A Unicast Packet Forwarding Accelerator Design Based on the RNS Algorithm

Lidan Zhu and Wen-Kang Jia(✉)

College of Photonic and Electronic Engineering, Fujian Normal University, Fuzhou 350000, China
qsx20210842@student.fjnu.edu.cn, wkjia@fjnu.edu.cn

Abstract. The challenge of achieving the Longest Prefix Match (LPM) in IP address lookup remains a bottleneck in high-speed routers. Classless Inter-Domain Routing (CIDR) utilizes member query algorithms within Packet Forwarding Engines (PFE) to determine whether a packet's next hop should be forwarded through a specific output port. Among these, Bloom Filters (BF), Scalar Vector Routing and Forwarding (SVRF), and their improved variants have all proven efficient for rapid packet forwarding. In this paper, we propose a Unicast Packet Forwarding Accelerator design that utilizes the Chinese Remainder Theorem (CRT), Residue Number System (RNS), and SVRF. This framework associates prefix lengths with processing blocks by partitioning n-elements into 22 columns (sub-blocks) based on prefix lengths. Within each sub-block, IP addresses are parallelly computed for output port indexing, optimizing the partitioning of scalar vectors to facilitate the determination of address membership in a prefix set sorted by prefix length. This scheme optimization simultaneously addresses the Longest Prefix Match problem while reducing memory space usage and forwarding latency. Furthermore, we perform an analysis of snapshots from real-world IPV4 BGP tables to instantiate and simulate the performance of our algorithm. Simulation evaluations demonstrate that our proposed algorithm offers significant advantages in terms of both time and space efficiency under optimized partitioning.

Keywords: Longest Prefix Matching (LPM) · Packet Forwarding Engine (PFE) · Scalar-pair Vectors Routing and Forwarding (SVRF)

1 Introduction

The explosive growth of the Internet has led to an exponential increase in forwarding table entries within routers [1, 2]. In this context, IP address lookup has become a crucial technology in router design to support high-speed communication over the Internet. Classless Inter-Domain Routing (CIDR) [3, 4] has been widely adopted as a means to extend the sustainable use of Internet Protocol version 4 (IPv4) by effectively utilizing the IP address space. Unlike traditional fixed-length subnetting methods, CIDR requires routers to perform variable-length address prefix searches. By flexibly utilizing variable prefix lengths, CIDR enables more versatile address allocation and route aggregation,

F. Gao et al. (Eds.): Chinacom 2023, LNICST 590, pp. 395–411, 2024.
https://doi.org/10.1007/978-3-031-67162-3_25

allowing routers to manage and forward packets more efficiently. This, in turn, reduces the size of routing tables and enhances routing performance, albeit at the cost of increased complexity in routing lookup operations [5].

The Hardware-assisted Packet Forwarding Engine (PFE) constitutes one of the most intricate components within large packet-switching devices, including high-end Ethernet switches and core IP routers[6]. The main function of the PFE is to swiftly transfer packets from the input queue to the output queue via the data plane's backplane, which functions as the switching fabric. When the PFE cannot locate a matching outgoing port for a packet in the forwarding table, it discards the packet with an unknown destination and notifies the routing engine to initiate the necessary processing[7, 8]. The control plane of the PFE relies on the forwarding table constructed by the upper-layer Routing Engine (RE)[9]. This forwarding table, also known as the Forwarding Information Base (FIB), aids in determining and specifying the egress protocols and routing tables for the data plane, referred to as the Routing Information Base (RIB). Due to the substantial system resources needed, PFE functionality is typically only achievable through software implementation.

In unicast route forwarding, router routing tables are required to store variable-length IP address prefixes along with corresponding output port information[10, 11]. Typically, forwarding tables are sorted based on prefix length and commonly formatted as $<$ destination prefix, forwarding information $>$, where the 'destination prefix' represents an IP prefix, and 'forwarding information' usually consists of the next-hop address or output port number. When an incoming packet's IP address matches multiple entries in the forwarding table, the router must determine the most precise match (i.e., the longest prefix length). Upon finding a match, the next-hop information linked to the matching prefix is retrieved. The objective of the Longest Prefix Match (LPM) algorithm is to identify the most specific entry from among all matching entries in the table. Since the target IP address may have multiple matching entries due to variable prefix lengths, all entries in the routing table must be checked to ensure that the packet is forwarded to the output port associated with the longest prefix match [12]. To address this challenge, we propose a Unicast Packet Forwarding Accelerator design based on the Residue Number System (RNS) algorithm [13]. By taking into account the differences in length distribution between actual forwarding entries and prefix sets, our algorithm, compared to other existing methods, achieves efficient packet forwarding with fewer resources. This solution meets the increasing demands for router performance in the context of the rapidly growing Internet.

Our paper is structured as follows. In Sect. 2, we discuss existing solutions to the LPM algorithm, providing an overview of relevant technical approaches. Section 3 introduces our proposed solution, encompassing the framework design and query forwarding process. In Sect. 4, we begin by analyzing the real-world routing tables we obtained, which includes an analysis of statistical results derived from processing the routing table dataset. We instantiate our designed solution and conduct a comparative study with prior work. Finally, in Sect. 5, we conclude our work with a summary of this paper.

2 Related Work

The IP lookup problem has been a central one in networking for a very long time. Maintaining a forwarding table based on IP destination address prefixes is the typical method used by IP routers to forward packets. Each prefix corresponds to the following hop in the direction of the destination. IP routers look up a packet's destination address in their databases and forward it depending on the longest prefix that matches. A large number of algorithms and techniques have been put forth in numerous works to overcome the difficulties with the longest prefix match (LPM) [7, 14, 15].

Hashing transforms each key into a smaller integer, known as a hash index, which serves as a pointer to an array [16, 17]. For IP address lookup, hash-based Longest Prefix Match algorithms [12, 18, 19] divide prefixes into different sets based on their prefix lengths and apply hashing to each prefix set. This allows for parallel execution of all hash operations and memory accesses to enhance performance, ultimately selecting the longest prefix match as the best match.

Most hash-based and hybrid Bloom Filter (BF) Longest Prefix Match solutions convert the LPM problem into an exact match problem for multiple subsets, incurring high costs for these hash tables [18, 20]. BF are probabilistic data structures used for membership queries. In the context of IP lookup, BF is first associated with prefix lengths and the forwarding table is sorted by prefix length. The essence of a BF is a vector of length m. It efficiently represents a set of messages. Suppose there are n members in a group whose messages are represented by X. The BF computes k hash functions on each message x_i in X, generating k values, each ranging from 1 to m. By examining the bits at positions corresponding to these k hash values in the m-bit length vector, if there is at least one 0 among these k bits, the message x_i is not a member of the set. However, if all the bits are set to 1, it means that the message x_i might belong to the set, but there is still a probability of false positives in the BF. In other words, there may be cases where all k bits are 1, but x_i is not a member of X. This probability of false positives, often referred to as the false positive probability or false alarm rate (denoted as f), is independent of the specific input message. The false positive probability can be calculated using an equation:

$$f = \left(1 - \left(1 - \frac{1}{m}\right)^{nk}\right)^{k} \tag{1}$$

For larger values of m, the above equation can be simplified as:

$$f \approx \left(1 - e^{\frac{-nk}{m}}\right)^{k} \tag{2}$$

BF as applied to IP lookups [21, 22] commences by sorting the forwarding table based on prefix length and linking a Bloom filter with each unique prefix length. Assuming that the system can support N target prefixes, has a total of M available slots for BFs, supports W different prefix lengths, assigns a size of m_i to each BF, uses k_i hash functions for each filter, and stores ni prefixes in each filter. If we denote f_i as the probability of misclassification for the $i-$ th BF:

$$f_i = f = \left(\frac{1}{2}\right)^{\left(\frac{m_i}{n_i}\right)\ln 2} = \left(\frac{1}{2}\right)^{\left(\frac{M}{N}\right)\ln 2}, \forall i \in [1 \ldots 31] \tag{3}$$

During the IP lookup process, parallel membership queries are performed on W- BF sorted by prefix length, allowing subsets of BF to report matches to determine possible prefix matches. One advantage of BF is their ability to compactly represent a set of items, making them suitable for on-chip storage [23].In the case of IPV4, W is 32, and for a given IP address, the prefix length is probed in an off-chip hash table. Due to the false positive probability of BF, false matches may exist, and when multiple BF indicate false positives, off-chip prefix table searches must be performed. The implementation of such architectures is highly complex due to the presence of BF and hash tables for each prefix length.

To overcome the limitations of BF, Jia and Jin introduced an error-free member query algorithm utilizing the Chinese Remainder Theorem [24] and RNS properties [13]. This algorithm is known as Scalar Vector Routing and Forwarding (SVRF) [25] and its variant Frac-*N* SVRF [26]. In SVRF, the process begins with the traversal of the Packet Forwarding Engine (PFE). It encodes the complete forwarding table into "scalar pairs," representing each forwarding entry as a "vector." Forwarding decisions are made by performing modulo operations using a chosen set of prime numbers known as "keys." These modulo operations are based on the Residue Number System (RNS) attributes within the PFE. Notably, SVRF handles both multicast and unicast packets using the same RNS attributes, allowing it to process them efficiently. SVRF achieves forwarding decisions by performing modular operations on common scalar pairs and prime keys specific to forwarding entries within pseudo-polynomial time. The SVRF algorithm comprises three key components: 1) Node-specific scalar pairs (M_{cp}, M_{crt}), , 2) Keys specific to flows/groups, and 3) Vectors specific to flows/groups. The assumption is made that all keys, denoted as $K = \{P(a_i) \mid \forall i \leq n, a_i \in A\}$, ,are stored within the set K. K represents the set containing keys for all unicast and multicast packet (flow) identifiers. These keys are chosen to be pairwise mutually prime and are generated using the perfect function $P(a_i)$, which generates prime numbers within a specified range for each route identifier a_i. In the context of the entire forwarding table vector, the set $B = \{b_1, b_2, ..., b_n\}$ is employed to store the output port bitmap, where b_x represents an integer value. Additionally, for the scalar pair (M_{cp}, M_{crt}), the first scalar M_{cp} is calculated as the continuous product of all keys:

$$M_{cp} = \prod_{i=1}^{n} k_i \tag{4}$$

The second scalar M_{crt}, , can be constructed using the Chinese Remainder Theorem (CRT) function:

$$M_{crt} = CRT\begin{pmatrix} \{k_1, b_1\}, \\ \{k_2, b_2\}, \\ ..., \\ \{k_n, b_n\} \end{pmatrix} \tag{5}$$

In the membership query process, the first step involves extracting the node-specific key k_x from the incoming packet identifier x. This key serves as the divisor for subsequent operations. Additionally, the scalar pair (M_{cp}, M_{crt}) is retrieved from the memory cell, and this pair is also used as a divisor. The final output, which represents the output port bitmap for the element x, is obtained through two long integer division operations, utilizing these divisors:

$$b_x = \begin{cases} (M_{crt} \bmod k_x), & \text{if}\left(M_{cp} \bmod k_x\right) = 0 \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

In this model, when sets K and M_{crt} are available, every element in set B can be recovered by linear congruence. For M_{crt} in the model, there is the following constant equation:

$$M_{crt} \equiv \begin{cases} b_1 (\bmod\, k_1) \\ b_2 (\bmod k_2) \\ \cdots \\ b_n (\bmod k_n) \end{cases} \tag{7}$$

The main idea underlying the ensuing improvement technique, Frac-*N* SVRF, is to divide the classic SVRF's processing units into several smaller sub-blocks in the same way that the large-scale, extremely complex scalar pairs (M_{cp}, M_{crt}) are divided into many smaller sub-scalar pairs. These *N*-SVRF sub-blocks all have a consistent structure and are not dependent on one another. When dealing with n forwarding entries, standard SVRF only uses one sub-block, a single pair of scalar pairs, to do membership queries; however, with Frac-*N* SVRF, each sub-block is in charge of processing roughly n/N members. This method effectively re-uses prime keys created for each sub-block and allows for the concurrent computation of output ports across N sub-blocks, resulting in a large reduction in computational complexity and a notable improvement in time efficiency. However, it is worth noting that Frac-*N* SVRF's design primarily focuses on forwarding multicast entries and assumes a uniform distribution of forwarding entries within each SVRF sub-block. This assumption may not align with the actual routing table entry distribution in real-world networks, limiting its practical applicability. To enhance the adaptability of packet forwarding in real router forwarding engines, we introduce our new scheme, Frac-22 SVRF, building upon the original SVRF.

3 Fractional-22 SVRF

In this section, we present a novel approach distinct from the original Frac-*N* SVRF. This new scheme is tailored for unicast forwarding in CIDR and aims to reduce memory consumption (space efficiency) and lookup latency (time efficiency) during packet forwarding while achieving the longest prefix match for IP lookup.

3.1 Construction of the Hardware Architecture Diagram for Frac-22 SVRF

The hardware design of the Frac-22 SVRF scheme is depicted in Fig. 1. The key components of Frac-22 SVRF include 1) input unit, 2) processing unit, 3) memory unit, and 4) output unit. In terms of the configuration function of the Routing Engine, we begin by sorting prefixes based on their prefix lengths and storing them in a prefix table. Here, the shortest prefix length within the prefix set is denoted as s and the longest prefix length is $l + s$, thus forming the prefix set as $S = \{s, s + 1, s + 2, ...s + l\}$. The distinctive characteristics of each unit in Frac-22 SVRF are described as follows:

The Input Unit consists of an extractor and a queue sequence with a capacity of W. When a flow arrives at the Input Unit, the extractor retrieves the corresponding flow identifier. The system employs W-SVRF sub-blocks, where W represents the span of prefix lengths and is determined as $W = l - s + 1$. Each SVRF sub-block is uniquely associated with a specific prefix length. Prefix length determines how the input queue is arranged, allocating flow identifiers with various prefix lengths to various sub-blocks inside the Processing Unit for forwarding operations.

The Processing Unit comprises W-SVRF sub-blocks and $(l - s)$ comparers, with each sub-block being equivalent to a conventional SVRF processing unit. Each sub-block consists of a prime number generator, two dividers, and a zero detector. Firstly, the prime number generator is used to produce random, non-repeating prime numbers within a specified range. Its input is the flow identifier, and its output is a random prime number. One of the divisors is used to compute the modulus of the scalar $M_{cp(i)}$ with the prime key $k_{x(i,x)}$. This calculation is used for two purposes: 1) to determine whether the data stream is in the forwarding table, i.e., to determine whether the entry being processed is in the list of forwarding table members; and 2) to allow the Multiplex Data Selector W-MUX to enable the Output Port Index (OPI) of the sub-block with the largest serial number (associated with the longest prefix), based on hits in each sub-block. The remaining divider is used to compute the modulus of M_{crt} with the prime key k_x, and the result of this divider is the unicast output port index OPI_i obtained by this IP address lookup computed in this sub-block.

The memory unit uses a $W \times 2$ storage scalar matrix. This matrix stores all pairs of scalars: $\{(M_{cp(1)}, M_{crt(1)}), (M_{cp(2)}, M_{crt(2)}), ..., (M_{cp(W)}, M_{crt(W)})\}$. The output unit is a Selector and a W-multiplexer (W-MUX), and the final OPI is output by the combined action of the Selector and the Multiplexer. In the output unit, there may be multiple sub-blocks hitting at the same time, generating multiple Forwarding port indexes $\{OPI_1, OPI_2, ..., OPI_x\}$, the Selector enables W-MUX and takes the output port index of the sub-block with the largest serial number among them as the final lookup result of this flow identifier. Finally, the forwarding engine forwards the packet from the port

pointed by this port index according to the final port index of the data stream based on the calculation result.

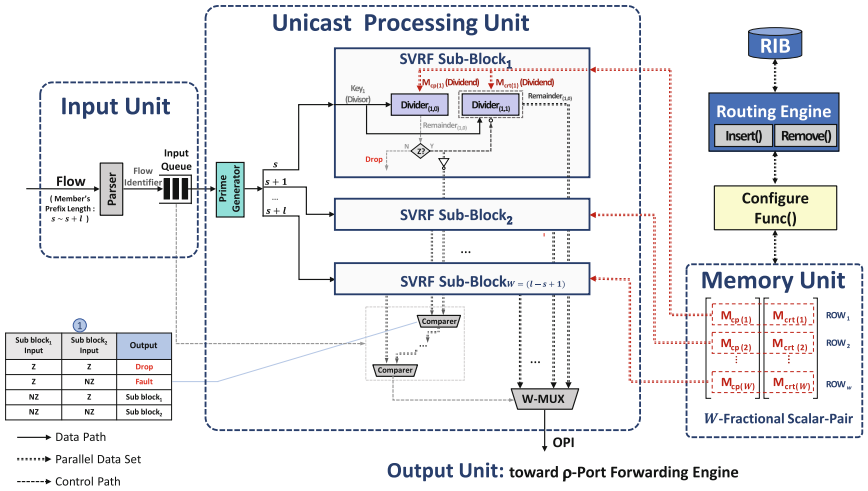


Fig. 1. Design of a Unicast packet forwarding Accelerator based on RNS algorithm.

3.2 Query Forwarding Process

The Routing Engine (RE) is responsible for constructing and maintaining the data structures in the memory cells, using a flow identifier (e.g., destination IP address) or some other information to identify a flow. The extractor of the Fraction-W SVRF extracts the flow identifier (e.g., the destination IP address) when a new flow enters the input unit. The input queue then caches the flow identifier information before it is mapped to each subblock of the processing unit in the order of the prefix length. For instance, the flow is mapped to the first SVRF subblock to carry out further forwarding operations when the prefix length is s (s is the shortest prefix length).

When dealing with a finite unicast forwarding table comprising n entries, SVRF is tasked with assigning an appropriate prime key to each forwarding entry based on specific assignment rules. We define the $i-$ th sub-block, where $i = 1,2, \cdots, W$. . And express all stream identifiers (keys) for the entire forwarding table entry in this simplified form: $\mathrm{K}_i = \left\{k_{(i,1)}, k_{(i,2)}, \ldots, k_{(i,n)}\right\}$. For unicast forwarding entries (UFEs) with their corresponding out port indexes (OPIs), we can generate a set that includes the OPIs of all forwarding entries: $U_i = \{u_{(i,1)}, u_{(i,1)}, ..., u_{(i,n)}\}$. OPI is a binary form of a ρ bit, where ρ indicates the number of ports supported by the PFE. The OPI of any UFE satisfies $u_i = \{u_{(i,1)}, u_{(i,2)}, ..., u_{(i,\rho)}\}$, each element $u_{(i,s)}$ s ∈ {1,2, ..., ρ} can only be "0" or "1". The OPI for each UFE is a positive integer value less than ρ. This value, represented as OPI_i for sub-block i, signifies the port to which the unicast packet will be forwarded. In addition, all keys are unique and are irreducible prime numbers that are mutually prime. In this context, it's worth highlighting that the key $k_{(r,x)}$ must be greater than the

corresponding i_x, and i_x must be equal to $log_2(\rho)$. So, for UFE, the SVRF distributes the prime key k_U bit length as $\lceil \log_2(\rho + 1) \rceil$. The sub scalar $M_{cp(i)}$ in SVRF sub-block i is defined as the concatenated product of the keys of all members of the sub-block:

$$\mathrm{M}_{\mathrm{cp}(i)} = \prod\nolimits_{\forall x \in n} k_{(i,x)}, \tag{8}$$

where $M_{crt(i)}$ is defined as the smallest positive integer solution to the set of one-time linear congruence equations for all member keys in the sub-block with the egress port vector OPI, meaning that $M_{crt(i)}$ satisfies the following conditions:

$$\mathrm{M}_{\mathrm{crt}(i)} \equiv \left\{ \begin{bmatrix} u_{(i,1)} \\ u_{(i,2)} \\ \vdots \\ u_{(i,n)} \end{bmatrix} \left(\mathrm{mod} \begin{bmatrix} k_{(i,1)} \\ k_{(i,2)} \\ \vdots \\ k_{(i,n)} \end{bmatrix} \right) \right. . \tag{9}$$

Using the Chinese Remainder Theorem (CRT), we can solve the smallest positive integer solution of a system of linear simultaneous equations for a given set of primes $k_1, k_2, \ldots, k_n$ with output port indices $u_1, u_2, \ldots, u_n$ as the remainder set and divisor set, respectively:

$$\mathrm{M}_{\mathrm{crt}(i)} = \mathrm{CRT}\big(\big\{k_{(i,1)}, u_{(i,1)}\big\}, \big\{k_{(i,2)}, u_{(i,2)}\big\}, \ldots, \big\{k_{(i,n_i)}, u_{(i,n_i)}\big\}\big). \tag{10}$$

When a specific SVRF sub-block, denoted as i, receives a query request, its first step is to identify the flow identifier of the data stream using the prime number generation function. This function allows it to obtain a unique and non-repeating prime key $k_{(i,x)}$ for that particular data flow. This prime key serves as the divisor, and the processing unit retrieves the target scalar pair $\{\mathrm{M}_{\mathrm{cp}(i)}, \mathrm{M}_{\mathrm{crt}(i)}\}$ from the i-th row of the memory scalar matrix, which acts as the dividend. Where each UFE uses only $\lceil \log_2(\rho + 1) \rceil$ bit. The two dividers in the processing unit concurrently execute modulo operations, resulting in two distinct remainders $\{\mathrm{R}_{(i,1)}, \mathrm{R}_{(i,2)}\}$ each with its respective significance:

$$\begin{cases} \mathrm{R}_{(i,1)} = \mathrm{M}_{\mathrm{cp}(i)} mod k_{(i,x)} \\ \mathrm{R}_{(i,2)} = \mathrm{M}_{\mathrm{crt}(i)} \bmod k_{(i,x)} \end{cases}, \tag{11}$$

where the first remainder $\mathrm{R}_{(i,1)}$ is the remainder of $\mathrm{M}_{\mathrm{cp}(i)}$ with $k_{(i,x)}$ and the second remainder $\mathrm{R}_{(i,2)}$ is the remainder of $\mathrm{M}_{\mathrm{crt}(i)}$ with $k_{(i,x)}$. On the one hand, if the value of the remainder $\mathrm{R}_{(i,1)}$ is 0, that means the data flow is in the forwarding table, then the value of $\mathrm{R}_{(i,2)}$ is the OPI_i computed by the corresponding subblock of a certain prefix length. When all sub-blocks have completed their calculations, there may be multiple sub-blocks that hit at the same time.

For sub-block$_i$, the zero adjudicator will activate the output $\mathrm{R}_{(i,1)}$, which represents OPI_i, when the remainder value $\mathrm{R}_{(i,1)}$ equals 0. Consequently, the sub-block's value will serve as the "1" hit signal, obtained by inverting the output of the remainder $\mathrm{R}_{(i,1)}$ when its value equals 0. The comparer will govern the output of the W-MUX based on the input values. The hit sub-blocks produce a sequence of hit matches delivered to the comparers. The comparer's role is to select the final hit, which corresponds to the

longest prefix, from this sequence of matches. It then activates the W-MUX to output the OPI_x associated with this sub-block. The input and output truth tables of the comparer are displayed as labeled "①" in Fig. 1, where the input variables represent the hits of the two sub-blocks entered based on the sub-block serial number size. When sub $block_1$ equals Z and sub $block_2$ equals Z (Z denotes a value of 0), it signifies that there is no hit for this IP address in both sub-blocks, and consequently, the packet should be dropped. If sub $block_1$ equals Z and sub $block_2$ is NZ (NZ indicates a value not equal to 0), it represents an error situation. This occurs because a packet should not have zero hits in the sub-block dealing with a shorter prefix length while receiving hits in the sub-block dealing with a longer prefix length. When sub $block_1$ is NZ and sub $block_2$ equals Z, it means that the comparer will output the hit information from sub $block_1$ and control the W-MUX to produce the OPI_1 of sub $block_1$. When both sub $block_1$ and sub $block_2$ are NZ, it signifies that the comparer will output the hit information from sub $block_2$ and control the W-MUX to output the OPI_2 of sub $block_2$. This implies that in cases of multiple consecutive hits, the OPI_x of the sub-block corresponding to the longest prefix will be selected as the final output port.

The forwarding engine then forwards the data flow based on the egress information provided by the final selected output port index. On the other hand, if the value of $R_{(i,1)}$ is not 0, it means that the flow is not in the forwarding table. In this case, the value of $R_{(i,2)}$ will be invalid. The system will discard the data flow and report this to the upper-layer router for processing. In this scheme, SVRF sub-block i processes incoming stream identifiers in parallel, resulting in the generation of two pieces of critical information: the output port index and the sub-block hit match value. The determination of the sub-block hit match value is carried out by the zero adjudicator. If the zero adjudicator determines that the value of $R_{(i,1)}$ is 0, it signifies both a sub-block hit (indicating that the stream is present in the forwarding table) and inputs the corresponding hit value into the Selector, following the order of the sub-blocks. Simultaneously, it represents a valid value for $R_{(i,2)}$ (indicating a valid OPI_i). With this scheme in place, a unitary scalar matrix equipped with a sufficient number of keys is capable of forwarding unicast packets to their respective multiple egress ports. Furthermore, each element within the set of OPI can be reconstructed using a linear congruence involving the scalar M_{crt} and the key set K. Importantly, this model offers a unique solution.

4 Performance Evaluation

4.1 Analysis of the Actual IPV4 Routing Information Table

In this section, we delve into the practical aspects of managing BGP prefixes within router forwarding engines. Fundamentally, we consolidate IP prefixes with numeric prefix matches and identical next-hop addresses, and we also aggregate multiple IP prefixes into a unified prefix. This process leverages BGP prefix reachability information sourced from the Route Views [27] project, a comprehensive infrastructure built upon BGP collectors for measurement purposes. We conducted our analysis using a subset of the Route Views project's 32 collectors, specifically selecting 10 Routing Information Base (RIB) based on their geographical location and size. To facilitate our examination, we employed BGPdump to parse these 10 RIBs, which are essentially routing tables

collected at 00:00 on August 17, 2023, featuring distinct geographical locations and varying sizes (from the 100,000 level to the 10 million level). Our analysis process was implemented in Python, enabling us to effectively analyze and process the collected RIBs both before and after prefix aggregation.

Table 1 provides a summary of the routing table dataset utilized in this paper, along with the associated data processing analysis. The dataset's name commences with "RV," signifying its origin from Route Views[27], with the first part of the table specifying the source host of the data. The second part of the table indicates the host's location. We have retained only the IPV4 portion of the collected routing table, excluding the IPV6 segment. The third and fourth sections of the table represent the IPV4 type of routing table entry (RTE) and the corresponding aggregated routing table entry (ARTE) for this dataset, respectively. Through the assessment of entry aggregability within the routing table, we distinguish between aggregable and non-aggregable entries. The aggregation outcome of the routing table maintains the results of both non-aggregable and aggregable entries. For instance, the second row of the table, labeled as "RV.chile," signifies that it was acquired from[27] via the host "route-views.chile.routeviews.org, "situated in Santiago, Chile. Within this dataset, there are a total of 939,811 IPV4 entries in the routing table (RTE), all of which have been aggregated, resulting in the same number of aggregated routing table entries (ARTE), totaling 939,811 entries.

We analyzed the routing tables after the aggregation process and performed a count of the distribution of various prefix lengths within these post-aggregation routing tables. We compiled statistical data regarding prefix distribution, and the outcomes, as anticipated, reveal an uneven distribution of IPV4 prefixes.

Table 1. RIB Datasets[27]

Host	Location	RTE	ARTE
RV. Chile	Santiago, Chile	939,811	939,811
RV. Nwax	NWAX, Portland, Oregon	1,833,868	1,833,798
RV. Kixp	KIXP, Nairobi, Kenya	2,998,596	2,776,625
RV. Sfmix	San Francisco Metro IX, California	3,944,550	3,631,361
RV. Uaeix	UAE-IX, Dubai, United Arab Emirates	4,928,858	4,696,891
RV. Rio	IX.br (PTT.br), Rio de Janeiro, Brazil	6,546,232	6,479,807
RV. Flix	FL-IX, Atlanta, Georgia	9,725,724	8,772,700
RV. Napafrica	NAP Africa, Johannesburg, South Africa	10,750,258	9,954,475
RV. Sydney	Sydney, Australia	12,692,359	11,874,154
RV. Chicago	Chicago, Illinois	13,826,495	13,028,551

Snapshot taken on August 17, 2023, at 00:00, Retrieved on September 3, 2023.

Figure 2 illustrates the average prefix distribution across all the routing tables in our dataset. The prefix lengths found in actual routing tables range from 8 to 32 bits, with approximately 60% of routing table entries falling within the 24-bit prefix length category. Notably, IP addresses with prefix lengths of 30 and 31 are typically utilized for point-to-point connections or individual IP address assignment to hosts, while IP addresses with a prefix length of 32 often denote a single host's address. In the context of the forwarding engine, it's worth noting that addresses with prefix lengths of 30, 31, and 32 enable the forwarding engine to perform direct forwarding based on specific address rules, obviating the need for additional lookup or matching processes. This capability enhances forwarding efficiency and speed. In IPv4 networks, it is a common practice to implement longest prefix matching in the forwarding engine framework design using techniques such as BF or hashing [22, 28]. Typically, these methods mechanically group prefixes into sets of 32 based on their length, which is regrettably recognized as an inefficient approach. This inefficiency arises because configuring redundant forwarding architectures for entries that can be directly forwarded would result in a wasteful allocation of resources, a scenario that is impractical in high-end switches.

Drawing from the observed prefix distribution in actual network routing tables, our proposed implementation of the architecture employs $W = 22$, meaning it utilizes 22 SVRF sub-blocks. Each sub-block is exclusively associated with a specific prefix length, spanning from 8 to 29 for the entries within these sub-blocks.

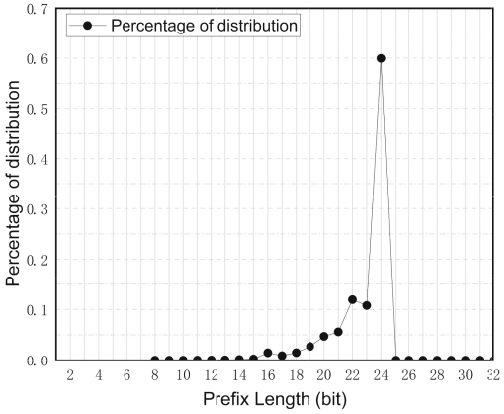


Fig. 2. Distribution of prefix lengths in the aggregated routing table.

4.2 Systematic Assessment

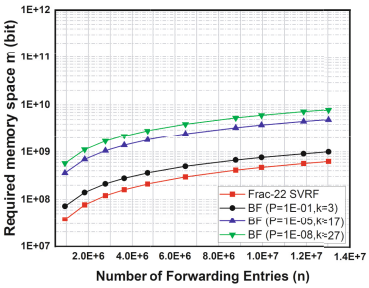
To assess the performance of Frac-22 SVRF and conventional BF across various port densities, we developed a representative Packet Forwarding Engine (PFE) and its corresponding membership query algorithm using the Python programming language. During

the simulation, we manipulated six distinct port density scenarios, ranging from typically 16 to 1024 ports each. In each scenario, we instantiated 10 different member capacities (shown in Table1 ATE) for the reference PFE, utilizing the actual routing table ARTE dataset as presented in Table 1. This simulation considers two metrics 1) space efficiency, and 2) time efficiency. A PFE constructed using the BF scheme denoted as BF (P, k), consists of k hash functions, each with a false positive ratio of ρ. In this context, when we refer to ρ-ports, it means that the BF construction will be duplicated ρ times. For all sizes of PFE, the number of partitions for our proposed scheme will be fixed at W = 22. We established three different false-positive rates for BF, each with its corresponding optimal number of hash functions parameterized as follows 1) BF (k = 3, P = 1E–01), 2) BF (k = 17, P = 1E–05) and 3) BF (k = 27, P = 1E–08). In each case, we adjusted the number of hash functions (k) to achieve the desired false-positive rate (P). In the hardware implementation, we employed SRAM-based 32-bit FPGAs for constructing the Frac-W SVRF, SVRF, and BF gas pedals. These FPGAs operated at a processing frequency of 2 GHz, and the data path connecting the memory and the processing unit had a width of 32 bits. For the SRAM within the FPGA, we assumed an access time of 10 ns for each bit. Additionally, we allocated one clock cycle for each compare and select operation, including operations involving the Selector and W-MUX components.

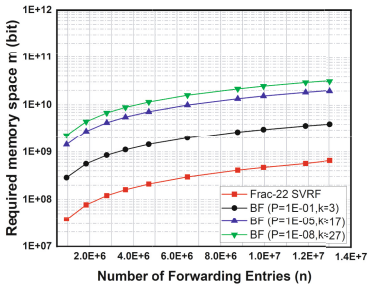
4.3 Space Efficiency

In the first simulation scenario, we examine the relationship between memory space (m) and the number of forwarding entries (n). To assess performance, we conduct a comparative analysis between Frac-22 SVRF and a general BF. It is important to highlight that system performance remains consistent regardless of the prefix distribution when each BF exhibits the same ratio of false positives to the number of hash functions. Figures 3(a)–(f) illustrate the relationship between memory utilization and the number of forwarding entries for scenarios involving each with varying port densities (16, 64, 128, 256, 512, and 1024 ports) PFEs. Simulations show that the memory footprint of the BF is consistently much higher than that of the Frac-22 SVRF.

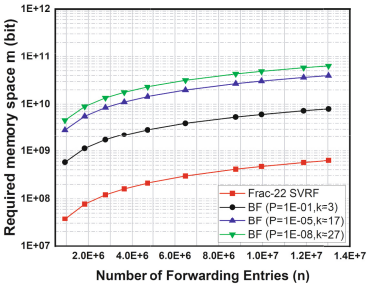
Frac-22 SVRF performs better when applied to PFEs with lower port density. On the one hand, in Frac-22 SVRF, the key length is related to the number of ports, not the number of forwarding entries (n). Frac-22 SVRF exclusively employs $\lceil \log_2(\rho + 1) \rceil$ length keys for unicast entries. The use of shorter keys allows Frac-22 SVRF to achieve more compact memory utilization and store forwarding membership information more efficiently, particularly when dealing with a smaller number of ports. In contrast, the BF scheme requires deploying a proportional number of BF based on the number of ports within the PFE. Consequently, as the port density increases, the BF scheme necessitates the deployment of more BF and consumes more memory space. In contrast, Frac-22 SVRF maintains a fixed number of sub-blocks executed in parallel at 22. Moreover, as the port density increases, the need for changing key lengths in Frac-22 SVRF remains relatively stable and follows a logarithmic pattern. This stability results in a more consistent memory footprint for Frac-22 SVRF compared to BF.



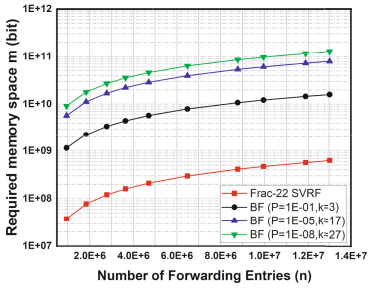
(a) ρ=16



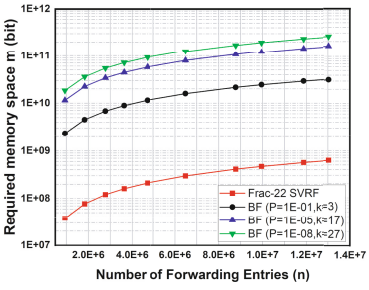
(b) ρ=64



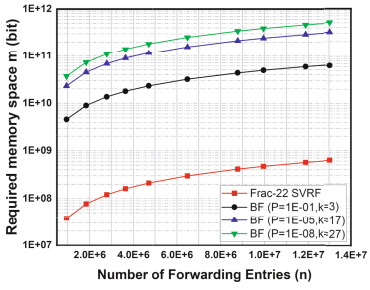
(c) ρ=128



(d) ρ=256



(e) ρ=512



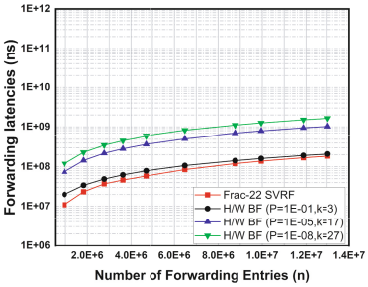
(f) ρ=1024

Fig. 3. Required memory space m vs. the number of forwarding entries n in the ρ-port PFEs.

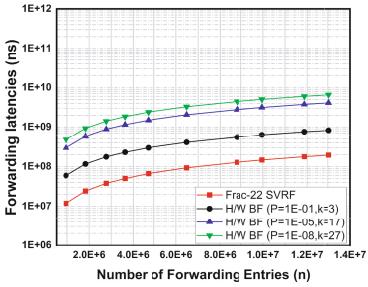
4.4 Time Efficiency

The forwarding delay stands as a crucial metric for assessing PFE performance. This delay is contingent upon the time complexity of the membership query, specifically the time it takes for the query process to ascertain whether a packet belongs to a forwarding member or not. In general, reducing the required memory space for a membership query

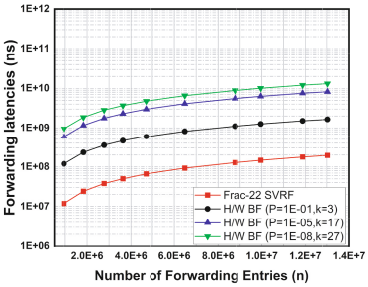
also leads to a reduction in the overall query process time. In theory, we can gauge the time complexity as O(k) for Frac-22 SVRF based on the total number of parallel executions of the division algorithm, where k signifies the membership queries that can be solved in constant time. However, in practical implementation, dealing with arithmetic operations involving large operands, such as requiring 10,000 bits of Mcrt



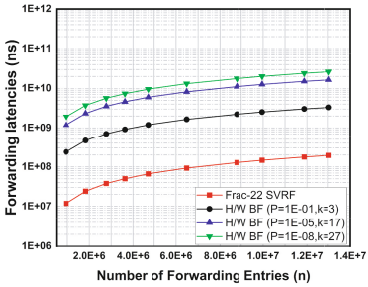
(a) ρ=16



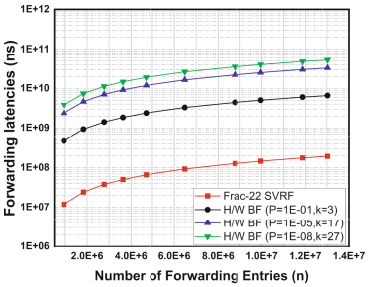
(b) ρ=64



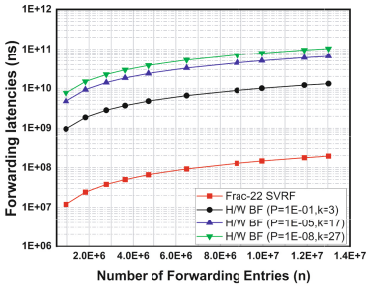
(c) ρ=128



(d) ρ=256



(e) ρ=512



(f) ρ=1024

Fig. 4. Forwarding latencies vs. number of forwarding entries n in the ρ-port PFEs.

as a divisor, presents challenges. Figures 4(a)–(f) depict the correlation between the steady-state average packet processing delay and the maximum membership capacity in PFEs across various port densities. The presented curves demonstrate that Frac-22 SVRF consistently delivers strong forwarding performance. It is noticeable that, as the number of ports increases, Frac-22 SVRF shows better performance.

This achievement can be attributed to our optimized partitioning strategy, which facilitates the independent execution of query operations within each sub-block. This approach minimizes the impact on memory access times and keeps memory changes insignificant even as the number of ports increases. Through its reduced memory consumption, Frac-22 SVRF succeeds in enhancing forwarding time efficiency. In contrast, BF requires scanning the entire predetermined memory space during a membership query, comparing each identified hash value. When the memory space is well-defined, BF takes a considerable amount of time to determine the final membership query result. The study's results unequivocally illustrate that Frac-22 SVRF is proficient at reducing the forwarding delay in PFEs that support classless inter-domain routing, thereby significantly enhancing the forwarding benefits.

5 Conclusion

This paper delivers into the implementation of classless inter-domain routing and forwarding engines, taking into account various port densities and the presence of large membership capacities. Our scheme primarily addresses scalability and efficiency concerns related to routing and forwarding engines. To enhance scalability and time efficiency, we conduct an evaluation of the proposed packet forwarding algorithms with a focus on memory utilization and forwarding delay. Among these algorithms, SVRF stands out as one of the representative schemes for membership querying, while the improved Frac-N SVRF scheme is designed to cater to multicast-enabled packet forwarding engines.

To enhance the applicability of classless inter-domain routing, we introduce an optimized Frac-22 SVRF scheme. Our approach involves the analysis of routing table data in real network scenarios, focusing on whether the routing table entries meet the aggregation criteria. This process distinguishes entries that can be aggregated from those that cannot, leading to the consolidation of forwarding tables. In a series of performance simulations, as opposed to relying solely on theoretical values, we utilize actual aggregated routing table datasets to instantiate our proposed scheme and compare it with other schemes. The spatial and temporal efficiency of Frac-22 SVRF outperforms traditional Bloom Filters (BF) across various port density PFEs.

References

1. Dharmapurikar, S., Krishnamurthy, P., Sproull, T., Lockwood, J.: Deep packet inspection using parallel Bloom filters. In: 11th Symposium on High Performance Interconnects, 2003. Proceedings, pp. 44–51 (2003)
2. BGP Reports. https://bgp.potaroo.net/
3. CIDR Report. https://www.cidr-report.org/as2.0/

4. Hinden, B.: Applicability statement for the implementation of classless inter-domain routing (CIDR). In: Internet Engineering Task Force (1993)
5. Waldvogel, M., Varghese, G., Turner, J., Plattner, B.: Scalable high-speed prefix matching. ACM Trans. Comput. Syst. **19**, 440–482 (2001)
6. Cerović, D., Del Piccolo, V., Amamou, A., Haddadou, K., Pujolle, G.: Fast packet processing: a survey. IEEE Commun. Surv. Tutor. **20**, 3645–3676 (2018)
7. Aweya, J.: Introduction to switch/router architectures. In: Switch/Router Architectures: Shared-Bus and Shared-Memory Based Systems, pp. 1–16. IEEE (2018)
8. Wei, J., Jiang, H., Zhou, K., Feng, D.: Efficiently representing membershipfor variable large data sets. IEEE Trans. Parallel Distrib. Syst. **25**, 960–970
9. Aweya, J.: Switch/Router Architectures: Shared-Bus and Shared-Memory Based Systems. Wiley, Hoboken (2018)
10. Radoslavov, P.I., Estrin, D., Govindan, R.: Exploiting the Bandwidth-Memory Tradeoff in Multicast State Aggregation
11. Li, Q., Xu, M., Wang, D., Li, J., Jiang, Y., Yang, J.: Nexthop-selectable FIB aggregation: an instant approach for internet routing scalability. Comput. Commun. **67**, 11–22 (2015)
12. Ghosh, S., Baliyan, M.: A hash based architecture of longest prefix matching for fast IP processing. In: 2016 IEEE Region 10 Conference (TENCON), pp. 228–231 (2016)
13. Zarandi, A.A.E.: RNS applications in computer networks. In: Molahosseini, A.S., de Sousa, L.S., Chang, C.-H. (eds.) Embedded Systems Design with Special Arithmetic and Number Systems, pp. 369–380. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-49742-6_14
14. Eatherton, W., Varghese, G., Dittia, Z.: Tree bitmap: hardware/software IP lookups with incremental updates. SIGCOMM Comput. Commun. Rev. **34**, 97–122 (2004)
15. Degermark, M., Brodnik, A., Carlsson, S., Pink, S.: Small forwarding tables for fast routing lookups. SIGCOMM Comput. Commun. Rev. **27**, 3–14 (1997)
16. Hasan, J., Cadambi, S., Jakkula, V., Chakradhar, S.: Chisel: A Storage-efficient, collision-free hash-based network processing architecture. In: 33rd International Symposium on Computer Architecture (ISCA 2006). pp. 203–215 (2006)
17. Pong, F., Tzeng, N.-F.: Concise lookup tables for IPv4 and IPv6 longest prefix matching in scalable routers. IEEE/ACM Trans. Netw. **20**, 729–741 (2012)
18. Tobola, J., Kořenek, J.: Effective hash-based IPv6 longest prefix match. In: 14th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems, pp. 325–328 (2011)
19. Tzeng, H.H.-Y., Przygienda, T.: On fast address-lookup algorithms. IEEE J. Sel. Areas Commun. **17**, 1067–1082 (1999)
20. Chazelle, B., Kilian, J., Rubinfeld, R., Tal, A.: The bloomier filter: an efficient data structure for static support lookup tables (2004)
21. Park, G., Kwon, M.: An enhanced bloom filter for longest prefix matching. In: 2013 IEEE/ACM 21st International Symposium on Quality of Service (IWQoS), pp. 1–6 (2013)
22. Lim, H., Lim, K., Lee, N., Park, K.-H.: On adding bloom filters to longest prefix matching algorithms. IEEE Trans. Comput. **63**, 411–423 (2014)
23. Song, H., Hao, F., Kodialam, M., Lakshman, T.V.: IPv6 lookups using distributed and load balanced bloom filters for 100 Gbps core router line cards. In: IEEE INFOCOM 2009, pp. 2518–2526 (2009)
24. Pei, D., Salomaa, A., Ding, C.: Chinese Remainder Theorem: Applications In Computing, Coding, Cryptography. World Scientific, Singapore (1996)
25. Jia, W.-K., Wang, L.-C.: A unified unicast and multicast routing and forwarding algorithm for software-defined datacenter networks. IEEE J. Select. Areas Commun. **31**, 2646–2657 (2013)

26. Jia, W.-K., Jin, Z.: Fractional- *N* SVRF forwarding algorithm for low port-density packet forwarding engines. IEEE Netw. Lett. **3**, 42–46 (2021)
27. Route Views Archive Project Page. http://archive.routeviews.org/
28. Lim, H., Seo, J.-H., Jung, Y.-J.: High speed IP address lookup architecture using hashing. IEEE Commun. Lett.Commun. Lett. **7**, 502–504 (2003)