hyesook Lim received the B.S. and M.S. degrees at the Department of Control and Instrumentation Engineering in Seoul National University, Seoul, Korea, in 1986 and 1991, respectively. She received the Ph.D. degree at the Electrical and Computer Engineering from the University of Texas at Austin, Texas, in 1996. From 1996 to 2000, she had been employed as a member of technical staff at Bell Labs in Lucent Technologies, Murray Hill, NJ, USA. From 2000 to 2002, she had worked as a hardware engineer for Cisco Systems, San Jose, CA, USA. She is currently a professor in the Department of Electronics Engineering, Ewha Womans University, Seoul, Korea, where she perform research on packet forwarding algorithms such as IP address lookup and packet classification, and in Content Centric Networks. She was awarded Year 2014 Women in Sciences and Technologies by the Ministry of Science, ICT and Future Planning of Korea. She is a senior member of the IEEE.