2009 Fifth International Conference on Networking and Services

Advanced Hash Algorithms with Key Bits Duplication for IP Address Lookup

Christopher Martinez
Department of Electrical and Computer Engineering
The University of New Haven

Wei-Ming Lin
Department of Electrical and Computer Engineering
The University of Texas at San Antonio

Abstract

Hashing technique have been widely adopted for various computer network applications such as IP address lookup and intrusion detection, among which XOR-hashing is one of most popular techniques due to its relatively small hash process delay. In all the current commonly used XOR-hashing algorithms, each of the hash key bits is usually explicitly XORed only at most once in the hash process, which may limit the amount of potential randomness that can be introduced by the hash process. This paper looks into the possibility in duplicating/reusing key bits to inject additional randomness into the hash process thus enhancing the overall performance further. However, when a key bit is reused, newly induced correlation arises among resulted hash bits where key bits are shared which may easily offset the intended benefit from bit duplication. The novel hash technique introduced in this paper shows how to intelligently apply bit duplication while minimizing the amount of correlation induced in the duplication process.

1. Introduction

A complete survey and complexity analysis on IP address lookup algorithms has been provided in [10]. A performance comparison of traditional XOR folding, bit extraction, CRC-based hash functions is given in [3]. Although most of the 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 are proved to be excellent means but are more complex to compute. Some schemes are hardware based that achieve an improvement in IP look-up by maintaining a subset of routing table in a faster cache memory [5], [6], while others are software based which improve their search performance mainly through efficient data structures [7], [11]. Waldvogel et al. [12] proposed an address look-up scheme based on a binary search of hash table, requiring an extra update process in a look-up table. Other hashing algorithms have also been widely adopted to provide for the address look-up process [1], [2], [9], [13]. All hashing algorithms inevitably suffer from unpredictable complexities involving conflicts among the data with the same hash result (hash collision). 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. The lower the number of hash collisions is created by the hash algorithm the better the performance becomes. Performance of a hashing algorithm is usually determined by two measurements: the MSL (maximum search length) and ASL (average search length) with the former one indicating the largest number of hash collisions for any single hash value and the latter denoting the average number of hash collisions for all hash values.

Hashing techniques using simple XOR operations have been very popular in applications where timely response is critical due to its relatively small hash process delay. In all the current commonly used XOR-hashing algorithms, when deriving the hash value, each of the hash key bits is usually explicitly XORed only at most once in the hash process, which may limit the amount of potential randomness that can be introduced by the hash process. When a key bit is reused (duplicated) for XORing in generating different hash value bits, there exists a potential that the overall randomness of new hash result may increase. This paper looks into the possibility in duplicating key bits to inject additional randomness into the hash process thus enhancing the overall performance further. However, when a key bit is reused, a newly induced correlation arises among hash bits where key bits are shared which may easily offset the intended benefit from duplication. This paper proposes a novel hash technique by intelligently applying bit duplication while minimizing

2. XOR-Hashing Methodology

Throughout this paper, the database under discussion 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 with each vector consisting of each respective bit from all entries. An example of $n = 8$ and $m = 3$ is shown in Figure 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.

978-0-7695-3586-9/09 $25.00 © 2009 IEEE
DOI 10.1109/ICNS.2009.68

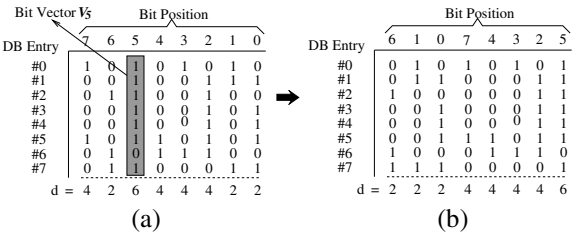


Figure 1. (a) Calculation of d Values (b) Sorted Database by d Value

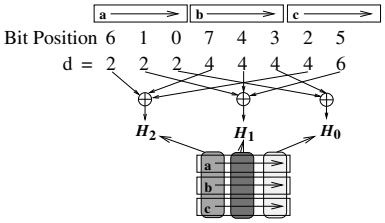


Figure 2. In-Order XOR (d-IOX) Hash Algorithm

XOR operator has been widely used for hashing and known to be an excellent operator in enhancing randomness in distribution. It also possesses a nice characteristic allowing for analytical performance analysis and thus better algorithm designing. A commonly used hashing technique is to simply hash the n-bit key into m-bit hash result through a simple process XORing every distinct $\frac{n}{m}$ 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. A much more effective hashing approach is proposed in [4] by preprocessing (and sorting) the database according to a parameter, the d value, that reveals a very useful insight into the degree of uniformity of the database. The d value of a bit vector is the absolute difference between the number of 0's and 1's in it (as shown in Figure 1(a)). 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. This leads us to employing a simple pre-processing step in re-arranging the n bit vectors according to their d values sorted into a non-decreasing order as shown in Figure 1(b). This sorted sequence then gives us an "order of significance" according to which each bit should be utilized.

A XOR-hashing algorithm based on the principle of d value is presented in [8]. This algorithm, the d-IOX (d value in-order XOR folding), involves the aforementioned preprocessing/sorting step before applying the simple in-order folding XOR hashing. Figure 2 shows the folding process in the d-IOX algorithm, with each of the H_i's referring to a hashing function in deriving a hash value bit. The d-IOX proves to be much better than the simple random Group-XOR approach by registering an improvement in ASL and MSL up to 30% in randomly generated database and up to 80% in real IP database.

3. Limitation in Non-Duplication XOR Hashing

First an experiment is performed to understand the effect in enhancing randomness by XORing a variable number of bits. These bits are randomly generated such that the d value for each bit is uniformly distributed in $[0, 2^{10} - 1]$. The benefit in XORing more bits is shown in Figure 3 in which the resulted d value is presented for a given *number of XORed bits* (NXB) to be XORed together. With bits relatively skewed significantly due to the non-uniform distributed database, randomness quickly becomes saturated as NXB reaches about eight. When the bits are not as

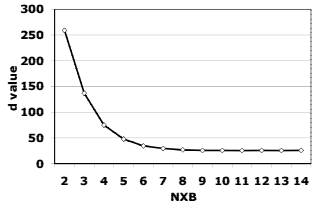


Figure 3. Analysis of NXB on d Values

skewed, one can expect the randomness from XORing tend to saturate even earlier. For such a data set with $m = 10$ and $n = 30$, a natural XOR hashing process leads to an NXB of 3 without any duplication, and the expected d value for each hash value bit drops from the original expected value of 512 (2^9, half of the range) for each hash key bit to 136. After this, it looks as though one can simply reuse (duplicate) the hash key bits to achieve more drop in the d value (e.g. two more NXB's would lead to a d value of about 50), which is actually far from the fact.

Note that when there is no bit duplication under standard XOR hashing, no bits are shared in XORing to lead to different hash value bits. That is, each hash value bit comes from XORing a distinct set of hash key bits. If one intends to reuse some key bits for XORing, then the overall effectiveness may be compromised due to the sharing. Here a notion is introduced to illustrate the induced effect from bit duplication. In obtaining two hash key bits, when there exist common bits between the two sets of hash key bits for their XORing, an Induced Duplication Correlation (IDC) arises between the two hash value bits. The reason that this correlation is regarded as "induced" because it is created through the artificial bit-sharing from duplication, in contrast to the inherent correlation that may already exist in the hash key bits. Figure 4 gives a simple illustration for IDC, where each of the letters (from A to F) denotes a distinct hash key bit.

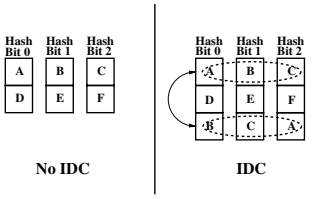


Figure 4. Induced Duplication Correlation (IDC)

The figure on the left shows that a hash value bit is obtained by XORing two distinct hash key bits, while the figure on the right shows that through duplication IDC occurs when every pair of hash value bits have some common bits being XORed. When more bits are duplicated for XORing, higher IDC tends to ensue.

4. Simple Bit-Duplication XOR Hashing

We first introduce several simple bit-duplication procedures to understand the their potential effectiveness and deficiencies. For the sake of simplicity in discussion, all are to be based on the example shown in Figure 5 with $n = 8$ and $m = 4$ and a baseline XORing with d-IOX without duplication. Note that a very important reason in

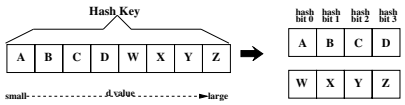


Figure 5. XOR Hashing with No Duplication

using the d-value-sorted bit sequence as the base is that one now may choose to duplicate the bit(s) with the smallest d value to maximize the benefit of duplication. This comes from an already-established theory from [4] which states that XORing with a bit with a smaller d value is more likely to lead to a smaller d value than XORing with a bit with a larger d. That is, in the example in Figure 5, bit A will be the the best candidate to duplicate.

Three different bit-duplication approaches are presented here to study patterns for potential beneficial mechanisms. Note that only one extra duplicated bit is considered here for XORing to produce each final hash value bit.

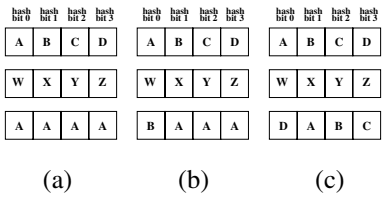
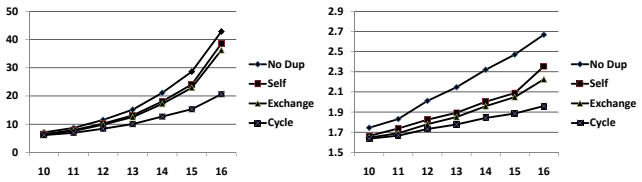


Figure 6. Three Simple Duplication Approaches: (a) Self-Duplication (b) Exchange-Duplication (c) Cycle-Duplication

- Self-Duplication
  The "self-duplication" process first follows a normal non-duplication XOR hash algorithm (Figure 5) and then picks a bit (bit A for the best result) to be duplicated for XORing with each of the four (m) hash value bits, as shown in Figure 6(a). This duplication process not only introduces IDC among all the bits due to the sharing of bit A, but also leads to an even worse scenario – in the XORing for hash bit 0, two A bits are used, which essentially nullifies the randomness from XORing bit A, since $A \oplus A = 0$. Self duplication can be unexpectedly destructive due to the "nullification" of the one bit – a loss of one additional potential bit for randomness. In this particular example, this leads to a hash value bit (bit 0) being degraded to a simple bit extraction (with bit W).
- Exchange-Duplication
  In order to render a quick fix on the nullification problem from the self-duplication approach, one may choose to pick one more bit, in additional to the originally duplicated one, solely for the XORing to avoid nullification. This approach is to be referred as the "exchange-duplication", demonstrated in Figure 6(b), where the second bit B is picked for this purpose. Although this approach eliminates the previous nullification problem, but it introduces another degree of IDC. From the example, the first two bits (bits 0 and 1) end up having an even higher IDC in between them (than the regular IDC induced, e.g. in between bits 2 and 3) since they share two bits in common (A and B). Since $A \oplus B = B \oplus A$, essentially the effect of this higher degree of IDC downgrades the XORing for these first two bits from XORing three bits to become XORing two bits with a normal degree of IDC.
- Cycle-Duplication
  None of the duplication approaches presented so far provides a "uniform" way of duplication without either increasing IDC or downgrading the XORing for some bit(s). One regular way to handle this is to use the so-called "cycle-duplication" in which m bits from the original hash key are selected for duplication with each duplicated exactly once and mapped into the hash bit positions in a rotated fashion. One example for this is shown in Figure 6(c) where the m duplicated bits are taken directly from one segment of m-bit hash key bits (the ones with the smallest d values) and rotated to the very next bit position for XORing. With this, nullification problem is completely eliminated and no downgrading in XORing exists any more. This results in a very uniform IDC among all pairs of two bits – every pair of two bits have exactly one hash key bit shared for XORing.

To show the potential improvement of duplication, a

simple simulation is conducted to measure the performance of each of the three duplication approaches compared to the one without duplication. Figure 7 shows the performance comparison in terms of MSL and ASL among the techniques. All three duplication techniques show improvement



(a) (b)

Figure 7. Performance Comparison in Terms of (a) MSL (b) ASL Among the Different Duplication Approaches on Randomly Generated Data

in the hash results. The self-duplication one has the largest amount of IDC while still leading to a reduction of 15% both in MSL and ASL. The cycle-duplication one, as expected, has the smallest amount of IDC with a reduction up to 50% in MSL and 27% in ASL. This preliminary test clearly shows that the cycle-duplication approach can be a very promising platform where even better bit-duplication techniques can be built upon.

5. Minimal IDC Duplication

All the duplication approaches discussed in the previous section assume to duplicate only one additional bit for XORing for each hash value bit. It is possible that more bits can be duplicated to allow for XORing to lead to more benefits. In order to take on this extension, several important observations from the previous analysis need to be noted carefully.

First, as mentioned earlier, the "cycle-duplication" approach seems to be a natural one to be based on for extension due to its uniformity in IDC and regularity in duplication pattern. The next natural step is to consider having more bit duplication process(es) added to the "cycle-duplication" basis, while avoiding the problems of nullification and downgrading. The nullification problem is relatively easy to avoid by not XORing identical bit(s) for any given hash result bit. However, if the next step of bit duplication is not employed carefully, the effect of downgrading will arise again easily. For example, built from the case of "cycle-duplication" in Figure 6(c), another set of m bits (A, B, C, and D) are duplicated again and "cycled" with another one-bit shift similar to the first step of duplication. Figure 8 illustrates such a case. With this, each pair of hash bits will have two key bits shared in their XORing (e.g. bits 0 and 1 sharing A and D), thus leading to a downgrading problem, or simply a higher degree of IDC. Note that, in our discussion

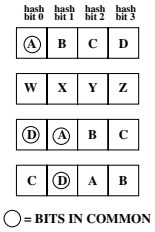


Figure 8. An Example of Downgrading from Additional Duplication

here, the same segment (the one with the smallest d values) is used for multiple duplication due to its low d values. Using another different segment (a segment of bits with larger d values) for the additional duplication has shown a far less potential for performance improvement due to an analysis result from [4] showing that XORing with bits of larger d values may even degrade the performance.

Therefore, in order to avoid any kind of downgrading, one has to first decide if there are sufficient number of bits for further duplication. As shown in the example in Figure 8, one may soon realize that it is impossible to duplicate the segment the second time without running into the downgrading problem. That is, the four (m) bits are not sufficient to support two times of duplication. This problem can be generalized into a question: "Given m, the number of bits to be cycle-duplicated, how many times can it be duplicated without causing the downgrading problem?" Viewing from the other direction of this problem, one may also ask: "In order to duplicate X times without the downgrading problem, what is the minimal m required?" First of all, the condition that needs to be satisfied to avoid the downgrading problem can be easily formulated into a scenario – at most one bit sharing exists in between any pair of hash value bits. In order to duplicate X times while avoiding multiple bits of sharing, the following condition has to be satisfied

$$m \geq X \cdot (X + 1) + 1.$$

Proof:

When the same m bits are to be duplicated X times, it means that each hash bit position is to be produced by XORing the original bit and X duplicated bits (by disregarding the key bits irrelevant to our discussion here, e.g. key bits W, X, Y nd Z in Figure 8). Note that all these $X+1$ bits have to be distinct to avoid nullification. Figure 9 shows an example when the original m-bit hash key are to be duplicated two times, i.e. $X = 2$. Take the case of bit position 0 – bit A, G and E are three ($X + 1 = 3$) distinct bits, with bits G and E duplicated into this bit position. In order to ensure no multiple bits of sharing among all m bits, the three ($X + 1$) bits are not allowed

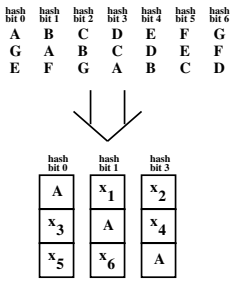


Figure 9. Number of Bits Needed for Two Duplications

to appear in the same column (bit position). Since each of the three bits are to appear another two times (X) in the other columns, the total number of columns (m) has to satisfy

$$m - 1 \geq X \cdot (X + 1). \tag{1}$$

Or simply, as shown in the figure, the three ($X+1$) columns where bit A appears, there are six ($X \cdot (X + 1)$) slots that require distinct key bits (in addition to bit A) to fill. That is, given m, the maximal number of duplication allowed without causing multiple bits of sharing is governed by

$$X \leq \lfloor \frac{\sqrt{4m - 3} - 1}{2} \rfloor \tag{2}$$

The example in Figure 9 shows the minimal m required, $m = 7$ to allow for two bits of duplication.

A sufficient condition has thus been established for the number of duplication steps that can occur given a size of hash value m. What remains to be determined is an algorithm to decide each of the X duplication patterns to avoid any of the aforementioned problems. Although a general algorithm is still not completely available, we have managed to translate the problem into a problem of graphics for easier visualization and processing. Let the set of the m bit position indices be denoted $S = \{0, 1, 2, \ldots, m - 1\}$, and these bits are to be duplicated X times such that X satisfies the condition in Equation 2. For the sake of simplicity without losing generality, assume that each of the X duplicated sequences of the m bits are to be *rotated* starting from a particular bit position to avoid the two problems. We can simply focus on the bit 0 position of each of the strings to analyze the whole pattern. That is, the bit 0 position of the original string is at position 0. Let the position of bit 0 of each of the $X+1$ strings be denoted as s_j where $0 \leq j \leq X$. Figure 10 shows an illustration for a case with $X = 3$ and $m = 13$, with 13 bit positions on a circle. In this case, the four starting locations are $s_0 = 0$, $s_1 = 1$, $s_2 = 3$ and $s_3 = 9$. With this notation, we can easily show that, in order to avoid any nullification problem, the following condition has to hold:

$$s_i \neq s_j, \ \forall i, j, \ 0 \leq i, j \leq m, \text{ and } i \neq j$$

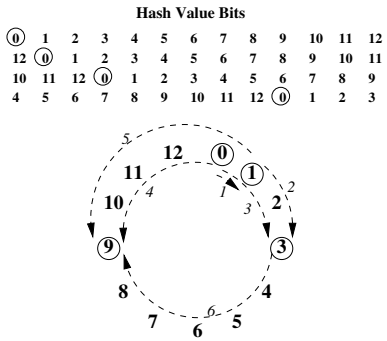


Figure 10. Three-Time Duplication with $m = 13$ Using Cycle Duplication with One Bit in Common

which guarantees that no bit position has two identical bits to be XORed. In order to avoid any sharing of multiple bits (i.e. the downgrading problem), the following condition has to be satisfied:

$$D_{ij} \neq D_{kl}, \ \forall i, j, k, l, \ 0 \leq i, j, k, l \leq m, \text{ and } (i, j) \neq (k, l).$$

where D_{ij} denotes the "shorter" distance from position s_i to position s_j; that is,

$$D_{ij} = \min((s_i - s_j) \bmod m, \ (s_j - s_i) \bmod m)$$

For example, between $s_1 = 1$ and $s_3 = 9$, their distance is

$$D_{13} = \min((1-9) \bmod 13, \ (9-1) \bmod 13) = \min(5, 8) = 5.$$

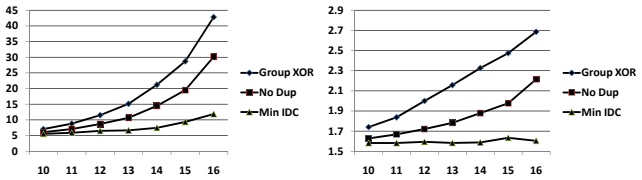
Essentially, this condition guarantees that the no two positions can share more than one bit in common.

Given m, there may exist more than one corresponding X-time duplication patterns that satisfy the aforementioned conditions. We have been able to show that feasible solution(s) can be found for any X up to four, i.e. with any m up to 30. Note that, when multiple solutions exist, each solution does not necessarily lead to the same hashing performance due to the d value distribution after sorting. This proposed duplication approach is thus named as the "Minimal IDC Duplication" technique.

6. Simulation Results

Simulation runs are performed on randomly generated data sets and real IP data sets to demonstrate the performance improvement of the minimal IDC duplication XOR hash technique over other techniques with no duplication. The Group-XOR algorithm which XORs groups of random key bits is the general base of our comparison, while the d-IOX [8] aforementioned will serve as the reference as a non-duplication technique (referred to as "No Duplication" in the rest of our discussion), which the minimal IDC algorithm is essentially based on.

The simulation results for $n = 32$ and $10 \leq m \leq 16$ are given in Figure 11 using a data set randomly generated such that the d value for each bit position is uniformly distributed. Performance comparison among the three techniques are in terms of MSL and ASL by taking an average of results from 1,000 runs. The minimal IDC algorithm delivers a



(a) (b)

Figure 11. Performance Comparison in Terms of (a) MSL (b) ASL on Randomly Generated Data Sets with $n = 32$ and $10 \leq m \leq 16$

vast improvement over the Group-XOR algorithm, up to 72% of reduction in MSL and 40% in ASL. True effect of duplication is demonstrated by its results compared to the d-IOX (No Duplication), showing a whopping 60% reduction This result clearly demonstrates the potential of the proposed duplication approach. What makes the duplication even more promising is that the Minimal IDC Duplication continues to increase its gain over the Group-XOR as m increases while the non-duplication one (d-IOX) saturates its gain when m reaches 13. This scenario may be explained by the fact that when $m \geq 13$, the Minimal IDC algorithm increases its duplication to three times, thus further improving its performance.

7. Conclusion

By duplicating and reusing hash key bits, our technique further enhances the randomness from the best known XOR-hashing techniques. Other potential applications of this approach include string matching, general database query, etc. There still exist many potential extensions along this line of research. This paper only approaches bit duplication in a cyclic pattern, while a mixture of different patterns may provide even more benefit. In addition, this paper examined only induced correlation from the duplication without considering the inherent correlation already existing in the target database, which may have a very significant impact on the design of hash algorithms. By providing initial groundwork for duplication in hashing, this paper has pointed out the potential areas to improve hashing algorithms and new ways to exploit specific characteristics of the target database.

References

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

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

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

[4] 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.

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

[6] X. Nie, D.J. Wilson, J. Cornet, G. Damm, Yiqiang Zhoa, "P Address Lookup Using A dynamic Hash Functio", *IEEE Electrical and Computer Engineering, Canadian Conference*, Page(s) 1646 - 1651, May 1-4, 2005.

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

[8] D. Pandya, C. Martinez, W.-M. Lin and P. Patel, "Advanced Hashing Techniques for Non-Uniformly Distributed IP Address Lookup", *Third IASTED International Conference on Communications and Computer Networks (CCN2006)*, October 2006, Lima, Peru.

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

[10] 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.

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

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

[13] 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.

[14] D. Yu, B. Smith, and B. Wei, "Forwarding Engine for Fast Routing Lookups and Updates," *Global Telecommunications Conference,*, Globecom '99, p.1556-p.1564.