Disjoint Superposition for Reduction of Conjoined Prefixes in IP Lookup for Actual IPv6 Forwarding Tables

Roberto Rojas-Cessa, Taweesak Kijkanjanarat, Wara Wangchai, Krutika Patil, Narathip Thirapittayatakul

***Abstract*—Helix is a recently-proposed scheme that performs IP lookup in a single memory access. Helix uses parallel prefix matching at the different prefix lengths and the position of prefixes in a binary tree for reducing the amount of memory used. The scheme enables fast table updates as prefixes are kept in their original form. In Helix, a large number of prefixes is stored in a very small amount of memory and route updates, as the lookup process, is performed in a single memory access. Most IPv6 testings of IP lookup schemes are performed on forwarding tables generated synthetically from IPv4 tables as IPv6 tables have a small prefix count. However, the prefix distribution of address blocks in actual tables may be correlated and that may result in using large amounts of memory to represent them. Here, we proposed the application of disjoint superpositions in Helix to further reduce the amount of memory used to represent these forwarding tables. We show that under IPv6 forwarding tables, Helix prevails in performing lookup operations in a single memory access time.**

***Index Terms*—Address lookup, helicoidal properties, single memory access, parallel matching, parallel search**

I. Introduction

The recently proposed Helix scheme for Internet Protocol (IP) lookup address performs lookup and table updates in a single memory access for both IPv4 and IPv6 prefixes [1]. Helix also uses small amounts of memory to represent routing tables. This scheme performs parallel prefix matching [2], [3] at the different prefix lengths.

Transmission speeds in the order of hundreds of gigabit per second [4], forwarding tables approaching the million count [5], and prefix lengths of 128 bits in IPv6 make IP lookup a challenging router function [6], [7].

The main performance goal of an IP lookup scheme is to minimize the number of memory accesses needed to perform this function as memory access time continues to lag. In addition, the amount of memory used by the scheme to represent prefixes must be small enough to fit on chip to avoid large off-chip delays or to run on static Random Access Memory (SRAM). Furthermore, a lookup scheme must keep up with the hundred thousands per second updates forwarding tables of backbone routers undergo [8] in the shortest time.

Research on IP lookup schemes has been extensive in recent years [9]–[16]. Many proposed schemes are centered on representing forwarding tables and the prefixes in them as binary trees. Although with economical memory usage, the number of memory accesses required to perform the longest prefix matching had remained proportional to the largest prefix length. Helix overcomes drawbacks of binary trees as a scheme that organizes a forwarding table's prefixes a concise structure such that IP lookup and table updates are attained in optimal time.

The case for IPv6 tables is particularly challenging as current actual tables are small; with about 17,000 entries in 2015. This limitation has prompted research to rely on the use of synthetic tables derived from IPv4 tables, without guarantees of future accuracy. As forwardng tables continue to grow and their prefix distribution may converge, it is not known wether the performance of IP lookup schemes will remain.

In this paper, we study the performance of Helix under IPv6 forwarding tables, including actual ones. We propose the use of a disjoint superposition in IPv6 routing tables where prefixes show heavy correlation (i.e., a large number of prefixes share a large number of most-significant bits, MSBs). The use of disjoint superposition reduces memory amount in actual IPv6 tables.

We present the performance of Helix under three IPv6 routing tables, one actual table with 3,188 prefixes but with prefix lengths of up to 128 bits, and two synthetic tables. One of the synthetic tables is derived from the IPv4 table of AS65000 holding 389,956 prefixes and the other from AS6447, with 512,104 prefixes. Both tables were recorded on May 8, 2014. The synthetic IPv6 forwarding tables are generated by applying a well-accepted method [17]. For completeness, we comment on results obtained for a legacy and two recent IPv4 forwarding tables. The legacy IPv4 table, holding 39K prefixes, was recorded on August 24, 1997. The two recent IPv4 routing tables are from AS6447, recorded on June 8, 2010 and on May 8, 2014, holding 334,159 and 512,104 prefixes, respectively. In all tested cases, Helix performs lookup in a single memory access and stores prefixes in very small amounts of memory.

The remainder of this paper is organized as follows. Section II introduces the proposed IP lookup scheme and the helicoidal properties of binary trees. Section III presents the lookup speed and memory usage of Helix under IPv6 forwarding tables and summarizes the performance of Helix under different IPv4 forwarding tables. We comment on the complexity of Helix for performing address lookup and table updates. We conclude

R. Rojas-Cessa and K. Patil are with Networking Research Laboratory, Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ, E-mail: rojas@njit.edu. T. Kijkanjanarat, W. Wangchai, and N. Thirapittayatakul are with the Department of Electrical and Computer Engineering, Thammasat University, Rangsit, Pathumtani, Thailand. Email: taweesak@engr.tu.ac.th

the paper in Section IV.

II. Proposed Scheme and Terminology

In a forwarding table, each prefix is associated with a next hop identifier (NHI), which is the port where a packet is forwarded after matching the packet destination to a prefix. A forwarding table has as many different NHIs as the number of the ports, k, of the router hosting the table.

A. IP Lookup Process

In this process, we first identify the prefix (or tree) levels that have one or more prefixes, or non-empty prefix levels. Prefixes are kept in their original length. Each level is represented and stored separately. We allocate a prefix table, or memory block, for each non-empty prefix level. Prefix matching is performed at each prefix level, in parallel, and the longest prefix is selected among them.

In a prefix table, each location stores the NHI of the prefix assigned to that location. The memory size of the prefix table for level y, $(S(y))$, is $S(y) = 2^y \log_2 k$, where 2^y is the number of locations (or addresses) in the table. This number is referred to as the *memory length*, and the number of bits stored in a location is referred to as the *memory width*. In this case, the memory width is $\log_2 k$, which is the number of bits representing an NHI. The amount of memory used by a prefix table is the memory length times the memory width. Note that this table provides room for the largest number of prefixes at the level rather than the exact amount of memory used by the number of existing prefixes.

B. Definitions and Helicoidal Properties

Long prefixes may also be numerous and they must be efficiently stored. Helix achieves this objective by using the concept of a torsion on the binary tree. The following definitions are used in the remainder of the paper.

Node index (i). Let n_i denote node i of a binary tree with $0 \leq i \leq m$, where $i = 0$ for the root node and i increases for level 1, while moving from left to right. For example, $i = 1$ for the 0-child of the root node $i = 2$ for the 1 child, where the 0 child is to the left of the parent node and the 1 child is to the right. Increase i for the following levels, in the same order.

Local node position. The position of node n_i respective to its prefix level is called local prefix position p_i, where $0 \leq p_i \leq 2^y - 1$. Here, p_i is represented with y bits: $x_1 \ \ldots \ x_v$, where x_1 is the most-significant bit (MSB) and x_y is the least-significant bit (LSB).

Torsion. A torsion is defined here as the process of radially rotating a node on a given level, called *torsion level* L_T, of the binary tree by an angle α. The actual value of α is irrelevant for this discussion, but for the sake of description, it may be considered as 90 degrees in a counter-clockwise direction. A node at L_T and its descendants define a new plane. In this paper, a torsion is applied to all nodes on L_T in for simplicity.

Family root of prefix x (FRx) and descendant bits (Dx). After a torsion, the bits from (*) to L_T become the compound root node of x at level y, where $y > L_T$, and it is called *family root* of x, or FRx. Bits from $L_T + 1$ to y are the portion the prefix used as the address of the location of x in memory of prefix level y. These bits are called *descendant bits of* x, Dx. FRx is stored together with the NHI of prefix x. Therefore, prefix x with a torsion at L_T is represented as FRx $x_{L_T+1} \ \ldots \ x_y$, where FRx is the string of bits $x_1 \ \ldots \ x_{L_T}$.

IP prefix	Prefix ID
000000*	**a**
001001*	**b**
010000*	**c**
010111*	**d**
011000*	**e**
100011*	**f**
101001*	**g**
110111*	**h**
111001*	**i**

TABLE I
Example of prefixes of a forwarding table.

Because all of the subtrees rooted to L_T are coplanar, they may be represented as a single subtree by a joint superposition, which is defined as follows.

Joint superposition. In subtrees rooted to L_T that are coplanar, two or more nodes may be superposed if they share the *same local position*. This process is defined here as joint superposition of subtrees (and nodes). Except for FRx, superposed nodes share the same descendant bits and, therefore, are distinguished from one another by their FRxs.

Conjoined prefix. Two or more prefix nodes are conjoined prefixes if they share a local position in a superposed tree. In the following figures, conjoined prefixes are shown filled in grey color. For completeness, the following definition is given.

Non-conjoined prefix. A prefix that does not share position with any other node is a non-conjoined prefix. In other words, a non-conjoined prefix is one and only one prefix in a local node position of a superposed subtree.

Table I shows an example of prefixes with length six of a forwarding table. Figure 1 shows the prefixes on prefix level 6, as listed in Table I, as a binary tree, as dark nodes. Prefix length y of prefix x is referred to as the level of the tree where the prefix is located. For completeness, we also show the nodes at this prefix level that are no prefixes, as white nodes, and the local position of nodes for the sake of description. Let us select level 3 ($T_L = 3$) as the torsion level to represent prefix level 6. This torsion then generates eight subtrees (rooted at T_L) and eight different family roots (the subtrees below the torsion level, or red line in the figure). Once the subtrees are superposed, the result is a tree of height three, as Figure 2 shows. The gray color of nodes on level 3 (which is level 6 in the original tree) indicates that the prefix is conjoined; holding two or more prefixes. A black node is a non-conjoined prefix. The root of this three is a compound node, holding up to eight different family roots, one per each subtree. Therefore, a torsion with a joint superposition reduces the height of the three from 6 to 3 levels, or an exponential decrease of memory, in this example. In this figure, all prefixes, except for **f**, are

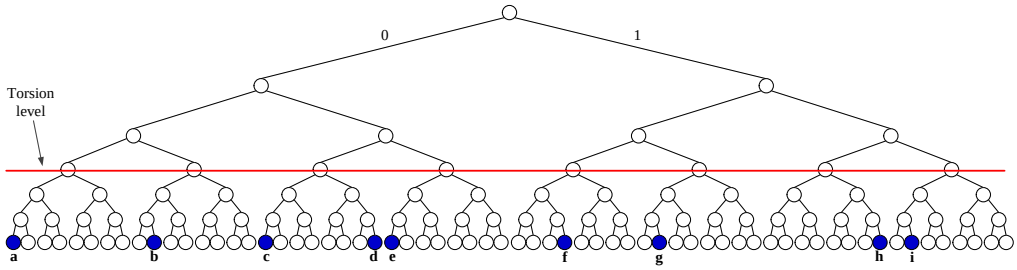


Fig. 1. Binary tree of the prefixes in Table I.

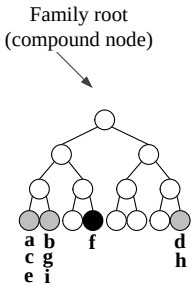


Fig. 2. Resulting three after the torsion and joint superposition.

conjoined prefixes. We can see that a conjoined node has up to three prefixes. The number of conjoined prefixes in a node determines the memory width needed to represent the resulting tree. Table II shows the prefix table of the (compressed) tree in Figure 2. This table has eight entries (instead of the original 64) and the widest entry stores three prefixes with their corresponding FRx.

1) Selection of L_T for each Prefix Level: A suitable L_T for a given prefix level v must satisfy two objectives: 1) To decrease the memory length (i.e., tree's height) to represent prefixes at level v. As L_T increases, the memory length decreases. 2) To keep the number of conjoined prefixes, or memory width, within a feasible number. As L_T increases, the number of conjoined prefixes may increase. Because each prefix level is represented by a self-contained prefix table and is independent of any other prefix level, each prefix level may select a particular torsion level to achieve the two objectives stated above.

Dx	$FRx1$	Prefix ID1	$FRx2$	Prefix ID2	$FRx3$	Prefix ID3
000	000	**a**	010	**c**	011	**e**
001	001	**b**	010	**g**	111	**i**
010						
011	100	**f**				
...						
111	010	**d**	110	**h**		

TABLE II
PREFIX TABLE OF TREE IN FIGURE 2.

Disjoint Superposition. In cases where prefixes of a prefix level are non-uniformly distributed, such that the number of conjoined prefixes remains large despite the use of a torsion, a disjoint superposition may be suitable. In a disjoint superposition, the subtrees are not superposed (we could select the groups of trees to be superposed, but that is out of the scope of this paper) onto a single tree. In the case of a disjoint superposition, we separate all the subtrees generated by a torsion, and a prefix table is allocated for each subtree. A subtree generated by a disjoint superposition is rooted to a family root, called in this case *root bits*, RB. For example, if a torsion is performed at level 2, the tree is divided into up to four subtrees, each indexed by a two-bit RB combination.

For prefix levels with non-uniformly distributed prefixes, the tree can be segmented into several subtrees and each portion of a level residing in each subtree can use a different torsion level. This process is called herein as a disjoint-joint superposition combination. The application of this combination may reduce the number of conjoined prefixes and memory length needed to concisely store the prefixes. In this combination, a disjoint superposition is performed around a first torsion, or L_{T1}. The result is separate subtrees with reduced height. The heights of these subtrees are further reduced by a second torsion, L_{T2}, where $L_{T2} > L_{T1}$. In general, a prefix table generated by a disjoint superposition is indexed by RB, or $x_1 \ \ldots \ x_{L_{T1}}$, and each of its locations is addressed by $x_{L_{T2}+1} \ \ldots \ x_y$ for $y > L_{T2}$. A joint superposition is applied to the subtrees generated after L_{T2}. After the joint superposition, a location stores the family root of a prefix, $x_{L_{T1}+1} \ \ldots \ x_{L_{T2}}$, and the NHI of each prefix.

Figure 3 shows the two subtrees resulting from a torsion on level 1, followed by a disjoint superposition. Each subtree how applies a second torsion and a joint superposition, independently. Figure 4 shows the resulting threes (i.e., Root Bit 0 and Root Bit 1) after the joint superposition. In this figure, subtree of root bit 0 has three conjoined prefixes, **a**, **c**, **e** and two non-conjoined prefixes, **b** and **d**. The subtree of root bit 1 has two conjoined prefixes, one comprising **g** and **i**, and the other with **f** and **h**. In this way, each subtree generated after a disjoint superposition can adopt different torsion level for the application of a joint superposition. This approach further reduces the amount of needed memory.

III. PERFORMANCE EVALUATION

Lookup and Table Update Speed. Helix performs IP lookup in one memory access by design. This performance is confirmed by testing whether a portion of the prefixes of a routing table can be used as the address of the prefix level tables. Helix was evaluated with several Boarder Gateway Protocol (BGP) forwarding tables; including IPv4 and IPv6 tables. The IPv4

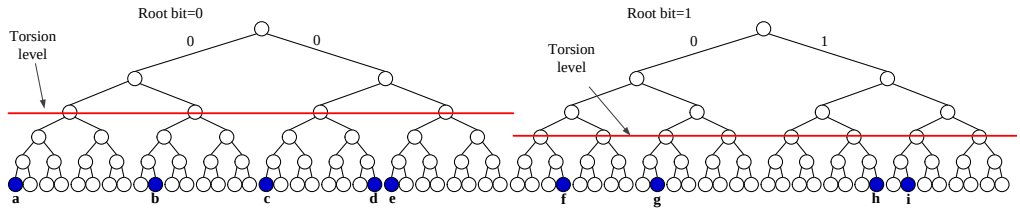


Fig. 3. Binary tree of the prefixes in Table I.

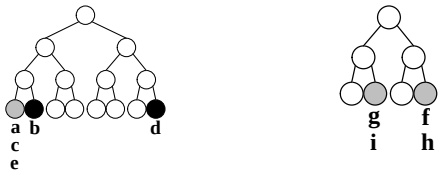


Fig. 4. Resulting threes of Root Bits (a) 0 and (b) 1 after a torsion on levels 3 and 4, respectively, and a joint superposition.

routing tables are Mae East (recorded in August 24, 1997) [18], holding 39,000 IPv4 prefixes, and two AS6447 tables, one recorded on June 8, 2010 and another in May 8, 2014, holding 334,159 and 512,104 IPv4 prefixes, respectively [5].

Helix was tested with three IPv6 routing tables, one actual table, with a small number of prefixes, and two synthetic tables. The actual table is from AS6447, recorded on June 8, 2010, with 3,188 prefixes. This table includes prefix lengths of 128 bits. The synthetic IPv6 routing tables was generated according to a well-accepted model [17] from IPv4 tables AS65000 and AS6447, both recorded on May 8, 2014 [5]. These tables are labeled as AS65000-S and AS6447-S. The AS65000-S table holds 389,956 prefixes and the AS6447-S table holds 512,104 prefixes. In all the tables, prefixes could be represented by using a portion of their bits as addresses, confirming that one memory access would be sufficient to perform prefix lookup by Helix.

For table updates, prefixes are not modified and each prefix table indicates the number of bits used for family roots. Therefore, modifications may require up to one memory accesses on a prefix table, so that updates can be performed very efficiently.

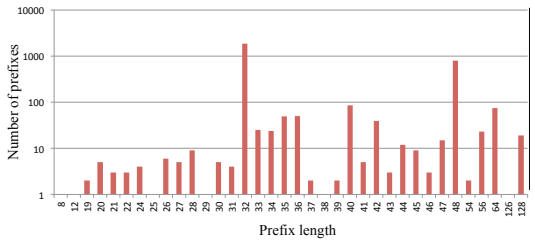
A. Memory used in IPv6 forwarding tables.

Because our objective in this paper is to show the performance of Helix on IPv6, we report the following memory usage by Helix on the IPv6 forwarding tables.

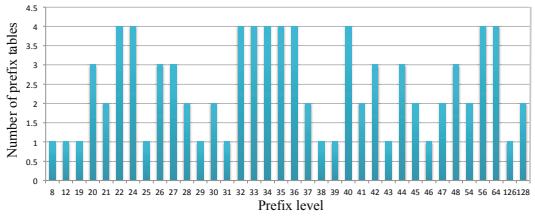
Memory for Actual AS6447 IPv6 Table. This is a small table and it is labeled as IPv6 AS6447 in the remainder of this paper. Figure 5(a) shows the prefix distribution. Prefix levels 32, 48, and 64 hold the largest numbers of prefixes. This forwarding table also contains 128-bit prefixes. Single torsions on the dense prefix levels reveal that prefixes are nonuniformly distributed. Therefore, we apply a double torsion and disjoint-joint superposition combination to this table to store the prefixes in a small amount of memory. The first torsion, followed by a disjoint superposition, is selected at level 6 (and hence the use of 6 root bits as subtree indexes) for the complete table. The disjoint superposition generates four (out of the possible 64) subtrees, or tables per prefix level. Figure 5(b) shows the number of tables per prefix level. As the figure shows, some prefix levels use fewer than four tables. The width of memory blocks for this routing table is set to 1024 bits. Figure 5(c) shows the smallest torsion levels per prefix level. These second-torsion levels are selected independently for each prefix level and for each subtree. Figure 5(d) shows the largest number of conjoined prefixes for each prefix level of the four subtrees. Level 48 has the largest number of conjoined prefixes, 17. Figure 5(e) shows the total memory per level. The required memory for the complete IPv6 table is 2.1Mbytes.

Memory for AS65000-S IPv6 routing table. This synthetic IPv6 table holds prefixes with lengths from 17 to 64 bits, as Figure 6(a) shows. The figure also shows that level 48 holds the largest number of prefixes; 113,750. The selection of the torsion levels for this table is performed by keeping the largest number of conjoined prefixes such that the largest memory width is also equal to or smaller than 1024 bits. We select this width as we targeted the implementation for an FPGA [1]. The FPGA permits the aggregation of several RAM blocks to build a very wide memory. Figure 6(b) shows the selected torsion levels per prefix level. The torsion level for level 48 is 34. Figure 6(c) shows the largest number of conjoined prefixes for each prefix level. Level 48 has 19 conjoined prefixes, as the figure shows. Figure 6(d) shows the total amount of memory per level. Prefix level 48, being the most populated, is provisioned with a memory block of about 1.6 Mbytes of memory. The total amount of required memory for this table is 5.2 Mbytes.

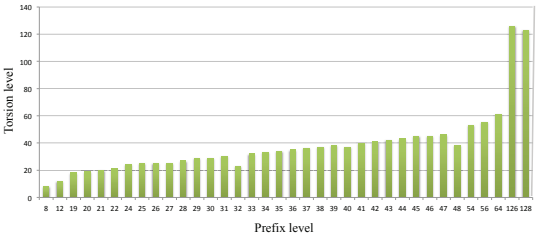
Memory for AS6447-S IPv6 Routing Table. In the IPv6 AS6447-S table, the prefix lengths are found to be from 17- to 65-bit long. Figure 7(a) shows the prefix distribution of this table. The figure shows that level 48 holds the largest number of prefixes (more than 200,000). The tree representing this table undergoes a torsion at level 5, followed by a disjoint superposition. Therefore, five bits are used as root bits, and this superposition produces four subtrees. Each of the levels on



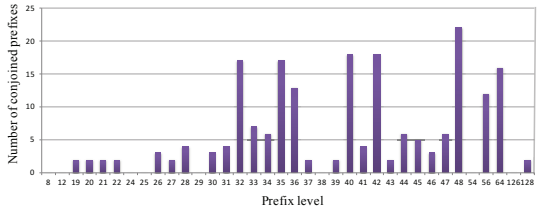
(a) IPv6 prefix distribution.



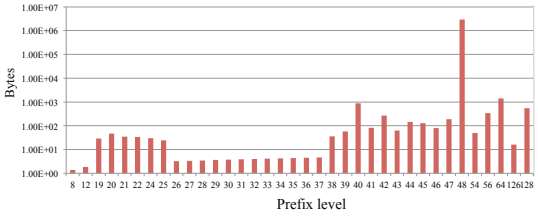
(b) Number of prefix tables



(c) Smallest torsion level per prefix length.

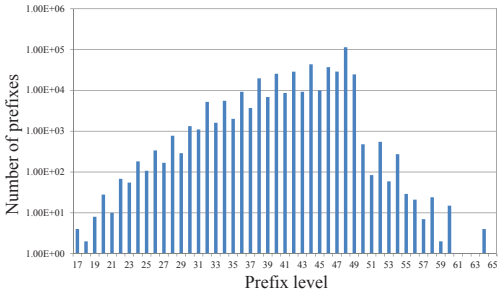


(d) Maximum number of conjoined prefixes.

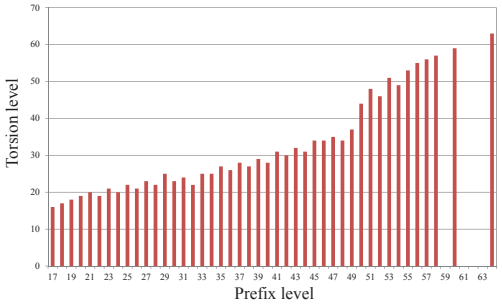


(e) Memory required per level.

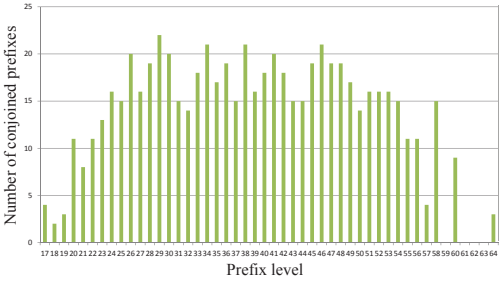
Fig. 5. Memory of Helix on Actual IPv6 AS6447 table.



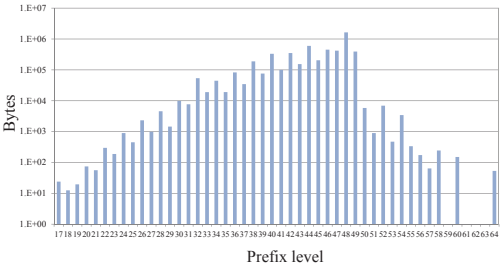
(a) IPv6 prefix distribution.



(b) Smallest torsion level per prefix length.



(c) Number of conjoined prefixes.



(d) Memory required per level.

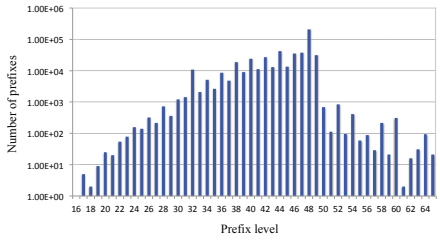
Fig. 6. Forwarding table AS65000-S with 389K prefixes.

the subtrees use a second torsion to separate the family roots from the prefixes. As in the previous table, the selections of the second torsion levels are performed by keeping the largest number of conjoined prefixes such that the largest memory width is equal to or smaller than 1024 bits. Figure 7(b) shows the smallest torsion levels per prefix level. These second-torsion levels are selected independently for each prefix level and for each subtree. The smallest torsion level for level 48 is 34 (a difference of 14 bits, which define the length of the memory block required). Figure 7(c) shows the largest number of conjoined prefixes for each prefix level of the four subtrees. Level 48 has about 28 conjoined prefixes, which is smaller than other levels, as the figure shows. Using a small number of conjoined prefixes on this populated level decreases the amount of memory required for this level and for the whole table. Figure 7(d) shows the total memory per level. The required memory for the complete IPv6 table is 4.3Mbytes.

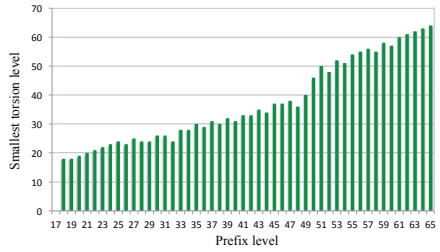
Table III shows a summary of the required memory for the IPv6 forwarding tables studied above.

IV. Conclusions

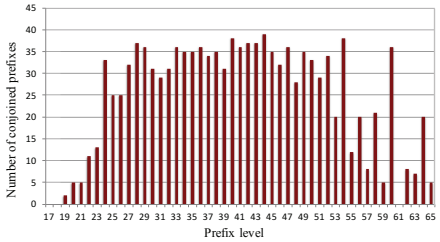
We have introduced the disjoint superposition operation for the Helix IP lookup scheme to reduce the memory used for an IPv6 forwarding table and studied the performance of Helix on IPv6. To demonstrate that Helix remains achieving



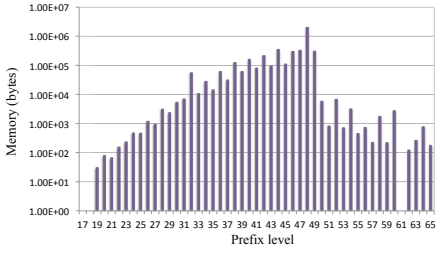
(a) IPv6 prefix distribution.



(b) Smallest torsion level per prefix length.



(c) Maximum number of conjoined prefixes.



(d) Memory required per level.

Fig. 7. Forwarding table IPv6 AS6447-S with 512K prefixes

TABLE III
MEMORY PROVISIONING IN HELIX FOR DIFFERENT IPV6 ROUTING TABLES.

Table	No. of Prefixes	Estimated memory (byte)
Actual AS6447	3,188	2.1M
Synthetic AS65000	389,956	5.3 M
Synthethic AS6447	512,104	4.3 M

lookup in a single memory access, we calculated the amount of memory that Helix would require on three IPv6 forwarding tables, which contain long and numerous prefixes. These tables comprise an actual and two synthetic tables. The actual table has a small number but long prefixes (up to 128 bit long) and the synthetic tables have numerous prefixes. In all the cases, prefixes are represented in a concise format and in their original form such that the lookup and table updates are performed in a single memory access. We also report the memory required by Helix for all IPv6 tables and show that memory is small.

REFERENCES

[1] R. Rojas-Cessa, T. Kijkanjanarat, W. Wangchai, K. Patil, and N. Thirapittayatakul, "Helix: IP lookup scheme based on helicoidal properties of binary trees," *Computer Networks*, vol. 89, pp. 78–89, 2015.

[2] M. Degermark, A. Brodnik, S. Carlsson, and S. Pink, "Small forwarding tables for fast routing lookups," in *Proceedings of the ACM SIGCOMM '97 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '97. New York, NY, USA: ACM, 1997, pp. 3–14. [Online]. Available: http://doi.acm.org/10.1145/263105.263133

[3] R. Rojas-Cessa, L. Ramesh, Z. Dong, L. Cai, and N. Ansari, "Parallel-search trie-based scheme for fast IP lookup," *Proc. IEEE GLOBECOM*, pp. 210–214, November 2007.

[4] J. McDonough, "Moving standards to 100 gbe and beyond," *Communications Magazine, IEEE*, vol. 45, no. 11, pp. 6–9, 2007.

[5] "Routing tables," June 2014, http://bgp.potaroo.net; accessed May, 2014.

[6] H. Song, F. Hao, M. Kodialam, and T. V. Lakshman, "IPv6 lookups using distributed and load balanced bloom filters for 100gbps core router line cards," in *INFOCOM 2009, IEEE*, 2009, pp. 2518–2526.

[7] Y. Qu and V. K. Prasanna, "High-performance pipelined architecture for tree-based IP lookup engine on FPGA," in *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International*, 2013, pp. 114–123.

[8] G. Huston, T. Bates, and P. Smith, "CIDR report," *Web site, http://www.cidr-report. org*, 2005.

[9] M. Degermark, A. Broadnik, S. Carlsson, and S. Pink, "IP-address lookup using lc-tries," *ACM SIGCOMM*, pp. 3–14, 1997.

[10] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable high speed IP routing lookups," *ACM SIGCOMM*, pp. 25–36, 1997.

[11] P. Gupta, S. Lin, and N. McKeown, "Routing lookups in hardware at memory access speeds," *Proc. IEEE INFOCOM*, pp. 1240–1247, 1998.

[12] B. Lampson, V. Srinivasan, and G. Varghese, "IP lookups using multiway and multicolumn search," *IEEE/ACM Trans. on Networking*, vol. 7, no. 3, pp. 324–334, June 1999.

[13] N.-F. Huang and S.-M. Zhao, "A novel IP-routing lookup scheme and hardware architecture for multigigabit switching routers," *IEEE J. Select. Areas Commun.*, vol. 17, no. 6, pp. 1093–1104, June 1999.

[14] P. Gupta, B. Prabhakar, and S. Boyd, "Near-optimal routing lookups with bounded worst case performance," *Proc. IEEE INFOCOM*, pp. 1180–1192, 2000.

[15] S. Nilsson and G. Karlsson, "IP-address lookup using LC-tries," *IEEE J. Select. Areas Commun.*, vol. 17, no. 6, pp. 1083–1092, June 1999.

[16] R. Sangireddy, N. Futamura, S. Aluru, and A. K. Somani, "Scalable, memory efficient, high-speed IP lookup algorithms," *IEEE/ACM Trans. on Networking*, vol. 13, no. 4, pp. 802–812, August 2005.

[17] M. Wang, S. Deering, T. Hain, and L. Dunn, "Non-random generator for IPv6 tables," in *High Performance Interconnects, 2004. Proceedings. 12th Annual IEEE Symposium on*. IEEE, 2004, pp. 35–40.

[18] "Mae East routing table (1997) [online]," 1997, http://www.nada.kth.se/~snilsson/software/router/Data/.; accessed June, 2010.