Normalizing Traffic Pattern with Anonymity for Mission Critical Applications

Dongxi Liu, Chi-Hung Chi, Ming Li
School of Computing
National University of Singapore
Lower Kent Ridge Road
Singapore 119260
Emails: chich@comp.nus.edu.sg

Abstract

Intruders often want to analyze traffic pattern to get information for his some malicious activities in ultra-secure network. This paper presents a general approach to prevent traffic pattern of IP-based network from being analyzed. It is an isolated scheme which can be used to prevent traffic analysis in overall network by achieving the same goal in each network segment independently. On each network segment, complementary traffic is generated according to its real traffic, and the combination of these two kinds of traffic constitutes the normalized traffic on each link. Main advantages of our approach are, from the performance viewpoint, 1) complementary traffic does not compete on the bandwidth with real traffic actively, and 2) complementary traffic does not consume the bandwidth of other network segment at all. In addition, by encrypting source and destination IP addresses of each packet, anonymous communication can be achieved and anonymous normalized traffic loses its value for the analysis of eavesdropped traffic by intruders.

1. Introduction

Generally, information security in the Internet is guaranteed by using security protocols, such as IP security protocol (IPSec) [1] and transport layer security (TLS) [2]. However, it is not enough for an ultra-secure communication network, such as a military network, to just encrypt and authenticate transmitted data. It is because in computer communication networks, a traffic series has its statistical patterns, which can provide extra useful information for intruders. For example, the research in traffic analysis exhibits ample evidence that traffic is a second-order stationary series with the statistical properties of long-range dependence (LRD), (asymptotical) self-similarity and heavy-tailed distribution at different layers [3-6]. Thus, intruders may obtain the statistical traffic pattern between any two network nodes, such as hosts or routers, by processing monitored data series. Using the known traffic patterns, he can distinguish special events happening from the pattern variations of a traffic series. Consequently, any methods used to prevent traffic pattern analysis on a packet-by-packet basis are beneficial for ultra-secure distributed systems.

Several approaches have been reported in this area, see for examples [7-11]. In this paper, we propose a novel approach based on normalizing traffic pattern with anonymity. Our scheme to prevent traffic pattern analysis for IP-based network is based on the following two aspects: 1) steadily normalizing traffic, and 2) hiding IP addresses in packets. By steadily normalizing traffic, we mean that traffic pattern remains unchanged regardless of what happens to it. By hiding IP address in every packet, we mean the IP address in every packet is confidential to intruders.

To achieve our goal, we insert a thin layer between the network layer and the link layer. Thus, during every unit time period, only one packet is sent (either a regular packet from the network layer or a padded packet when no regular packet is available, i.e. a dummy packet). Moreover, if the packet to be sent is shorter than an expected length, some extra bytes are padded to its rear so that packets sent by one network node always have constant size with constant rate, i.e., normalized traffic, in which dummy packet and extra bytes together make up the complementary traffic. In addition, when sending a packet, its source and destination IP addresses will be encrypted for anonymity so that intruders cannot trace the packet flow in network. Upon receiving a packet, a reverse procedure should be taken to recover the real packet. In this way, a normalized anonymous traffic is generated such that traffic pattern losses its analysis value for intruders from the security view.

This paper is organized as follows. Section 2 introduces some work related our study. Our scheme is discussed in Section 3. Conclusions are given in Section 4.

Proceedings of the 37th Annual Simulation Symposium (ANSS'04)
1080-241X/04 $ 20.00 © 2004 IEEE

2. Related Work

There are two categories of countermeasures to resist traffic analysis. One is anonymous communication and the other traffic camouflaging. Anonymous communication can prevent intruders from knowing who is communicating with whom, such as Onion Routing project [10-11], but it cannot hide the traffic pattern of the overall network. Traffic camouflaging [7-8] can prevent traffic analysis by padding and rerouting so that the traffic pattern between two hosts are not related to the operation status of application for intruders. However, it cannot provide anonymity and intruders can infer the differences of traffic patterns between the two hosts.

In comparison with the previous work, the main advantage of our approach in function is that it can provide anonymous communication and hide traffic pattern *simultaneously*. Moreover, our approach is an isolated scheme. By isolated scheme, we mean two points: 1) to achieve its goal in the overall network by achieving the same goal in each network segment, and 2) not to assume any global parameters, such as the capacity between any two hosts like that in [7-8]. Being an isolated scheme, our approach, from the performance viewpoint, has two additional advantages: 1) complementary traffic does not compete on the bandwidth with the real traffic *actively,* and 2) complementary traffic does not consume the bandwidth of other network segment at all. Our approach can be deployed in an overall network by deploying it in each network segment gradually, thus making it to be practical in engineering. It should be mentioned that the link cover mode in [9] is to achieve constant traffic rate on each link, too. However, it depends on a finite end-to-end flow set to compute the link cover mode. We argue that this parameter is very difficult (if not impossible) to get because the two ends may have infinite flows. In [14], Kung et. al. propose an IP-layer anonymizing infrastructure to hide the server address from all clients to resist DOS attack, which is different from the anonymity in this paper.

3. Methodologies

Our approach consists of three components: a protocol to negotiate the parameters, such as the encryption algorithm and key; a procedure to normalize traffic; and cache management. In what follows, we will introduce each part in detail.

3.1 Notations and Definitions

As usual, a network can be described by a directed graph (N, E), where $n \in N$ is a network node (e.g., hosts or routers). For $n_1, n_2 \in N$, if $(n_1, n_2) \in E$, then (n_1, n_2) is a link. For $e \in E$, we denote len_e as the maximum length (bytes) of packets transmitted in e without fragment, i.e. MTU of e (we sometimes use len_e to denote the payload length of a packet for brevity), and σ_e as the minimum interval time between two frames.

Definition 1 (*Link Feature*): Let $G = (N, E)$ be a directed graph. Let $e \in E$. Then the pair (len_e, σ_e) is the feature of link e.

If the feature of a link is known, the goal of normalizing traffic on this link can be determined. Usually, a link can have other parameters and we abstract them away since they are not relevant with our work here.

Let $x_e[t(i)]$ be the length of the i^{th} packet to be sent to link e at time $t(i)$ $(i = 0, 1, 2, \ldots)$. Define the normalized traffic as follows.

Definition 2 (*Normalized Traffic*): Let $G = (N, E)$ is a directed graph. For $\forall e \in E$, $x_e[t(i)]$ is the normalized traffic if the followings hold:

1) (len_e, σ_e) is e's feature;
2) $x_e[t(i)] = len_e$ for each i;
3) $t(i + 1) - t(i) <= \sigma_e$ for $i>0$;
4) $t(i + 2) - t(i) > \sigma_e$ for $i > 0$.

From the above definition, each frame in a normalized traffic has constant size len_e and only one packet is sent in each time interval σ_e.

Remark 1. A normalized traffic is characterized by the feature of its link.

Encryption operation needs keys and algorithms. With the notion of security association stated by the IKE protocol [13], we define the link security association below.

Definition 3 (*Link Security Association*): Let $G = (N, E)$ is a directed graph. For $\forall e \in E$, a pair $sa_e = (Enc_e, k_e)$ is called the security association of link e, where Enc_e is

the encryption algorithm and k_e is the relevant key.

The address of packet transmitting on link e can be encrypted by the algorithm Enc_e with key k_e. In addition, for simplicity, our approach does not support authentication. That is, the spoofing of the link address is not prohibited.

3.2 Parameter Negotiation

To normalize traffic in a link e, we need to know e's feature (len_e, σ_e) and the security association sa_e. Given link e, len_e depends on the characteristics of the link medium and link protocol. It is always configured by hand or negotiated by link protocol such as PPP. Therefore, we assume this parameter has been instantiated before the traffic normalization begins. For parameter σ_e, we just use it to characterize the normalized traffic and our scheme does not rely on it to implement normalization. Thus, it is also unnecessary to negotiate it with the parameter negotiation protocol. Therefore, sa_e is the only parameter we need to care, and this can be discussed by the following two cases:

1) Link frame with encryption: For example, MPPE Protocol is used as the link protocol, or WEP encryption scheme is used in IEEE 802.11 wireless LAN. In this case, we are free to negotiate sa_e because the traffic encryption will be done by the link protocol rather than by our scheme.

2) Link frame without encryption: For example, ordinary Ethernet frame or HDLC frame belongs to this case. In order to provide anonymity and prevent intruders from detecting padding data or packets, our approach has to encrypt the source and destination addresses of packet from the network layer. For this purpose, the link security association needs to be negotiated.

In theory, we can use IKE protocol for our purpose. In practice, however, we found that it might not be suitable for link communication since it is designed for Internet service. For example, we need not to worry about the denial of service attack at the link layer service. Below, we will describe the parameter negotiation protocol. For a link $e = (n_1, n_2)$, this protocol defines two roles. One is master played by n_1 and the other slave played by n_2. The message sequence in this protocol is expressed by

1. $n_1 \rightarrow n_2$: g, $g^x \bmod g$, Enc
2. $n_2 \rightarrow n_1$: $g^y \bmod g$, $\{g, g^y, g^x\}k$
3. $n_1 \rightarrow n_2$: $\{g, g^x, g^y\}k$.

Obviously, this protocol is developed from the Diffie-Hellman key exchange protocol, in which g is a big prime, $g^x \bmod g$ and $g^y \bmod g$ being the two Diffie-Hellman public values, and g^{xy} will be the symmetric session key. This protocol directly runs on the link layer. Hence, the link layer should be able to identify the frames for this protocol.

After running the protocol, n_1 and n_2 can use the encryption algorithm Enc with key $g^{xy} \bmod g$ to provide anonymity for the network layer packet. In addition, we can assume that master n_1 has a clock, which periodically invokes the protocol so that n_1 and n_2 can periodically refresh their link security associations.

Though the man-in-the-middle attack of the Diffie-Hellman protocol is quite unlikely in the link layer, we still authenticate g, g^y, g^x by encrypting them with a key $k = g^{x(i-1)y(i-1)} \bmod g$, where $i > 1$ indicates the i^{th} run of this protocol. That is, k negotiated in the previous run will be used to authenticate messages in this run. It should be noted that the first run depends on $g^{x(0)}$ and $g^{y(0)}$ which must be configured on n_1 and n_2, which are long term keys.

3.3 Normalizing Traffic with Anonymity

Suppose (len_e, σ_e) is the link e's feature. Normalized traffic on e consists of frames with constant size len_e and constant interval σ_e between them. Thus, during a period of σ_e, if there is no packet to transmit on e, we need to send a dummy packet with length len_e, or if the size of packet transmitting on e is less than len_e, then some extra bytes are needed to append at the rear of the packet such that it has size len_e. Therefore, the following problems have to be solved:

1) How dummy packets should be marked so that they are distinguishable from real packets for receiver and not for intruders?

2) How extra bytes should be appended in a real packet so that receiver can distinguish them, but intruders cannot?

3) How should the payload of dummy packets and extra bytes be constructed?

4) How should the interval between two frames be guaranteed to be σ_e?

With the assumption that the network protocol is IP, we just concern the four fields in an IP header, namely, total length, header checksum, source IP address and destination IP address. Therefore, a packet can be represented by <*srcIP*, *dstIP*, *len*, *checksum*, *payload*>. Let $sa_e = (Enc_e, k_e)$. Then, to provide anonymity, *srcIP* and *dstIP* must be encrypted: <{*srcIP*, *dstIP*}k_e, *len*, *checksum*, *payload*>.

Mark dummy packets

Obviously, we can insert a protocol indicator to tell whether the encapsulated packet is a dummy packet. That is a common way in protocol design. Here, we take a more efficient way to avoid the overhead of encapsulating packets as follows. When an IP packet is arrived, the receiver checks its header checksum. If this fails, the packet will be dropped. Therefore, we only need to negate a bit in the header checksum field of a real packet. In case of a dummy one, it will be dropped by the receiver.

Formally, suppose <*srcIP*, *dstIP*, *len*, *checksum*, *payload*> is a real packet. Then, <{*srcIP*, *dstIP*}k_e, *len*, *checksum*⊕*r*, *payload*>, where ⊕ denotes *exclusive-or*, *r* is a member chosen from the following set randomly:

{0x0001, 0x0002, 0x0004, 0x0008,
0x0010, 0x0020, 0x0040, 0x0080,
0x0100, 0x0200, 0x0400, 0x0800,
0x1000, 0x2000, 0x4000, 0x8000}

Moreover, intruders cannot know whether a packet is a dummy one or not because *srcIP* are *dstIP* are encrypted and he is unable to compute the checksum.

Append extra bytes in a real packet

We can append enough random bits to the rear of a real packet such that it has the expected packet length. Suppose the link *e*'s feature is (len_e, σ_e) and <*srcIP*, *dstIP*, *len*, *checksum*, *payload*> is a real packet. Then, <*srcIP*, *dstIP*, *len*, *checksum*, *payload+extra bytes*> is a normalized packet, where *length*(*payload+extra bytes*) = len_e.

After doing this, we do not update the value of the field *len*. Otherwise, the real length is impossible to recover. However, if this field is transmitted as plaintext, intruders will know the real length even the real bytes and extra bytes are transmitted in one frame. Therefore, the value of this field needs to be confidential to intruders. That is, the field should be encrypted if the link does not provide privacy. After appending the extra bytes, the packets will be <{*srcIP*, *dstIP*}k_e, {*len*} k_e, *checksum*, *payload+extra bytes*>, where *length*(*payload+extra bytes*) = len_e and (Enc_e, k_e) is the link *e*'s security association.

Construct payload of dummy packets and extra bytes

Dummy packets including more bytes than extra bytes make the payload in full length. Here, we discuss a way to construct the payload of dummy packets because the extra bytes can be drawn from the constructed dummy packet. A simple way is to use the last real packet as the dummy packet by just flipping one bit in the header checker sum. However, the similarity between them can be observed by intruders, thus helping him to distinguish the dummy packet. Here, we would like to introduce some randomness to the payload. Suppose < {*srcIP*, *dstIP*}k_e, {*len*}k_e, *checksum*, *payload*> is the last real packet, where *length*(*payload*) = len_e, and l is a random value and $1 < l < len_e$. Then, *payload* = $payload_1$ + $payload_2$, where *length*($payload_1$) = l and *length*($payload_2$) = $len_e - l$. In this way, the new payload in the dummy packet is $payload' = payload_2 + payload_1$. Therefore, a complete dummy packet can be represented as:

<{*srcIP*, *dstIP*}k_e, {*len*}k_e, *checksum*⊕*r*, *payload*'>,

where *r* is defined above.

Of course, we can generate a series of random bits with length len_e as the payload of dummy packets, but it is an expensive computation to do [12].

Guarantee the interval between two frames to be σ_e

For a node *u*, if it has a timer more precise than σ_e, it is easy to transmit a frame during every σ_e. Unfortunately, σ_e is always much less than the timer in system. For example, σ_e in fast Ethernet is less than 120 μs, but a tick in Linux is about 10 ms. So, σ_e is useless when practically normalizing traffic and it will not be negotiated in the parameter negotiation protocol. Here, we propose the best-effort transmitting scheme to make the interval time as small as possible, which will

approach σ_e.

In our scheme, we suppose a node u has two queues. One is for real packets, called *real queue*, and the other for dummy packets, called *dummy queue*. Of course, the packets in *real queue* are the output of u's packet scheduling algorithm while *dummy queue* has only one packet which is the last real packet transmitted on the link. Those in *dummy queue* are sent only if there are no packets in *real queue*. At the same time, each packet is normalized and this is done by the *traffic normalization scheduler*. The following algorithm can illustrate this procedure.

Algorithm 1: *traffic normalization scheduler*

INPUT: Enc_e, k_e, len_e, *real queue*, *dummy queue*
OUTPUT: a normalized packet p

1. let $< \{srcIP, dstIP\}k_e, \{len\}k_e, checksum, payload>$ be the packet in *dummy queue*;
2. choose a random integer l, $1 < l < len_e$;
3. $payload' = payload_2 + payload_1$, where $payload_1 + payload_2 = payload$ with $length(payload_1) = l$ and $length(payload_1) = len_e - l$.
4. if *real queue* is empty, then
   4.1 $p = < \{srcIP, dstIP\}k_e, \{len\}k_e, checksum \oplus r, payload'>$, where r is defined as that in 1);
   4.2 return p;
5. else
   5.1 let $<srcIP, dstIP, len, checksum, payload>$ be the first packet in *real queue*;
   5.2 if $length(payload) < len_e$, then
      5.2.1 $payload'' = payload + payload_1$, where $payload_1$ is the prefix of $payload'$ with length $len_e - length(payload)$);
      5.2.2 $p = < \{srcIP, dstIP\}k_e, \{len\}k_e, checksum, payload''>$;
   5.3 else
      5.3.1 $p = < \{srcIP, dstIP\}k_e, \{len\}k_e, checksum, payload>$;
   5.4 substitute p for the packet in dummy packet;
   5.5 return p;

The output p in the above algorithm can be directly transmitted by the link layer. After sending a packet, the interrupt event will be generated, by which the above algorithm can be called in turn. Thus, a continuous normalized traffic with anonymity is generated.

3.4 Cache Management

Cache is usually a good way to improve the performance. In our approach, cache is used to improve the performance of cryptography operations, including encryption and decryption. We first introduce our cache management scheme and then discuss two problems from the point of view of cache hit rate. One problem is why the total length field and source/destination IP addresses are encrypted separately and the other why we do not identify dummy packets by modifying the value of the total length field in ciphertext.

There are two cache tables in our approach. One is for the total length field and the other for the source/destination IP addresses.

Cache table for total length field

Each entry in this cache table has the format (*len_plaintext*, *len_ciphertext*, *valid*), where *len_plaintext* is the *len* in a real packet; *len_ciphertext* is the result that *len* is encrypted under the current *link security association* sa_e; *valid* = 1 indicates that this entry is valid while *valid* = 0 invalid. This table is called *lencache*, which can be viewed as a set.

The size of this table is bounded by len_e. For example, the payload length in Ethernet frame is from 46 to 1510 bytes. Hence, the number of entries in this table is at most 1464. For *lencache*, it is unnecessary for cache replacement algorithm.

For a packet $<srcIP, dstIP, len, checksum, payload>$, when encrypting *len* field, if $\{len\_ciphertext | (len\_plaintext, len\_ciphertext, valid) \in lencache, valid = 1 \text{ and } len = len\_plaintext\} \neq \Phi$ (null set), then the cache will be hit and the *len_ciphertext* in this set will be the encryption result of *len*. In this way, we can avoid the encryption operation. Otherwise, cache miss occurs and we encrypt *len* under the current *link security association* (Enc_e, k_e) and then append $(len, \{len\}k_e, 1)$ to *lencache*.

For a packet $< \{srcIP, dstIP\}k_e, \{len\}k_e, checksum, payload>$, when decrypting $\{len\}k_e$, if $\{len\_plaintext | (len\_plaintext, len\_ciphertext, valid) \in lencache, valid = 1 \text{ and } \{len\}k_e = len\_ciphertext\} \neq \Phi$, then the cache will be hit and the *len_plaintext* in this set will be the decryption result of $\{len\}k_e$. Again, we can avoid the decryption operation. Otherwise, we decrypt $\{len\}k_e$ under the current *link security association* (Enc_e, k_e) and append $(len, \{len\}k_e, 1)$ to *lencache*.

Obviously, if the *link security association* is renegotiated, all the entries in *lencache* will have to be invalidated.

Cache table for source/destination IP addresess

This cache is called *IPcache*, which consists of entries with the form of (*srcIP_plaintext*, *destIP_plaintext*, *IP_ciphertext*, *route*, *valid*), where *srcIP_plaintext* and *destIP_plaintext* are the *srcIP and destIP* in a real packet respectively; *IP_ciphertext* is the encryption result of *srcIP and destIP* under the current *link security association* sa_e; *valid* = 1 indicates this entry is valid while *valid* = 0 invalid; *route* is the route information for the *destIP*.

For a packet <*srcIP*, *dstIP*, *len*, *checksum*, *payload*>, when encrypting *srcIP and destIP* fields, if {*IP_ciphertext*| (*srcIP_plaintext*, *destIP_plaintext*, *IP_ciphertext*, *route*, *valid*) ∈ *IPcache*, *valid* = 1, *srcIP* = *srcIP_plaintext* and *destIP* = *destIP_plaintext*} ≠ Φ, the *IP_ciphertext* in this set will be the encryption result of *srcIP and destIP*. Otherwise, we encrypt them under the current *link security association* (Enc_e, k_e) and append (*srcIP*, *destIP*, {*srcIP*, *destIP* }k_e, *route*, 1) to *IPcache*, where *route* is looked up from the route table.

For a packet <{*srcIP*, *dstIP*}k_e, {*len*}k_e, *checksum*, *payload*>, when decrypting {*srcIP*, *dstIP*}k_e, if {(*srcIP_plaintext*, *destIP_plaintext, route*)|(*srcIP_plaintext*, *destIP_plaintext*, *IP_ciphertext*, *route*, *valid*) ∈ *IPcache*, *valid* = 1 and {*srcIP*, *dstIP*}k_e = *IP_ciphertext*} ≠ Φ, the cache will be hit and the *srcIP_plaintext* and *destIP_plaintext* in this set will be the decryption result of {*srcIP*, *dstIP*}k_e and *route* gives the route information for this packet. Otherwise, we decrypt {*srcIP*, *dstIP*}k_e under the current *link security association* (Enc_e, k_e) and append (*srcIP*, *destIP*, {*srcIP*, *destIP* }k_e, *route*, 1) to *IPcache*, where *route* is looked up from the route table.

Like *lencache*, all the entries in *IPcache* have to be invalidated if the *link security association* is renegotiated.

Two Problems

The first problem is why the total length field and source/destination IP addresses are encrypted separately. Given the current *link security association*, because the *len* range is finite, we can expect the probability of the cache hit rate to be 1 after some time. If we combine the total length field and IP address field, then even for two packets with the same *len*, their source/destination IP addresses are not necessarily the same. Thus, we cannot obtain benefits from the finite range of *len*. In a flow, the packets in it must have the same source/destination IP addresses, but their lengths can possibly be (very) different. Therefore, a cache hit must occur for the successive packets in the flow if we only encrypt and cache source/destination IP addresses. Otherwise, we cannot attain the benefits from the feature of the packet flow. As a result, we choose to encrypt and cache the total length field and source/destination IP addresses separately.

The second problem is why we do not identify dummy packets by modifying the value of the total length field in ciphertext. Although it is feasible to mark dummy packets by this way, it might increase the rate of cache miss. Since a dummy packet is evolved from a real packet with the same length between them, after normalizing the real packet, its length value must be in the *lencache*. Consequently, if we modify the length value in ciphertext, it is possible that upon receiving the dummy packet we might have to decrypt the {*len*}k_e because of the cache miss. Therefore, we choose to modify the checksum field in the IP header instead of the total length field to mark dummy packets.

4. Conclusions

In this paper, we have presented and explained a general approach to prevent traffic analysis. Our approach works between the link layer and network layer and it includes three parts. Firstly, a parameter negotiation protocol is used to negotiate the link security association (i.e., an encryption algorithm and the corresponding key). To counter the cryptanalysis, these parameters should be negotiated periodically. If a link encryption is provided, this protocol is not necessary. Secondly, traffic is normalized with anonymity and this is the main part of our work. It concerns how to generate complementary traffic and combine it with real traffic to normalize the traffic on a link. By encrypting the IP address of each packet, anonymity is achieved. Thirdly, cache management is used to improve the efficiency of encryption operation when normalizing traffic. A high rate of cache hit can be attained because the packet length has a finite range and the packets in one flow have the same source and destination IP addresses. Each run of the parameter negotiation protocol will invalidate all the entries in *lencache* and *addrcache*. Since our approach is an isolated scheme and it does not need any global parameters, it can be easily deployed in each network segment independently. From the performance

viewpoint, by using the cache, the encryption can be done quickly and complementary traffic does not compete on the bandwidth with real traffic actively, nor does it consume the bandwidth of other network segments at all. Hence, it is an effective and practical approach to prevent traffic pattern analysis by intruders.

5. References

[1] S. Kent and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401.

[2] T. Dierks and C. Allen. "The TLS protocol version 1.0," RFC 2246.

[3] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, "On the Self-Similar Nature of Ethernet Traffic (Extended Version)," *IEEE Trans on Networking*, 2 (1), Feb. 1994, 1-15.

[4] V. Paxson and S. Floyd, "Wide Area Traffic: The Failure of Poison Modeling," *IEEE Trans. on Networking*, 3 (3), June 1995, 226-244.

[5] E. Crovella and A. Bestavros, "Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes," *IEEE/ACM Trans. on Networking*, 5 (6), Dec. 1997, 835-846.

[6] B. Tsybakov and N. D. Georganas, "Self-Similar Processes in Communications Networks," *IEEE Trans. on Information Theory*, 44 (5), Sep. 1998, 1713-1725.

[7] Y. Guan, D. Xuan, P. U. Shernoy, R. B. Bettati, and W. Zhao, "NetCamo: Camouflaging Network Traffic for QoS-Guaranteed Mission Critical Applications," *IEEE Trans. on Systems, Man, and Cybernetics, Part AL Systems and Humans*, 31 (4), 253-265.

[8] Y. Guan, C. Li, D. Xuan, R. Bettati, and W. Zhao, "*Preventing Traffic Analysis for Real-Time Communication Networks*," *Proc., The IEEE Military Communication Conference (MILCOM'99)*, Nov. 1999.

[9] S. Jiang, N. Vaidya, and W. Zhao. "Preventing Traffic Analysis in Packet Radio Networks," *Proc., DARPA Information Survivability Conference and Exposition (DISCEX II'01)*, 2001.

[10] D. M. Goldschlag, M. G. Reed, and P. F. Syverson, "Onion Routing for Anonymous and Private Internet Connections," *Communications of the ACM*, 42 (2), 1999.

[11] M. G. Reed, P. F. Syverson, and D. M. Goldschlag, "Anonymous Connections and Onion Routing," *IEEE Journal on Selected Areas in Communication Special Issue on Copyright and Privacy Protection*, 1998.

[12] A. J. Menezes, P. C. van Oorschot and S. A Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.

[13] D. Harkins and D. Carrel, The Internet Key Exchange (IKE), RFC 2409.

[14] H. T. Kung, C. M. Cheng, K. S. Tan and S. Bradner, "Design and Analysis of an IP-Layer Anonymizing Infrastructure", *The Third DARPA Information Survivability Conference and Exposition (DISCEX 3)*, April 2003.