On Designing Fast Nonuniformly Distributed IP Address Lookup Hashing Algorithms
Christopher J. Martinez, Devang K. Pandya, and Wei-Ming Lin

Abstract—Computer networks have continued to make substantial advances in the past couple of decades through better technologies and methodologies employed. As the usage of the networks continues to increase exponentially, high throughput of the networks has to be maintained with various performance-efficient network algorithms. IP address lookup is one of the processes, the performance of which dearly affects the overall network performance. Hashing has been widely used for fast IP address lookup due to its simplicity, but mostly assuming on hashing from an address set with uniformly distributed key values. Performance from these known hashing techniques is far from optimal due to the high nonuniformity in actual IP address distribution. In this paper, we propose a preprocessing method for the IP address databases to extract certain regularity to allow for design of more efficient hashing algorithms based on XOR operations. Simulation results show an improvement in performance ranging from 35% to 72% on randomly generated addresses and several sample IP address databases. The paper also shows that the proposed algorithms deliver comparable performance to other well-known hashing algorithms such as the CRC and RS hashing while requiring much less hardware to implement and a much shorter time to perform.

Index Terms—Hashing algorithm, IP address lookup, nonuniform distribution.

I. INTRODUCTION

IN THE PAST decade, the usage of applications in computer networks has dramatically increased. Most applications involve some form of network activity and heavily rely on accesses to networks. Researchers have been pushing transmission speeds to higher levels in hope of matching the increasing need for throughput. In order to fully exploit the transmission potential, many algorithms that make up the backbone of the network devices need to be further improved. One hindrance that all network components—such as routers, firewalls, intrusion detection systems, and others—have faced is that they all need to be matched quickly in real time with the network devices, requiring an extra update process in a lookup table. Other lookup schemes have also been proposed, such as the one in [1], in which addition and multiplication steps are used iteratively to create a hash value. The multiplication is used as a way of transporting changes from high-bit positions to low-bit positions. All hashing algorithms have also been widely adopted to provide for the address lookup process [2], [4], [13], [20].

Fast address lookup and identification matching has become critical to the feasibility of many modern network applications. In a general form, this problem involves a search process through a large database to find a record (or records) associated with a given key. One modern example is in the routers in a wide-area network having to maintain a large database and routing table and to search through it for a forwarding link that matches the given destination address. Another example that calls for imminent attention is in the area of network security, in which intrusion detection demands rapid evaluation of client requests. In this, rules are established to allow for the intrusion detection system to check for wrongdoing. Usually, packet headers need to be matched quickly in real time with the rule database. Such a matching process usually is carried out through a hashing process to reduce the otherwise potentially excessively long search time.

There have been many schemes developed for the IP address lookup problem using a hash function. A complete survey and complexity analysis on IP address lookup algorithms has been provided in [14]. A performance comparison of traditional exclusive OR (XOR) folding, bit extraction, CRC-based hash functions is given in [5]. Although most of the regular hash functions, such as the simple XOR folding and bit extraction, are relatively inexpensive to implement in software and hardware, their performance tends to be far from desirable. CRC-based hash functions have proven to be excellent means, but have some potential shortcomings. Compared to a simple XOR folding hash algorithm that can be implemented in a fast parallel circuit, the CRC-based hash function requires a sequential circuit and a much longer time to determine the hash value. Some schemes are hardware-based that achieve an improvement in IP lookup by maintaining a subset of routing table in a faster cache memory [9], [11], while others are software-based that improve their search performance mainly through efficient data structures [12], [17]. In [18], an address lookup scheme is proposed based on a binary search of a hash table, requiring an extra update process in a lookup table. Other hashing algorithms have also been widely adopted to provide for the address lookup process [2], [4], [13], [20]. All hashing algorithms inevitably suffer from unpredictable complexities involving conflicts among the data with the same hash result. A search for matching a given query could end up with a sequential search through the number of maximal conflicts in the database. This may result in a long search process time that exceeds the time limitation imposed by design specifications.

Another class of hashing algorithms are multiplicative algorithms, in which addition and multiplication steps are used iteratively to create a hash value. The multiplication is used as a way of transporting changes from high-bit positions to low-bit positions. One well-known multiplicative hashing algorithm is the RS hash [16], which uses a series of multiplication and addition steps to reach a hash value. Besides the basic hashing algorithms...
used in fast address lookup, there also exist a set of well-established hash algorithms such as MD4, MD5, SHA-1, and SHA-2, which have found use in the cryptography field [3], [15]. These algorithms rely on a series of addition, bit rotation, and logic operations through many cycles. These algorithms, though very useful in the area of cryptography, have limited applications in the address lookup study due to the drastic difference between the two areas in terms of requirement in collision rate and processing time.

Many research approaches in the literature have been based on an assumption that records in the database are key-wise uniformly distributed. Under this assumption, any regular hashing algorithm would easily lead to the same probabilistically expected performance in terms of search time required. On the other hand, if these records are instead not uniformly distributed, then different regular hash functions would lead to different expected performance. Hash results delivered by traditional hashing algorithms usually are far from optimal when the database presented is not uniformly distributed. This paper proposes a unique hash algorithm that considers specific features within each database that facilitates the design of an efficient hashing approach. By preprocessing the original database to reorder the keys, our proposed designs allow the application of a standard XOR folding hashing to produce a significantly improved performance, which is clearly demonstrated by our simulations on address databases either randomly generated or extracted from real IP traffic.

II. HASHING FOR ADDRESS LOOKUP

Basically, hashing is a process that allows the search to go through a statistically smaller number of steps than a simple sequential straightforward search would have performed. A hash function, usually a mathematical one, maps a number with a large value range into another number with a smaller range. For example, a simplified one as shown in Fig. 1, a database of eight given records (e.g., addresses in our application) are to be matched against for any incoming record. Due to the large size of each record and potentially large number of records in the database under real situations, searching through the whole database one at a time could be merely impractical. One may choose to use a portion of the record (or its entirety) as a key to hash into the target value (a 3-bit value, as shown in this example) using the hash function (operator) \( H \). Therefore, a database of eight records are now grouped into bins of records according to their corresponding hash results. With this, any incoming record would go through the same hashing process to identify the one bin it would need to search through, instead of the whole database. Perfect hashing would guarantee that every bin contains exactly one record, which leads to a search process of exactly one matching to take place. A hash function is considered better than the other if it leads to a smaller expected number of matching steps required. Note that if the records in the database follow a uniform distribution with respect to value of the key, any regular hash function—e.g., nonoverlapped XOR, bit extraction, etc.—would simply lead to a uniform distribution of these records onto the bins. That is, it should always lead to best expected performance probabilistically in search time required. The goal of this paper is to develop a universal hashing methodology applicable to nonuniformly distributed data sets.

Due to the nature of different applications, databases (data sets) to be hashed usually do not follow a pure random distribution on their key fields. For example, in computer networks address lookup where IP addresses (and some other fields used for key) are known to follow a nonuniform probability density function (pdf) due to the way the addresses are artificially assigned. Fig. 2 shows the binary value distribution of the most significant byte of the Subnet ID field from packets (of three different classes) entering a network router in a duration of a few hours, a clear demonstration of nonuniformity in potential key value distribution in this application.

A. Simulation Setup

Our simulation is devoted to comparing among technique on several practical performance indicators. A system is assumed to have a database of \( 2^{n} \) entries, each of which contains a key of \( n \) bits to be used for hashing into \( m \) bits. A standard performance comparison uses the required number of search steps through the hash cells, or hash bins (as shown in Fig. 1). Thus, each of the \( 2^{n} \) entries is mapped to a hash cell for matching.
Two widely used performance measurements are adopted in this paper: 1) maximal search length (MSL); and 2) average maximal search length (ASL). MSL denotes the largest number of entries that are mapped into any hash bin, thus representing the longest possible required search time. ASL is the average maximum number of matching steps needed for any given record to match. Data sets are generated either randomly or extracted from real IP traffic.

III. PROPOSED HASHING ALGORITHMS

The general methodology of the proposed hashing algorithm is to extract critical distribution information in the target database so that it can be rearranged for a straightforward application of a generic XOR folding hashing algorithm.

A. Preprocessing of Database

As aforementioned, the database is defined as consisting of \( M = 2^n \) entries, with each entry having \( n \) bits in length. It can also be viewed as having \( n \) \( M \)-bit vectors labeled as \((V_{n-1}, V_{n-2}, \ldots, V_0)\), with each vector consisting of each respective bit from all entries. An example of \( n = 8 \) and \( m = 3 \) is shown in Fig. 3(a). The target hashing process is to hash each of the \( n \)-bit entries (an IP address or part of it in this application) into an \( m \)-bit hash value. These hash values need to be distributed as evenly as possible so as to minimize the eventual search time. In order to produce the best (uniform) distribution in the final hashed data set, each of the \( m \) bits in the final hashing value should demonstrate a distribution as probabilistically random as possible, i.e., evenly distributed between 0’s and 1’s. This will then provide the best distribution of the target records into the hash cells. To quantify the randomness of each bit vector in the original database, we define \( d_i \), for bit vector \( V_i \), as the absolute difference between the number of 0’s and 1’s in \( V_i \) across the data set. Fig. 3(a) displays the \( d \) values thus determined for the shown database example. The value \( d \) gives us a very useful insight into the degree of uniformity of the database. A bit vector \( V_i \) with \( d_i = 0 \) has the same number of 0’s and 1’s, while a \( d = M \) indicates that all the bits in the vector have the same value.

Translated to effect of hashing, in the final \( m \)-bit hash result, a bit of \( d = 0 \) gives an even hashing distribution (i.e., evenly divided) among the entire address space, allowing other bits to hash to it, while a bit of \( d = M \) will limit the hashing to only one half of the hash space. Intuitively, using the bits with smaller \( d \) values for hashing would lead to a probabilistically better hash distribution, i.e., less potential conflict in the final mapping. Ideally, if one can identify (or through a combination to obtain) \( m \) bits with all their \( d \) values equal to 0, it should lead to the best potential performance, assuming no correlation among the bit vectors. This leads us to employing a simple preprocessing step in rearranging the \( n \)-bit vectors according to their \( d \) values sorted into a nondecreasing order as shown in Fig. 3(b). This sorted sequence then gives us an “order of significance” according to which each bit should be utilized.\(^1\)

An immediate benefit from this preprocessing step can be demonstrated by observing the performance difference between applying a simple bit-extraction hashing process on the nonpreprocessed database and the preprocessed one. A bit-extraction hashing is to simply extract \( m \) bits from the \( n \)-bit entry as its hash value. Although it represents the simplest hashing approach, its performance usually is not comparable to other standard hashing techniques that utilize all the \( n \) bits. Assume that the bit sequence before sorting is \((b_{n-1}, b_{n-2}, \ldots, b_1)\) and the bit sequence afterward is \((s_{n-1}, s_{n-2}, \ldots, s_0)\). Bit-extraction on the nonpreprocessed database would be arbitrarily extracting \( m \) bits, or simply extracting the first \( m \) bits \((b_{n-1}, b_{n-2}, \ldots, b_{n-m})\) if no information on the database has been provided. After the sorting, the first \( m \) bits that have the \( m \) smallest \( d \) values become obvious candidates for bit extraction. That is, the final hash value is \((s_{n-1}, s_{n-2}, \ldots, s_{n-m})\). Fig. 4 depicts the two bit-extraction techniques compared here: bit-extraction without preprocessing (EXT) and bit-extraction with preprocessing by sorting with \( d \) values (\( d \)-EXT). Fig. 5 gives the performance comparison between the two bit extraction approaches, when \( n = 32 \) and \( m \) is varied, using a data set randomly generated. The data set is generated by randomly assigning each bit vector a \( d \) value ranging from 0 to \( 2^n - 1 \), thus leading to a uniform distribution of \( d \) values. Afterward, the bit vector is then generated by randomly assigning 0’s and 1’s on it indepen-
1’s to the \( m \) bits to reflect its chosen \( d \) value. The \( d \)-EXT easily outperforms the random EXT approach by an extensive margin. Reductions of up to 92% in MSL are obtained, signified by the case in \( m = 14 \) where the MSL is reduced from 292 to 23. This easily could lead to a hardware implementation achieving a real-time processing speed 10 times faster. Although bit-extraction approaches cannot fully exploit the hashing potential due to their using only a portion of the bits, the above observation clearly indicates the “order of significance” among the bits in how they should be used when all bits are to be used for hashing.

B. XOR Hashing

The XOR operator has been widely used for hashing and has been known to be an excellent operator in enhancing randomness in distribution. It also possesses a nice characteristic, allowing for analytical performance analysis and, thus, a better algorithm design. Group-XOR is a commonly used hashing technique by simply grouping the \( n \)-bit key into \( m \)-bit hash result through a simple process XORing every \( n/m \) key bits into a final hash bit. Such a random XORing process (so-called “group-XOR” in this paper) may not always lead to a desirable outcome. If randomness (i.e., degree of even distribution) is not improved when bits are XORed, then hash performance is degraded. The goal of this paper is to use the extracted information from the preprocessing (\( d \) values) to facilitate a better hash design with the XOR operator.

To precisely quantify the benefit when XORing two bit vectors with their \( d \) values being \( d_i \) and \( d_j \), a formula can be derived to find the expected resultant \( d \) for the new bit vector after XORing, denoted as \( d_{[d_i \oplus d_j]} \)

\[
d_{[d_i \oplus d_j]} = \sum_{k=0}^{x_i} |M - 2\Delta - 4k| \cdot \frac{C_{x_i}^k \cdot C_{M-x_i}^{M-k}}{C_M^M}
\]

where

\[
x_i = \frac{M - d_i}{2}, \quad x_j = \frac{M - d_j}{2}, \quad \Delta = x_i - x_j.
\]

This equation is derived by taking into account all possible combinations of the two bit vectors with \( d_i \) and \( d_j \). Note that the term \( C_{x_i}^k \cdot C_{M-x_i}^{M-k} \) indicates the probability for the resultant vector to have a \( d \) value equal to \( |M - 2\Delta - 4k| \). A complete spectrum of \( d_{[d_i \oplus d_j]} \) for all possible integral values of \( d_i \) and \( d_j \) for \( M = 32 \) is given in Fig. 6. Ideally, in order not to degrade the hash performance, every intended XOR operation to be taken between two bits \( s_i \) and \( s_j \) after sorting, with \( d_i \leq d_j \), should lead to a \( d_{[d_i \oplus d_j]} \) value such that \( d_{[d_i \oplus d_j]} < d_i \) assuming no correlation exists in between bit vectors \( V_i \) and \( V_j \). To further quantify the gain (or damage) caused by XORing two bits, the percentage of reduction in \( d \) value, denoted as \( p_{ij} \), between \( d_i \) and \( d_j \) is determined by the following formula:

\[
p_{ij} = \frac{\min(d_i, d_j) - d_{[d_i \oplus d_j]}}{\min(d_i, d_j)}.
\]

Fig. 7 plots the reduction percentage in \( d \) value for the entire range of \( d_i \) and \( d_j \) with \( M = 64 \). In this plot, one can roughly see that XORing bits with small \( d \) values can easily lead to a negative reduction, i.e., a larger \( d \) value. Also, XORing bits with very large \( d \) values leads to minimal reduction. Although we believe that information in this plot can be fully exploited when designing a hashing approach using a complex ad hoc bit selection process, the goal here is to provide a very generic hashing scheme in using all the bits by taking advantage of the information from our preprocessing step.

C. \( d \)-Value-Based XOR Folding Hashing

With the results from Figs. 6 and 7, an obvious approach to utilize all the bits for XORing similar to the group-XOR one is to first establish an \( m \)-bit base, the first (and the best) \( m \) bits in the sorted sequence, and then XOR the rest into the base, hoping to provide more reduction in their \( d \) values for more randomness. Two straightforward ways to exploit the benefit from the \( d \)-value-based sorted sequence are to perform XOR hashing on the preprocessed database in the following order:

- \( d \)-Value-In-order XOR (\( d \)-IOX): As shown in Fig. 8(a), in which all successive segments are aligned in the same ascending order for XORing, this provides a straightforward XOR hashing sequence as in a normal group-XOR hashing algorithm.
- \( d \)-Value-Snake-Order XOR (\( d \)-SOX): As shown in Fig. 8(b), the order of the \( d \)-SOX algorithm is decided...
by bending the $m$-bit segments in a snake-order fashion: the first segment (segment $a$) is taken in order, the second segment (segment $b$) then aligned with the first segment in reverse order, and alternating between the two orders in subsequent segments. This folding approach is designed to deliver a better (more balanced) combination among all hash bits at the end by taking advantage of the symmetry displayed in the $d$-value reduction percentage graph in Fig. 7.

In general, these two folding-XOR approaches should lead to improvement over the group-XOR hashing due to the sorting preprocess. The traditional group-XOR process may XOR bits that easily lead to detrimental effect, while both $d$-IOX and $d$-SOX avoid XORing two bits both with small $d$ values (the worst possible XORing) and XORing two bits with large $d$ values (the XORing leading to minimal gain). By having the database sorted by $d$ value, each set of bits to XOR now has a wide range of $d$ values, which tends to minimize the chance of destructive XORing. These two proposed techniques tend to XOR bits from the smaller range with bits from the larger range, which is more likely to lead to a beneficial effect. In addition, the $d$-SOX algorithm is able to take advantage of one more scenario demonstrated in Fig. 7, in which a pyramid shape of reduction ratio indicates that XORing bits with similar $d$ value in the mid range provides the best reduction. With the wrapping around in the snaking order, the $d$-SOX is able to line up bits with similar $d$ values close to the wrapping (folding) points for XORing.

D. $d$-Value-Based Natural-Folding XOR Hashing

While the $d$-SOX and the $d$-IOX, compared to the traditional group-XOR, provide a more “balanced” XOR combination among the sorted bit vectors in general, specific grouping among the bits for XORing is still far from optimal.

To further exploit the somewhat symmetric pyramid-shaped feature in Fig. 7, one should try to wrap the sorted bit sequence as “symmetric to the center” as possible, assuming that the $d$-value distribution among the bit vectors is somewhat uniformly spread around in the $d$-value space. By doing so, bit vectors with smaller $d$ values are XORed with larger $d$-value bits in order to have a better chance for further reduction, and more importantly, bit vectors in the middle range are XORed together to provide the most reductions available. Detailed implementation of such a “symmetric wrapping” approach depends on the relation between $n$ and $m$. For example, when $n = 3m$ as shown in Fig. 8, segment $a$ and segment $c$ are supposed to be reversely grouped to lead to the best balancing; instead, they are in-order matched. This leads to a potentially beneficial conversion process demonstrated in Fig. 9. This proposed technique is called “natural-fold XOR” [$d$-NFX, as shown in Fig. 9(b)]. Instead of snake-ordering from the beginning bits as in $d$-SOX, the $d$-NFX folds the sorted bit sequence from both ends’ matching pair of bits accordingly. Thus, segments $a$ and $c$ are paired in a “natural folding” order. In this case of $n = 3m$, the middle segment will be folded in half. After running preliminary testing on this approach, the performance is not significantly improved due to loss in uniformity in terms of number of bits XORed to produce the final hash bits. That is, in this example, half of the hash bits are from XORing two bits each, and the other half from XORing four bits each. To remedy this problem, one simple way is to simply duplicate the middle subsegments, $b_1$ and $b_2$, to patch up the missing portion for uniformity, which leads to the final proposed technique, the “natural-fold with duplication XOR” [$d$-NFD, as shown in Fig. 9(c)]. This technique may lead to overduplication or underduplication on the center subsegments. These two situations are illustrated in Fig. 10. In case (a), when $n = 32$ and $m = 10$, the length of each of the $b_1$ and $b_2$ is 6; thus, duplicating the center segment once would overshoot the boundary of the final $m$ bits. A simple method is adopted in simply truncating the bits overshot (the shaded portion). On the other hand, as shown in (b), when $n = 32$ and $m = 7$, the center subsegments $c_1$ and $c_2$ each have only 2 bits left, thus requiring a duplication of more than once. The overshot portion is disregarded as well.

Note that, due to the potential duplication, the number of actual bits to be XORed, denoted as $n'$, can be decided as

$$n' = \left\lfloor \frac{n}{2m} \right\rfloor \times 2m.$$

When $n$ is an integral multiple of $2m$, $n'$ is equal to $n$; that is, the proposed $d$-NFD becomes identical to $d$-SOX if this condition is satisfied.

IV. SIMULATION RESULTS

To demonstrate the performance improvement from the three proposed XOR hashing techniques—$d$-IOX, $d$-SOX, and $d$-NFD—over the group-XOR technique, a series of simulation runs are performed on a variety of sets of data. Additional
simulation runs are also performed to compare the proposed XOR hashing techniques with two other well-known hash functions, the CRC and RS hashing. The usage of CRC as an IP address lookup algorithm is closely examined in [5]. The standard CRC-32 hash function requires 32 iterations to generate the final hash value for a given hash key, requiring additional control logic to properly maintain the sequential process. The CRC-32 will be used in our simulation for comparison, which always results in a 32-bit hash value, with the lower \( m \) bits being used in our comparison. The RS hash function [16] is a multiplicative hash algorithm that requires two multiply and one addition steps for every 8 bits of hash key to generate a hash value. For IP addresses with a 32-bit hash key, RS hash will require a total of eight multiply and four addition sequential steps. Similar to CRC, only the least significant \( m \) bits in the hash value generated by the RS hash are used in the simulation.

### A. Randomly Generated Data Set

The first set of data used for our simulation are randomly generated such that the \( d \) value for each bit position is uniformly distributed. This randomly generated data set gives a distribution of \( d \) values covering a larger range than a typical network address set would have. The data set is generated by randomly assigning each bit vector a \( d \) value ranging from 0 to \( 2^m - 1 \), thus leading to a uniform distribution of \( d \) values. Afterward, the bit vector is then generated by randomly assigning 0’s and 1’s to the \( m \) bits to reflect its chosen \( d \) value. Fig. 11 shows the \( d \) values of all bits in a sample data set with \( m = 14 \) and \( n = 32 \) thus generated. Motivation of using this type of data set is to provide a more general comparison result for potential performance gain. Since no sorting is applied, all bits using the current protocol setting.

The simulation results for \( n = 32 \) and \( 10 \leq m \leq 15 \) are given in Fig. 12 in terms of MSL and ASL by taking an average of results from 1000 runs. Table I gives a summary of performance gain in MSL from each of the three proposed techniques and the two reference techniques over the group-XOR. These results show that for each of the three proposed techniques, performance gain increases as \( m \) increases. The reason for this is due to the decrease in number of bits XORed to obtain each bit in the hash result when \( m \) increases. As more bits are XORed, it becomes less likely to lead to meritorious gain from each additional bit XORed, with some even creating detrimental effect. These gains clearly show the benefit from the proposed preprocess. For example, when \( m = 14 \), the expected maximal search length is reduced from 21 to about 15 with \( d\)-IOX, to about 12 with \( d\)-SOX, and to 9 with \( d\)-NFD. This translates to a saving of search time by up to 58%. In general, the \( d\)-SOX hashing can achieve an additional improvement of 3% to 17% in MSL than \( d\)-IOX due to the snake-order wrapping. By adopting a more natural folding process versus the \( d\)-SOX, the \( d\)-NFD posts an additional gain of 3% to 12% in MSL. Note that part of performance gain actually should be attributed to the duplication process. When compared against the supposedly much superior CRC-32 and RS hash functions, the proposed folding algorithms deliver very comparable performance numbers, especially from the \( d\)-NFD, which shows a drop no more than 10% in performance gain.

To further verify the effect of reducing \( d \) values on the overall performance, Fig. 13 shows the average \( d \) value of each final hash bit for \( m = 14 \). The regular group-XOR technique leads to an obvious two-step distribution due to the different number of bits XORed for final hash bits between the two regions. For this example, when \( n = 32 \) and \( m = 14 \), the first 10 hash bits are from the XOR result of only two key bits, while the last four hash bits are from that of three key bits, resulting in a jump of \( d \) values. Since no sorting is applied, all bits using the same number of bits for XORing lead to the same expected \( d \) value, indicated by the relatively flat curve within each of two steps. On the other hand, after the sorting is applied as in the \( d\)-IOX, \( d\)-SOX, and \( d\)-NFD, \( d \) value distribution all follows an increasing pattern, obviously dominated by the first \( m \)-bit base, the first \( m \) bits of smallest \( d \) values also in increasing order. The \( d\)-IOX shows the sharpest increase due to its “imbalanced” XORing pattern, i.e., favoring the leading bits where \( d \) values are smaller. Compared to the \( d\)-IOX, the \( d\)-SOX is more balanced in \( d \) values attained, thus leading to an overall better performance, although it has slightly larger \( d \) values in the beginning section (the first seven bits in the example). The \( d\)-NFD in...
general reduces the overall $d$ values from the group-XOR by up to six times, $d$-IOX by up to three times, and from $d$-SOX by up to two times. The combination of a more balanced folding and the extra duplication provides a much smoother increase and overall reduced values in $d$.

### B. Real IP-Address Data Set

Simulation is also performed on a collection of real IP addresses gathered from three different sources:
- general IP traffic addresses;
- ad/spam IP addresses;
- P2P IP addresses.

The general IP traffic addresses are collected from packets entering a local network router in a duration of a few hours, while the ad/spam and P2P IP addresses are gathered from the IP filtering open source software project PeerGuardian [22]. For the simulation on these data sets, $2^m$ IP addresses were randomly taken from the trace and then used as a database to perform the hashing. Results are obtained by averaging those from 1000 runs.

Fig. 14 shows the $d$ values of all bits in a sample data set thus collected. Note that the distribution of $d$ values in the real IP-address data set is very different from that of the one randomly generated as shown in Fig. 11. The one randomly generated shows a wide range of $d$ values, while the IP-address one displays a dense cluster of $d$ values in the low-value range and a few scattered ones in higher ranges, though in general much smaller than the values from the randomly generated one. This discrepancy eventually plays an important role in deciding the effectiveness of different hashing techniques. Note that, in this distribution chart, the IP address bit positions are displayed in network address order, with bit 0 being the LSB of the IP address and bit 31 the MSB of it. The reason for the high $d$ value spikes in bit positions 29 and 30 is due to the classification of the IP addresses with these bits (and bit 31). Bit 29 has the highest $d$ value, indicating a large portion of these IP addresses being Class C. Bit 31 has a relatively small $d$ value, which in turn shows that there are also a good portion of Class A addresses in the database.

Performance comparison in MSL and ASL among techniques for each of the three IP-address data sets is given in Figs. 15–17, respectively. Tables II–IV each display a summary of performance gains correspondingly. All three proposed XOR hashing approaches post significant gains over the traditional group-XOR approach, ranging from 35% to 71% in reduction in MSL when $m_m$ is relatively large. Among the three IP-address data sets, the ad/spam one leads to the highest gain numbers in both the MSL and ASL. Although the P2P IP address data set shows a different trend of performance gain in terms of varying
it posts the most consistent spectrum of gains throughout the range of $m$.

Note that there is very little difference in performance among the three proposed techniques, $d$-IOX, $d$-SOX, and $d$-NFD, usually no more than 1% of difference. Such a scenario can be explained by looking into the $d$-value spectrum of the IP-address data set in Fig. 14. Since there are only a small group of bit vectors with significantly large $d$ values, while all the others are relatively similar in their small $d$ values, the order of these bit vectors for XORing (through our sorting preprocess) still matters to ensure the larger ones get XORed with smaller ones. However, more sophisticated folding approaches (as in $d$-SOX and $d$-NFD) will not lead to any more significant improvement in performance. Again, when compared to the CRC and RS hash under this real IP simulation setup, $d$-IOX, $d$-SOX, and $d$-NFD functions all produce very comparable performance improvements over the group-XOR, trailing by no more than 2% in improvement. This result clearly shows that the proposed folding-XOR hashing algorithms, under the real-world scenario, can compete with the much more expensive and longer processing algorithms.

To further analyze potential performance difference between the $d$-value XOR folding algorithms and the well-established CRC and RS hashing algorithms, the $\chi^2$ analysis is conducted. Tables V–VII show the $\chi^2$ results with each of the three real IP address database for all hashing algorithms that are evaluated. Less than 2% of difference in general separating the proposed algorithms and the reference algorithms again verifies our claim in the effectiveness of our designs without resorting to using more costly hardware or lengthy computation processes.

V. IMPLEMENTATION

In order to implement the desired hashing function in hardware to allow fast search and comparison, we propose a simple design allowing the needed mapping flexibility from the original $m$-bit key to the final $d$-bit hash value. Fig. 19 shows a block diagram of the design, where each bit from the input is connected to a 1-to-$m$ demultiplexer (DMUX) selected by a
The above simple design, however, cannot be used to implement the $d$-NFD technique due to the extra potential duplication. A modification is made to incorporate such an expansion. Fig. 21 gives an illustration of a sample design for the case of $d$-SOX. Note that if a bit-extraction or partial key hashing is used, then the bit that is not used will have its DMUX “disabled.”

The wide applicability of the proposed methodology is clearly demonstrated by our simulation results. Other potential applications include general database query, string matching, etc. There are still many potential extensions along this line of research, including:

1) Correlation among bit vectors: When there exists correlation among bit vectors, as most real data sets exhibit, the best hashing produced by relying solely on the $d$ values may turn out to be far from the best. How to come up with a standard and reliable calibration mechanism to quantify the correlation between the two bit vectors XORed and derive a function to decide whether or not the intended XOR is beneficial remains a challenge for this research.

VI. CONCLUSION

This paper has outlined several new ideas to design efficient hashing functions for nonuniformly distributed IP address lookup. With minor overhead in terms of preprocessing, substantial performance gain can be attained over traditional standard hashing techniques. The wide applicability of the proposed methodology is clearly demonstrated by our simulation results. Other potential applications include general database query, string matching, etc. There are still many potential extensions along this line of research, including:

example of the resulting $RSV$ values, given that the sorted sequence is provided as shown.

Note that in an IP address filter where IP addresses can be inserted and deleted dynamically, the database has to be resorted (offline) before all entries are mapped to the $2^m$ memory locations. Only the $RSV$ needs to be changed accordingly to reflect the new sorting result. The processing time for this step, including the sorting, is very small compared to the time interval between successive modifications to the database.
2) Effect of bit reusing (duplication): Obviously, the proposed technique has taken a positive step toward exploiting the benefit of bit duplication. Once bits are reused, correlation between bit positions sharing the bit(s) arises. How to decide on whether the ensuing correlation is too big an investment to compensate from the gain through using the misleadingly smaller $d$ values remains a challenge for this research.

By providing initial groundwork for nonuniform distribution hashing, this paper has pointed out the potential areas to improve hashing algorithm and new ways to exploit specific characteristics of the target database.

REFERENCES


Christopher J. Martinez received the B.S. degree in electrical engineering from the University of Texas at Austin in 2003, and the M.S. and Ph.D. degrees in electrical engineering from the University of Texas at San Antonio in 2005 and 2008, respectively. Currently he is with the Department of Electrical and Computer Engineering, University of New Haven, West Haven, CT. His areas of interest include computer architecture, computer networks and computer programming.

Dr. Martinez has received a Best Paper Award from the 3rd International Conference on Cybernetics and Information Technologies, Systems, and Applications.

Devang K. Pandya received the Diploma in industrial electronics from Vivekananda Education Society’s Polytechnic, Mumbai, India, in 1997, the Bachelor’s degree in electronics engineering from Mumbai University, Mumbai, India, in 2000, and the M.S. degree in electrical engineering from the University of Texas at San Antonio in 2006.

He was with Meco Instruments Pvt. Ltd. as a Testing Engineer in the Research and Development Department until the end of Summer 2003. He is currently with Dibon Solutions Inc., Newark, NJ, in the field of embedded system and imaging product development. His research interests include computer architecture, computer networks, and Internet security.

Wei-Ming Lin received the B.S. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, in 1982, and the M.S. and Ph.D. degrees in electrical engineering from the University of Southern California, Los Angeles, in 1986 and 1991, respectively.

He was an Assistant Professor in the Department of Electrical and Computer Engineering, Mississippi State University, Mississippi State, before joining the University of Texas at San Antonio (UTSA) in 1993.

He has been a Professor of Electrical Engineering at UTSA since 2004, and also the Associate Dean for Graduate Studies in the College of Engineering since 2006. His research interests are primarily in the area of distributed and parallel computing, computer architecture, computer networks, and Internet security.

Prof. Lin has served in program committees for many international conferences and also as the Program Chair for the International Conference on Computer Applications in Industry and Engineering 2004. He received three Best Paper Awards from the 1995 International Phoenix Conference on Computer and Communications, the Fifth International Conference on Intelligent Systems, and the 3rd International Conference on Cybernetics and Information Technologies, Systems, and Applications.