applied sciences

MDPI

Article

Vectored-Bloom Filter for IP Address Lookup: Algorithm and Hardware Architectures

Hayoung Byun, Qingling Li and Hyesook Lim *

Department of Electronic and Electrical Engineering, Ewha Womans University, Seoul 03760, Korea; hayoung77@ewhain.net (H.B.); liqingling05@gmail.com (Q.L.)

* Correspondence: hlim@ewha.ac.kr; Tel.: +82-2-3277-3403

Received: 28 September 2019; Accepted: 25 October 2019; Published: 30 October 2019

Abstract: The Internet Protocol (IP) address lookup is one of the most challenging tasks for Internet routers, since it requires to perform packet forwarding at wire-speed for tens of millions of incomming packets per second. Efficient IP address lookup algorithms have been widely studied to satisfy this requirement. Among them, Bloom filter-based approach is attractive in providing high performance. This paper proposes a high-speed and flexible architecture based on a vectored-Bloom filter (VBF), which is a space-efficient data structure that can be stored in a fast on-chip memory. An off-chip hash table is infrequently accessed, only when the VBF fails to provide address lookup results. The proposed architecture has been evaluated through both a behavior simulation with C language and a timing simulation with Verilog. The hardware implementation result shows that the proposed architecture can achieve the throughput of 5 million packets per second in a field programmable gate array (FPGA) operated at 100 MHz.

Keywords: Bloom filter; IP address lookup; vectored-bloom filter; FPGA; hardware accelerator

1. Introduction

The global IP traffic forecast provided by Cisco Systems reported that the annual run rate for global IP traffic was 1.5 Zeta bytes per year (122 EB per month) in 2017, and the global IP traffic will reach 4.8 ZB per year by 2022, which means it will increase nearly threefold over the next five years. The rapid growth of traffic has made packet forwarding in routers a bottleneck in constructing high-performance networks.

An IP address is composed of a network part and a host part. The network part indicates a group of hosts included in a network while the host part indicates a specific host [1]. The network part is called a prefix and hosts connected to the same network have the same prefix. In a class-based addressing scheme, the prefix length is fixed at 8, 16, or 24 bits and routers perform an exact match operation for an IP address lookup. However, because of the excessive address wasting caused by the inflexibility of the prefix lengths under the class-based addressing scheme, a new addressing scheme called classless inter-domain routing (CIDR) has been introduced. In the CIDR scheme, arbitrary prefix lengths are allowed and routers identify the longest prefix among all matching prefixes as the best matching prefix (BMP) for an IP address lookup [2–4].

Various IP address lookup algorithms have been researched, including trie-based [2,3], hash table-based [5] and Bloom filter-based algorithms [6,7]. Since an access to an off-chip memory is 10-20 times slower than an access to an on-chip memory [8], reducing the number of off-chip memory accesses required for looking up an IP address is the most effective strategy [6,7]. While the trie and the hash table are generally stored in off-chip memories due to their sizes, a Bloom filter is an efficient structure that can be stored in an on-chip memory.

In building a packet forwarding engine, various hardware components, such as application- specific integrated circuits (ASIC), ternary content addressable memory (TCAM) and field-programmable gate arrays (FPGA), are used to satisfy the wire-speed packet forwarding requirement. As a flexible and programmable device, the FPGA has a matrix of configurable logic blocks connected through programmable interconnects. In particular, FPGAs have been widely used to build the prototypes of packet forwarding engines [9–13] and intrusion detection systems [14–16].

The contribution of this paper is as follows. We propose a vectored-Bloom filter (VBF) architecture, which is a multi-bit vector Bloom filter for the purpose of IP address lookup. The VBF is proposed to obtain lookup results by accessing only an on-chip memory. An off-chip hash table is rarely accessed when the VBF fails to provide the results. We have evaluated our proposed architecture in two steps. The construction procedures of the VBF and the hash table are implemented using C at behavior level, since the construction procedure is not necessarily performed in real-time. The performance at the behavior level has been evaluated in terms of on-chip memory requirement, off-chip memory requirement, the average number and the worst-case number of memory accesses, indeterminable rate and false port return rate. The search procedure that should be performed in real-time is implemented using Verilog with a single FPGA. The performance of the FPGA has been evaluated in terms of block-RAM (BRAM) requirement, resource utilization and throughput.

The remainder of this paper is organized as follows. Section 2 briefly explains the Bloom filter and previous IP address lookup algorithms utilizing on-chip memories. Section 3 describes our proposed IP address lookup algorithm using a vectored-Bloom filter including theoretical analysis on the search failure probability of our proposed algorithm. Section 4 describes the hardware architecture implemented on an FPGA. Section 5 shows behavior simulation results comparing the performance of the proposed VBF with other structures. Section 6 shows the hardware implementation details of our proposed structure. Finally, Section 7 concludes the paper.

2. Related Works

2.1. Bloom Filter

A Bloom filter [17] is a multi-bit probabilistic data structure used for membership querying to determine whether an input is the element included in a given set. Bloom filters have been applied to many network applications due to their space-efficient attributes [18–21] and hardware-friendly features [22–26].

A Bloom filter has two operations: programming and querying. In programming, the membership information of each element in a given set is stored using a number of hash indexes, which are obtained by entering each specific element as the input of a hash function. In querying, the membership of each input is checked using the same hash function as used in programming [27]. Even though a Bloom filter requires multiple hash indexes for the programming and the querying, the hash indexes can be easily obtained using a single hash function and a few simple operations [28,29].

In programming set $S = \{x_1, x_2, \cdots, x_n\}$, every bit of the m-bit Bloom filter is initially 0 and k different hash indexes are used to map each element to k locations of the m-bit array. Let n be the number of elements in the programming set. The optimal number of hash indexes is defined as follows.

$$k = \frac{m}{n} \ln 2 \tag{1}$$

In order to program an element in the set, the k bits pointed by the k hash indexes obtained using the element are all set to 1. If a specific bit location in the m-bit array was already 1, it is not changed.

A Bloom filter provides whether or not an input is a member of the programmed set by querying. In querying input y, the k hash indexes obtained by the same procedure as in the programming are used. The querying to the Bloom filter produces two types of results: *negative* or *positive*. If any of the k-bit locations are 0, y is definitively not a member of set S and is thus termed a *negative*. If all of the

k-bit locations are 1, y is considered a member of set S and thus termed a *positive*. However, a Bloom filter can produce a false positive owing to hash collision even if $y \notin S$. For n elements, the false positive rate f of an m-bit Bloom filter is obtained as follows [27].

$$f = \left(1 - (1 - \frac{1}{m})^{kn}\right)^k \approx (1 - e^{-kn/m})^k \quad (2)$$

As shown, the false positive rate can be reduced by increasing the size of the Bloom filter but cannot be completely eliminated. Larger Bloom filters require larger k according to Label (1), resulting in more overhead. However, since the on-chip processing overhead for the Bloom filter stored in an on-chip memory is much smaller compared with the off-chip processing overhead for accessing off-chip memories, this paper aims to reduce the number of off-chip memory accesses by utilizing an on-chip Bloom filter.

2.2. IP Address Lookup Algorithms Utilizing On-Chip Memories

In the parallel Bloom filter (PBF) architecture proposed by Dharmapurikar et al. [6], W Bloom filters associated with each prefix length are maintained, where W is the number of distinct prefix lengths. In programming, prefixes with the same length are programmed into the same Bloom filter. In search procedure, for an input IP address, all of the Bloom filters stored in on-chip memories are first queried in parallel. For the lengths where positive results are returned by the Bloom filters, an off-chip hash table is sequentially accessed starting from the longest length to find the output port of the best matching prefix.

In the Bloom filter chaining (BF-chaining) architecture proposed by Mun et. al. [20], a binary trie is primarily constructed. Every node in the binary trie is programmed into an on-chip Bloom filter, while prefixes are only programmed in an off-chip hash table. For an input IP address, the Bloom filter is sequentially queried from the root node until a negative result is produced. Since a node cannot exist without an ancestor node, the trie level (prefix length) of the last positive would have the longest matching node for the input (if the positive is true positive). The off-chip hash table is accessed for this length. If the positive is a false positive or if the node is an empty node without storing a prefix, a back-tracking should occur, which accesses for a shorter length. The back-tracking by the false positive cannot be avoided but the back-tracking by the empty node can be avoided by pre-computation, which makes each empty node store the output port of the direct ancestor prefix. An IP address lookup is achieved by a single off-chip memory access if the Bloom filter positive is true.

However, the PBF and BF-chaining architectures require at least one off-chip memory access because the on-chip Bloom filter does not return the output port of the longest matching prefix.

Yang et al. [30] proposed a splitting approach for IP address lookup (SAIL). The SAIL architecture applies the splitting along with prefix lengthes and focuses to speed-up the IP address lookup with short lengths using on-chip bitmaps. The SAIL also requires at least one off-chip access because the on-chip bitmap does not return the matching output port.

3. IP Address Lookup Using a Vectored-Bloom Filter

The basic idea of the proposed vectored-Bloom filter (VBF) structure for IP address lookup was briefly introduced in References [31] and [32]. The VBF consists of m multi-bit vectors and each vector comprised of l bits contains an output port. The proposed structure completes an IP address lookup by querying only the on-chip VBF without accessing the off-chip hash table. Depending on how output ports are programmed to the VBF, two different structures are possible [31]. The first structure involves making the l-bit vector represent up to $2^l - 2$ output ports, zero and a *conflict* value, which is the value of $2^l - 1$. The *conflict* indicates that the vector is programmed by two or more different output ports.

In programming a prefix, the k vectors pointed by the k hash indexes of the prefix are written with the output port of the prefix. If any of the vectors already have an output port other than the

output port, all of the bits in the vector are set to 1 in order to represent the *conflict* ($2^l - 1$). In querying an input, if all the k vectors located by the k hash indexes of the input have the *conflict* values, the VBF cannot return an output port and the result is termed an *indeterminable*. If any of the k vectors has 0, the input is not a prefix and hence the result is termed a *negative*. If the vectors other than *conflict* have the same value, the input is a prefix and the value is the output port of the prefix and hence the result is termed a *positive*.

The second structure involves making the l-bit vector represent l output ports by assigning each bit in a vector an output port. Hence, two or more different output ports can be represented in a vector by setting the corresponding bits and the *conflict* value is not necessarily defined.

The motivation of the proposed VBF structure is to complete the IP address lookup by only searching the VBF implemented with an on-chip memory, without accessing a hash table implemented with an off-chip memory. The hash table is infrequently accessed only when the VBF fails to provide the output port. The VBF structure is shown in Figure 1. Assuming that output ports are uniformly distributed, for n given prefixes, the optimal number of hash functions for the VBF is defined as follows, similarly to Label (1).

$$k_v = \frac{ml}{n} \ln 2 \tag{3}$$

since a VBF can be considered with l m-bit Bloom filters, each of which is programmed by n/l prefixes.

In this paper, we describe the second structure in detail including the hardware architecture which was not described in Reference [31]. Hence, Section 4 is completely a new section compared with Reference [31]. In Reference [32], even though the hardware architecture of the VBF implemented on the FPGA is briefly described, the proposed structure has not been evaluated through a behavior simulation and not been compared with other algorithms. In this paper, we describe our algorithm and the proposed hardware structures in detail and the proposed architecture has been evaluated through both a behavior simulation with C language and a timing simulation with Verilog. In addition, we compare our algorithm with other algorithms with a large scale. Hence. the Section 5 is completely a new section and Section 6 is largely extended compared with Reference [32].

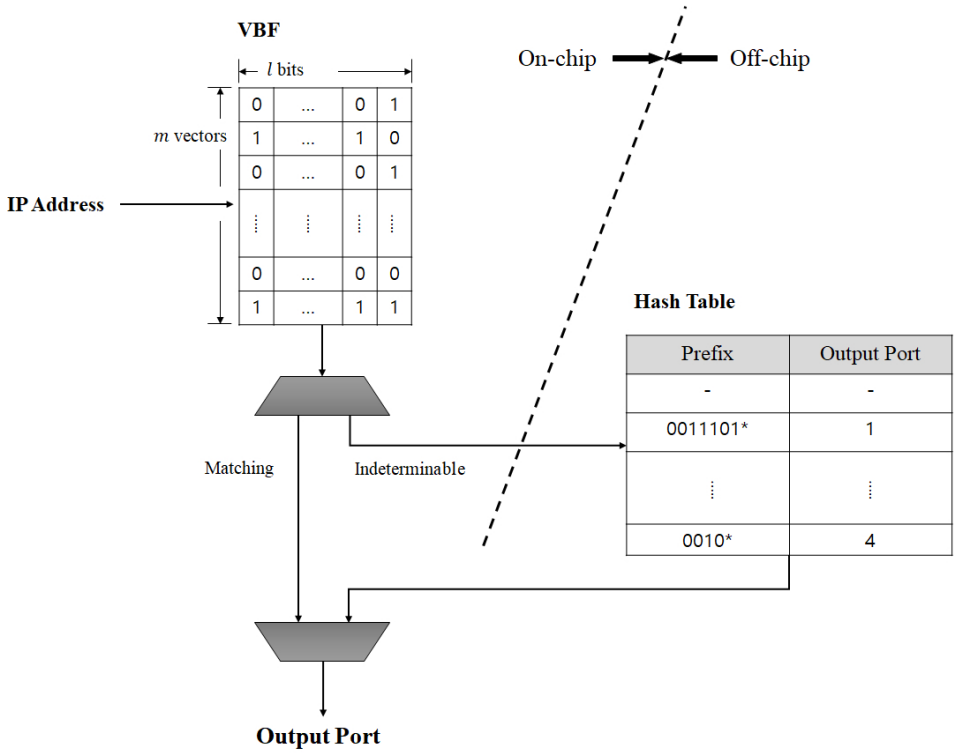

Figure 1. Overall structure of vectored Bloom filter (VBF) algorithm.

3.1. VBF Programming

Algorithm 1 describes the construction procedure of the VBF. All of the prefixes in a routing set are programmed into the VBF and also stored in a hash table. For prefix x in a routing set with output port $x.port$, k_v hash indexes are first obtained. In order to program the output port to the k_v vectors pointed by the k_v indexes, each bit location corresponding to $x.port$ is set to 1.

Algorithm 1: Programming Procedure of VBF

```
Function programVBF(x)
    for (i = 1 to k_v) do
        BF[h_i(x)][ x.port − 1] = 1;
    end
end
```

3.2. VBF Querying

Algorithm 2 describes the querying procedure of the VBF in detail. For a given input key, k_v indexes are first obtained and a bit-wise AND operation is then performed for k_v vectors obtained from the VBF in order to obtain a l-bit result vector, *check*. The querying has three possible results: *negative*, *positive* or *indeterminable*. If all of the bits in the *check* are 0, the input was definitively not programmed to the VBF and the result is termed a *negative*. If the *check* has a single set bit, the input is considered a member of the routing set and the location of the set bit is returned as the matching output port; the result is then termed a *positive*. If the *check* has plural set bits, the VBF cannot return a value and the result is termed an *indeterminable*. In Algorithm 2, outPort is the matching output port and *counter* is the number of set bits in the *check*. Therefore, if *counter* is 1, it means a *positive* and outPort is returned as the matching port. If *counter* is larger than 1, it means an *indeterminable* and a matching port cannot be returned from the VBF.

Algorithm 2: Querying Procedure of VBF

```
Function queryVBF(y)
    check = BF[h_0(y)]&BF[h_1(y)]&...&BF[h_{k_v−1}(y)];
    counter = 0;
    for (i = 0 to l − 1) do
        if ( check[i] == true ) then
            outPort = i + 1;
            counter = counter + 1;
            if ( counter >1 ) then
                break;
            end
        end
    end
    if ( counter == 0 ) then
        return 0; // negative
    else if ( counter == 1 ) then
        return outPort; // positive
    else
        return -1; // indeterminable
    end
end
```

3.3. Hash Table Implementation

In implementing the hash table, hash collisions should be carefully considered. In order to reduce the number of collisions, we use two hash indexes and a linked list to store a prefix into the hash table. In other words, storing a prefix can have a maximum of three choices of hash entries. The first hash index has the highest priority. If both entries indicated by the two hash indexes have already been filled by other prefixes, the prefix is stored in the entry indicated by the linked list. Each hash table entry stores a prefix, the length of the prefix, a corresponding output port and a linked list.

3.4. IP Address Lookup Using VBF

Algorithm 3 describes the IP address lookup procedure using the VBF in detail. For a given input IP address, the VBF is queried by gradually decreasing the querying length, starting from the longest prefix length.

If the VBF returns a *negative* at the current length, the querying is continued at a shorter length (if the current length is not already the shortest).

If the result of the VBF is a *positive*, the VBF returns a matching output port (BMPport) and the IP address lookup procedure is completed. Note that the VBF can generate a *false positive* for a substring of the given input even though the substring is not included in the routing set and a false output port can be returned for the input. As will be shown in simulation, the *false port return rate* converges to zero when the sizing factor of the VBF is larger than two.

If the VBF returns an *indeterminable*, the off-chip hash table should be accessed because the matching output port at the current length of the given input cannot be determined. In searching the hash table, two hash indexes are used for multi-hashing. If the matching entry is found from the entry pointed by the first hash index, the search is complete. If the matching entry is not found, the procedure continues to search the hash entry pointed by the second hash index. If the matching entry is not found, the procedure continues to search the hash entry pointed by the linked list. The search procedure is completed if the matching entry is found. Otherwise, the search procedure should go back to the VBF.

Algorithm 3: IP Address Lookup Procedure Using VBF

```
Function Search(DstAddr)
    for (length = longestLen to shortestLen) do
        if ( queryVBF(DstAddr.length) == 0 ) then
            continue; //negative
        else if ( queryVBF(DstAddr.length) != 0 ) & ( queryVBF(DstAddr.length) != -1) then
            //positive
            BMPport = queryVBF(DstAddr.length);
            break;
        else
            //indeterminable
            BMPport = searchHT(DstAddr.length);
            if (BMPport != NULL ) then
                break; // no more Bloom filter access
            end
        end
    end
    return BMPport;
end
```

3.5. Theoretical Analysis on Search Failure Probability

In this section, we present the theoretical probability of search failure of the proposed VBF. The search failure in the VBF occurs in two cases; *false port return* and *indeterminable*. The theoretical

probability of the search failure for the first architecture of the VBF (summarized in the first paragraph of Section 3) has been provided in Reference [21] and this section provides the theoretical search failure probability for the second structure of the VBF. The VBF consists of m multi-bit vectors and each vector comprised of l bits contains l different ports. For n elements included in programming set S, assuming that the elements are equally distributed to each port, the number of elements in each port set, n', is equal to n/l.

In querying, while *false positives* can occur for non-programmed inputs, *indeterminables* can occur for all inputs including programmed inputs. If the VBF returns a value for a non-programmed input, it is a *false positive*, which means that a single port is returned among l values. If the VBF returns two or more values, it is an *indeterminable*, which means that more than one port is returned among l ports in one querying.

Let p represent the probability that a specific bit in a vector is set at least once by n' elements, after programming all elements by k hash functions. If the hash functions are assumed to be perfectly random, p can be calculated as

$$p = 1 - (1 - \frac{1}{m})^{kn'}. \tag{4}$$

When querying input x not included in set S, a *false positive* can occur if the same bit locations in each of k vectors are 1 and other bit locations in each of k vectors are 0. Hence, the *false port return* probability, $P(F)$, is as follows.

$$P(F) = P(S^c)P(F|S^c) = P(S^c) \cdot \left(l \cdot (p^k) \cdot (1 - p^k)^{l-1}\right) \tag{5}$$

An *indeterminable* can occur if there exist more than one port regardless of true or false ports. The *indeterminable* probability, $P(I)$, is defined as follows.

$$P(I) = P(S)P(I|S) + P(S^c)P(I|S^c) \tag{6}$$

$P(I|S)$ is the *indeterminable* probability for the inputs included in S. In case of inputs included in S, since one true port must occur, an *indeterminable* can occur if the number of false ports is from 1 to $l - 1$ times. Therefore, $P(I|S)$ is as follows.

$$P(I|S) = 1 - (1 - p^k)^{l-1} \tag{7}$$

$P(I|S^c)$ is the *indeterminable* probability for the inputs not included in S. In case of inputs not included in S, an *indeterminable* can occur if the number of false ports is from 2 to l times. Therefore, $P(I|S^c)$ is as follows.

$$P(I|S^c) = 1 - (1 - p^k)^l - P(F|S^c) \tag{8}$$

From Labels (7) and (8), the *indeterminable* probability is

$$\begin{aligned} P(I) &= P(S)\left(1 - (1 - p^k)^{l-1}\right) + P(S^c)\left(1 - (1 - p^k)^l - P(F|S^c)\right) \\ &= P(S)\left(1 - (1 - p^k)^{l-1}\right) + P(S^c)\left(1 - (1 - p^k)^l - l \cdot (p^k) \cdot (1 - p^k)^{l-1}\right) \end{aligned} \tag{9}$$

From Label (4), since p is the function of the size of the Bloom filter, m, the *false port return* probability in Label (5) and the *indeterminable* probability in Label (9) can be controlled by the size of the Bloom filter.

4. Hardware Architecture

The construction procedures of the VBF and the hash table are implemented at the behavior level using C language, since they are not necessarily performed in real-time. The IP address lookup

procedure described in Algorithms 2 and 3 is implemented using Verilog on a single FPGA. In this section, we describe the hardware architecture of the VBF which is implemented on the FPGA.

4.1. Basic IP Address Lookup Module

Figure 2 shows the block diagram of our proposed IP address lookup module in its basic form. The basic IP address lookup module contains a hash index generator, a VBF search block and a hash table search block. The hash index generator is realized with a 64-bit cyclic redundancy check (CRC-64) generator. The CRC-64 is implemented with a shift register and few exclusive-OR gates, which is much simpler than MD5 and SHA-256 and hence the CRC-64 is used in our implementation.

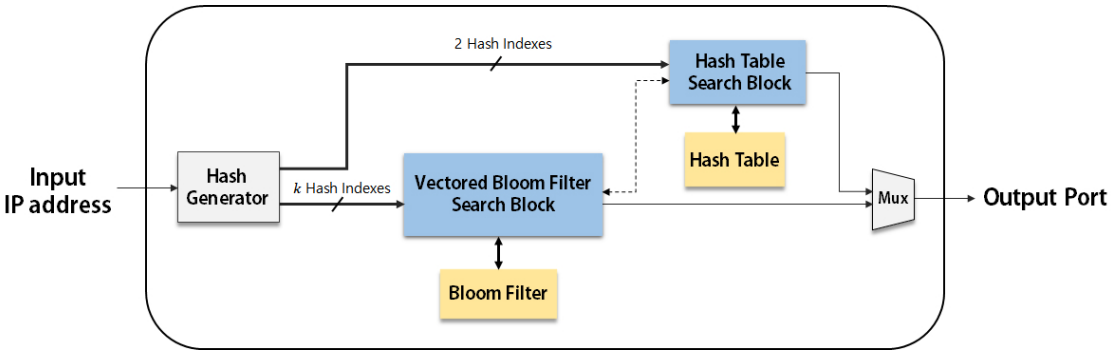

Figure 2. Block diagram of *Basic IP Address Lookup Module*: the bold lines represent data paths and the dotted line represents a control path.

Figure 3 shows the state machine representing the IP address lookup procedure in detail. When the reset signal is given at state **Start**, the state machine is ready to accept an input address. When an IP address is given to the module, the lookup procedure starts from the longest prefix length and iterates by decreasing the length until a matching prefix is found.At state **CRC Gen**, for a substring of the input IP address, the CRC code is first obtained when the corresponding number of input bits are serially entered into the CRC generator starting from the most significant bit. Multiple hash indexes are obtained by combining multiple bits in the CRC code. In our implementation, to obtain the CRC code within a single cycle, the operations required in the 64-bit CRC generator are implemented in parallel.

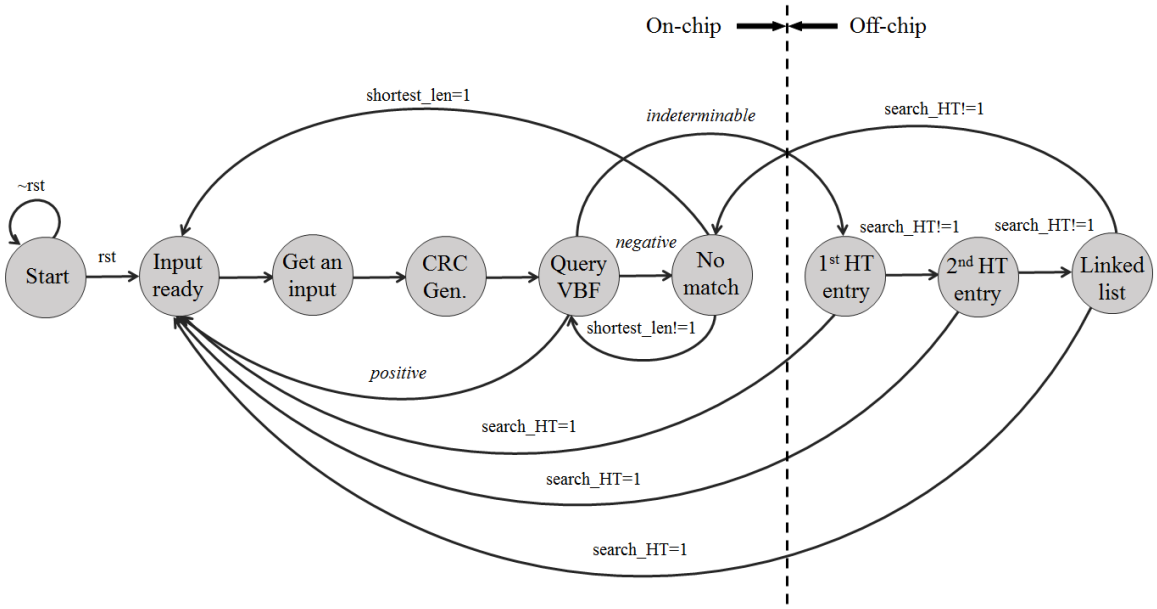

Figure 3. State machine for Internet Protocol (IP) address lookup procedure.

The vectors of the VBF pointed by hash indexes are bitwise-ANDed to obtain a result vector at state **Query VBF**. The vector can produce three different results: *negative*, *positive* or *indeterminable*. The negative result means no matching output port at the current length; the search procedure goes on to state **No match** and then continues to query the VBF with a shorter length. The positive result means a matching output port from the VBF and hence the search is completed for the current input; in this case, the search procedure goes to state **Input ready** for the next input. The indeterminable result means that an output port cannot be returned from the VBF and hence the search procedure needs to proceed to the hash table.

At State **1st HT entry**, the hash entry pointed by the first hash index is accessed. If the entry does not have a matching prefix, another entry pointed by the second hash index is accessed at state **2nd HT entry**. If the entry pointed by the second hash index does not have a matching prefix, the other entry pointed by the linked list of the second entry is accessed at state **Linked list**. If the output port is not found in any of these hash entries, the matching prefix does not exist in the current length and the search procedure goes back to state **No match** and continues to query the VBF with a shorter length. If a matching prefix is found from any of these entries, the search is completed for the current input and hence the search procedure goes to state **Input ready** for the next input.

In terms of time complexity, since the query for the vectored-Bloom filter performs the linear search on the number of bits in an IP address, the on-chip search performance is $O(W)$, where W is the length of the IP address. However, the off-chip hash table access which mainly determines the IP address lookup performance does not occur for a sufficiently large VBF.

4.2. Parallel Architecture

The IP address lookup performance using the VBF can be effectively improved by applying parallelism. Figure 4 describes the parallel architecture using two basic IP address lookup modules. Two new blocks are implemented for this parallel architecture: a *Distributer* and an *Output Queue*. The *Distributer* provides inputs to each module when the module is ready for a new input, while the *Output Queue* provides the output ports according to the order of arrived inputs.

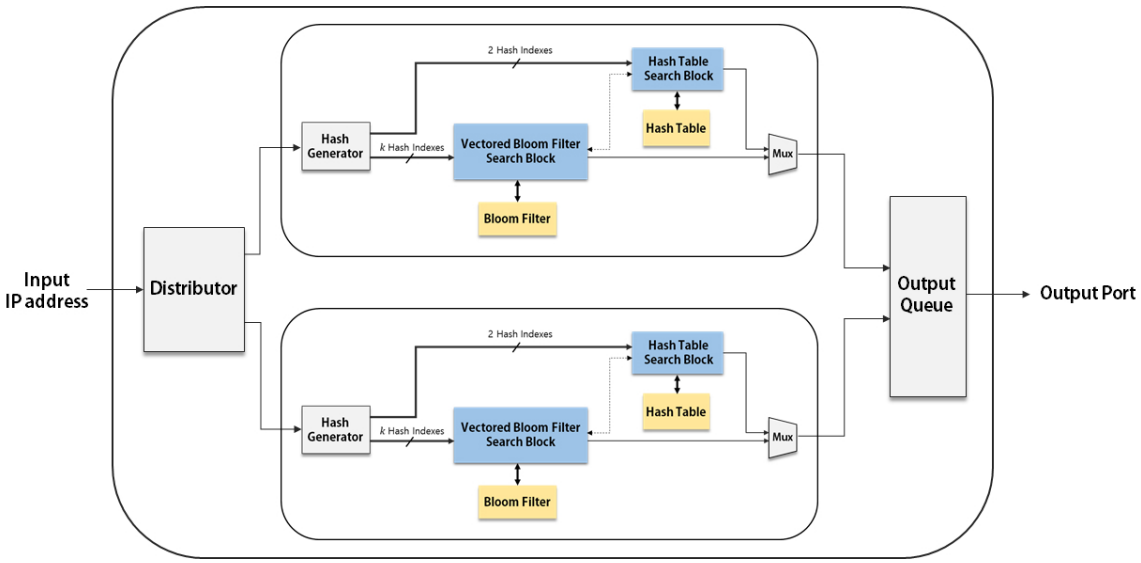

Figure 4. Parallel architecture consisting of two basic IP address lookup modules: the bold lines represent data paths and the dotted lines represent control paths.

4.3. Parallel Architecture with a Single Hash Table

Since the parallel architecture has multiple copies of the basic IP address lookup modules, the block-RAM (BRAM) utilization is rapidly increased when the number of modules is increased. The BRAM thus becomes the bottleneck in increasing the degree of parallelism. Since the hash table is infrequently accessed in our proposed IP address lookup architecture, multiple VBF blocks can share one hash table. In other words, we can separate the hash table from the basic IP address lookup module and make multiple VBF search blocks share a single hash table.

Figure 5 shows the parallel architecture with a single hash table, in which two VBF search blocks share a single hash table. The hash table search block is now composed of a *Hash Table Queue* and the hash table.

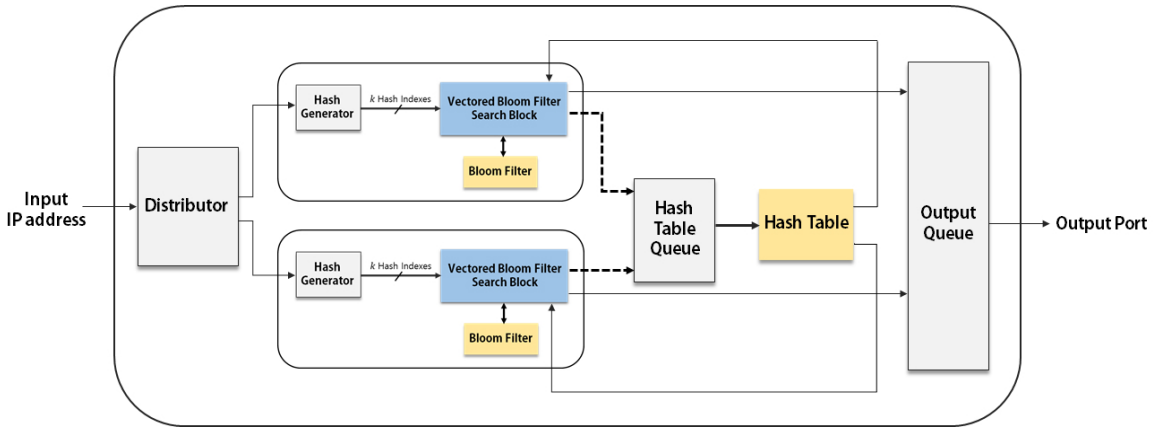

Figure 5. Parallel architecture employing a single hash table: the bold lines represent data paths and the dotted lines represent control paths.

The *Hash Table Queue* is a waiting place before accessing the hash table in preparing the case that two or more input addresses need to access the hash table at the same time. If the output port for an input is not determined by the VBF search block, the index of the input and two hash indexes for the hash table are stored in the *Hash Table Queue*. When the input arrives to the front of the queue, the stored indexes for each input are used to access the shared hash table. The input is removed from the queue after accessing the hash table.

5. Behavior Simulation

Performance evaluation was carried out using routing sets downloaded from backbone routers [33] at the behavior level with C language and at the hardware level with Verilog on an FPGA. We have created four routing sets and the number of prefixes (N) in each set is 1000, 5000, 14,553 and 30,000 (called 1 k, 5 k, 14 k and 30 k, respectively). Note that the number of prefixes included in actual backbone routers can be several hundred thousands. The simulation results of our proposed algorithm for large routing sets have been already shown in our previous paper [31]. This paper focuses on verifying the feasibility of the hardware implementation using an FPGA for our proposed architecture. Since the number of prefixes that can be handled by the FPGA is limited by the size of BRAM, we have used sets with a small number of prefixes in this paper.

The number of inputs to test is three times the number of prefixes in each routing set. Assuming that the number of output ports is eight, 8 bits are allocated for each vector of the vectored-Bloom filter.

5.1. Performance of the Proposed Structure

Table 1 shows the data structures of the VBF and a hash table. Let N be the number of prefixes in each set. For the *depth* of the VBF, $m = \alpha N'$, where $N' = 2^{\lceil log_2 N \rceil}$ and sizing factor α = 1, 2 and 4. The *width* of the VBF is the vector size (l), which is determined by the number of output ports.

The *depth* of the hash table, $B = 2N'$ and the *width* of the hash table is the size of a hash entry. A single hash entry has four fields. In storing a prefix, 32 bits are allocated assuming IPv4. The prefix length has five bits calculated by $log_2 32$. The output port uses three bits calculated by $log_2 l$ because the number of output ports l is assumed to be 8. Since the number of entries in the hash table is $2 \cdot 2^{\lceil log_2 N \rceil}$, the linked list has $\lceil log_2 N \rceil + 1$ bits.

Table 1. Data structures.

VBF				**HT**		
Depth	No. of vectors (m)	$\alpha N'$		No. of entries (B)	$2N'$	
Width (bits)	Vector size (l)	output port	8	Entry size (E)	prefix	32
					length	5
					ouput port	3
					linked list	$\lceil log_2 N \rceil + 1$

Since the VBF querying procedure starts from the longest length existing in the set and stops when a positive result is returned, the performance is affected by the distribution of the prefix lengths. Figure 6 shows the distribution of prefixes according to their lengths in each set.

Table 2 shows the on-chip memory requirement (M_b) for a VBF and the off-chip memory requirement (M_h) for a hash table. It is shown that the VBF requires less memory than a hash table even when the sizing factor α is 4, since each vector of the VBF only represents an output port while the hash entry stores a prefix, a length, an output port and a linked list. The optimal number of hash functions for the VBF (k_v) is shown as well.

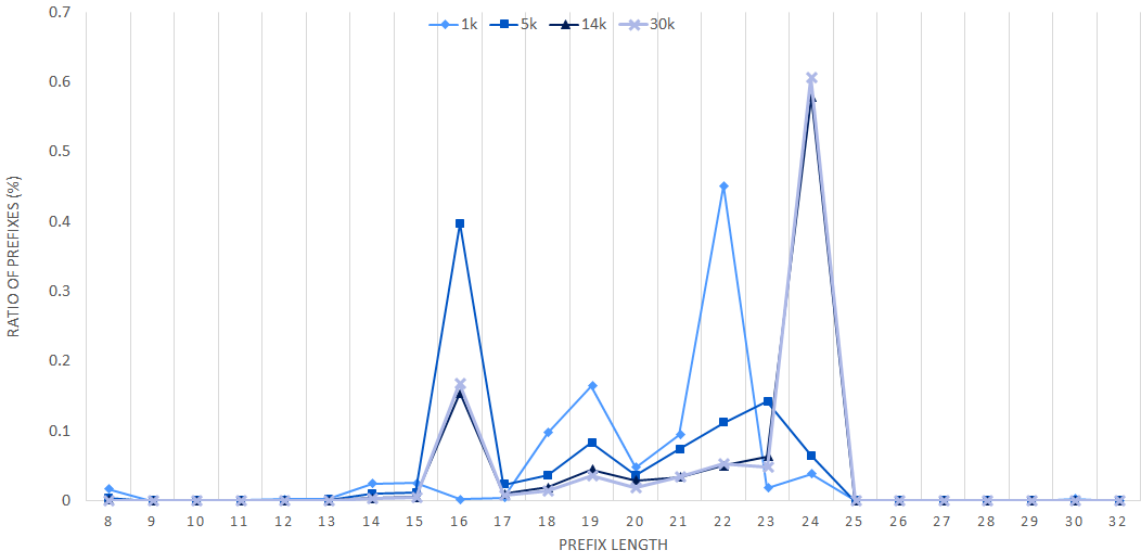

Figure 6. Distribution of the number of prefixes according to prefix lengths.

Table 3 shows the *indeterminable rate* and *false port return rate* according to the size of the VBF. The *indeterminable rate* (I) is defined as the number of inputs causing *indeterminable* over the number of inputs. Since the output ports for these inputs are not identified by the VBF because of multiple set bits in the resulting vector, the search procedure should perform an access to the hash table. The *false port return rate* (F) is defined as the number of inputs having false return over the number of inputs. False ports can be returned by the false positives of the VBF. Note that the *indeterminable rate* and the *false port return rate* are zero when the sizing factor of the VBF is larger than two as shown in Table 3.

Table 2. Memory requirement of proposed structure.

Routing Set (N)	α	VBF				HT		
		k_v	Depth	Width (bits)	M_b (KBytes)	Depth	Width (bits)	M_h (KBytes)
1000	1	6	1024	8	1	2048	51	12.75
	2	11	2048	8	2			
	4	22	4096	8	4			
5000	1	6	8192	8	8	16,384	54	108
	2	11	16,384	8	16			
	4	22	32,768	8	32			
14,553	1	6	16,384	8	16	32,768	55	220
	2	11	32,768	8	32			
	4	22	65,536	8	64			
30,000	1	6	32,768	8	32	65,536	56	448
	2	11	65,536	8	64			
	4	22	131,072	8	128			

Table 3. Indeterminable and False port return rate.

Routing Set (N)	α	I	F
1000	1	0.122	0.444
	2	0.003	0.019
	4	0.000	0.000
5000	1	0.017	0.135
	2	0.000	0.001
	4	0.000	0.000
14,553	1	0.067	0.424
	2	0.002	0.013
	4	0.000	0.000
30,000	1	0.072	0.451
	2	0.003	0.016
	4	0.000	0.000

5.2. Performance Comparison with Other Structures

Table 4 shows the comparison on Bloom filter characteristics of the proposed VBF structure with other structures such as BF-chaining with pre-computation [20] and PBF [6]. For fair comparison, the performances of each algorithm should be compared under the same amount of the on-chip memory required for constructing each Bloom filter. Based on the memory amount of the VBF shown in Table 2, in constructing Bloom filters for the BF-chaining and the PBF, sizing factor α and the number of hash indexes for each structure are calculated, when the same amount of the memory is used. As sizing factor α increases, the number of off-chip hash table accesses decreases because the number of the search failures decreases.Tables should be cited in sequential numerical order. Please confirm if it's invalid citation, if no, please revise the order. The VBF has the width of 8 but the number of elements for each output port is $\frac{N}{8}$ and hence the VBF is basically the same as a standard Bloom filter as in the BF-chaining and the PBF structures. Since all nodes of a binary trie are stored in the Bloom filter of the BF-chaining, the k of the BF-chaining is smaller than k_v. Since the PBF requires up to W independent Bloom filters [6], where W is the valid lengths of prefixes, the space efficiency is degraded.

Table 4. Comparison on Bloom filter characteristics.

Routing Set (N)	M_b (KB)	BF-Chaining		PBF		VBF	
		α	k	α	k	α	k_v
1000	1	1	1	6.30	4	1	6
	2	2	1	12.60	9	2	11
	4	4	3	25.20	17	4	22
5000	8	2	1	10.13	7	1	6
	16	4	3	20.24	14	2	11
	32	8	6	40.49	28	4	22
14,553	16	1	1	5.11	4	1	6
	32	2	1	10.22	7	2	11
	64	4	3	20.44	14	4	22
30,000	32	2	1	5.07	4	1	6
	64	4	3	10.14	7	2	11
	128	8	6	20.28	14	4	22

Table 5 shows the comparison of off-chip memory requirements. The hash table in the BF-chaining stores all of the nodes (T) in a binary trie, while the PBF and the VBF only store prefixes (N) in each routing set. Thus, the off-chip memory requirement of the PBF and the VBF is much smaller than that of the BF-chaining.

Table 5. Comparison of off-chip memory requirement.

Routing Set (N)	BF-Chaining		PBF	VBF
	T	M_h (KB)	M_h (KB)	M_h (KB)
1000	7678	108	13	13
5000	29,534	448	108	108
14,553	76,708	1856	220	220
30,000	127,576	1856	448	448

Table 6 shows the comparison of the on-chip search performance in terms of the average number and the worst-case number of Bloom filter accesses, represented by A_b and W_b, respectively. Since the Bloom filter querying is not performed for lengths not including any prefix, W_b is the number of valid prefix lengths for each set. Note that the Bloom filter querying of the BF-chaining proceeds from the shortest length, while the querying of the PBF and the VBF proceeds from the longest length. Thus, as the size of each Bloom filter (M_b) increases, the average number of Bloom filter accesses (A_b) of the BF-chaining decreases but that of the VBF increases, because the number of *negatives* increases. The larger number of *negatives* results in the smaller number of hash table accesses in both structures.

In case of the PBF, for the fair comparison with other structures, we assume that all of the BFs in the PBF structure are sequentially queried until the longest prefix is matched as in the VBF. Hence, the A_b is constant regardless of the size of the Bloom filter because Bloom filter querying always stops at the length of the matching longest prefix. In other words, starting from the longest length, the Bloom filter querying always continues until a true positive occurs and hence the average number of Bloom filter querying is not related to the size of the Bloom filter. The A_b of the VBF in 5 k is greater than that in other sets, because the 5 k set has many short length prefixes, such as length 16, as shown in Figure 6. Since the search procedure proceeds from the longest length, if the prefix matching occurs in a short length, the A_b becomes large.

Table 6. Comparison on the number of on-chip Bloom filter querying.

Routing Set (N)	M_b (KB)	BF-Chaining		PBF		VBF	
		A_b	W_b	A_b	W_b	A_b	W_b
1000	1	14.6	18	6.7	18	4.4	18
	2	13.9				6.3	
	4	13.5				6.7	
5000	8	13.5	21	10.0	21	8.9	21
	16	13.1				9.5	
	32	13.0				10.0	
14,553	16	16.6	22	8.2	22	5.5	22
	32	16.2				7.5	
	64	15.9				8.2	
30,000	32	16.5	22	8.1	22	5.2	22
	64	16.1				7.2	
	128	15.9				8.1	

Table 7 shows the comparison on off-chip search performance in terms of the average number and the worst-case number of hash table accesses, represented by A_h and W_h, respectively. The number of off-chip memory accesses is the most important performance criterion in the IP address lookup problem. To improve the IP address lookup performance, the number of off-chip accesses should be minimized. In the BF-chaining and the PBF, even if the size of the Bloom filter (M_b) increases, the off-chip hash table is accessed at least once in order to obtain the matching output port. However, hash table accesses occur infrequently in our proposed VBF architecture, since the Bloom filter stored in an on-chip memory returns the matching output port in every case except the *indeterminable* cases. Only when the VBF produces an *indeterminable* result, the hash table is accessed. It is shown that the average number of hash table accesses becomes zero as the size of the Bloom filter (M_b) increases.

Table 7. Comparison on the number of off-chip hash table accesses.

Routing Set (N)	M_b (KB)	BF-Chaining		PBF		VBF	
		A_h	W_h	A_h	W_h	A_h	W_h
1000	1	2.385	9	1.170	5	0.203	5
	2	1.633	9	1.004	2	0.004	3
	4	1.153	4	1.000	1	0.000	0
5000	8	1.565	10	1.052	3	0.024	4
	16	1.140	4	1.001	2	0.000	0
	32	1.015	3	1.000	1	0.000	0
14,553	16	1.784	12	1.491	6	0.110	5
	32	1.335	6	1.054	4	0.003	5
	64	1.046	4	1.001	2	0.000	0
30,000	32	1.619	13	1.322	6	0.124	6
	64	1.160	6	1.037	3	0.004	6
	128	1.019	3	1.001	2	0.000	0

6. Hardware Implementation

This section shows the hardware implementation details of our proposed structure. Especially, the *parallel architecture with a single hash table* shown in Figure 5 has been implemented. The hardware implementation has been carried out with Verilog language using Vivado 2017.4 development tool. Our target device is NetFPGA CML operating at 100 MHz. The size of the VBF is $m = 2N'$. The VBF and the hash table implemented with BRAMs are loaded with values obtained from the simulation at the behavior level.

Figure 7 shows the hardware test flow. Generally, the input IP addresses should be provided by an external input generater, as shown in Figure 7a. However, in our experiment, the input IP addresses are stored in a BRAM, as shown in Figure 7b, in order to provide the input at the operation rate of the hardware implemented on the FPGA. The input IP addresses are stored in dual-port RAMs and hence the parallel architecture can process two IP addresses in parallel.

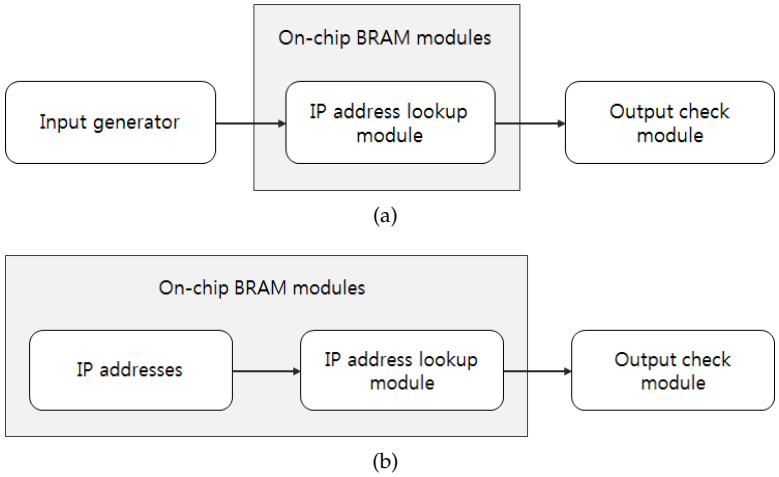

Figure 7. Test flow of field programmable gate array (FPGA)-based IP address lookup: (**a**) General case. (**b**) Our case.

Since the VBF search block in the basic IP address lookup module has 11 hash indexes obtained from Label (3) and a VBF is implemented with a dual-port RAM, six duplicates of the VBF are required. Since the parallel architecture has two basic IP address lookup modules, the VBF duplication should be doubled again and hence the 12 copies of the VBFs are implemented in total.

Table 8 shows the memory requirement in the hardware implementation. The BRAM has 18-Kbit and 36-Kbit blocks and the blocks are automatically allocated for each component. It is shown that a VBF is implemented with a single 18-Kbit block for the 1 k set, with four 36-Kbit blocks for the 5 k set and so on. Since each parallel architecture requires 12 copies of VBFs, the total number of VBFs is multiplied by 12. Similarly, it is shown the number of blocks used to store a hash table and input IP addresses for each routing set.

The values in total BRAMs represent the summation of the BRAM in KBytes. The utilization rate of the BRAM is based on the BRAM capacity of 16,020 Kbits. Since each component is allocated in blocks, the memory requirement of a VBF or a hash table in hardware implementation is greater than that in behavior simulation.

Table 9 shows resource utilization. *Capacity* means the amount of available resources and *Used* means the amount of resources used in our implementation. *Utilization* represents the ratio between *Used* and *Capacity*. It is shown that the utilization of the BRAM reaches up to 84.0% for the 30 k set.

Table 8. Memory requirement in hardware.

Routing	Input Set (#. of blks)		VBF (#. of blks)		Total VBFs (#. of blks)		HT (#. of blks)		Total BRAMs	
Set	18-Kbits	36-Kbits	18-Kbits	36-Kbits	18-Kbits	36-Kbits	18-Kbits	36-Kbits	(KBytes)	(%)
1000	0	3	1	0	12	0	0	3	54	2.7
5000	4	12	0	4	0	48	0	24	387	19.3
14,553	7	36	0	8	0	96	0	49	830	41.5
30,000	0	80	0	16	0	192	0	102	1683	84.0

Table 9. Resource utilization.

Resource	Capacity	1 k		5 k		14 k		30 k	
		Used	Utilization (%)	Used	Utilization (%)	Used	Utilization (%)	Used	Utilization (%)
LUT	203,800	5560	2.7	6440	3.2	6995	3.4	7419	3.6
FF	407,600	2824	0.7	3029	0.7	3084	0.8	3087	0.8
BRAM	445	12	2.7	86	19.3	184.5	41.5	374	84.0
IO	400	6	1.5	6	1.5	6	1.5	6	1.5
BUFG	32	3	9.4	3	9.4	3	9.4	3	9.4

Table 10 shows the total on-chip power and the worst negative slack reported in Vivado 2017.4 development tool.

Table 10. Power consumption and the worst negative slack.

	1 k	5 k	14 k	30 k
Total on chip power (W)	0.172	0.181	0.188	0.262
Worst negative slack (ns)	88.51	82.35	84.30	79.56

Figure 8 graphically shows the resource utilization. Since the utilizations of IO and BUFG do not depend on the sizes of the routing sets, their utilizations are not shown. The utilizations of LUT and FF are almost constant, even when the size of the routing set increases. *BRAM w/o IP* refers to the BRAM utilization when input IP addresses are not stored in BRAMs, as shown in Figure 7a. In this case, the BRAM utilization reaches up to 66.1%, which is substantially lower than 84.0%.

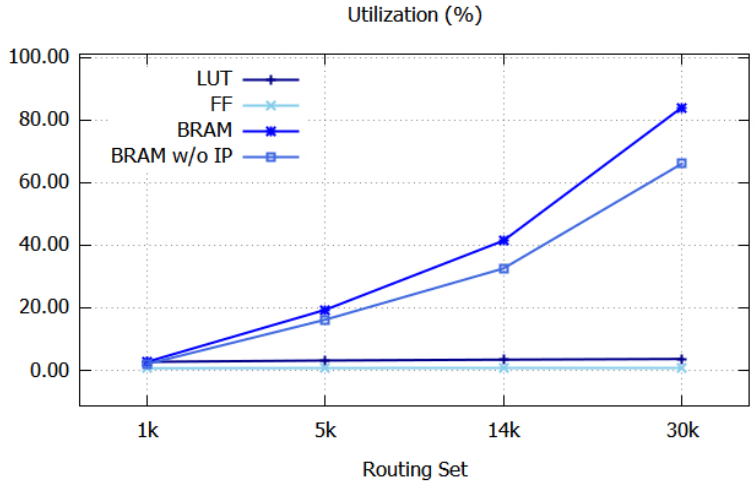

Figure 8. Resource utilization.

Figure 9 shows the average number of cycles needed to perform an IP address lookup and the throughput of our implementation. The average number of cycles is related to the search performance and the search performance is affected by the prefix length distribution shown in Figure 6. The throughput is the maximum number of processed packets per second (p/s or pps) and it is inversely related to the average number of cycles. The best-case of the throughput is 4.92 million packets/sec for the 1 k set, while the worst-case of the throughput is 3.32 million packets/sec for the 5 k set, because the 5 k set has more short-length prefixes than other sets as shown in Figure 6. Note that the throughput performance depends more on the prefix length distribution rather than the number of prefixes. The prefix length distribution is related to the role of routers, whether the router is a backbone router or an edge router. Backbone routers have the more number of short-length prefixes than edge routers, since edge routers connect mostly access networks, which has a more specific (longer-length) prefix.

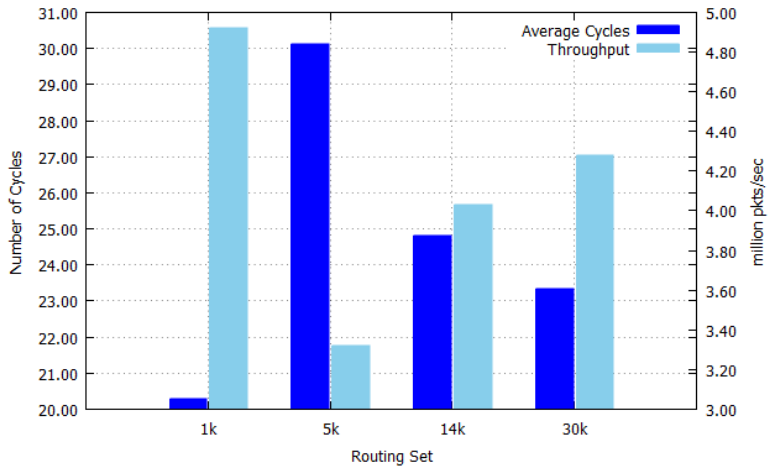

Figure 9. Relationship between average number of cycles and throughput.

Considering that the number of prefixes is scaled to several hundred thousands reflecting real backbone routing tables, if the proposed architecture is implemented with an ASIC, which has at least 4 Mbytes on-chip memory operating at 500 MHz [6], the expected throughput can be improved by 5 times. The proposed architecture can provide the wire-speed IP address lookup at the rate of about 15 to 25 million packets/sec, since the routing sets used in our simulation has the similar characteristics as actual routing sets by being downloaded (and randomly selected) from real backbone routers.

7. Conclusions

In this paper, we have proposed a new efficient IP address lookup architecture using a Bloom filter. The proposed architecture is based on a vectored-Bloom filter, which stores an output port in each vector. We have also proposed the use of parallel architectures to improve the search performance. In the proposed parallel architectures, multiple VBF search blocks are implemented with sharing a single hash table, since the hash table is infrequently accessed. Hence the memory efficiency is increased. The performance of the proposed architecture has been evaluated at the behavior level using C and at the hardware level using Verilog. The behavior evaluation shows that the proposed architecture can perform the IP address lookup without accessing the off-chip hash table for a reasonably sized Bloom filter. The hardware evaluation shows that the proposed architecture provides the throughput of 4.92 million packets/sec on an FPGA operating at 100MHz. The proposed hardware architecture can be a promising candidate to provide the wire-speed packet forwarding since a more degree of parallelism can be pursued when it is implemented with an ASIC operating at a higher frequency.

Author Contributions: Conceptualization, H.B. and H.L.; methodology, H.L.; software, H.B.; hardware, Q.L.; validation, H.B. and Q.L.; investigation, H.L.; resources, H.B. and Q.L.; data curation, H.B.; writing—original draft preparation, H.B.; writing—review and editing, H.L.; visualization, H.B.; supervision, H.L.; project administration, H.L.; funding acquisition, H.L.

Funding: This research was funded by the National Research Foundation of Korea (NRF), NRF-2017R1A2B4011254.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

Abbreviations

IP	Internet Protocol
CIDR	Classless Inter-Domain Routing
BMP	Best Matching Prefix
ASIC	Application-Specific Integrated Circuits
TCAM	Ternary Content Addressable Memory
FPGA	Field-Programmable Gate Arrays
HT	Hash Table
BF	Bloom Filter
VBF	Vectored-Bloom Filter
PBF	Parallel Bloom Filter
SAIL	Splitting Approach for IP address Lookup
CRC	Cyclic Redundancy Check

References

1. Chao, H. Next Generation Routers. *Proc. IEEE* **2002**, *90*, 1518–1588. [CrossRef]
2. Lim, H.; Lee, N. Survey and Proposal on Binary Search Algorithms for Longest Prefix Match. *IEEE Commun. Surv. Tutor.* **2012**, *14*, 681–697. [CrossRef]
3. Yang, T.; Xie, G.; Li, Y.; Fu, Q.; Liu, A.; Li, Q.; Mathy, L. Guarantee IP Lookup Performance with FIB Explosion. In Proceedings of the ACM SIGCOMM, Chicago, IL, USA, 17–22 August 2014; pp. 39–50. [CrossRef]
4. Vagionas, C.; Maniotis, P.; Pitris, S.; Miliou, A.; Pleros, N. Integrated Optical Content Addressable Memories (CAM) and Optical Random Access Memories (RAM) for Ultra-Fast Address Look-Up Operations. *Appl. Sci.* **2017**, *7*, 700. [CrossRef]
5. Gupta, P.; Lin, S.; Mckeown, N. Routing Lookups in Hardware at Memory Access Speed. In Proceedings of the IEEE INFOCOM '98, the Conference on Computer Communications. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Gateway to the 21st Century, San Francisco, CA, USA, 29 March–2 April 1998; pp. 1240–1247. [CrossRef]
6. Dharmapurikar, S.; Krishnamurthy, P.; Taylor, D. Longest Prefix Matching Using Bloom Filters. *IEEE/ACM Trans. Netw.* **2006**, *14*, 397–409. [CrossRef]
7. Lim, H.; Lim, K.; Lee, N.; Park, K. On Adding Bloom Filters to Longest Prefix Matching Algorithms. *IEEE Trans. Comput.* **2014**, *63*, 411–423. [CrossRef]
8. Panda, P.; Dutt, N.; Nicolau, A. On-Chip vs. Off-Chip Memory: The Data Partitioning Problem in Embedded Processor-Based Systems. *ACM Trans. Des. Autom. Electron. Syst.* **2000**, *5*, 682–704. [CrossRef]
9. Jiang, W.; Prasanna, V. Sequence-preserving parallel IP lookup using multiple SRAM-based pipelines. *J. Parallel Distrib. Comput.* **2009**, *69*, 778–789. [CrossRef]
10. Erdem, O.; Bazlamacci, C. Array Design for Trie-based IP Lookup. *IEEE Commun. Lett.* **2010**, *14*, 773–775. [CrossRef]
11. Pérez, K.; Yang, X.; Scott-Hayward, S.; Sezer, S. Optimized packet classification for Software-Defined Networking. In Proceedings of the 2014 IEEE International Conference on Communications (ICC), Sydney, Australia, 10–14 June 2014; pp. 859–864. [CrossRef]
12. Song H.; Lockwood, J. Efficient packet classification for network intrusion detection using FPGA. In Proceedings of the 2005 ACM/SIGDA 13th International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 20–22 February 2005; pp. 238–245. [CrossRef]

13. Sanny, A.; Ganegedara, T.; Prasanna, V.K. A Comparison of Ruleset Feature Independent Packet Classification Engines on FPGA. In Proceedings of the IEEE International Symposium on Parallel and Distributed Processing, Workshops and PhD Forum (IPDPSW), Cambridge, MA, USA, 20–24 May 2013; pp. 124–133. [CrossRef]
14. Viegas, E.; Santin, A.; França, A.; Jasinski, R.; Pedroni, V.; Oliveira, L. Towards an Energy-Efficient Anomaly-Based Intrusion Detection Engine for Embedded Systems. *IEEE Trans. Comput.* **2017**, *66*, 163–177. [CrossRef]
15. Gidansky, J.; Stefan, D.; Dalal, I. FPGA-based SoC for real-time network intrusion detection using counting Bloom filters. In Proceedings of the IEEE Southeastcon, Atlanta, GA, USA, 5–8 March 2009; pp. 452–458. [CrossRef]
16. Hieu, T.; Thinh, T.; Vu T.; Tomiyama, S. Optimization of Regular Expression Processing Circuits for NIDS on FPGA. In Proceedings of the Second International Conference on Networking and Computing, Osaka, Japan, 30 November–2 December 2011; pp. 105–112. [CrossRef]
17. Bloom, B. Space/time tradeoffs in in hash coding with allowable errors. *Commun. ACM* **1970**, *13*, 422–426. [CrossRef]
18. Waldvogel, M.; Varghese, G.; Turner, J.; Plattner, B. Scalable High Speed IP Routing Lookups. In Proceedings of the ACM SIGCOMM '97 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, Cannes, France, 14–18 September 1997; pp. 25–35. [CrossRef]
19. Mun, J.; Lim, H.; Yim, C. Binary Search on Prefix Lengths for IP Address Lookup. *IEEE Commun. Lett.* **2006**, *10*, 492–494. [CrossRef]
20. Mun, J.; Lim, H. New Approach for Efficient IP Address Lookup Using a Bloom Filter in Trie-Based Algorithms. *IEEE Trans. Comput.* **2016**, *65*, 1558–1565. [CrossRef]
21. Byun, H.; Lim, H. A New Bloom Filter Architecture for FIB Lookup in Named Data Networking. *Appl. Sci.* **2019**, *9*, 329. [CrossRef]
22. Moralis-Pegios, M.; Terzenidis, N.; Mourgias-Alexandris, G.; Vyrsokinos, K. Silicon Photonics towards Disaggregation of Resources in Data Centers. *Appl. Sci.* **2018**, *8*, 83. [CrossRef]
23. Lin, P.; Lin, Y.; Lai, Y.; Zheng, Y.; Lee, T. Realizing a Sub-Linear Time String-Matching Algorithm With a Hardware Accelerator Using Bloom Filters. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2009**, *17*, 1008–1020. [CrossRef]
24. Lai, B.; Chen, K.; Wu, P. A High-Performance Double-Layer Counting Bloom Filter for Multicore Systems. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2015**, *23*, 2473–2486. [CrossRef]
25. Reviriego, P.; Pontarelli, S.; Maestro, J.; Ottavi, M. A Synergetic Use of Bloom Filters for Error Detection and Correction. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2015**, *23*, 584–587. [CrossRef]
26. Chen, Y.; Schmidt, B.; Maskell, D. Reconfigurable Accelerator for the Word-Matching Stage of BLASTN. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2013**, *21*, 659–669. [CrossRef]
27. Tarkoma, S.; Rothenberg, C. E.; Lagerspetz, E. Theory and practice of Bloom filters for distributed systems. *IEEE Commun. Surv. Tutor.* **2012**, *14*, 131–155. [CrossRef]
28. Lu, J.; Yang, T.; Wang, Y.; Dai, H.; Jin, L.; Song, H.; Liu, B. One-Hashing Bloom Filter. In Proceedings of the IEEE 23rd International Symposium on Quality of Service (IWQoS), Portland, OR, USA, 15–16 June 2015; pp. 289–298. [CrossRef]
29. Qiao, Y.; Li, T.; Chen, S. One Memory Access Bloom Filters and Their Generalization. In Proceedings of the IEEE INFOCOM Shanghai, China, 10–15 April 2011; pp. 1745–1753. [CrossRef]
30. Yang, T.; Xie, G.; Liu, A.; Fu, Q.; Li, Y.; Li, X.; Mathy, L. Constant IP Lookup With FIB Explosion. *IEEE/ACM Trans. Netw.* **2018**, *26*, 1821–1836. [CrossRef]
31. Byun, H.; Lim, H. IP Address Lookup Algorithm Using a Vectored Bloom Filter. *Trans. Korean Inst. Electr. Eng.* **2016**, *65*, 2061–2068. [CrossRef]
32. Byun, H.; Li, Q.; Lim, H. Vectored-Bloom Filter Implemented on FPGA for IP Address Lookup. In Proceedings of the ICEIC 2019, Auckland, New Zealand, 22–25 January 2019; pp. 967–970. [CrossRef]
33. Prefix Names Available online: http://www.potaroo.net (accessed on 29 October 2019).