

FlashLook: 100-Gbps Hash-Tuned Route Lookup Architecture

Masanori Bando, N. Sertac Artan, and H. Jonathan Chao

Department of Electrical and Computer Engineering
Polytechnic Institute of NYU, Brooklyn, NY

Abstract—Since the recent increase in the popularity of services that require high bandwidth, such as high-quality video and voice traffic, the need for 100-Gbps equipment has become a reality. In particular, next generation routers are needed to support 100-Gbps worst-case IP lookup throughput for large IPv4 and IPv6 routing tables, while keeping the cost and power consumption low. It is challenging for today’s state-of-the-art IP lookup schemes to satisfy all of these requirements. In this paper, we propose FlashLook, a low-cost, high-speed route lookup architecture scalable to large routing tables. FlashLook allows the use of low-cost DRAMs, while achieving high throughput. Traditionally, DRAMs are not known for their high throughput due to their high latency. However, FlashLook architecture achieves high-throughput with DRAMs by using the DRAM bursts efficiently to hide DRAM latency. FlashLook has a data structure that can be evenly partitioned into DRAM banks, a novel hash method, HashTune to smooth the hash table distribution and a data compaction method called verify bit aggregation to reduce memory usage of the hash table. These features of the FlashLook results in better DRAM memory utilization and less number of DRAM accesses per lookup. FlashLook achieves 100-Gbps worst-case throughput while simultaneously supporting 2M prefixes for IPv4 and 256k prefixes for IPv6 using one FPGA and 9 DRAM chips. FlashLook provides fast real-time updates that can support updates according to real update statistics.

I. INTRODUCTION

Everyday life has become more dependent on the Internet in recent years, especially with the introduction of new services such as Voice over IP (VoIP), Video on Demand (VoD), and Peer-to-Peer (P2P) file sharing. The traffic increase caused by these new services forces data centers and carriers to upgrade their networks from currently deployed 10/40-Gbps equipment to 100-Gbps equipment [1]. Accordingly, IEEE 802.3ba working group is expected to complete the standard for 100-Gbps Ethernet by 2010 [2]. Our dependence on the Internet is also reflected in the skyrocketing numbers of Internet users around the globe. Computer Industry Almanac Inc. reports that Internet users in the world already exceeded one billion in 2005 and it is estimated to reach to two billion users in 2011 [3]. Considering there were only 45 million Internet users in 1995 and 420 million users in 2000, it is clear that the steep trend that we are facing challenges the current Internet infrastructure. In particular, this trend challenges the current routers, especially the core routers, in demanding support for higher throughput and larger routing tables. An essential packet-processing task for routers is IP route lookup to forward the packets to their destinations. In IP route lookup,

each incoming packet’s destination IP address is extracted and the longest prefix matching against entries in the routing table is performed to find the next hop to which to send the packet. For a 100-Gbps link, 250 million lookups per second are required in the worst-case. In other words, each IP route lookup operation must be finished within 4 ns, which means we can only afford one or two memory accesses to perform an IP route lookup with state-of-the-art memory technology [4], [5]. Furthermore, as a result of the steep trend in the number of Internet users, the size of the typical IP routing table for core routers, supporting both IPv4 and IPv6, is expected to grow to 1 – 2 million routes in the near future. To support the needs of future networks, next generation routers will require large memory and very compact and fast data structures. Previous schemes tend to use TCAMs since they are simple to operate or they use SRAMs since they have high throughput with low latency. Unfortunately, neither TCAMs nor SRAMs have enough capacity to store future routing tables. Additionally, TCAMs are expensive and power-hungry. Recently, DRAM-based schemes, such as tree bitmap from Cisco [6], have been proposed. However, these schemes need many DRAM chips, especially for IPv6. The memory utilization of these chips is unbalanced, increasing the cost of implementing such systems. Besides, due to the latency of DRAMs, previous DRAM-based methods have limited throughput. So, currently there is no low-cost method to support large routing tables at speeds future networks will require. Apart from the memory technology used, another issue of designing next generation routers is the lack of high-speed compact data structures for IP lookup. Hash-based IP lookup schemes are promising, yet the following three shortcomings of such schemes prevent them from achieving the requirements of future routers. (1) Routes have different lengths. To use a hash function to look up routes, either all the routes need to be expanded to a fixed length using a technique called *prefix expansion* [7] described in detail below, which leads to impractically large memory requirements for traditional on- or off-chip SRAM-based methods. Alternatively, one hash function can be used for each route length, which requires many parallel accesses to the memory, which is also impractical, especially for IPv6 and for very high speeds. (2) It is hard to find a perfect hash function that can distribute routes evenly to bins in a hash table, especially when the route set stored in the hash table is not static but updated frequently. To minimize bin

overflows, the hash tables need to be designed with larger bins, which leads to unacceptable memory usage. (3) Even when the bin sizes are large, the bin overflows are inevitable. A special memory needs to be reserved for such overflows (*i.e.*, black sheep routes). A small on-chip CAM is used traditionally to accommodate these black sheep. However, on-chip CAMs cannot be too large, which introduces an additional constraint to the hash table. In this paper, we propose *FlashLook*, a low-cost, high-speed route lookup architecture scalable to large routing tables. *FlashLook* is a DRAM-based architecture. However, unlike previous DRAM architectures, *FlashLook* can use the DRAM burst access efficiently, so that memory access latency can be hidden between two lookups, leading to high throughput IP lookup with DRAM. Since, *FlashLook* uses DRAMs, which provide abundant and low-cost storage, *FlashLook* can easily accommodate future routing tables for both IPv4 and IPv6. *FlashLook* is also a hash-based method; however, it eliminates the three shortcomings of previous hash-based methods by introducing (1) a data compaction method called *verify bit aggregation* that counter balance the memory increase caused by prefix expansion, (2) a novel hash method called *HashTune* to evenly distribute elements in a hash table, and (3) a RAM-based black sheep memory (BSM) to replace the expensive on-chip CAMs. *FlashLook* can provide route lookup for a 100-Gbps link on an IPv4 routing table with 2 million prefixes and on an IPv6 routing table with 256 thousand prefixes. *FlashLook* achieves this performance by using a single FPGA and 9 DRAM chips resulting in a low-cost system. *FlashLook* provides fast real-time updates that can support updates according to real update statistics.

The rest of the paper is organized as follows. Section II summarizes the related work in the area and also presents some of our observations on the publicly available routing tables. Section III introduces the *FlashLook* architecture. Section IV gives details of the implementation of the *FlashLook* architecture, whereas Section V discusses the issues specific to IPv6. Section VI demonstrates the performance of *FlashLook*. Section VII concludes the paper.

II. RELATED WORK AND OBSERVATIONS

Many IP route lookup schemes have been proposed in the past decade such as TCAM (Ternary Content Addressable Memory)-based schemes [8], [9], direct lookup (DL) schemes [10]–[13], trie-based schemes [6], [14]–[18], and hash-based schemes [19]–[25].

TCAM-based schemes are popular for middle range routers because of their high worst-case query throughput and simpler design. However, TCAM’s high cost and huge power consumption make these schemes unattractive particularly for the routers with large routing tables.

DL schemes store the next hop for each prefix in a table or multiple tables that are addressed by the prefix. To address the table properly with a fixed size address, all prefixes need to be at the same length, which can be achieved by a technique called *prefix expansion*. Prefix expansion replaces one short prefix with a set of longer prefixes. Repeating the same

expansion for all prefixes with different lengths leads to a single set of prefixes, all with the same length. As a result, a single longest prefix matching problem can be replaced with a set of exact matching problems. Since each prefix expands to many longer prefixes, prefix expansion requires a larger memory to store the prefixes. In [10], P. Gupta *et al.* proposed DIR-24-8, where prefixes with lengths up to 24 are all extended to 24 bits and stored in a 16-Mbyte direct lookup table on a DRAM. Later, N-F Huang *et al.* reduced the memory requirement of DIR-24-8 using the Lulea algorithm [11], [12]. However, the memory requirement for DL is still significant for IPv4 and prohibitive for IPv6.

Hardware trie-based schemes can achieve high speed, however they require many memory chips in parallel to accommodate the pipeline stages required by the many levels of the trie, which has a height proportional to the number of bits in the IP address. This is especially a problem for IPv6, which has larger number of bits in the address. A recently proposed tree bitmap architecture [6] improves the lookup performance of trie-based schemes, but this architecture still requires a considerable amount of memory and many memory accesses. Tree bitmap requires 6 and 12 memory accesses for IPv4 and IPv6, respectively. To reduce the number of memory accesses of tree bitmap, H. Song *et al.* proposed Shape Shift Tries (SST) [16], which allow the number of trie levels that can be retrieved in one access to be flexible. This way, SST reduce the number of memory accesses by retrieving long paths from the trie with fewer memory accesses compared to tree bitmap and leading to approximately 50% reduction in the number of memory accesses. Although this reduction is significant, the number of memory accesses required by the SST is still considerable.

Another drawback of the trie-based schemes including the tree bitmap architecture is the uneven distribution of their data structure in memory. Characteristic of the tries, the lower level contains many more prefixes than the higher level. Since, each level (or group of consecutive levels) constitutes one pipeline stage, which is typically stored in one memory bank, the size of the pipeline stages is not balanced, causing very low memory utilization. In [18], the authors propose a solution for balancing the pipeline memory. However, their scheme requires 25 independent memory chips, which significantly increases the pin counts for the FPGA/ASIC. The number of memory chips required in [18] is even larger when IPv6 support is considered. Additionally, it is also proposed in [18] that throughput can further be improved by further dividing the pipeline stages such that one level in the trie is divided into multiple pipeline stages. Although the architecture is enriched with caching, it is still limited with the throughput of one pipeline.

Hash-based schemes usually require either parallel lookup engines for every single prefix length or they need *prefix expansion* [7]. The former approach is not practical, especially for IPv6 lookup, due to the large number of parallel memory accesses required. Prefix expansion, as given above, requires larger memory. Additionally, when the prefixes after expan-

sion are stored in a hash table, collisions impact the query performance of existing schemes. As this paper describes, our solution also uses prefix expansion. However, by allowing the use of low-cost DRAMs instead of SRAMs, and with the help of our novel prefix aggregation scheme, the memory increased by prefix expansion is counter balanced. By using our novel approach, HashTune, the impact of collisions is also minimized.

1) *Observations:* Prefix length distribution is an important parameter for prefix expansion, which we present in this section. Figure 1 shows the IPv4 prefix length distribution for four different geographical locations, which is obtained from RouteView Project [26]. We will investigate IPv6 in Section V. Some key observations to assist in the design of a scalable IP lookup architecture can be made by examining Figure 1. First of all, it is obvious from the figure, that the four distributions are practically the same, regardless of their geographical location. As a result, in this paper, we will only use the Oregon routing table, which is the largest among the four, as a representative routing table.

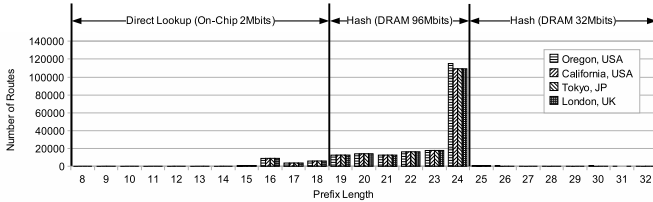


Fig. 1: IPv4 Prefix length distribution of four real routing tables, located at four different geographical locations.

It can be observed that the bulk of the prefixes (89.40%) have lengths between 19 bits and 24 bits [7], [26], [27], whereas most of the prefixes (55%) have a length of 24 bits. The remaining prefixes with lengths less than 19 bits and more than 24 bits account for only around 10% of all prefixes.

In order to observe the variation of routing tables in time, we take snapshots of routing tables once for each year between 2002-2008. Then we compare their prefix length distribution with the distribution of a snapshot in 2009. We observe that the ratio of number of prefixes in any prefix length in any table does not deviate more than 5.56% from the 2009 values. On average, this deviation is only 0.51%. As a result, we can conclude that the prefix length distribution has changed little in the last 8 years and it is reasonable to assume that this distribution will stay stable in future routing tables.

Based on the above observations, we expand prefixes into 3 lengths. We expand prefixes shorter than 18 bits to 18 bits, prefixes with lengths between 19 and 23 bits to 24 bits, and prefixes with lengths between 25 and 31 bits to 32 bits. We denote the set of all prefixes with lengths 18, 24, and 32 bits after prefix expansion as IPv4/18, IPv4/24, and IPv4/32, respectively.

An on-chip direct index table with $2^{18} = 256K$ entries can comfortably accommodate IPv4/18 prefixes. Furthermore, by distributing this direct index table to multiple parallel on-chip memory blocks, the updates can be quickly completed by

parallel accesses to these memory blocks. Table I shows the number of prefixes before and after prefix expansion for the Oregon routing table and the corresponding expansion rates. For instance, the 190,692 prefixes between lengths 19 and 24 bits are replaced with 889,371 prefixes with a length of 24 after prefix expansion giving an expansion rate of 4.66. Obviously, since one prefix is replaced with a set of prefixes in most cases, the number of prefixes will increase after prefix expansion. Although the on-chip memory is not large enough to store all the prefixes after expansion (IPv4/24 and IPv4/32), off-chip DRAMs can accommodate them easily.

TABLE I: Number of prefixes before and after the expansion.

Original Prefixes		Expanded Prefixes		Expansion Rate	
/19-/24	/25-/32	IPv4/24	IPv4/32	IPv4/24	IPv4/32
190,692	2,968	889,371	152,794	4.66	51.48

III. FLASHLOOK ARCHITECTURE

In this section, we introduce the FlashLook architecture and describe the details of the three main features of FlashLook. For the sake of simplicity, this section is based on IPv4. Design issues related to IPv6 are discussed in Section V.

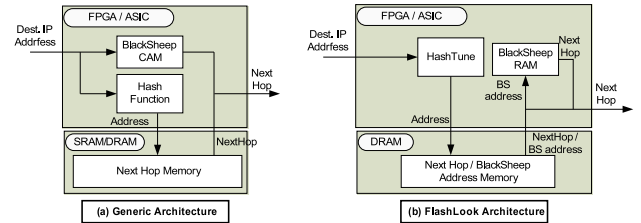


Fig. 2: A generic architecture for hash-based IP Lookup Schemes and the FlashLook Architecture.

FlashLook is a hash-based IP lookup scheme. A typical hash-based IP lookup scheme is based on the generic architecture shown in Figure 2 (a). In such an architecture, there are one or multiple hash functions on-chip, which given a destination IP address from a packet, generates a memory address. This memory address usually points to a bin in a hash table on a DRAM or SRAM chip, where the next hop information for this IP address and a partial or complete prefix are stored. When information for the next hop is retrieved from the hash table, the packet is sent to this next hop. If there is an overflow in the hash table, the overflowed next hop information is generally stored on a CAM that resides on-chip (*i.e.*, on FPGA or ASIC). The CAM, which is also known as Black Sheep Memory (BSM), is queried with the same IP address given to the hash function. If the CAM has the next hop information (*i.e.*, the next hop information is overflowed to the CAM), the packet is sent to the next hop given by the CAM and the hash table is not accessed. Hash-based schemes can also deploy additional on-chip structures for optimizing performance.

The FlashLook architecture, as shown in Figure 2 (b), is also based on this generic architecture but it provides the following contributions to transform the generic architecture into a low-cost, high-speed, and scalable architecture: (1) *Verify Bit*

Aggregation for compact hash bin structure, (2) *HashTune*, a novel feature that replaces the single hash function with a pool of hash functions and allows the flexibility to choose a separate hash function out of the hash function pool for different groups of prefixes. As a result, hash collisions are reduced and the memory utilization of the hash table is maximized. (3) A *RAM-based BSM* replacing the expensive CAM-based BSM by introducing a unique bin structure to store the next hop information in the DRAM. The RAM-based BSM also relaxes the constraints on the hash table, allowing a simpler and smaller hash table with more expected collisions, yet with the same performance¹.

A. FlashLook Hash Table

To accommodate hash overflows, the bins in the FlashLook hash table have two different organizations. Assume a bin has a capacity of c . If there are at most c elements hashed to this bin, the bin stores these elements back-to-back. If, on the other hand, there are $c + v$ elements hashed to this bin, the first $c - 1$ elements hashed to this bin are stored in the bin and the remaining part of the bin will be used as a pointer to a BSM location that stores the remaining $v + 1$ elements. In this paper, we assume $c = 3$ and $v = 7$ for IPv4/24 and $c = 4$ and $v = 7$ for IPv4/32 as these values provide a good memory utilization for IP lookup using current DRAM technology, as explained in Section VI-A.

Although, FlashLook improves the performance of the generic architecture significantly with the above features, the operation of FlashLook stays very similar to the generic hash-based IP lookup scheme. Once a packet arrives, HashTune picks the relevant hash function for this IP address and retrieves the next hop information from the address pointed to by this hash result. If there is an overflow in the hash table, the corresponding address also points to the location of the overflow routes in the BSM. If the next hop is not in the DRAM, the next hop is retrieved from the RAM-based BSM.

B. Verify Bit Aggregation

In this section the Verify Bit Aggregation for compacting the routing table is explained. In a routing table, the crucial information to store for each prefix is the next hop for packets matching to this prefix. In a direct lookup table, there is one bin available for each prefix (possibly after prefix expansion), so storing the next hop for each prefix in the corresponding table bin is enough. However, this may require prohibitively large memory space as the routing table size grows. The traditional solution is to reduce the number of bins by projecting multiple prefixes onto one bin in the form of a hash table. However, in a hash table, it is not enough to simply store the next hop, since it is not possible to distinguish which prefix corresponds to this next hop among many possible prefixes that can be projected onto this bin. For instance, if a packet destined for the default next hop falls into this bin, it

is necessary to guarantee that the packet will not be sent to the stored next hop, but instead sent to the default next hop. To distinguish between the prefixes in a hash-based scheme, typically, the prefix is also stored along with the next hop. Storing the prefix increases the space requirement of each bin, however this increase is justified by the huge decrease in the number of bins, thus the decrease in the overall table memory. Also note that, the inevitable result of using hash tables is possible collisions, which can be resolved by accommodating multiple slots in each hash bin.

As a middle ground, the hash table can be arranged in such a way that only a small set of prefixes with known properties can be stored in one particular bin. In this case, only enough bits to distinguish between these prefixes, called *verify bits*, need to be stored along with the next hop, further reducing the space requirement of the hash table. However, verify bits are still a considerable overhead. Here, we propose *verify bit aggregation* to reduce this overhead. Before getting into details of verify bit aggregation, note that FlashLook only stores an 8-bit Next Hop ID to distinguish next hops instead of storing the full 32-bit next hop address.

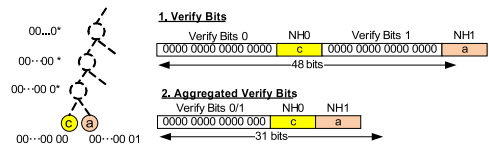


Fig. 3: An example hash bin with two routes (c and a) from the tree on the left is shown both before and after route aggregation. The route aggregation reduces the bin size from 48 bits to 31 bits.

Traditionally, verify bits are always stored with their next hop information as illustrated in Figure 3, **1. Verify Bits**. Two next hop information is shown in the Figure as c and a. In this approach, 16 verify bits are stored for each next hop information. However, if a child node has a sibling, one can reduce the size and the number of elements by storing and verifying the common verify bits of the children at the parent node. This property can readily be exploited to achieve significant savings in an expanded routing table, where parent nodes are likely to have both child nodes. Only one set of the verify bits is needed at the parent element and the number of those verify bits is also reduced by one bit. The above operation is illustrated in Figure 3 as **2. Aggregated Verify Bits**. The nodes with next hop information c and a share the same parent node. Thus, an element of their parent node contains the 15 common bits as the verify bits. The 16th bit is used to identify the correct child to traverse in the next level (i.e., c if the 16th bit is “0” and a otherwise.) The reduction in the number of elements with this optimization is shown in section VI.

C. HashTune

HashTune is a notable feature of FlashLook that provides a hash table with evenly distributed prefixes leading to better hash memory utilization and a smaller number of overflows.

¹Our experiments on the Xilinx ISE 10.1 show that a CAM with a capacity of 4096 elements requires 512 Block RAMs on a Virtex-4 FPGA, while our architecture only requires 8 Block RAMs for the same capacity.

Figure 4 (a) shows a hash table with 12 bins, which is used to store 20 prefixes. Each row represents a bin and each dark square in a row shows a prefix hashed to that bin. Typically, a naïve hash function may lead to non-uniform distribution of prefixes in the hash table as in Figure 4 (a) (b). This non-uniformity forces unnecessarily large bin sizes, which requires multiple memory accesses to retrieve the contents of one bin while reducing the memory utilization of the hash table. More importantly, this leads to bin overflows (*i.e.*, black sheep prefixes) requiring a large BSM. Even a good hash function is found for a particular table, after some updates, the distribution of prefixes in the hash table can still become non-uniform.

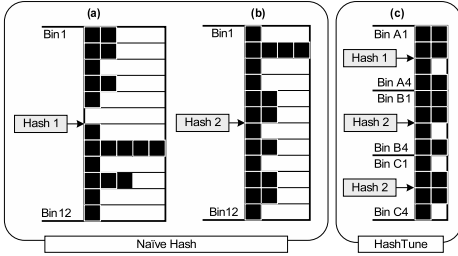


Fig. 4: HashTune Example.

HashTune employs multiple hash functions for different parts of a hash table and locally optimizes the uniformity of prefix distribution in these parts by picking up a hash function for these parts that reduces the hash collisions. In other words, FlashLook “tunes” the hash function to prefixes similar to tuning a radio to the frequency that gives the clearest sound for a radio station. Thus, we named this feature *HashTune*. The first step for HashTune is to partition the hash table into groups of bins by means of index bits. Then HashTune dynamically assigns a separate hash function to each bin group, from a pool of universal hash functions. Each hash function assigned to a bin group is selected such that the number of collisions between the prefixes in that bin group at the time is minimized. After prefix updates, if the hash function assigned to a bin group no longer provides good uniform distribution of prefixes, HashTune replaces that hash function with a more suitable function from the hash pool, *i.e.*, rehashing. Note that this rehashing only affects a small portion of the prefixes, the prefixes that hashed to the particular bin group and can be handled in a reasonable time without disturbing the performance, as shown in Section VI-A. In Figure 4 (c), HashTune partitions the hash table into three bin groups (A, B, and C) and uses the two hash functions from Figure 4 (a) and (b) to achieve a more even distribution of prefixes in the hash table. Section VI-A shows that HashTune reduces the number of overflows 98% compared to the single hash function case, and the on-chip memory requirement to a quarter of the single hash function case.

D. RAM-based Black Sheep Memory (BSM)

In this section, we show how FlashLook allows using a RAM-based BSM instead of the conventional CAM-based BSMs. Most hash-based matching engines suffer from the unlikely but possible event of collisions exceeding a bin size,

thus overflowing the bin. The straightforward and accepted approach is to minimize such a possibility when designing the hash table and use a small on-chip CAM to store the overflowed elements [28]–[30]. On-chip CAMs are especially useful since they provide constant time access to the prefixes by their content-addressable nature. Unfortunately, since, on-chip CAMs require a significant amount of resources, the on-chip CAM-based BSM should be very small. Thus, the number of allowed overflows at a given time should also be very small. In the hash table design, this leads to an additional constraint for limiting the number of expected overflows. An on-chip RAM-based BSM can relax this constraint, providing a more efficient hash table. However, RAMs cannot provide the flexibility of CAMs for constant time access to the prefixes.

In this paper, we propose an addressing scheme that utilizes the DRAM, without any performance penalty, to address the on-chip BSM, thus allowing an on-chip RAM-based BSM. Our addressing scheme eliminates the need for using expensive on-chip CAMs for BSM, thus leading to less resource usage per prefix on-chip. As a result, a RAM-based BSM can be used to store more prefixes with less resources compared to the CAMs. This relaxes the constraint introduced by the CAM approach onto the hash table.

The proposed addressing scheme stores a pointer in a hash bin in the external DRAM when the bin overflows. This pointer points back to an address in the BSM that stores those black sheep prefixes hashed to this bin that cannot be stored due to the overflow as shown in Figure 2. The RAM-based BSM allows multiple prefixes to be stored and queried in parallel for each overflow bin. Since up to $v + 1 = 8$ elements can be stored in the RAM-based BSM in the current FlashLook architecture, the possibility of overflowing the BSM is unlikely. Even if such an unlikely event occurs, only the elements in the partition will be rehashed with a new hash function from the hash pool, rather than the entire routing table, which is reasonable.

IV. IMPLEMENTATION OF A 100-GBPS IP ROUTE LOOKUP ARCHITECTURE

In FlashLook, the next hop information of almost all prefixes of IPv4/24, IPv4/32, IPv6/32, IPv6/40, and IPv6/48 are stored in the DRAM chips with some overflows stored in the on-chip BSM. To achieve high-speed operations, multiple copies of each *Next Hop* (NH) table are stored in the DRAM chips. Figure 5 shows a *basic configuration* of 3 DRAM chips, assuming each DRAM chip consists of four banks. Each chip stores 2 copies of IPv4/24 and IPv4/32, where each copy is stored in a separate bank. As a result, 6 copies of IPv4/24 and IPv4/32 in total are stored in the 3 DRAM chips. For IPv6, four copies of each of IPv6/32, IPv6/40, and IPv6/48 are stored in the 3 DRAM chips. The figure also shows the size of burst access. For the common DDR DRAM, there are two burst access lengths: 4 and 8 bursts, which are equivalent to 64 bits and 128 bits, respectively, for a 16-bit data bus. FlashLook uses 8 bursts as the default burst length, but since most DRAMs are now supporting an interrupt after 4 burst

access, we prefer the burst access pattern of 4 – 8 – 4 – 8 to allow the best throughput for reading 64 bits for IPv4/24 (a 4-burst using interrupt) from the first bank followed by reading 128 bits for IPv4/32 from the second bank. The third and fourth banks follow the same pattern.

In DDR DRAM, since 2 words can be read at each clock cycle, bursts of length 4 and 8 can be completed in 2 and 4 clock cycles, respectively. As a result, accessing four banks required 12 clock cycles, when the first and third banks have interrupted after 4 burst accesses (*i.e.*, burst access pattern of 4 – 8 – 4 – 8 for the four banks). If the DRAM clock frequency is 200 MHz, which is equivalent to 5 ns clock period, access to all four blocks can be completed in 60ns.

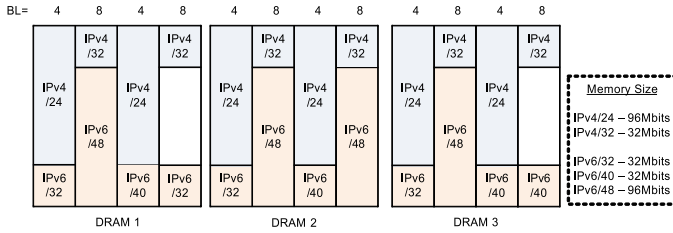


Fig. 5: Data Allocation of DRAMs

Now, let us see how many such basic configurations of 3 DRAM chips are required to achieve a 100-Gbps line rate. For IPv4, we have to finish each IP lookup in 4 ns, assuming the minimum packet size of 40 bytes. With the access restriction of the DRAM chips that the same DRAM bank can only be accessed once in every 60 ns [4], [5], we need $60/4 = 15$ copies of next hop information for each route length. Therefore, we need three basic 3 DRAM chip configurations (9 DRAMs) to achieve a 100-Gbps line rate for IPv4 route lookup. Similarly for IPv6, to achieve a 100-Gbps line rate, we have to finish each IP lookup in 6 ns, assuming a minimum packet size of 64 bytes. With the access restriction of the DRAM chips, we need 10 copies of next hop information for each route, which means that also three basic DRAM configurations are required to achieve a 100-Gbps line rate for IPv6 route lookup. In conclusion, three basic DRAM configurations can achieve a 100-Gbps line rate for both IPv4 and IPv6 route lookup. We generated a IPv4 routing table with 2 million prefixes and an IPv6 routing table with 256 thousand prefixes based on the current prefix distributions [7], [26], [27], [31] to approximate future routing tables. Simulation results show that FlashLook with 9 DDR2 DRAM chips can support these two forwarding tables simultaneously².

V. IPv6

We describe our FlashLook architecture in the previous sections. In this section, we discuss the expansion of FlashLook architecture to accommodate for IPv6. The FlashLook architecture for IPv6 is almost identical to the architecture for IPv4. There are two main differences. First, we need to

²This is based on a 512-Mbit DDR2 DRAM chip with 4 128 Mbits banks. Recent DDR3 DRAM chips have 8 256 Mbits banks with clock frequency up to 800 MHz. Only 5 DDR3 DRAM chips are enough to achieve 100 Gbps.

determine the characteristics of IPv6 routing tables. Although, there are some IPv6 routing tables available from research projects such as the RouteView project, the IPv6 network is still mostly experimental and lacks adequate data to represent future commercial routing tables. Instead, we use the future IPv6 routing table predictions from [31]. There are 128 bits in IPv6 IP addresses, but the first 64 bits are used as the network address. In the IPv6 routing tables more than 95% of the prefixes have lengths between 25 to 48 bits. Majority of the prefixes have a length of 48 bits similar to the case in IPv4 for prefixes with a length of 24 bits. Based on this observation, we expand prefixes to 32, 40, and 48 bits and denote the set of prefixes after prefix expansion as IPv6/32, IPv6/40, and IPv6/48, respectively. Since prefixes with lengths up to 24 bits and between 49 and 64 bits constitute no more than 5% of the total prefixes, we store these prefixes in on-chip hash tables, while storing the other prefixes in off-chip hash tables.

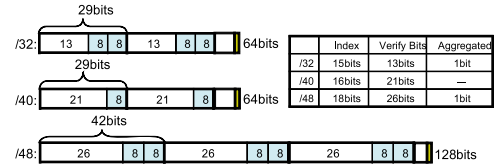


Fig. 6: DRAM bin structure for IPv6 /32, /40, and /48.

Secondly, the hash table bin structure on the DRAM is also modified from the IPv4 structure to accommodate IPv6 tables as shown in Figure 6. IPv6/32 and IPv6/40 use a 4 burst (*i.e.*, 64 bits bin size) and IPv6/48 uses an 8 burst (*i.e.*, 128 bits bin size). IPv6 allows one additional DRAM access compared to IPv4, since the minimum IPv6 packet size is larger than minimum IPv4 packet size (*i.e.*, 64 bytes vs. 40 bytes). Thus, there is 50% more time for IPv6 for route lookup compared to IPv4.

VI. PERFORMANCE EVALUATION

In this section, we discuss performance of FlashLook including the route update performance.

A. HashTune and BSM Performance

To evaluate the performance of HashTune, we hashed the 2, 685, 348 IPv4/24 prefixes from the IPv4 table we generated to a hash table with 1, 572, 864 bins, where each bin has a capacity $c=3$. Then we observed the overflows (*i.e.*, number of prefixes hashed to a bin in excess of $c=3$ prefixes). The results are shown in Figure 7. For HashTune, results for different pool sizes, ranging from 2 hash functions to 64 hash functions, are shown. For comparison, the best result without the HashTune using a single hash function is also shown. This single hash function is the one with the minimum number of overflows among the 64 hash functions in the pool. Additionally, since bin occupancy distribution in a hash table follows the Poisson Distribution, the figure also shows the corresponding Poisson Distribution (with $\lambda = 1.707$). Figure 7 also shows the on-chip memory requirement for the HashTune including BSM. As expected, the number of bins with overflows is high for

the single hash function case, and decreases sharply as the hash pool size grows. For instance, using a hash pool of size eight, 18,917 prefixes overflow, requiring 563 kbits for BSM, whereas the overflows reduce to 669 prefixes and require only 20 kbits of BSM memory for the hash pool size of 64, corresponding to a 96% reduction in the overflows and the BSM size. In particular, the HashTune provides the best performance trade-off between number of overflows and on-chip memory when it uses 8 or 16 hash functions, when 98% of overflows is eliminated and three quarters of the on-chip memory is saved compared to a single hash function case. It should also be noted that even 64 hash functions can easily be implemented on-chip without any significant resource usage at high speed.

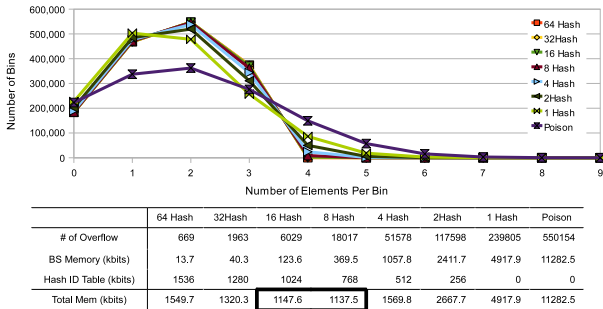


Fig. 7: HashTune Performance

B. Index Bits

Selecting good index bits is important to achieve a balanced hash table. Index bits identify groups and we define the good index bits as any bit that can disperse routes equally among all the bins. To determine the possible good index bits, we calculated the binary entropy of each IP address bit for the Oregon routing table as $H_i = -p_i \log_2 p_i - (1 - p_i) \log_2 (1 - p_i)$, where H_i is the entropy of the bit at position i ($1 \leq i \leq 32$) and bit position 1 corresponds to the most-significant bit. p_i is the percentage of 1s at bit position i . If a bit with an entropy of 1 is used to distribute routes into two bins, each bin will receive an equal number of routes. As the bin entropy becomes smaller, the distribution becomes biased. The results of the binary entropy calculations are plotted in Figure 8 for IPv4/24 and IPv4/32 prefix sets separately before and after prefix expansion. The y -axis shows the entropy and the x -axis represents the bit position in the IP address. Before prefix expansion, only the bits in the middle, namely bits 8 – 23, provide good randomness ($H_{8-23} \approx 1$). However, after prefix expansion all the bits in the range 8 – 32 provide good randomness. Thus, we use these bits as index bits. More specifically, IPv4/24 uses bit positions 6 – 23 and IPv4/32 uses bit positions 16 – 31 as the index bits. Prefix expansion improves the randomness of less-significant bits because, each expanded prefix produces an equal number of ones and zeros at less-significant bits. These expanded prefixes dominate the small number of long prefixes that were there before expansion, and have a non-uniform distribution in these bits.

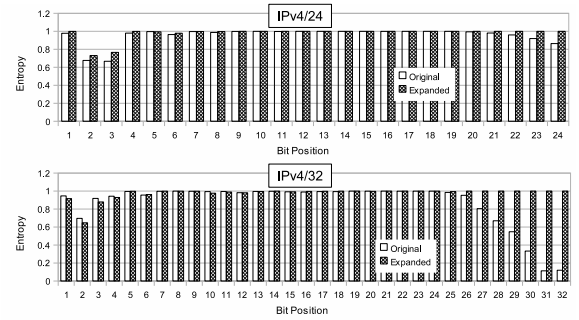


Fig. 8: Bit entropy for IPv4/24 and IPv4/32 before and after prefix expansion.

C. Verify Bit Aggregation

Recall that verify bit aggregation reduces the number of elements by aggregating verify bits from child nodes. One concern for this optimization is that when a parent node only has one child, one of the next hop information sets has to keep unimportant information, such as the default route. If most of the parents have one-child, the savings obtained by the reduction in the number of elements becomes insignificant compared to the overhead in the increased element size. Using real routing table entries, we constructed the corresponding FlashLook data structure. The number of elements is shown in Table II. Ideally, each level of aggregation reduces the number of elements by half.

The original numbers of prefixes are 190,692 and 2,968 for IPv4/24 and IPv4/32, respectively. The number of elements after route expansion but before aggregation (non-aggregated element) is listed in the second column. The columns that follow show the resulting number of elements when up to 2, 4, 8, and 16 verify bits are allowed to be aggregated into one element. The simulation result shows that the numbers of aggregated elements reduced by almost half at each increase of aggregation level. This shows that the verify bit aggregation operation significantly reduced the number of elements.

TABLE II: Verify Bit Aggregation

	Non-Aggregated Element	Aggregated Element			
		2	4	8	16
IPv4/24	889,371	463,765	247,751	136,313	77,117
IPv4/32	152,794	76,431	38,244	19,365	9,958

D. Memory Usage

Based on the results given above, FlashLook requires 2 Mbits for the IPv4 direct lookup table, 3.5 Mbits for IPv6 Hash (/4- /24 and /49-/64), 1 Mbit for BSM, and 2.5 Mbits for the Hash ID table. As a result, the total on-chip memory requirement of FlashLook is 9 Mbits, which can comfortably fit into today's on-chip RAMs on off-the-shelf FPGAs.

E. Update

The update speed is a major parameter for next generation routers. There are three types of updates: (1) Changing the next hop for a prefix, (2) adding a new prefix to the table and (3) withdrawing a prefix from the table. According to

CIDR statistics obtained during Dec. 2008, to Jan. 2009 for a period of 31 days, current routing tables have 3.64 prefix updates per second on average, where updates can be either changing a next hop or adding a new prefix. The peak update and withdrawal rate can go as high as 2,101 and 32,282 updates, respectively.

For FlashLook (1) and (3) can be handled with simple memory updates in constant time. On the other hand, inserting a new route may have more overhead because of the possibility of rehashing. For each index group the best hash function was chosen by HashTune, but the insertion process may cause unbalancing in the memory occupancy for an index group. Then, multiple memory updates need to be done after a rehashing process. As a solution, we do not immediately rehash an unbalanced index group. Instead, we first put the new items, which cause the unbalance, to the BSM without rehashing. Second, the FlashLook design reserves enough time for updates, 50 Mpps, which can be used for updates and other maintenance. The updates can also be scheduled whenever a packet larger than a minimum-sized packet is received. Since FlashLook is designed to support even traffic consisting of only minimum-sized packets, any larger packet will spare FlashLook additional maintenance time.

VII. CONCLUSION

In this paper, we propose a low-cost next generation route lookup architecture that can support 2 million and 256 thousand routes for IPv4 and IPv6, respectively. A flexible hash scheme named HashTune and DRAM based architecture permit programming large routing tables while achieving 250 million packet lookups per second. We have implemented a preliminary prototype of FlashLook on a Xilinx Virtex-4 FPGA chip, which can run up to a clock rate of 250 MHz.

REFERENCES

- [1] P. M. M. Cvijetic, "Delivering on the 100GbE Promise," *IEEE Communications Magazine*, vol. 45, no. 12, pp. 2–3, Dec. 2007.
- [2] IEEE P802.3ba 40Gb/s and 100Gb/s Ethernet Task Force. [Online]. Available: <http://www.ieee802.org/3/ba/>
- [3] Computer Industry Almanac Inc. [Online]. Available: <http://www.c-i-a.com/pr0207.htm>
- [4] Samsung DRAM. [Online]. Available: <http://www.samsung.com/>
- [5] Micron DRAM. [Online]. Available: <http://www.micron.com/>
- [6] 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, 2004.
- [7] V. Srinivasan and G. Varghese, "Fast address lookups using controlled prefix expansion," *ACM Trans. Comput. Syst.*, vol. 17, no. 1, pp. 1–40, 1999.
- [8] F. Zane, G. Narlikar, and A. Basu, "CoolCAMs: Power-Efficient TCAMs for Forwarding Engines," in *Proc. of the IEEE Computer and Communications Societies (INFOCOM 2003)*, vol. 1, Mar/Apr 2003, pp. 42–52 vol.1.
- [9] K. Zheng, C. Hu, H. Lu, and B. Liu, "A TCAM-Based Distributed Parallel IP Lookup Scheme and Performance Analysis," *IEEE/ACM Trans. Netw.*, vol. 14, no. 4, pp. 863–875, 2006.
- [10] P. Gupta, S. Lin, and N. McKeown, "Routing Lookups in Hardware at Memory Access Speeds," in *Proc. of the IEEE Computer and Communications Societies (INFOCOM 1998)*, vol. 3, Mar/Apr 1998, pp. 1240–1247 vol.3.
- [11] N.-F. Huang and S.-M. Zhao, "A Novel IP-Routing Lookup Scheme and Hardware Architecture for Multigigabit Switching Routers," *IEEE Journal on Selected Areas in Comm.*, vol. 17, no. 6, pp. 1093–1104, Jun. 1999.
- [12] N.-F. Huang, S.-M. Zhao, J.-Y. Pan, and C.-A. Su, "A Fast IP Routing Lookup Scheme for Gigabit Switching Routers," in *Proc. of the IEEE Computer and Communications Societies (INFOCOM 1999)*, vol. 3, Mar. 1999, pp. 1429–1436 vol.3.
- [13] Y.-C. Liu and C.-T. Lea, "Fast IP Table Lookup and Memory Reduction," in *Proc. of the IEEE Workshop on High Performance Switching and Routing (HPSR 2001)*, 2001, pp. 228–232.
- [14] S. Sikka and G. Varghese, "Memory-Efficient State Lookups with Fast Updates," in *Proc. of the ACM Special Interest Group on Data Communication (SIGCOMM 2000)*. New York, NY: ACM, 2000, pp. 335–347.
- [15] R. Sangireddy, N. Futamura, S. Aluru, and A. K. Somani, "Scalable, Memory Efficient, High-Speed IP Lookup Algorithms," *IEEE/ACM Trans. Netw.*, vol. 13, no. 4, pp. 802–812, 2005.
- [16] H. Song, J. Turner, and J. Lockwood, "Shape Shifting Tries for Faster IP Route Lookup," in *Proc. of the IEEE International Conference on Network Protocols (ICNP2005)*, Nov. 2005, pp. 10 pp.–367.
- [17] A. Basu and G. Narlikar, "Fast Incremental Updates for Pipelined Forwarding Engines," *IEEE/ACM Trans. Netw.*, vol. 13, no. 3, pp. 690–703, 2005.
- [18] W. Jiang and V. K. Prasanna, "Multi-Terabit IP Lookup Using Parallel Bidirectional Pipelines," in *Proc. of the ACM International Conference on Computing Frontiers (CF 2008)*. New York, NY: ACM, 2008, pp. 241–250.
- [19] J. van Lunteren, "Searching Very Large Routing Tables in Fast SRAM," in *Proc. of the IEEE International Conference on Computer Communications and Networks (ICCCN 2001)*, 2001, pp. 4–11.
- [20] H. S. Srihari Cadambi, Srimat Chakradhar, "Prefix Processing Technique for Faster IP Routing," US Patent 2006/0 200 581 A1, 7 398 278, May., 2005, issue date Jul 8, 2008.
- [21] S. Kaxiras and G. Keramidas, "IPStash: A Set-associative Memory Approach for Efficient IP-Lookup," in *Proc. of the IEEE Computer and Communications Societies (INFOCOM 2005)*, vol. 2, Mar. 2005, pp. 992–1001 vol. 2.
- [22] J. Hasan, S. Cadambi, V. Jakkula, and S. Chakradhar, "Chisel: A Storage-efficient, Collision-free Hash-based Network Processing Architecture," in *Proc. of the International Symposium on Computer Architecture (ISCA 2006)*, 2006, pp. 203–215.
- [23] X. Hu, X. Tang, and B. Hua, "High-Performance IPv6 Forwarding Algorithm for Multi-Core and Multithreaded Network Processor," in *Proc. of the ACM SIGPLAN symposium on Principles and Practice of Parallel Programming (PPoPP 2006)*. New York, NY: ACM, 2006, pp. 168–177.
- [24] H. Song, S. Dharmapurikar, J. Turner, and J. Lockwood, "Fast Hash Table Lookup Using Extended Bloom Filter: An Aid to Network Processing," in *Proc. of the ACM Special Interest Group on Data Communication (SIGCOMM 2005)*. New York, NY, USA: ACM, 2005, pp. 181–192.
- [25] S. Dharmapurikar, P. Krishnamurthy, and D. E. Taylor, "Longest Prefix Matching Using Bloom Filters," *IEEE/ACM Trans. Netw.*, vol. 14, no. 2, pp. 397–409, 2006.
- [26] University of Oregon Route Views Project. [Online]. Available: <http://www.routeviews.com/>
- [27] CIDR Report. [Online]. Available: <http://www.cidr-report.org/>
- [28] S. Kumar and P. Crowley, "Segmented Hash: An Efficient Hash Table Implementation for High Performance Networking Subsystems," in *Proc. of the ACM Symposium on Architecture for Networking and Communications Systems (ANCS 2005)*, 2005, pp. 91–103.
- [29] A. Kirsch and M. Mitzenmacher, "The Power of One Move: Hashing Schemes for Hardware," in *Proc. of the IEEE Conference on Computer Communications (INFOCOM 2008)*, Apr. 2008, pp. 106–110.
- [30] N. Artan, H. Yuan, and H. Chao, "A Dynamic Load-Balanced Hashing Scheme for Networking Applications," in *Proc. of the IEEE Global Communications Conference (GLOBECOM 2008)*, Dec. 2008, pp. 1–6.
- [31] M. Wang, S. Deering, T. Hain, and L. Dunn, "Non-Random Generator for IPv6 Tables," in *Proc. of the IEEE Symposium on High Performance Interconnects.*, Aug. 2004, pp. 35–40.