HASHING FOR FAST IP ADDRESS LOOKUP UTILIZING INTER-KEY CORRELATION

APPROVED BY SUPERVISING COMMITTEE:

Wei-Ming Lin, Ph.D., Chair

Bao Liu, Ph.D.

Ram Krishnan, Ph.D.

Accepted:

Dean, Graduate School

DEDICATION

This thesis is dedicated to my beloved parents for their love, encouragement and endless support.

HASHING FOR FAST IP ADDRESS LOOKUP UTILIZING INTER-KEY CORRELATION

by

RAMIN SAHBA, M.Sc.

DISSERTATION
Presented to the Graduate Faculty of
The University of Texas at San Antonio
In Partial Fulfillment
Of the Requirements
For the Degree of

MASTER OF SIENCE IN ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT SAN ANTONIO
College of Engineering
Department of Electrical and Computer Engineering
December 2013

ACKNOWLEDGEMENTS

I would like to thank my supervising professor Dr. Wei-Ming Lin who kindly let me work under his supervision. Dr. Lin helped me to learn more subjects in computer architecture and network security, and his great advises made me a better researcher. I am thankful to research committee Dr. Eugene John, Dr. Bao Liu and Dr. Ram Krishnan for reading and evaluating my thesis.

December 2013

HASHING FOR FAST IP ADDRESS LOOKUP UTILIZING INTER-KEY CORRELATION

Ramin Sahba, M.Sc.
The University of Texas at San Antonio, 2013

Supervising Professor: Wei-Ming Lin, Ph.D., Chair

Hashing algorithms long have been widely adopted to design a fast IP address look-up process which involves a search through a large database to find a record associated with a given key. When the target database has a highly skewed distribution as demonstrated by most of the real-time IP databases, performance delivered by regular hashing algorithms, such as the bit-extraction, bit-wise XOR, etc., usually becomes far from desirable. This research aims at designing a very simple bit-extraction hashing algorithm by exploiting the correlation among non-uniformly distributed key-bits in the database in order to minimize the search time. Application scope of the proposed methodology reaches beyond typical database systems. General computer network routing, high-speed Hardware-based Intrusion Detection Systems (HIDS) are among significant others that could benefit from this result. Pattern matching, identification recognition and all sorts of queries for general database systems would also see their required processing time drastically curtailed.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1: INTRODUCTION

The computer and communication networks community has seen extensive advancement in its research and commercial fields in the past few decades. To further increase the speed of networks takes more than just physical advances in transmission speed through a given medium. One hindrance that all network components, such as routers, firewalls, intrusion detection, and others, have faced and suffered from it more as the network size grows is from the required search and lookup process through a large address space. Fast address lookup or identification matching has become critical to the feasibility of many modern 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 that the routers in wide-area networks have to look through a large database, a routing table, for a forwarding link that matches the given destination address. Another example that calls for imminent attention these days is in the area of internet security, in which intrusion detection demands rapid evaluation of client requests. In this, rules are established to allow the intrusion detection system to check for wrong-doing. Usually, packet headers need to be matched quickly in real time with rule database. Such a matching process usually is carried out through a hashing process to reduce the otherwise potentially excessively long search 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. In many various IP databases, there exists a high degree of similarities among groups of IP addresses. For example, class- C IP addresses for all clients in a typical local area network, i.e. in the same subnet, will have the same network prefix (subnet address) which is 24 bits out of the 32 address bits in IPv4. Class-A and class-B addresses in the same subnet have the

same property except with a smaller network prefix. Such a scenario is very common in a typical database for intrusion detection in which groups of IP addresses on the blacklist may share the same network prefixes. Other similar duplicities exist in a typical router database, e.g., the forwarding table, in which the packet headers from different packets usually contain similar or even identical byte(s). Relying on a generic search algorithm or a hashing function without taking this duplicity information into consideration would definitely compromise its potential effectiveness and/or efficiency. This research proposes a methodology following which a hashing algorithm can be designed to lead to a more uniformly distributed hash result from a non-uniform database than a generic hashing algorithm. An analytical pre-process on the original database is first performed to extract critical information that would significantly benefit the design of a better hashing algorithm. This process includes identifying correlation among key bits and a simple sorting to facilitate the eventual bit extraction hashing algorithm. A significant improvement from our simulation results has been obtained on typical data sets.

1.1 Goal and Contributions

The goal of this research is to propose a methodology following which a hashing algorithm can be designed to lead to a more uniformly distributed hash result from a non-uniform database than a generic hashing algorithm.

1.2 Organization of Thesis

The rest of the report is organized in the following way. Chapter 2 describes related work. Hashing for address lookup is introduced in Chapter 3 and it briefly describes using a hash function to generate hash keys and for address lookup. Chapter 4 shows Simulation setup. Chapter 5 describes d-EXT extraction algorithm. Chapter 6 explains the proposed hashing algorithm and provides analytical simulations and finally, the report is concluded with summarizing remarks and future work.

Chapter 2: RELATED WORK

There have been many schemes developed for IP address lookup problem using a hash function. A complete survey and complexity analysis on IP address lookup algorithms has been provided in [15]. A performance comparison of traditional XOR (exclusive OR) folding, bit extraction, CRC-based hash functions is given in [6]. 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 proved to be excellent means but have some potential shortcomings. Compared to a simple XOR folding hash algorithm which 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 [10], [12], while others are software based which improve their search performance mainly through efficient data structures [13], [18]. Waldvogel et al. In [19] an address lookup scheme is proposed based on a binary search of 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], [5],[14], [21]. 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 [17] 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 algorithm such as MD4, MD5, SHA-1 and SHA-2 algorithms which have found use in the

cryptography field [3], [16]. 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.

Chapter 3: HASHING FOR ADDRESS LOOKUP

3.1 Hashing

A hash function, usually a mathematical one, maps a number with a large value range into another number with a smaller range [4]. Figure 3.1 shows a simple example, in which a database of eight given records (e.g., addresses in our application) are to be matched against for any incoming record. 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. Due to the large size of each record and a 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 hash value (a three 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 value 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 another 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. a simple bit-extraction, grouped XOR, 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 research is to develop a universal hashing methodology applicable to non-uniformly distributed data sets.

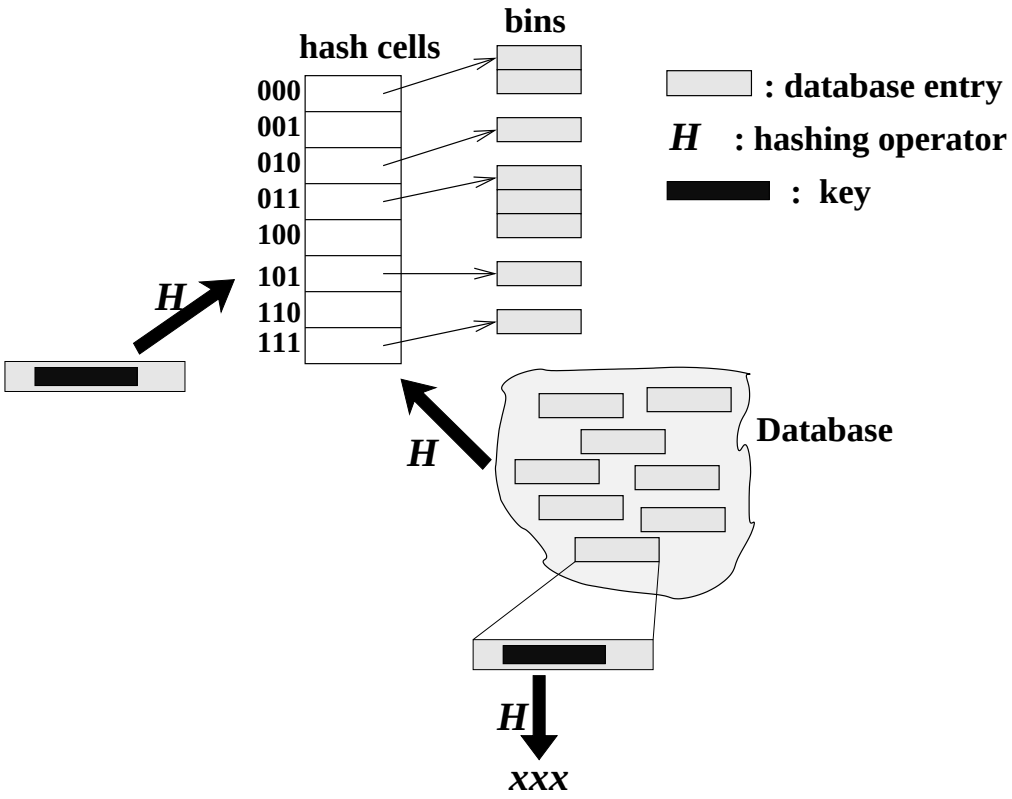


Figure 3.1: An Example of Hashing

3.2 Simulation Setup

Our simulation is devoted to comparing among techniques on several practical performance indicators. A system is assumed to have a database of 2^m 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 Figure 3.1). Thus, each of the 2^m 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.

Chapter 4: D-EXT EXTRACTION ALGORITHM

4.1 Extraction Algorithm

The simplest hashing process is to simply extract (select) the necessary m hash bits directly from the n key bits. As aforementioned, if all bit vectors (see definition and illustration of a bit vector in Figure 4.1) are uniformly distributed, then such a extraction process would always lead a probabilistically identical hash performance regardless which m bits are chosen. However, if there exists any inherently non-zero "correlation" among bit vectors, the selection of hash bits would make a difference in the resulted hash performance.

The d-EXT method proposed in [4] is a simple and efficient way to generate hash keys especially for databases of non-uniformly distributed IP addresses. This algorithm is briefly described here for the sake of completeness and as a basis for our design and performance comparison.

4.2 Preprocessing of Database

As aforementioned, the database is defined as consisting of $M = 2^m$ entries with each entry having n bits in length. It can also be viewed as having n M-bit vectors labelled 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 Figure 4.1(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 then will provide the best distribution of the target records into the hash cells. To quantify the randomness of each bit vector in the original database, the d value was first introduced in [4], defined as d_i for bit vector V_i, which is the absolute difference between the number of 0's and 1's in V_i across the data set. Figure 4.1(a) displays the d values thus determined

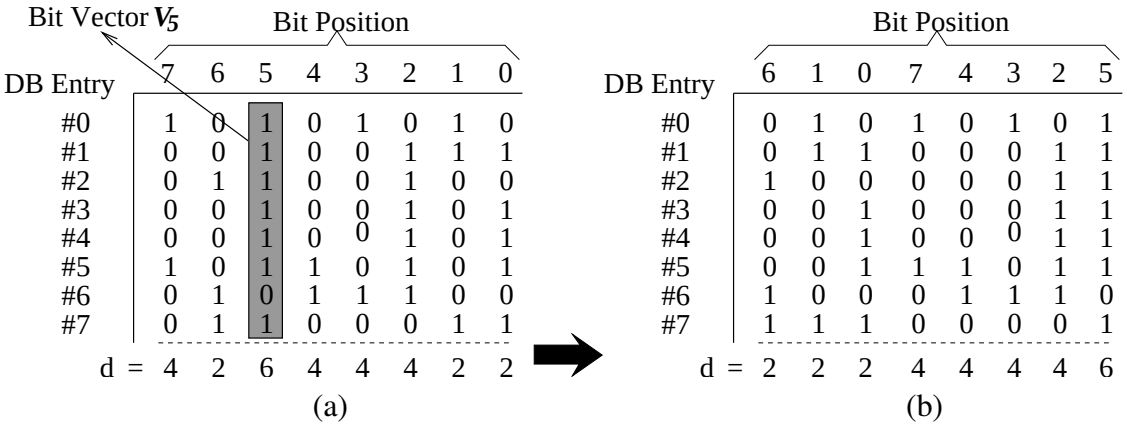


Figure 4.1: (a) Calculation of d Values (b) Sorted Database by d Value

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 to the employment of a simple pre-processing step to re-arrange the n bit vectors according to their d values sorted into a non-decreasing order as shown in Figure 4.1(b). This sorted sequence then gives us an "order of significance" according to which each bit should be utilized.

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 non-preprocessed 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

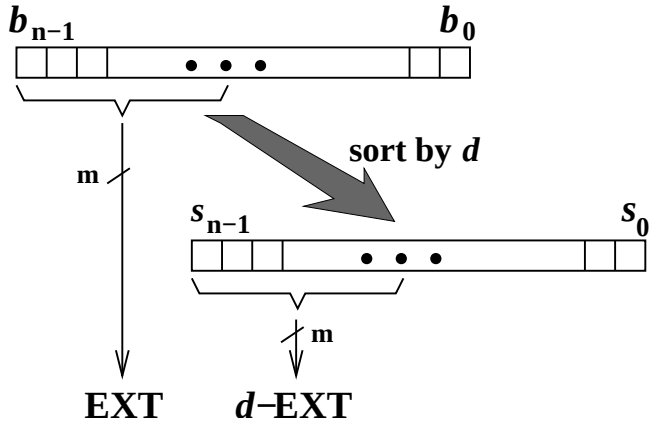
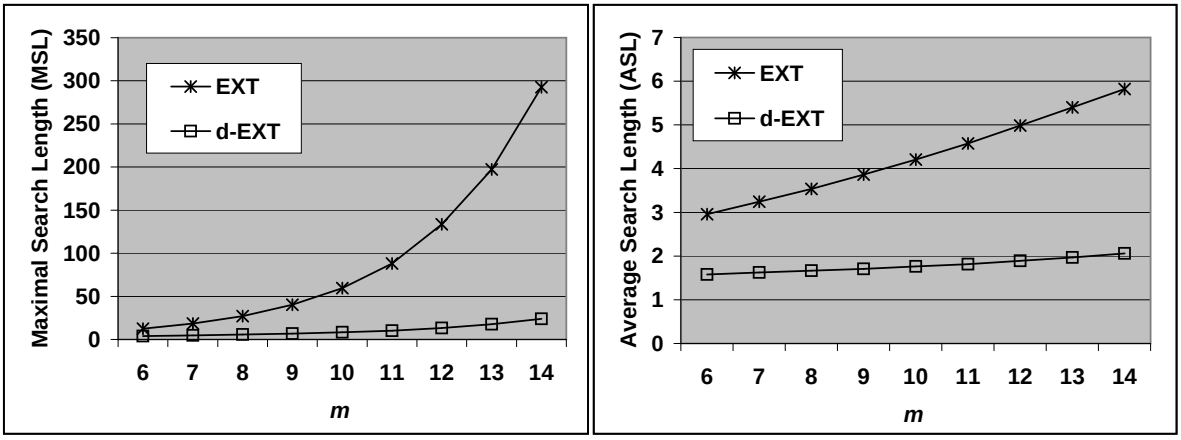


Figure 4.2: Bit Extraction Hashing: EXT vs. d-EXT

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 afterwards is $(s_{n-1}, s_{n-2}, \ldots, s_0)$. Bit-extraction on the non-preprocessed 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 which have the m smallest d values become obvious candidates for bit extraction [4]. That is, the final hash value is $(s_{n-1}, s_{n-2}, \ldots, s_{n-m})$. Figure 4.2 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). Figure 4.3 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 (an even number) ranging from 0 to 2^m, thus leading to a uniform distribution of d values. Afterwards, the bit vector is then generated by randomly assigning 0's and 1's to the 2^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 maximal search length is reduced from 292 to 23. This easily could lead to a hardware implementation achieving a real-time processing speed of 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.



(a) (b)

Figure 4.3: Performance Comparison for Bit-Extraction Hashing on Random Data Set: (a) MSL (b) ASL

Chapter 5: PROPOSED HASHING ALGORITHM

5.1 PPDV Algorithm

Note that, although the d-EXT clearly demonstrates a very significant performance gain over the simple extraction technique (EXT), the algorithm is based on the assumption that there is no "correlation" between any two bit vectors. That is, a pair of two bit vectors with their d values both equal to 0 should always be expected to form bit pairs with a uniform distribution among the four combination patterns: 00, 01, 10 and 11. For example, shown in Figure 5.1, two bit vectors x and y in an 8-entry database show four differenct levels of correlation between the two vectors. Note that the term "correlation" used here is not strictly correct in the theory of statistics, but instead is adopted to represent the degree of "pattern distribution variation" (PDV) between the two composing bits. Case (a) shows the least amount of correlation where all four combination patterns each having exactly two occurrences, leading to a zero variation in the distribution of the four patterns. Case (b) has a higher correlation with two patterns having 3 occurrences and the others with 1. Case (c) shows an even higher correlation with only two patterns and (d) shows the highest correlation with only one pattern throughout.

Out of these four cases, (a), (b) and (c) all have both vectors x and y with a zero d value. According to the selection criteria in the d-EXT technique, both x and y should be among the key bits to use (select) for the bit-extracting process hoping to lead to a perfectly distributed search space with respect to the space composed by x and y, with each of the four patterns occupying exactly $1/4$ of the search space. However, only case (a) produces such a result, with (b) and (c) both leading to a non-ideal search space partitioning which in turns causes more collisions to occur than expected in the hashing process. Figure 5.2 shows a complete hashing example where there are $M = 8$ entries in the database. Thus three ($m = \log_2 M = 3$) hash bits will be extracted from the n key bits ($n > m$). An ideal scenario from applying the d-EXT technique is shown in Figure 5.2(a), where the first three bits (all with a zero d value) are used (extracted) as the hash

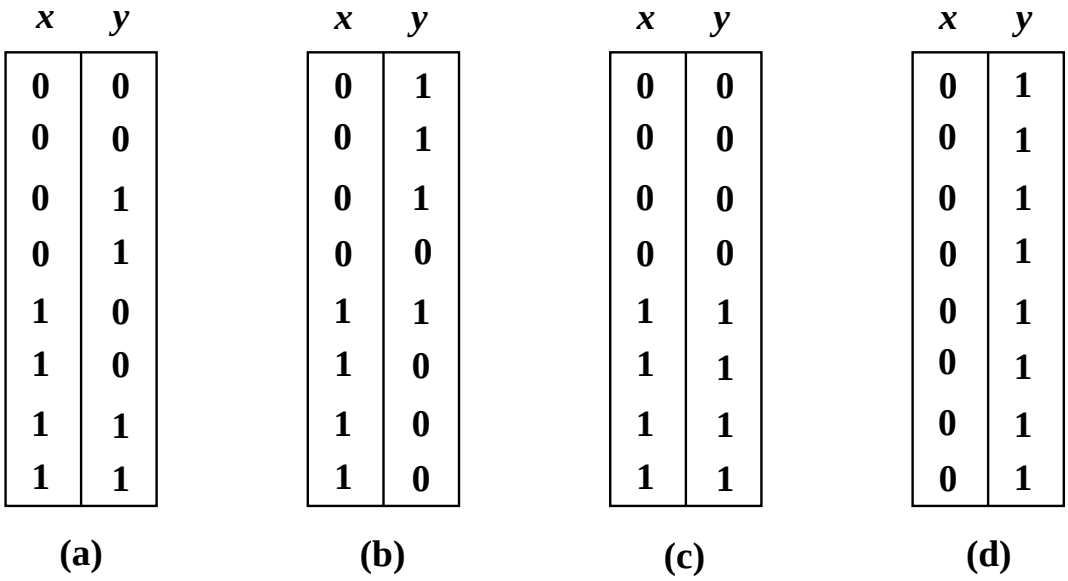


Figure 5.1: Examples of Correlation between Two Bit Vectors

bits. One can see that with these three bits the hash value is perfectly distributed leading to the best search space. In this case, the maximum search length (MSL) is 1 and the average maximum search length (ASL) is also 1. However, a similar extraction process may lead to the cases in (b) or (c), where the three selected bits are with zero d value but their hash value distribution is far from ideal, each leading to a search space heavily skewed toward a few of the eight patterns. In case (b), MSL$= 3$ and ASL$= 2$, and in case (c) MSL$= 4$ and ASL$= 4$, both having a much worse hash conflict and longer search time. This clearly shows that an extraction technique cannot rely on solely the d value for hash bit selection.

The underlying concept of the proposed hashing algorithm is to apply a simple pre-process to identify a "sufficient" amount of correlation information to facilitate the design of an effective bit extraction hashing algorithm. We elect to adopt the concept directly from the example demonstrated in Figure 5.1 in evaluating the PDV of every pair of two bits, and thus this proposed technique is to be referred to as the "Pair-wise Pattern Distribution Variation" (PPDV) algorithm. First of all, in order to give a proper quantification of the PDV, we choose to use the standard deviation value of the distribution among the four patterns between the two bit vectors. Let w_{xy} denote the number of occurrences for each of the four possible pairs, 00, 01, 10 and 11 which xy

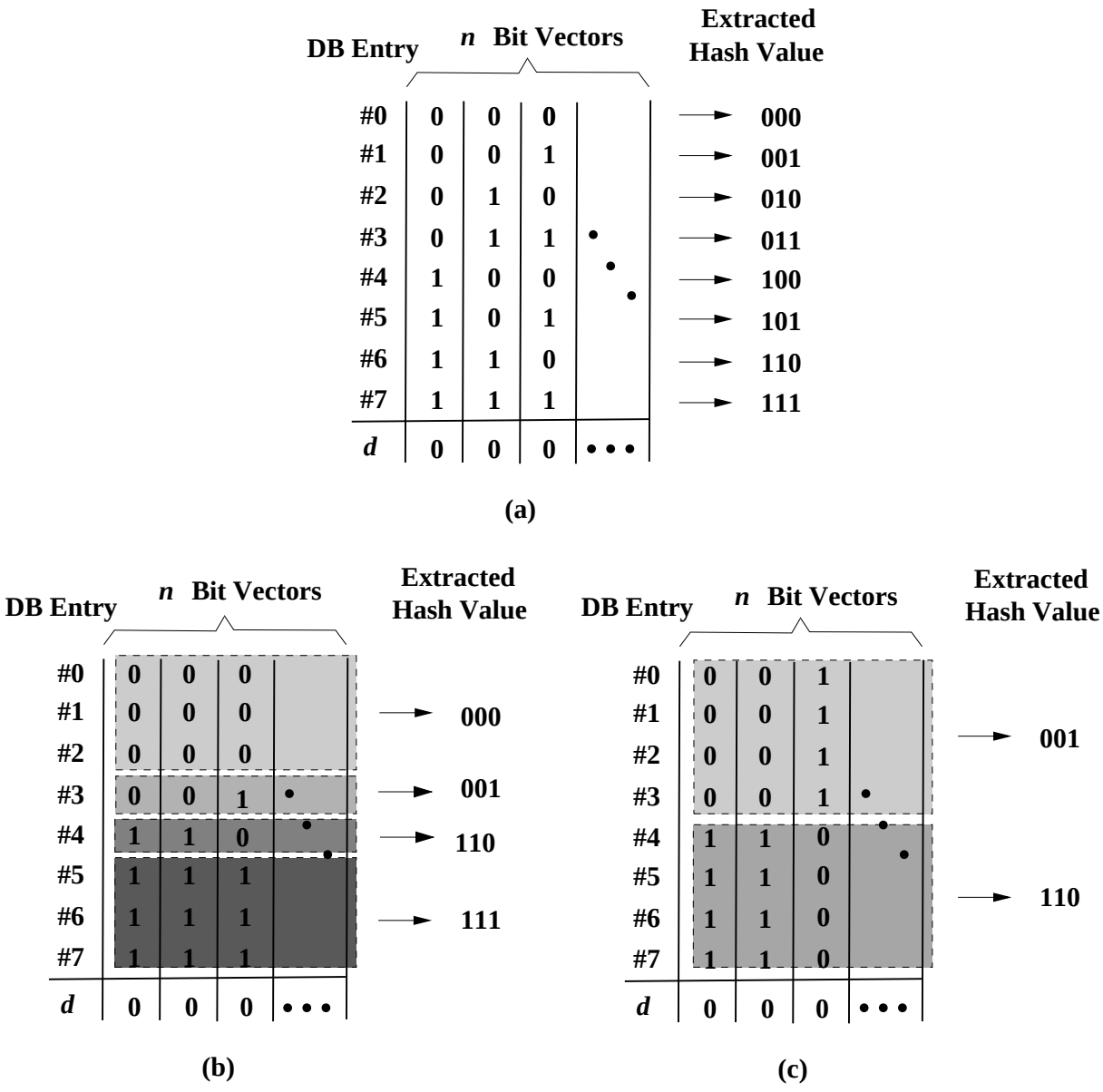


Figure 5.2: Examples of Hashing with d-EXT

takes. Since each bit vector has 2^m elements, we have

$$\sum_{xy \in \{00,01,10,11\}} w_{xy} = 2^m \tag{5.1}$$

The PDV of the corresponding pair of bit vectors is then

$$\begin{aligned} \text{PDV} &= \sqrt{\tfrac{1}{4} \textstyle\sum_{xy \in \{00,01,10,11\}} (w_{xy} - \mu_{\text{PDV}})^2} \\ \text{where } \ \mu_{\text{PDV}} &= \tfrac{2^m}{4} \end{aligned} \tag{5.2}$$

For the examples shown in Figure 5.1 where $2^m = 8$ and $\mu_{\text{PDV}} = 2$, case (a) has a PDV of 0 indicating the best distribution provided by the two bits. In case (b), $(w_{00}, w_{01}, w_{10}, w_{11}) = (1, 3, 3, 1)$, thus its PDV is 1. The PDVs of cases (c) and (d) are 2 and $2\sqrt{3}$, respectively. Obviously, the higher the PDV is between the two bits, it becomes less "meaningful" to include both bits in the hash bits. When all 2^m patterns are identical (as in case (d)), one of the two bits is completely redundant when they both are used for hashing.In order to establish a threshold of PDV for hash bit selection, we use the maximum of PDV as a reference which corresponds to the case where all 2^m patterns are identical and can be derived as

$$\text{PDV}_{\text{M}} = \sqrt{\tfrac{1}{4}(3(0 - \mu_{\text{PDV}})^2 + (2^m - \mu_{\text{PDV}})^2)} = \sqrt{3} \cdot \mu_{\text{PDV}} \tag{5.3}$$

For the same examples in Figure 5.1, the maximum PDV is $\text{PDV}_{\text{M}} = 2\sqrt{3}$.Translated to be a fraction of PDV_{M}, there is a 0% "correlation" in case (a), 28.9% in case (b), and 57.7% in (c) and 100% in (d).

Figure 5.3 shows a concise description of the proposed PPDV algorithm. The first step (lines 1-2) is to find the d values of all the key bits and then sort them into an array in ascending order, denoted as Array_D, as the basis for hash bit extraction. Similar to the d-EXT algorithm, the goal is to extract the first m bits out of this array except that an additional elimination process is

PPDV() {
1. find $d(i)$ for each bit i, $0 \leq i \leq n-1$;
2. sort all $d(i)$'s in ascending order to
   Array_D={b_0,b_1, ..., b_{n-1}};

3. find PDV for each bit-pair $p_i = (b_j, b_k)$,
   $0 \leq j, k \leq n-1$, $j \neq k$;
4. sort all PDV's in descending order to
   Array_PDV={p_0,p_1, ..., p_{v-1}},
   $v = n(n-1)/2$, $p_i = (b_j, b_k)$ & $d(b_j) \leq d(b_k)$;

5. for ($n - m$ iterations) {
6. do {
7. let $p_i = (b_j, b_k)$ be the next element in Array_PDV;
8. } while ($b_k \notin$ Array_D(m))
   // Array_D(m): the first m elements in Array_D;

9. if PDV(p_i) > PDV_{th}
10. Array_D = Array_D - {b_k};
11. else break;
12. }
13. return(Array_D(m));
}

Figure 5.3: The PPDV Algorithm

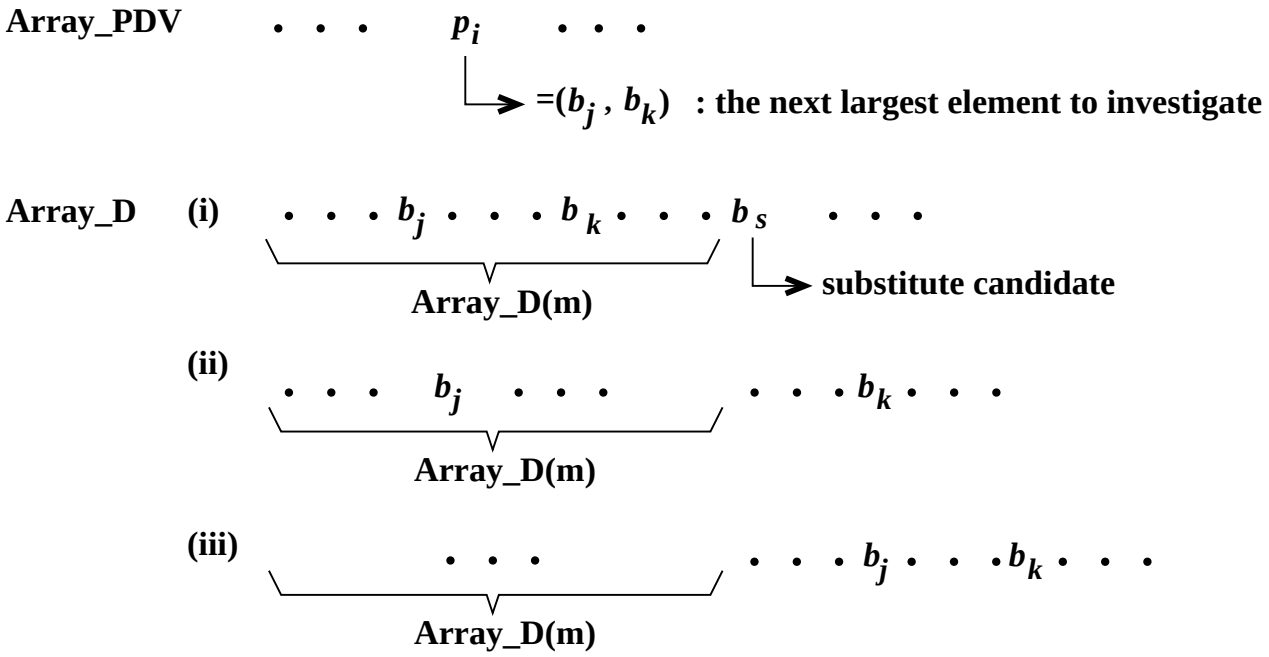


Figure 5.4: The Membership Criterion

incorporated into the algorithm using the PDV as a secondary criterium. This "extraction window" ("the first m bits" in Array_D) will continue to evolve if a bit from this window is eliminated and the bit with the next smallest d value is then included. The second step (lines 3-4) is then to find the PDV value for each pair of bits (using the formula in Equation 6.2, and sort them into an array in descending order, denoted as Array_PDV. The secondary criterium is used to eliminate a bit from the extraction window when this bit is deemed not as useful as a substitute with an even larger d value.

The elimination process is repeated for up to $n - m$ times (the "for loop" in lines 5-12) to eliminate a maximum of $n - m$ bits from the array so that final array has at least m bits for the final extraction. Each elimination will continue in Array_PDV to investigate the next bit-pair, $p_i = (b_j, b_k)$ and $d(b_j) \leq d(b_k)$ (i.e. b_j is in front of b_k in Array_D) with the largest PDV (lines 6-8). This process is shown in Figure 5.4. A suitable pair is identified when both corresponding bits are within the current extraction window (denoted as Array_D(m)), as shown in the case (i) in Figure 5.4. This is to be referred to as the "membership" criterium. That is, if both bits are outside

of the window as in case (iii), then there is no need for elimination, and, if one of two bits is outside of the window (case (ii)), then there is no issue of "redundancy" present. Whether or not the bit-pair thus identified should be considered for elimination depends on if its PDV is large enough to offset the benefit of having both bits as part of the hash bits versus replacing one with another bit of larger d value (bit b_s in case (i)). Here we choose to preset a threshold of PDV, denoted as PDV_{th}, for such a filtering process. If the identified bit-pair passes this threshold criterium, then the bit with the larger d value (bit b_k) is eliminated from the current Array_D (lines 9-10), since the bit with the smaller d value has a probabilistically better hashing capability. The underlying idea is to replace b_k with b_s (although b_s has a larger d value) hoping to compensate the loss in d value of b_k with a better PDV from b_s with others.

Once the next largest PDV value in Array_PDV is below the threshold, the whole elimination process is terminated (line 11), and the final extraction is carried out (line 12). The PDV threshold is preset using a fraction of the maximum PDV value (PDV_{M}):

$$\text{PDV}_{\text{th}} = \gamma \cdot \text{PDV}_{\text{M}} \tag{5.4}$$

where γ is the threshold percentage that may have to be adjusted according to different values of n, m and even inherent data distribution "skewness" in the database. Without further delving into more complexities, γ is set to 50% in our simulation runs.

Note that the computation time requirement for the proposed technique is still considered minimal when compared to the potential saving in the processing time searching for all the incoming queries. Versus the $O(n \log n)$ pre-processing time for the d-EXT algorithm, the proposed PPDV algorithm requires a longer pre-processing time of $O(n^2 \log n)$ in sorting the n^2 elements into the Array_PDV.

5.2 Simulation Results

To evaluate our proposed algorithm, we compare it with the d-EXT algorithm. As aforementioned, the database targeted by this study possesses a high degree of similarities among groups of bits. Thus, two types of random data sets are generated to mimic such a characteristic displayed in a typical IP address database.

- [Type-I] A database with key bits having their d values randomly (evenly) distributed
- [Type-II] A Type-I database with some identical key bits among groups of database entries

Note that the Type-I database is one that reflects some degree of the skewness in typical IP address distribution, where some bits are more skewed than others. For example, the more significant bits are more likely to be skewed than the lower bits, partly due to the subnet divisions. The uneven distribution of the three classes (A, B and C) also leads to a very significant skew in the first three bits. Depending the scope of a subnet and the number of hosts in it, the significant portion of its host ID tends to be extremely skewed as well. That is, if the number of hosts is much smaller than the assigned host ID space, the leading bits in the host ID tend to be mostly zero. A Type-II database further addresses the skewness typically demonstrated inside a local IP address database, where groups of IP entries have identical byte(s). In all our simulations, each record (database entry) is assumed to be 32 bits ($n = 32$), with the final hash result size (m) ranging from 10 to 13. MSL and ASL performance indicators are used for comparison between the two techniques.

For Type-I data sets, Figure 4.3 already shows the performance enhancement from using the d-EXT over the straightforward EXT technique. Compared to the d-EXT, the proposed PPDV algorithm displays additional benefits as shown in Figure 5.5 on MSL performance and Figure 5.6 on ASL performance. The PPDV approach outperforms the d-EXT by about 6% in MSL and about 7% in ASL. The ranges of performance gain are expected – although the PPDV takes bit-wise pattern PDV into consideration, the randomly generated databases rarely show high PDV values that satisfy the elimination threshold. That is, in most of the cases (about 80%), the PPDV delivers exactly the same m-bit hash bits as the original one from d-EXT. Note that, even there may

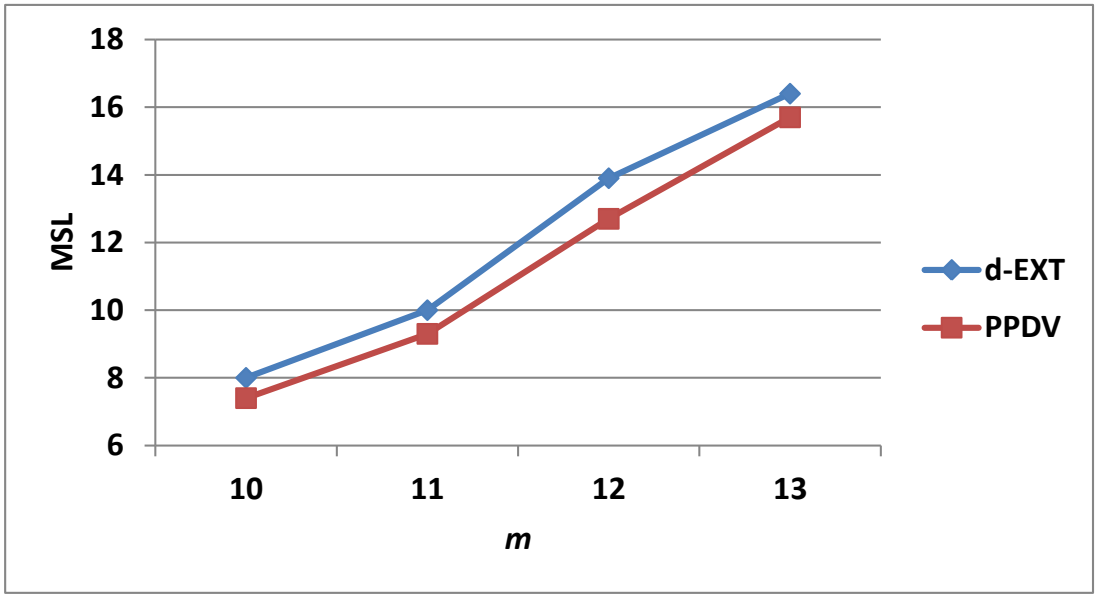


Figure 5.5: PPDV versus d-EXT on Type-I Databases: MSL

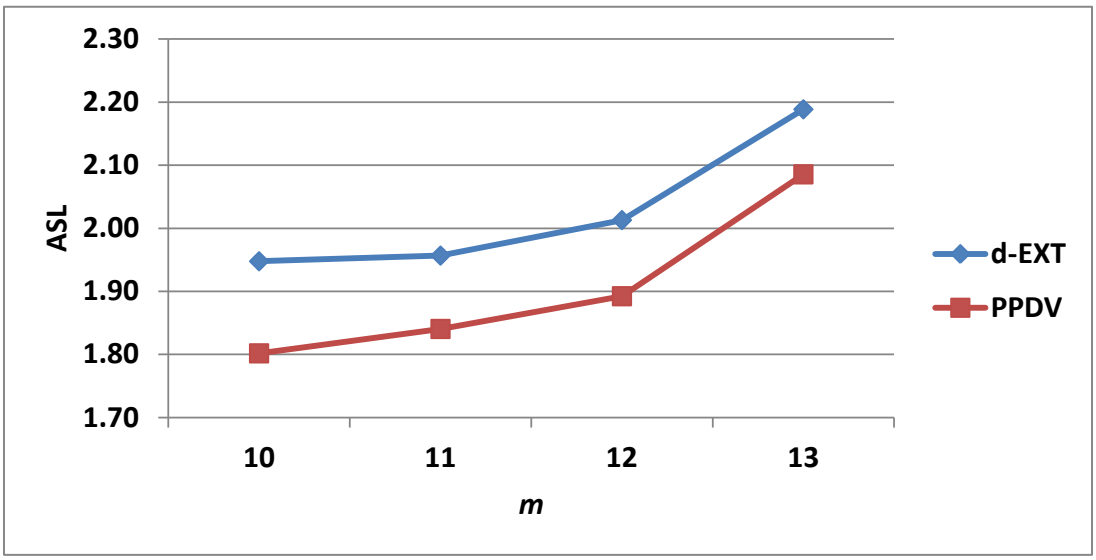


Figure 5.6: PPDV versus d-EXT on Type-I Databases: ASL

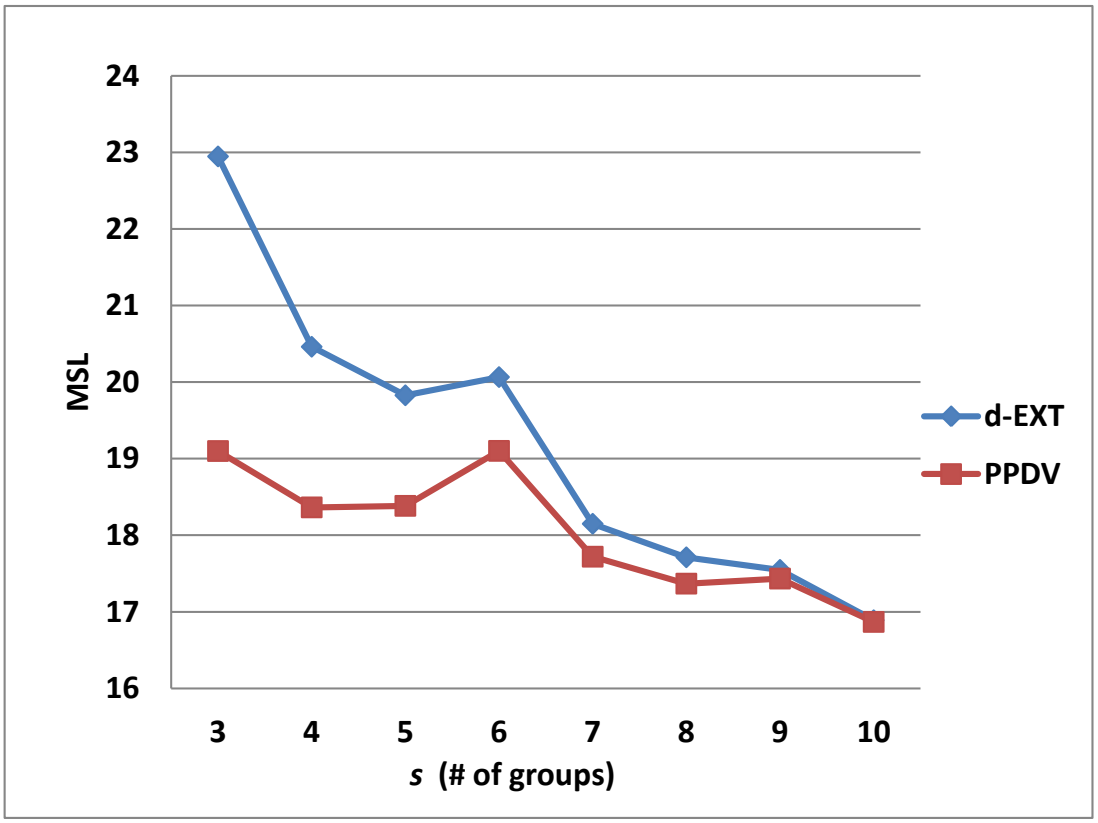


Figure 5.7: MSL Comparison between PPDV and d-EXT with $m = 13$, and $k = 4$

exist high PDV values, it is probabilistically less likely for such a value to exist between two bits of small d values, thus leading to a small likelihood for it to satisfy the "membership" criterium.

Type-II databases are generated using the following guidelines to mimic the characteristic of high PDV in a typical IP database:

- k out of the n key bits are randomly selected to be the "bits of duplication" on which a group of entries share the same value;
- the 2^m entries are randomly divided into s arbitrary-sized groups, and within each group all entries take an identical value (randomly generated) on the bits of duplication.

Different databases are generated for our simulation, with k ranging from 4 to 10 and s varying from 3 to 10. Figures 5.7–5.10 show the MSL performance comparison between the PPDV and the d-EXT. As expected, the smaller the number of groups (s) is, the gain from PPDV is larger,

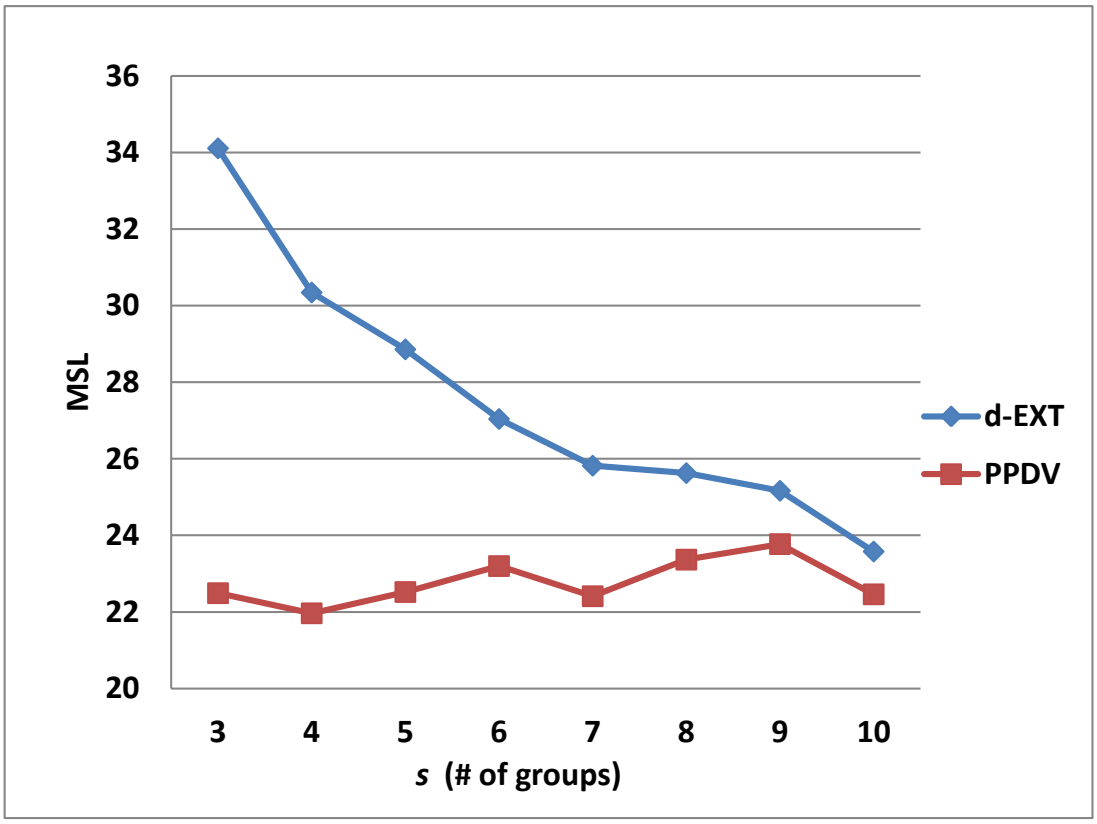


Figure 5.8: MSL Comparison between PPDV and d-EXT with $m = 13$, and $k = 6$

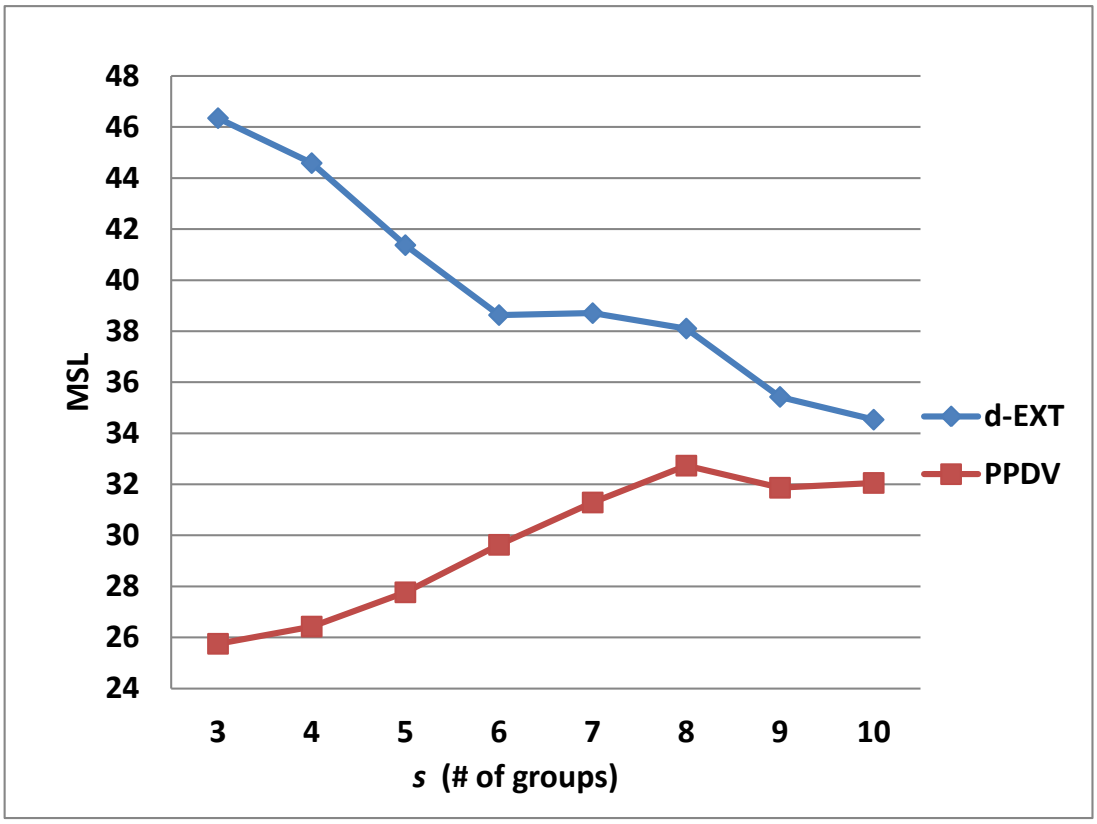


Figure 5.9: MSL Comparison between PPDV and d-EXT with $m = 13$, and $k = 8$

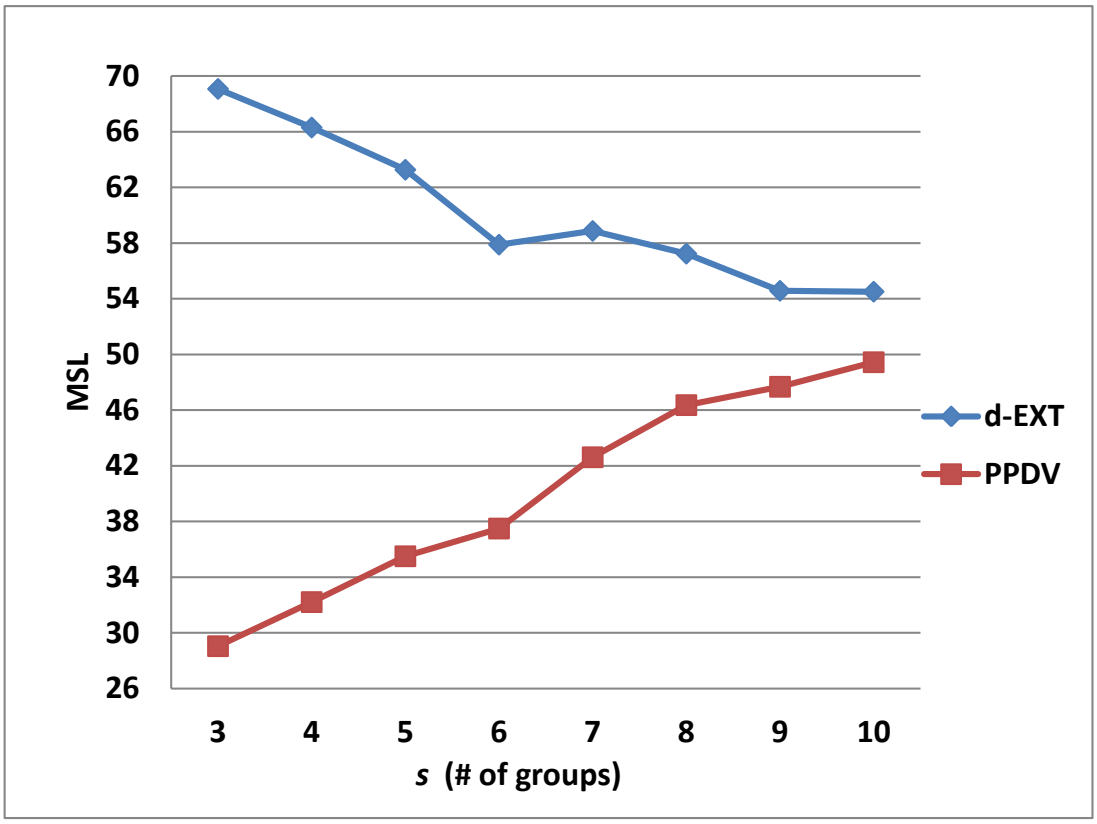


Figure 5.10: MSL Comparison between PPDV and d-EXT with $m = 13$, and $k = 10$

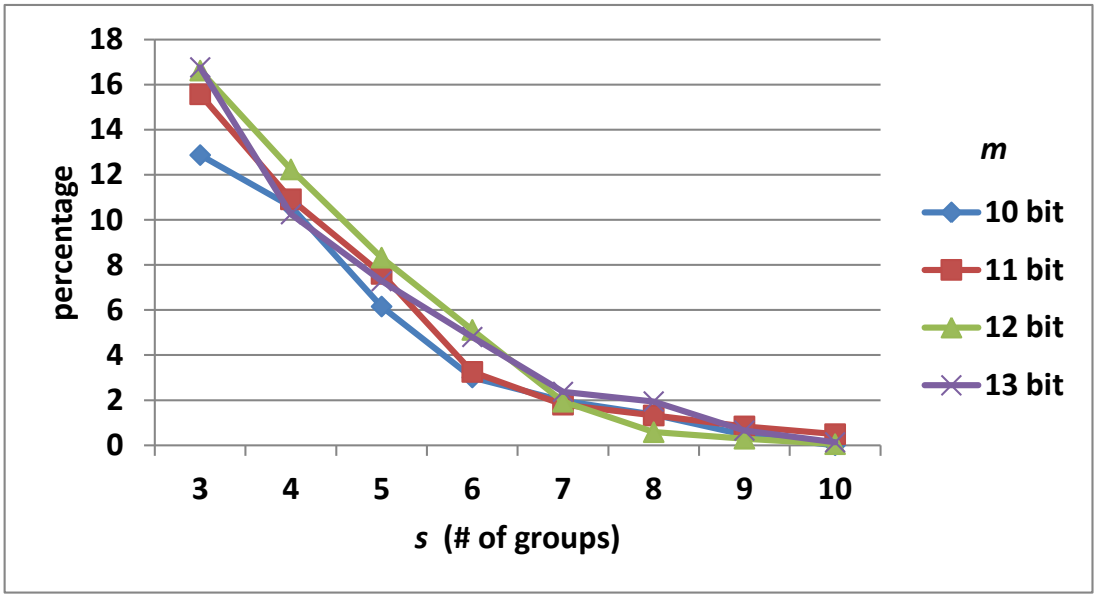


Figure 5.11: Percentage of Performance Gain of PPDV over d-EXT with m varying and $k = 4$

clearly indicating the effectiveness of the proposed bit elimination process. Also, the larger the number of bits of duplication (k) is, the gain becomes more significant.

The respective performance gain percentages from PPDV applied to cases where m is also varied are shown in Figures 5.11–5.14. Another interesting trend observed here is that, the larger the number of hash bits (m) is, the gain from PPDV is more. This can be explained by that when m is larger the more chances that bit elimination process in PPDV can be applied due a looser membership in the "membership" criterium. The improvement percentage reaches up to 58% when $m = 13$, $k = 10$ and $s = 3$. Note that in our simulations the value of k is limited to 10; however, in some realistic cases the k value can easily go up to 24 or even higher depending on the site where the database is located, where the benefits of the proposed PPDV will be even more significant.

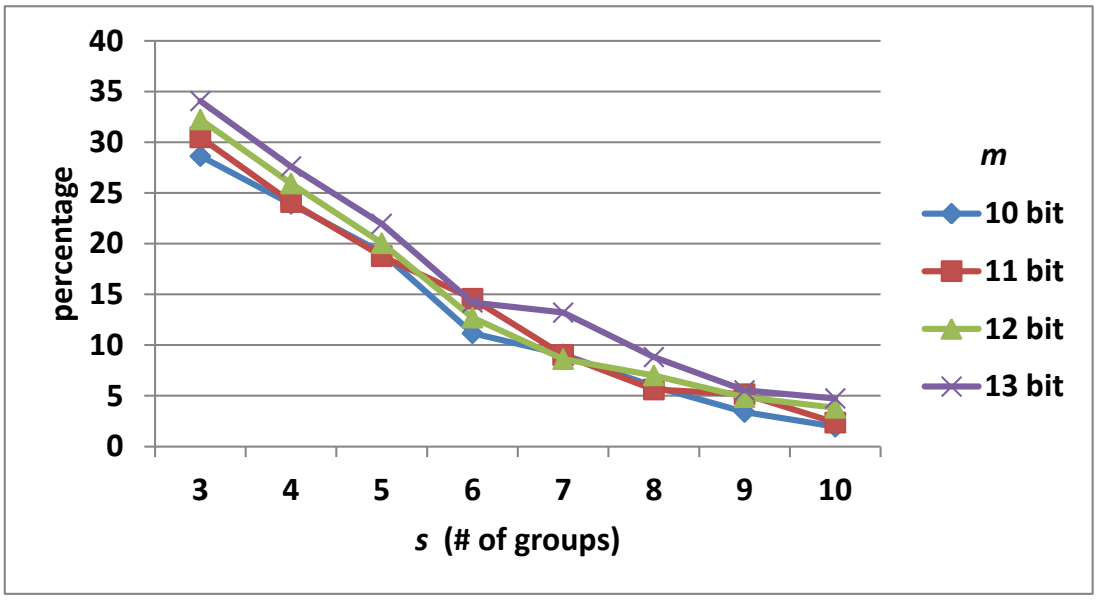


Figure 5.12: Percentage of Performance Gain of PPDV over d-EXT with m varying and $k = 6$

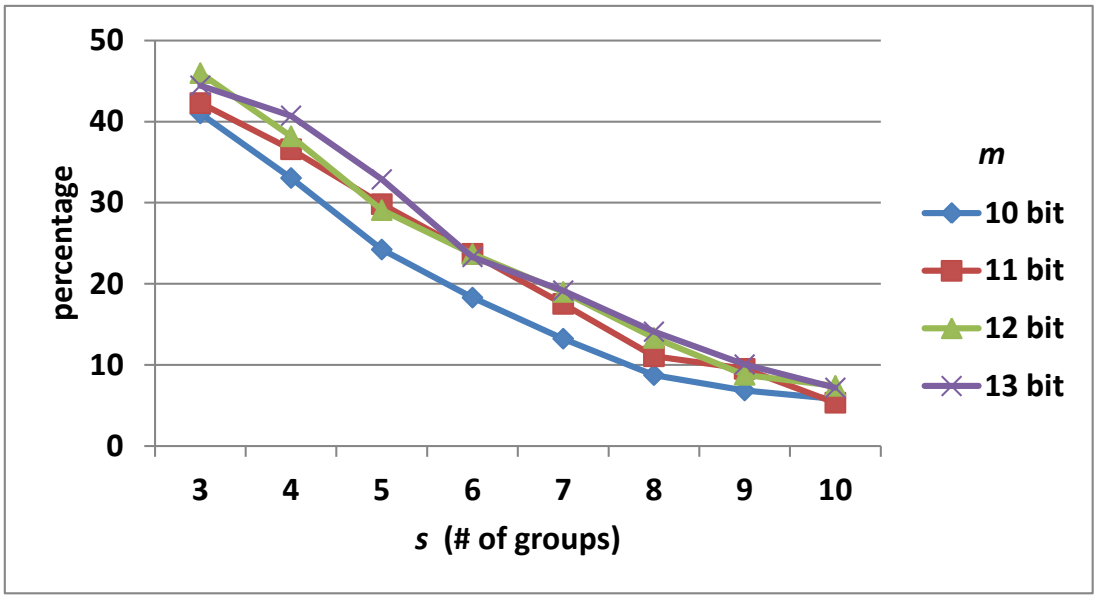


Figure 5.13: Percentage of Performance Gain of PPDV over d-EXT with m varying and $k = 8$

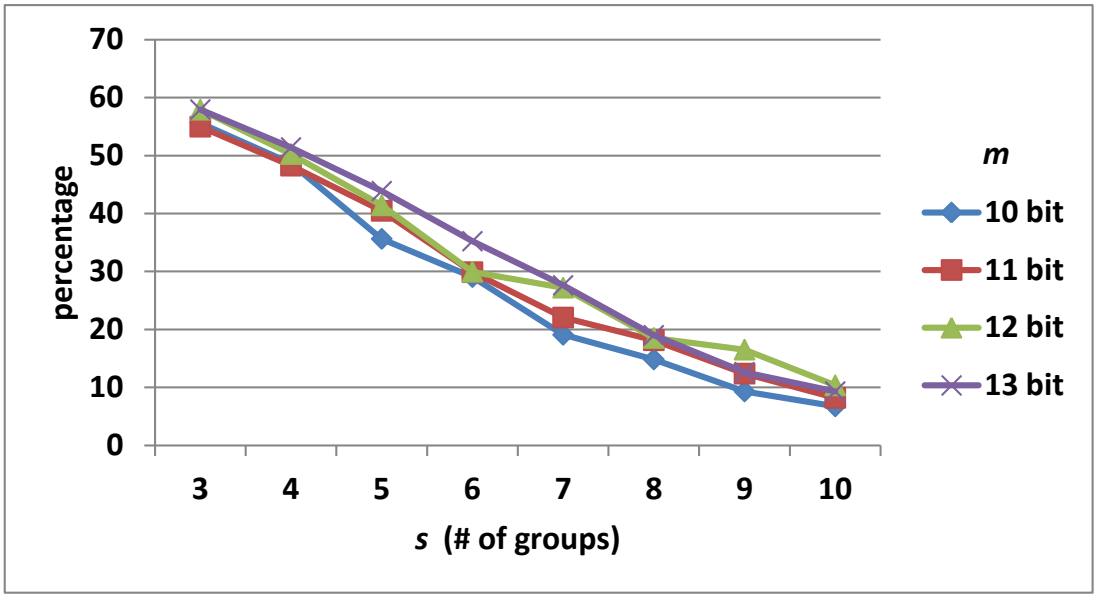


Figure 5.14: Percentage of Performance Gain of PPDV over d-EXT with m varying and $k = 10$

Chapter 6: CONCLUSION

This research shows that, without having to employ a computationally extensive process, PPDV can improve on an already-effective d-EXT process by a very significant margin. The investment of the pre-processing is minimal compared to the saving in time processing all incoming queries. Also, it is likely that, with a heavily skewed database, the PPDV technique can actually match the performance of a more complicated hashing technique, such as the ones that involve all the hash key bits in the generation of hash bits.

BIBLIOGRAPHY

[1] O. Amble and D. E. Knuth, "Ordered Hash Tables," *The Computer Journal*, 17(2):135-142, 1974.

[2] A. Broder and M. Mitzenmacher, "Using Multiple Hash Functions to Improve IP Lookups," *IEEE INFOCOM*, 2001.

[3] W.E. Burr, "Cryptographic Hash Standards: Where Do We Go from Here?" *IEEE Security and Privacy Magazine,* Vol. 4, Issue 2, pp. 88-91, March-April 2006.

[4] C. Martinez, D. Pandya, and W.-M. Lin, "On Designing Fast Non-Uniformly Distributed IP Address Lookup Hashing Algorithms," *IEEE Transactions on Networking (ICON 2006)*, Vol.17 Issue 6, pp.1916-1925, Dec. 2009.

[5] S.-H. Chung, J. Sungkee, H. Yoon, J.-W. Cho, "A Fast and Updatable IP Address Lookup Scheme," *International Conference on Computer Networks and Mobile Computing*, 2001.

[6] R. Jain, "A Comparison of Hashing Schemes for Address Lookup in Computer Networks," *IEEE Transactions on Communications,*, Vol. 40, No. 10, Oct. 1992.

[7] D. E. Knuth, "The Art of Computer Programming," *Vol 3: Sorting and Searching*, 2nd Ed., Addison-Wesley, Reading MA, 1998.

[8] C. Martinez, W.-M. Lin and P. Patel, "Optimal XOR Hashing for A Linearly Distributed Address Lookup in Computer Networks," *Symposium on Architectures for Networking and Communications Systems*, Oct., 2005, Princeton, New Jersey.

[9] C. Martinez and W.-M. Lin, "Adaptive Hashing Technique for IP Address Lookup in Computer Networks," *14th IEEE International Conference on Networks (ICON 2006)*, September 2006, Singapore.

[10] A. Moestedt, P. Sjodin, "IP Address Lookup in Hardware for High-speed Routing," *IEEE Hot Interconnects 6 symposium*, pp.31-39, August 1998, Stanford, California.

[11] P. Newman, G. Minshall, T. Lyon, and L. Hutson, "IP Switching and Gigabit Routers," *IEEE Communications Magazine*, pp.64-69, January 1997.

[12] X. Nie, D. J. Wilson, J. Cornet, G. Damm, Yiqiang Zhoa, "P Address Lookup Using A Dynamic Hash Function," *IEEE Electrical and Computer Engineering, Canadian Conference*, pp. 1646-1651, May 2005.

[13] S. Nilsson and G. Karlsson, "IP Address Lookup Using LC-Tries," *IEEE Journal on Selected Areas in Communications*, pp. 1083-1092, June 1999.

[14] D. Pao, C. Liu, L. Yeung and K. S. Chan, "Efficient Hardware Architecture for Fast IP Address Lookup," *IEEE INFOCOM*, 2002.

[15] M. A. Ruiz-Sanchez, E. W. Biersack, and W. Dabbous, "Survey and Taxonomy of IP Address Lookup Algorithms," *IEEE Network*, Vol.15, pp.8-23, Mar./Apr. 2001.

[16] A. Satoh and T. Inoue, "ASIC Hardware Focused Comparison for Hash Functions MD5, RIPEMD-160, and SHS," *International Conference on Information Technology: Coding and Computing 2005,* pp. 90-94, June 2005.

[17] R. Sedgewick, *Algorithms*, Addison-Wesley, 1988.

[18] V. Srinivasan and G. Varghese, "Faster Ip Lookups Using Controlled Prefix Expansion," *SIGMETRICS 98*, pp. 1-10, Madison, 1998.

[19] M. Waldvogel, G. Varghese, J. Turner and B. Plattner, "Scalable High Speed IP Routing Lookups," in *ACM SIGCOMM'97*, pp. 25-35, Sept. 1999.

[20] G. B. White and M. I. Huson, "A Peer-based Hardware Protocol for Intrusion Detection Systems," *Military Communications Conference (MILCOM)*, 1996.

[21] P. A. Yilmaz, A. Belenkiy, N. Uzun, N. Gogate and M. Toy, “A Trie-based Algorithm for IP Lookup Problem,” *Global Telecommunications Conference (GLOBECOM)*, 2000.

[22] D. Yu, B. Smith, and B. Wei, “Forwarding Engine for Fast Routing Lookups and Updates,” *Global Telecommunications Conference*, pp. 1556-1564, 1999.

VITA

Ramin Sahba is from Karaj, Iran. He earned both Bachelor's and Master's degrees in Computer Engineering respectively from Sheikh Bahaee University and Science and Research University, Iran. Besides, he received a Master's degree in Electrical Engineering from The University of Texas at San Antonio. His future plans include attending a Ph.D. program.