On the Design of Fast Prefix-Preserving IP Address Anonymization Scheme

Qianli Zhang, Jilong Wang, and Xing Li

CERNET Center, Tsinghua University
zhang@cernet.edu.cn

Abstract. Traffic traces are generally anonymized before used in analysis. Prefix-preserving anonymization is often used to avoid privacy issues as well as preserve prefix relationship after anonymization. To facilitate research on real time high speed network traffic, address anonymization algorithm should be fast and consistent. In this paper, the bit string based algorithm and the embedded bit string algorithm will be introduced. Bit string based algorithm uses precomputed bit string to improve the anonymization performance. Instead of only using the LSB of each Rijndael output, the embedded bit string algorithm will take advantage of the full size Rijndael output to anonymize several bits at the same time. The implementation can be downloaded from https://sourceforge.net/projects/ipanon.

1 Introduction

There has been a growing interest in internet traffic research. However, real-world internet traffic traces are still very rare, only a few organizations would share their traffic traces (NLANR/MOAT Network Analysis Infrastructure (NAI) project [2], WIDE project [9], and ACM ITA project [3]). Even with these traces, there still lack of the most recent traces of high speed network. To make the research on most recent traffic traces possible, DragonLab (Distributed Research Academic Gigabit Optical Network Lab) [1] began to establish the real-time traffic analysis environment.

In DragonLab, network traffic is collected from Tsinghua university campus network border router. Tsinghua University campus network is the first and the largest campus network in China, it is connected to China Education and Research Network with two gigabit links. Experimenters can assign incoming or outgoing traffic from one of these two links, and then this traffic will be replayed to the experimenter's measurement point.

To avoid the leak of users' privacy information, traffic traces are subject to an anonymization process [4] [5] [6] before being studied: payload will be erased, the source IP address and destination IP address of packets will be anonymized. IP address anonymization is one of the major steps in this process.

There have been many anonymization schemes available. A straightforward approach is to map each distinct IP address appearing in the trace to a random 32-bit address. The only requirement is that this mapping be one-to-one.

However, the loss of the prefix relationships among the IP addresses renders the trace unusable in situations where such relationship is important (e.g., routing performance analysis, or clustering of end systems [7]). It is, therefore, highly desirable for the address anonymization to be prefix-preserving. That is, if two original IP addresses share a k-bit prefix, their anonymized mappings will also share a k-bit prefix.

Inconsistent mapping is also undesirable. For inconsistent mappings, same original address may be mapped into different anonymized addresses when applied independently on more than one traces. Consistent mapping is important because of the following reasons. First, if the traffic anonymization process stops and restarts after a while, the previous and current anonymized traffic traces will take different mappings, thus make the consistent research impossible; secondly, there is a real need for simultaneous (yet consistent) anonymization of traffic traces in different sites, e.g., for taking a snapshot of the Internet. It would be very cumbersome if hundreds of traces have to be gathered first and then anonymized in sequence.

Speed of IP address anonymization is also worth a serious consideration in research on real time traffic. Even for off-line anonymization, speed is important since slow anonymization algorithm may require traffic to be stored to disk beforehand, which is time consuming and inconvenient.

In this paper, we will propose a group of novel prefix-preserving IP address anonymization algorithms; they are all based on the precomputation of random bits. The rest of this paper is organized as follows. In section 2 we briefly introduce related works, including the operation of TCPdpriv and Crypto-pan. In section 3 we describe our schemes in details, section 4 will discuss some concerns in implementation and its performance. The paper is concluded in section 5.

2 Related Works

2.1 TCPdpriv

One possible prefix-preserving approach is adopted in TCPdpriv developed by Greg Minshall [8] and further modified by K. Cho [9]. TCPdpriv can be viewed as a table based approach. It stores a set of $< raw, anonymized >$ binding pairs of IP addresses to maintain the consistency of the anonymization. When a new raw IP address a needs to be anonymized, it will try to find the longest prefix match and anonymize the rest in random. The new generated pair will be added to the binding table. Since only the memory lookup and random generation are required in this algorithm, it may operate very fast.

However, this algorithm is not consistent: the mappings are determined by the raw IP addresses and the relative order in which they appear in a trace. Therefore, a raw address appearing in different traces may be mapped into different anonymized addresses by TCPdpriv, hence the inconsistency. Also, to store all the binding pairs a large amount of memory will be consumed.

2.2 Crypto-pan

Crypto-pan [10] [11] is a deterministic mapping function from raw addresses to anonymized addresses based on the Canonical Form Theorem [11]. With the same key, it can anonymize traffic traces consistently. In this algorithm, $f_i, i = 1 \ldots n$ are defined as follows: $f_i(a_1a_2 \ldots a_i) := L(R(P(a_1a_2 \ldots a_i); K)), i = 0, 1, \ldots, n-1$, where L returns the least significant bit, R is a pseudo-random function or a pseudo-random permutation (i.e., a block cipher) such as Rijndael [12], and P is a padding function that expands $a_1a_2 \ldots a_i$ into a longer string that matches the block size of R. K is the cryptographic key used in the pseudo-random function R. Since the cryptography based anonymization function is uniquely determined by K, same address appearing in two different traces will be mapped to the same anonymized address if the same key is used.

However, to anonymize an IP address, Crypto-pan needs 32 rounds of Rijndael encryption, thus makes it unsuitable for real time anonymization without special hardware. It can only anonymize 10000 IP addresses per second with a PIII machine [11]. Consider the overhead of packet capture, the Crypto-pan is not practical for anonymization in wire speed. Thus, unlike Tcpdpriv, Crypto-pan can only be used off-line.

3 Bit String Based Schemes

3.1 Methodology

Assume S is a random bit string of length L_S, and P_i is a function from $\{0,1\}^i$ to $\{0, L_S - 1\}$, for $i = 1, 2, \ldots, n-1$ and $P_0 \equiv 0$. Let $B(S, n)$ be the n'th bit of S, define $f_i(a_1a_2 \ldots a_i) = B(S, P_i(a_1a_2 \ldots a_i))$, The anonymization process would be:

Given an IP address $a = a_1a_2 \ldots a_n$, let $F(a) = a'_1a'_2 \ldots a'_n$ where $a'_i = a_i \bigoplus f_{i-1}(a_1, a_2, \ldots, a_{i-1})$, and $\bigoplus$ stand for the exclusive-or operation, for $i = 1, 2, \ldots n$.

According to Canonical Form Theorem [11], the map is prefix-preserving. This is also straightforward since given $a = a_1a_2 \ldots a_n$, the anonymized IP address $a'_1a'_2 \ldots a'_n$ is generated with $a'_i = a_i \bigoplus B(S, P_{i-1}(a_1a_2 \ldots a_{i-1}))$, which only depends on $a_1a_2 \ldots a_{i-1}$.

The length of bit string S is crucial for the security of anonymization. We have the following results:

Lemma 1. *If for any $a_1a_2 \ldots a_i \neq b_1b_2 \ldots b_j$, $P_i(a_1a_2 \ldots a_i) \neq P_j(b_1b_2 \ldots b_j)$, $0 \leq i, j \leq n-1$, string S is at least $2^n - 1$ bits size.*

Proof. For any $a_1a_2 \ldots a_i \neq b_1b_2 \ldots b_j$, $P_i(a_1a_2 \ldots a_i) \neq P_j(b_1b_2 \ldots b_j)$, imply:

1. If $i \neq j$, $1 \leq i, j \leq n-1$, $P_i(a_1a_2 \ldots a_i) \neq P_j(b_1b_2 \ldots b_j)$.
2. If $i = j$, $a_1a_2 \ldots a_i \neq b_1b_2 \ldots b_i$, $1 \leq i \leq n-1$, $P_i(a_1a_2 \ldots a_i) \neq P_i(b_1b_2 \ldots b_i)$

Consider 2, since the number of all possible i bits prefix is 2^i, $P_i(a_1a_2\ldots a_i)$ has at least 2^i different return values. Now consider 1 and $P_0 = 0$, string S has at least $1 + 2 + 4 + \ldots + 2^{n-1} = 2^n - 1$ different positions, thus the length of S is at least $2^n - 1$ bits.

To prefix-preserving anonymize the complete 32 bits IPv4 address, a string of $2^{32} - 1$ bits (or about 512M bytes) long is required for the maximum security level. The bit string S could be precomputed and preloaded to accelerate the anonymization process. Since memory is rather cheap now, this algorithm can operate very fast with commodity hardware. In situations where anonymizing the first 24 bits is enough, a shorter string with $2^{24} - 1$ bits (or about 2M bytes) long is required.

3.2 Construction of P_i Function

Now the problem is how to construct bit string S and find the proper position mapping function P_i. An ideal group of $P_i, i = 1, 2, \ldots, n-1$ should be easy to present and fast to calculate. We propose the binary tree traversal based method to find such mapping functions. The tree is formed as:

- the root node of the tree is $P_0 = 0$;
- the left child node of $P_i(a_1a_2\ldots a_i)$ is $P_{i+1}(a_1a_2\ldots a_i0)$ and the right child node is $P_{i+1}(a_1a_2\ldots a_i1)$.

Thus the problem becomes to assign values of 1 to $L_S - 1$ to all nodes (except the root node) of this binary tree. We can think of this problem to assign a traversal sequence number to each node. Though the assignment scheme may be arbitrary, to be simple in implementation, we only consider two typical schemes in this paper: the breadth first scheme and the depth first scheme(Fig. 1).

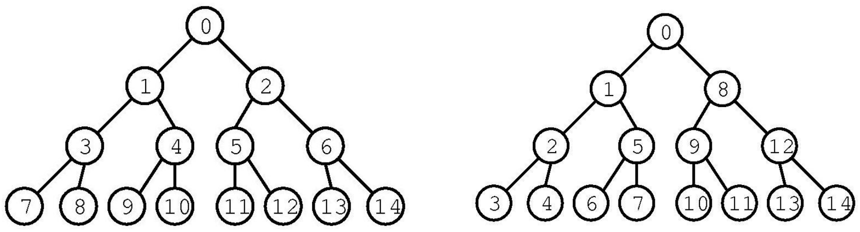

Fig. 1. Breadth first (left) scheme and depth first (right) scheme

For the breadth first scheme, the P_i function is: $P_i(a_1a_2\ldots a_i) = 2^i - 1 + VAL(a_1a_2\ldots a_i)$, $i = 1, 2, \ldots, n-1$. $VAL(a_1a_2\ldots a_i)$ is the value of $a_1a_2\ldots a_i$. Consider

$$\begin{aligned} P_i(a_1a_2\ldots a_i) &= 2^i - 1 + VAL(a_1a_2\ldots a_i) \\ &= 2(2^{i-1} - 1 + VAL(a_1a_2\ldots a_{i-1})) \\ &\quad +1 + a_i \\ &= 2P_{i-1}(a_1a_2\ldots a_{i-1}) + a_i + 1 \end{aligned} \tag{1}$$

Thus, P_i function can also be presented as:

$$\begin{aligned}P_i(a_1a_2\dots a_i) &= 2P_{i-1}(a_1a_2\dots a_{i-1})+1\\ &\quad \textit{if } a_i=0\\ P_i(a_1a_2\dots a_i) &= 2P_{i-1}(a_1a_2\dots a_{i-1})+2\\ &\quad \textit{if } a_i=1\\ &\quad i=1,2,\dots,n-1\end{aligned} \tag{2}$$

For the depth first scheme, the P_i function is:

$$\begin{aligned}P_i(a_1a_2\dots a_i) &= P_{i-1}(a_1a_2\dots a_{i-1})+1\\ &\quad \textit{if } a_i=0\\ P_i(a_1a_2\dots a_i) &= P_{i-1}(a_1a_2\dots a_{i-1})+2^{n-i}\\ &\quad \textit{if } a_i=1\\ &\quad i=1,2,\dots,n-1\end{aligned} \tag{3}$$

In both schemes $P_0 = 0$.

3.3 Reuse Distance Based Data Locality Analysis

Access to large memory often incurs many cache misses and thus seriously affects the performance. Since the cache policy is often complicated and highly dependent on the specific CPU's architecture, we will use the reuse distance [13] to measure the cache behavior. The reuse distance of a reference is defined as the number of distinct memory references between itself and its reuse. For bit string base algorithms, to anonymize the first k bits prefix, a total of k memory accesses (the address of each access is $S[0], \dots, S[P_{i-1}(a_0a_1\dots a_{k-1})/8]$) are required. Since each anonymization round starts from the access to $S[0]$, we will evaluate the reuse distance to anonymize one IP address. Note the fact that $P_i(a_1a_2\dots a_ia_{i+1}) > P_i(a_1a_2\dots a_i)$ always holds for depth first and breadth first schemes, only consecutive accesses may result in accesses to the same cache line. We have the following results.

Lemma 2. *For a cache line size of $c = 2^m$ bits, $L_S = 2^k - 1$ bits and $k \le c \ll L_S$, To anonymize k bits, for breadth first scheme, the reuse distance N for each address' anonymization satisfy*

$$k-1-m \le N \le k-m \tag{4}$$

For depth first scheme, N satisfy

$$1 \le N \le k-m \tag{5}$$

Proof. Defined $Diff(i) = P_{i+1}(a_1a_2\dots a_{i+1}) - P_i(a_1a_2\dots a_i)$. For breadth first scheme, since

$$\begin{aligned}Diff(i) &= P_{i+1}(a_1a_2\dots a_{i+1}) - P_i(a_1a_2\dots a_i)\\ &= P_i(a_1a_2\dots a_i)+1+a_{i+1}\end{aligned} \tag{6}$$

For $i \geq m, Diff(i) \geq 2^m$, and $Diff(i) = 2^m$ only when $i = m, a_1 = a_2 = \ldots = a_{i+1} = 0$. Thus if $i > m + 1$, each access is in a different cache line. Since $i \leq k - 1$, it is easy to see $N \geq k - 1 - (m + 1) + 1 = k - 1 - m$ and $N \leq k - 1 - (m + 1) + 2 = k - m$.

For depth first scheme, since $Diff(i) = 1$ if $a_{i+1} = 0$, and $Diff(i) = 2^{k-1-i}$ if $a_{i+1} = 1$. For $i \leq k - 1 - m$, if and only if $a_i = 1$, $Diff(i) \geq 2^m$, an access is in a different cache line. When IP address is 0, only one memory read is required ($k \leq c$). The worst case happens when all bits are 1, in which case $k - m$ cache misses. If for each bit 1 and 0 have the same possibility, the expectation of reuse distance is $(k - m)/2$.

It looks like that depth first algorithm will be faster than breadth first algorithm in average for single IP address anonymization given that the cache miss number is the only affecting factor. When anonymizing a number of IP addresses, the scenario will be a little different. For breadth first algorithm, the most frequently used bits are all located in the beginning of the bit string S. It is generally easier to cache the most frequently used memory. In contrast, the most frequently used bits in depth first algorithm are relatively sparsely located in the bit string. For random generated IP addresses, suppose the size of cache is C bits and the size of of cache line is C_L bits, in situation where half of the cache size is used to cache the most frequently used $C/2$ bit string, for breadth first algorithm, the first $log_2(C/2)$ bits' anonymization will not incur a cache miss, while for depth first algorithm, since half of the most frequently used bit (the 1 branch) are sparsely located across the bit string, thus only the first $log_2(C/2/C_L) + 1$ bits will be anonymized without cache miss. In experiment, we find that depth first algorithm is generally slower when $k < 31$. This indicates that to anonymize single IP address and to anonymize a large number IP address consecutively are quite different.

3.4 Block Tree Based Prefix Preserving Algorithms

From the above analysis, it can be inferred that if the bit string can be completely loaded into cache, the algorithm would be greatly accelerated. Thus a group of block tree based algorithms are designed. Block tree algorithm is constructed based on the depth first or breadth first bit string algorithms. The basic idea behind this is to divide one IP address into several parts, for each part, there is a correspondent bit string block. These blocks are also organized as a tree. The algorithm is defined by each part's bit number, the position mapping function among blocks and the position mapping function inside each block.

Assume to anonymize the first k bits of IP address, there are n parts and the i'th part has $L_i, i = 0, \ldots n - 1$ bits, $\sum_{i=0}^{n-1} L_i = k$, the part i of IP address IP is $part(i, ip)$, and the bit string is S_i, the algorithm is:

```
$for\ i = 0 : n - 1$
    $anonymize(part(i, ip), L_i, S(P_i(ip)))$        (7)
$end\ for$
```

$P_i(ip)$ determines which block of the string S should be used to anonymize the i'th part of IP address ip. P can be breadth first position function or depth first position function. The anonymization function can also be depth first or breadth first algorithms. About the length of bit string S, it can be proved that string S also requires at lease $2^k - 1$ bits long.

It is because, to anonymize the L_i bits of IP address, about $2^{L_i} - 1$ bits are required. Assume there are B_i blocks for the i'th part of the IP address. For the $part(i, ip)$ anonymization, the total number of bits required is $B_i(2^{L_i} - 1)$ bits. After anonymization, each block has 2^{L_i} branches. Thus:

$$\begin{aligned} B_0 &= 1 \\ B_i &= B_{i-1}2^{L_{i-1}} \end{aligned} \tag{8}$$

Thus the total number of bits required is

$$\begin{aligned} N &= \textstyle\sum_{i=0}^{n-1} B_i(2^{L_i} - 1) \\ &= \textstyle\sum_{i=0}^{n-1} (B_i 2^{L_i} - B_i) \\ &= \textstyle\sum_{i=0}^{n-1} (B_{i+1} - B_i) \\ &= B_n - B_0 \\ &= \textstyle\prod_{i=0}^{n-1} 2^{L_i} - B_0 \\ &= 2^k - 1 \end{aligned} \tag{9}$$

If $L_i = 1, i = 0, \ldots, n-1$, it becomes the depth first or breadth first algorithm.

Comparing to the simple depth first or breadth first algorithms, with proper parameters, block tree based algorithms may further minimize cache misses. For example, for CPU with $512K$ cache and 128 bytes size cache line, to anonymize 29 bits prefix,split the 29 bits into two parts: $21, 8$, in the optimal situation, the first bit string ($256K$ bytes) is loaded into cache and only one cache miss will be incurred to anonymize one address.

3.5 Embedded Bit String Based Prefix Preserving Algorithms

Embedded bit string based approach is another variant that aims to reduce the memory required. As described before, to anonymize 32 bits IPv4 addresses, about 512M bytes memory is required. It makes the algorithm infeasible for memory-limited devices like network processors. Also, it is impossible to anonymize IPv6 addresses with this algorithm, even if only anonymize the first 64 bits prefix (subnet prefix). Embedded bit string based algorithm can be thought as to divide one IP address into several parts, for each part, there is a correspondent bit string block. Unlike block tree based approach, these blocks are generated dynamically with cryptographical secure method.

Assume to anonymize the first k bits of IP address, there are n parts and the i'th part has $L_i, i = 0, \ldots n-1$ bits, $\sum_{i=0}^{n-1} L_i = k$, the part i of IP address IP is $part(i, ip)$, and the bit string is S_i, off_i is the first bit offset of the i'th parts' in IP address, the algorithm is:

```
$for\ i = 0 : n-1$
    $S_i = encrypt((ip >> (32 - off_i)) << (32 - off_i))$
    $anonymize(part(i, ip), L_i, S_i)$
$end\ for$
```
(10)

For example, if the encrypt function is 128 bits block cipher, each time 7 bits can be anonymized. For an IPv4 address, about 5 rounds are needed. To anonymize the first 64 bits prefix of IPv6 addresses, about 10 rounds are necessary, comparing to 64 rounds of Rijindael encryption of Crypto-Pan. Obviously, in this algorithm, cache misses are not the deciding factor for performance. It is welcome since a widening gap between processor and memory speeds has been witnessed in recent years.

4 Implementation and Experiment Results

4.1 Implementation

The bit string S is generated with some pseudo-random number generator. The selection of such PRNG is arbitrary, in this paper, we use the ISAAC [14] algorithm. ISAAC (Indirection, Shift, Accumulate, Add, and Count) generates 32-bit random numbers. Averaged out, it requires 18.75 machine cycles to generate each 32-bit value. The results are uniformly distributed, unbiased, and unpredictable. ISAAC is a secure pseudo random number generator for practical applications and hasn't been broken since it was published 5 years ago. No bias has been detected either. Recent research indicates an estimated known plain text attack on ISAAC may require a time of 4.67×10^{1240} [15]. The initial seed of ISAAC is generated from secret key K via a cryptographic secure pseudo-random number generator. For example, HMAC [16] algorithm or some block cipher.

The anonymization process will load the large bit string S in advance, then for each input IP address, calculate the $f_i(a_1a_2\dots a_i) = B(S, P_i(a_1a_2\dots a_i), i = 1, 2, \dots, n, f_0 = B(S, 0)$

Now consider the B and P function. $B(S, bit)$ function is defined as:

```
((S[bit >> 3]&(0x80>>(bit & 0x07)))!= 0)
```

We implement two $P(ip, i)$ functions corresponding to the breadth first scheme and the depth first scheme. $P(ip, i)$ for breadth first scheme and depth first are:

```
(ip&(0x80000000>>i))?
                ((P(ip,i-1)<<1)+1)
                :((P(ip,i-1)<<1)+2)
(ip&(0x80000000>>i))?
                (P(ip,i-1)+(1<<(32-i))
                :(P(ip,i-1)+1)
```

For block tree based algorithm, we use the breadth-breadth approach, that is, the construction inside the blocks or among blocks are all breadth first based. The selection of number of parts and each part's bit number often depend on the specific CPU's cache size. In this paper, we use the two level approach: the first bit string is $64K$ bytes and will anonymize the first 19 bits, the rest bits will be anonymized by another bit string.

For embedded bit string algorithm, we use Rijindael/128 algorithm, thus each round will anonymize 7 bits.

The implementation can be downloaded from https://sourceforge.net/projects/ipanon.

4.2 Experiment Results

We evaluate the these implementations in two systems: one is PIV 2.8G intel CPU with 1G memory(machine A) and the other is PIV 1.8G intel CPU with 1G memory (machine B). Both of them have a L2 cache size of $512K$ bytes and a cache line size of 128 bytes. We also modify Crypto-pan for a comparison. For each scheme, we generate 16,777,216 (16M) sequential or random IP addresses, and measure the elapsed time in anonymization. The input IP addresses are in 32 bits integer format. We measure the elapsed time after the initialization (for bit string based scheme, after the bit string is generated and loaded). The results are shown in table 1 and table 2. The first row is from machine A and the second row is from machine B.

Table 1. Experiment results for anonymization(random input)

	depth	breadth	cpan	block tree	embedded bit string
Time(us)	1.91	1.98	7.75	1.12	1.45
Time(us)	2.32	2.77	11.34	1.60	2.26

Table 2. Experiment results for anonymization (sequential input)

	depth	breadth	cpan	block tree	embedded bit string
Time(us)	0.24	0.24	7.24	0.37	1.20
Time(us)	0.37	0.37	11.05	0.58	1.84

Experiment result shows that bit string based schemes(depth first, breadth first, block tree, embedded bit string) are 3 to 6 times faster than Crypto-pan in the worst case. For 32 bits random input anonymization, depth first scheme is a little faster than breadth first scheme, while embedded bit string algorithm is faster than both. Block tree algorithm is considerably faster than all the other schemes. A comparison of the machine A and machine B is also interesting. For CPU sensitive algorithms like crypto-pan and embedded bit string, a performance increase of 40% is gained from slower machine B to A; while for memory access sensitive algorithms like breadth first or depth first algorithm, only about 20% is gained. Although the performance of CPU has increased a lot, the speed of memory access is roughly the same. Previous experiments in a PIV 1.4G machine indicate that depth first algorithm is more than 10 times faster than Crypto-pan. For 1.8G machine this ratio is 4.89 and for 2.8G machine this ratio is 4.06.

We evaluate the time required vs. number of bits to be processed, the results are shown in Fig.2. For algorithms like depth first algorithm, breadth first algorithm and block tree algorithm, the property of input can affect the performance dramatically. For example, sequential input may be more than 6 times faster than random input for depth first algorithm. The reason is that 16,777,216 sequential IP addresses share a common 8 bits prefix, thus has far less cache misses and the deciding factor is the computation overhead. For schemes that are not cache miss sensitive like Crypto pan and embedded bit string algorithm, there is little difference between sequential and random scenarios.

For random input, the computation time of breadth first and depth first anonymization algorithms are roughly linear before 22 bits with a fixed slope of computation overhead per bit. The time is roughly linear after 22 bits with the increasing cache misses, though the slope is a lot steeper. The embedded bit string based algorithm, is a staircase function since there is little overhead inside each 7 bits block. For block tree based approach, it can be separated into 3 lines, for the first line (1-22), since the complete bit string can be loaded into cache, there is little cache misses. Because Intel CPU's cache line size is 128 bytes, from 22 to 29, there is only one cache misses. From 30 to 32 bits, there will be one cache misses added per bit increased.

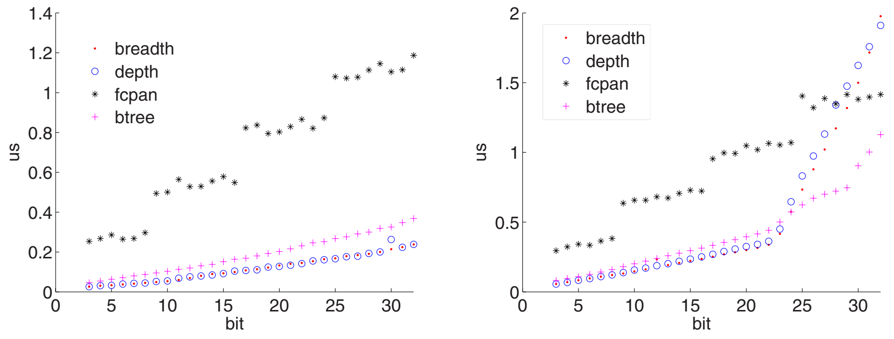

Fig. 2. Processing time per address (averaged from 16,777,216 IP addresses anonymization), sequential input (left) and random input (right). Depth first algorithm is shortened as depth, breadth first algorithm as breadth, Crypto-pan as cpan, block tree algorithm as btree and the embedded bit string algorithm as fcpan.

The IP addresses in real traces are often more complex: addresses inside ISP are heavily clustered while addresses outside ISP may be very random. To evaluate the performance of these anonymization scheme in real environment, we anonymize a public 24 hours traffic trace provided by WIDE [9]. The trace is captured on Feb 27, 2003 and is stored in pcap format. It contains a total of 364,483,718 packets. After gzip compression, the traffic trace occupies about 10G disk space. The proposed schemes are applied on it. Traces after anonymization are also stored in disk in pcap format. As shown in table 3, bit string based schemes are much faster than Crypto-pan.

Table 3. Experiment results for anonymization of real traces

	Time	PPS
Depth first	3437.3447s	105,566
Breadth first	3544.4881s	102,357
Crypto-pan	20338.0471	17,841
Depth first 24	2607.2846s	139,175
Breadth first 24	2570.0872s	141,189
Crypto-pan 24	15757.2146s	23,028
Block tree	3195.3225s	113,562
No anonymization	1923.2942s	188, 670

5 Conclusion

In this paper, we propose a group of novel prefix-preserving IP address anonymization algorithms which all base on the bit string based algorithm. Experiment results indicate that these algorithms are all much faster than Crypto-pan.

More research is still going on to accelerate the anonymization speed so that anonymization of IPv6 addresses in gigabit wire speed is possible.

Acknowledgment

This research was supported by the research Program of China (863) under contract number 2005AA112130 and research Program of China (973) under contract number 2003CB314807.

References

1. DragonLab, `http://www.dragonlab.org/`
2. McGregor, T., Braun, H., Brown, J.: The NLANR network analysis infrastructure. IEEE Communications Magazine 38(5), 122–128 (2000)
3. The Internet traffic archive (April 2000), `http://ita.ee.lbl.gov/`
4. Patarin, S., Makpangou, M., Pandora, M.: A flexible network monitoring platform. In: Proceedings of the 2000 USENIX Annual Technical Conference (June 2000)
5. Peuhkuri, M.: A Method to Compress and Anonymize Packet Traces. SIGCOMM IMW (2001)
6. Pang, R., Paxson, V.: A high-level programming environment for packet trace anonymization and transformation. SIGCOMM (2003)
7. Krishnamurthy, B., Wang, J.: On network-ware clustering of web clients. In: SIGCOMM (2000)
8. Minshall, G.: TCPdpriv Command Manual (1996)
9. Cho, K., Mitsuya, K., Kato, A.: Traffic data repository at the wide project. In: Proceedings of USENIX 2000 Annual Technical Conference: FREENIX Track, San Diego, CA (June 2000)
10. Xu, J., Fan, J., Ammar, M.H., Moon, S.B.: On the design and performance of prefix-preserving IP traffic trace anonymization. In: SIGCOMM IMW (2001)

11. Xu, J., Fan, J., Ammar, M.H., Moon, S.B.: Prefix-preserving IP address anonymization: measurement based security evaluation and a new cryptography-based scheme. In: ICNP (2002)
12. Daemen, J., Rijmen, V.: AES proposal: Rijndael, Tech. Rep., Computer Security Resource Center, National Institute of Standards and Technology (February 2001), `http://csrc.nist.gov/encryption/aes/rijndael/Rijndael.pdf`
13. Beyls, K., D'Hollander, E.: Reuse distance-based cache hint selection. In: Procceedings of the 8th International Euro-Par Conference (August 2002)
14. Jenkins, B.: ISAAC: a fast cryptographic random number generator, `http://burtleburtle.net/bob/rand/isaac.html`
15. Pudovkina, M.: A known plaintext attack on the ISAAC keystream generator, `http://eprint.iacr.org/2001/049.pdf`
16. Krawczyk, H., Bellare, M., Canetti, R.: RFC 2104: HMAC: Keyed-Hashing for Message Authentication (February 1997)
17. Ylonen, T.: Thoughts on how to mount an attack on tpcpdriv's "-50" option, in TCPpdpriv source distribution (1996)