Improvements to Covert Channels in TCP Timestamps

D. Anderson and P. Lee

***Abstract*—In this paper we analyze DEVCC, the most commonly referenced implementation of covert channels using steganography in TCP timestamps. We identify detectable patterns in the timestamps such as an abnormal frequency in the least significant bit (LSB) of 5:3 ones to zeroes, and autocorrelation that does not parallel normal TCP traffic, in particular one-bit autocorrelation of 0-0 and 1-1 pairs being unequal, and propose multiple alternative transmission algorithms in the hopes that they not only improve upon DEVCC but that they are also useable in real-world scenarios. Finally we analyze DEVCC and each alternative to support the strength of the best of these alternatives, showing that they show far less abnormal patterning than DEVCC, and we find that three proposed algorithms are statistically indistinguishable from normal TCP/IP traffic in low-bandwidth situations.**

***Index Terms*—Computer security, covert channel, steganography, TCP timestamp.**

I. Introduction

THE need for hiding the existence of data transmission is often overlooked as maintaining the secrecy of visible transmissions is focused upon. However, simply hiding the contents of a message can be insufficient; in some cases, simply knowing that information was sent is revealing too much data. Scenarios such as a political blogger who wishes to remain anonymous or a governmental spy who does not want to be exposed are immediately obvious; for the former, detecting encrypted file uploads at times that posts were made would be a strong signal, and for the latter even sending encrypted data at all might draw too much attention.

The solution is to hide the very transmission of data using a covert channel. A covert channel is deemed useful if it is practically undetectable when used. We will not consider a channel to have failed if an active warden model can prevent transmission, as prevention is not equivalent to detection, and is in fact much easier [1].

We focus upon the potential channel of timestamps in TCP headers. TCP over IP is the most frequently used protocol on the internet, and its optional timestamp field is being used by increasing numbers of operating systems. Its purported purpose is to allow hosts to accurately calculate round trip times (RTT) and to prevent sequence number (SEQ) wraparound difficulties [2]. We chose this channel because of its ubiquity and because there seems to be less published material on it compared to media such as images and sound.

It should be noted that to embed covert data we must delay packets. We cannot simply reduce their timestamps, since it is often trivial to detect packets being sent 10ms (the granularity on of a Linux system clock) after their timestamp.

Other techniques to use sections of TCP packets as covert channels have been presented. Type-of-Service channels flip bits in rarely-used fields of the header, but are trivial to detect precisely because these fields are rarely used, and even more rarely changed mid-stream [2]. The initial sequence number field, or ISN, can be used as well, but it can send at most four bytes per connection (not per packet), and undetectable versions must send significantly fewer than this [2]. Other methods exist, but are also detectable, as described in [3]. Mechanisms for embedding a channel at a lower level [2] have been proposed, but as these are unusable in an even mildly complicated network, their usefulness is diminished.

The only implementation of covert channels over TCP timestamps that we were able to find is DEVCC. “It is a protocol for sending data… at a rate of one bit per packet” [4]. It chooses which plaintext bit to send using a hash of the SEQ (which for our consideration can be effectively considered a unique number assigned to each packet). It determines a cypherkey using a hash of the timestamp (without its LSB) and SEQ. The cyphertext bit is the xor of the cypherkey and the plaintext key, and so if necessary it delays the packet to get the appropriate LSB as cyphertext. If a higher order bit is changed by this delay it begins calculations again. Unfortunately this recalculation leads to easily detectable patterning in all traffic conditions. In addition, timing characteristics caused by delaying half of all packets by 10ms forms a strong fingerprint – so strong that watermarking schemes have been proposed which do almost exactly this [5]. The data is recovered from the resulting packet in a symmetric fashion by the receiver.

We have implemented four alternative algorithms for manipulating TCP timestamps. The first algorithm is effectively half DEVCC, half normal traffic, as it simply generates a pseudorandom bit from the packet header and decides, based on that, whether or not a data bit should be sent.

The second algorithm is also a variant of DEVCC. It decreases the chance that a recalculation is needed by using fewer bits of the timestamp during the cyphertext calculation.

The third algorithm reduces the randomness of the cypherkey by ignoring the timestamp during its generation.

The final algorithm simply computes a hash of the entire

packet header, including all the bits of the timestamp, and if the LSB of the hash is the cyphertext LSB it is sent.

As will be seen below, each of the bit-encode algorithms will also decode correctly because they are symmetric. For brevity the decode steps will not be explicitly listed.

II. Potential areas for improvement in DEVCC

A. Algorithm

DEVCC's sending algorithm is as follows:

```
DEVCCEncodePacket(Packet P, TimeStamp T)
  GetHeader(P) → PacketHeader
  GetSeqNum(PacketHeader) → SequenceNumber
  SHA1(SequenceNumber) → Index
  SHA1(PacketHeader, T & 0xfffffffe) → KeyBit
  MessageBlock[Index] → PlainTestBit
  PlainTextBit ⊕ KeyBit → CipherTextBit
  If T[0] ≠ CipherTextBit then
    T + 1 → T
    If T[0] = 0 then
      Return DEVCCEncodePacket(P,T)
    End if
  End if
  SendPacket(P,T)
```

Figure 1: DEVCC embedding algorithm

As discussed in the introduction, this uses a hash of the SEQ to choose the plaintext bit, it uses a hash of the SEQ and the timestamp to get a cyphertext bit, and if the current timestamp does not have a matching LSB it delays one tick and recalculates. A full discussion of the algorithm can be found in [2].

B. Areas Open To Improvement

The DEVCC algorithm contains a serious flaw in any traffic conditions: it will transmit far fewer 0s than 1s in LSBs. When the LSB is 0, it has .5 odds of remaining a 0 or of becoming a 1. In no case does it become a 0 again, as it busy-waits in the kernel where it cannot be preempted. When the LSB is a 1, it also has .5 odds of remaining a 1 and .5 odds of being increased. When it is increased, however, the second-to-least significant bit is modified, and the cypherbit is recalculated based on new parameters, with even odds of being a 0 or 1. Therefore an original LSB of 1 has .75 odds of having a cypherbit of 1 and only .25 odds of having a cypherbit of 0.

A simple frequency analysis shows this plainly, with a 0s:1s sent-LSB ratio of .75:1.25, or 3:5. In traffic that is even mildly bursty (two-packet bursts are enough) this will be evident. A one-bit autocorrelation test will also reveal the discrepancy immediately if a sample of high speed traffic is taken, as normal traffic has equal odds for the case of an LSB of 1 following a 1 and for a 0 following a 0.

A statistical anomaly in high-speed high-density situations is the average number of unique timestamps used versus the transmission duration. Normal TCP traffic in these conditions is theorized to approach a ratio of 1:1, as each timestamp should be used [2]. DEVCC, however, approaches a number under .75 due to its need for time delays [2].

The number of groups of identical timestamps, by size of group, in high speed situations, is approximately exponentially decreasing (exponent of ½), whereas normal traffic approximately fits a bell curve under the same traffic conditions, so this can be used to differentiate the two.

Finally, under moderately consistent network conditions in small time intervals (<< 60,000ms), round-trip-time is normally unimodal with a significant spike at the average round trip time (RTT). This spike characteristically has a width of less than 10ms [6][7]. Because DEVCC delays, on average, 3/8 of the packets by 10ms and 1/8 by 20ms, its RTT pattern is trimodal, with the second spike roughly half as high and the third spike significantly smaller than the first, with a 10ms gap between each. Because RTT is a Poisson process this patterning can be detected independently from transmission speed or packet burst patterns.

III. Alternative Algorithm: Half-DEVCC

A. Motivation

DEVCC has very detectible patterns; by mixing an even amount of regular traffic with the covert channel we hoped to make patterning more difficult to observe.

B. Algorithm

```
HalfDEVCCEncodePacket(Packet P, TimeStamp T)
  GetHeader(P) → PacketHeader
  GetSeqNum(PacketHeader) → SequenceNumber
  SHA1(SequenceNumber, Constant) → RandomBit
  If RandomBit = 1 then
    Return DEVCCEncodePacket(P, T)
  End if
  SendPacket(P,T)
```

Figure 2: Half-DEVCC embedding algorithm

C. Results

Because this relies on the DEVCC algorithm, it still has very noticeable patterns; the LSB ratio has improved but frequency analysis on even a short transmission still shows something is amiss. For the same reason, autocorrelation reveals an unwanted pattern like DEVCC's. There is only a mild improvement upon its unique timestamp ratio—not enough to be undetectable. Unique timestamp count binning is substantially closer to normal traffic than DEVCC, but does not conform to a normal curve, and so may still be detectable. It also obviously requires twice as many packets as DEVCC to send a given message, which is a significant drawback.

IV. Alternative Algorithm: Timestamp-Masked Keybit Generation

A. Motivation

The frequency problem in DEVCC is a direct result of its frequent recomputations of the keybit due to non-LSB bits in

the timestamp being changed when the LSB is incremented from 1. By reducing the dependency on the lower-order bits of the timestamp by masking some out, this rollover problem can be reduced. We tested using 4 bits to ignore hoping to decrease the problem to $1/16^{th}$ while ensuring that there was still a nontrivial dependency on the timestamp.

B. Algorithm

```
TSMaskedEncodePacket(Packet P, Timestamp T)
  GetHeader(P) → PacketHeader
  GetSeqNum(PacketHeader) → SequenceNumber
  SHA1(SequenceNumber) → Index
  SHA1(PacketHeader, T & MaskBits) → KeyBit
  MessageBlock[Index] → PlainTestBit
  PlainTextBit ⊕ KeyBit → CipherTextBit
  If T[0] ≠ CipherTextBit then
    T + 1 → T
    If T[0] = 0 then
      Return TSMaskedEncodePacket(P,T)
    End if
  End if
  SendPacket(P,T)
```

Figure 3: Timestamp-Masked embedding algorithm

C. Results

As expected, this alternative solved the bit frequency problem, being statistically indistinguishable from normal traffic. With respect to unique timestamps, the ratio was over 0.9—higher than our actual measurement of unmodified traffic—and obviously improved greatly over DEVCC. Unfortunately it does not have timestamp count patterns like normal traffic, and autocorrelation on one and two bits reveals a distinctly nonstandard pattern.

V. Alternative Algorithm: Cypherbit Independent of Timestamp

A. Motivation

Using the timestamp to determine the cypherbit seems to introduce many problems; by using the timestamp solely as a transmission medium we aim to avoid this.

B. Algorithm

```
TimelessEncodePacket(Packet P, Timestamp T)
  GetHeader(P) → PacketHeader
  GetSeqNum(PacketHeader) → SequenceNumber
  SHA1(SequenceNumber) → Index
  SHA1(PacketHeader) → KeyBit
  MessageBlock[Index] → PlainTestBit
  PlainTextBit ⊕ KeyBit → CipherTextBit
  If (T[0] ⊕ CipherTextBit) = 1 then
    T + 1 → T
  End if
  SendPacket(P,T)
```

Figure 4: Time independent encryption embedding algorithm

C. Results

This algorithm has a bit frequency pattern indistinguishable from normal traffic. Its repeated timestamp binning pattern is similar to normal traffic's, though with a smaller standard deviation. For both one and two bits it's autocorrelation patterns are within 1.5 standard deviations of normal traffic, often better.

VI. Alternative Algorithm: Bernoulli Experiment

A. Motivation

We wish to treat the embedding of data as a coin-toss so as to make it a Bernoulli experiment, in part because it is well-suited for theoretical analysis.

B. Algorithm

```
BernoulliEncodePacket(Packet, Timestamp T)
  GetHeader(P) → PacketHeader
  GetSeqNum(PacketHeader) → SequenceNumber
  SHA1(SequenceNumber) → RandomBit
  SHA1(PacketHeader, T)→ KeyBit
  MessageBlock[Index] → PlainTestBit
  PlainTextBit ⊕ KeyBit → CipherTextBit
  If RandomBit ≠ CipherTextBit then
    T + 1 → T
    Return BernoulliEncodePacket(P,T)
  End if
  SendPacket(P,T)
```

Figure 5: Bernoulli embedding algorithm

C. Results

Frequency analysis and one-bit autocorrelation tests cannot distinguish this from normal traffic. Unfortunately, this algorithm still exhibits an exponential decrease in repeated timestamp count bins by size. Furthermore, two-bit autocorrelation shows small but statistically significant spikes where normal traffic has none. Since it is a Bernoulli experiment with .5 probability of usage, its unique timestamp to total possible timestamp ratio is 0.5, far below the theoretical "good" ratio of 1.

VII. Methodology and Data

We inserted one kernel module at a time into a Linux system to test each algorithm. The module intercepted outgoing TCP packets immediately before they are sent to hardware and, if applicable, modifies them before sending them to the hardware. We ensured that CPU usage and conditions in the surrounding network did not affect data collection.

To provide a suitably hostile environment for testing the algorithms, all experimental data was gathered on the fastest possible connection, a local loopback. Brief confirmation tests were run on internet-crossing traffic and a LAN. It should be noted that many of the results, however, are independent of transmission speed – in particular, DEVCC is theoretically and practically trivial to detect at any speed, as it's frequency

distribution does not change.

All data were gathered across identical network conditions on a low-traffic, high-bandwidth, low-latency network in conditions which make detection easier. Results for many tests would be less discriminating in a network with more friendly characteristics. For each dataset, data gathering was enabled and a 50Mb file was downloaded 30 times consecutively or until the entire message (32 bytes, including checksum) was sent. This was repeated 10 times per algorithm, as well as for an unmodified TCP stack. All bar graphs use means for their heights and standard deviations for their error bars.

In each legend, Norm is normal traffic, VOrg is DEVCC, V1 is Half-DEVCC, V4 is Timestamp-Masked, V5 is Timestamp-Independent, and V8 is Bernoulli.

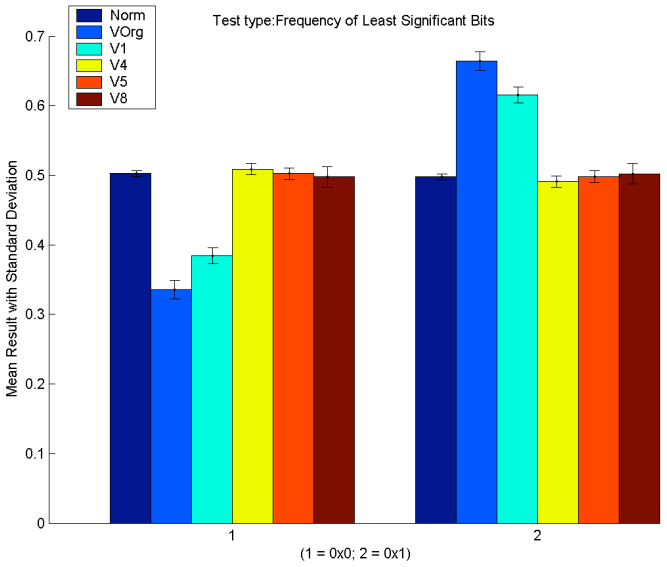

Figure 6: Frequency analysis on the least significant bit

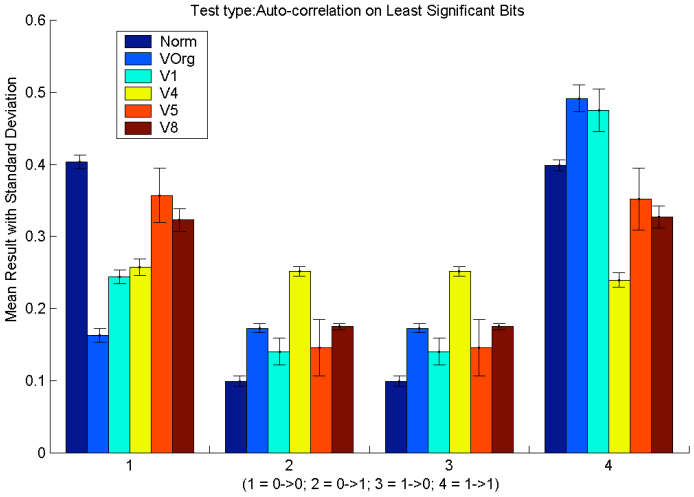

Figure 7: Auto-correlation analysis on the least significant bit

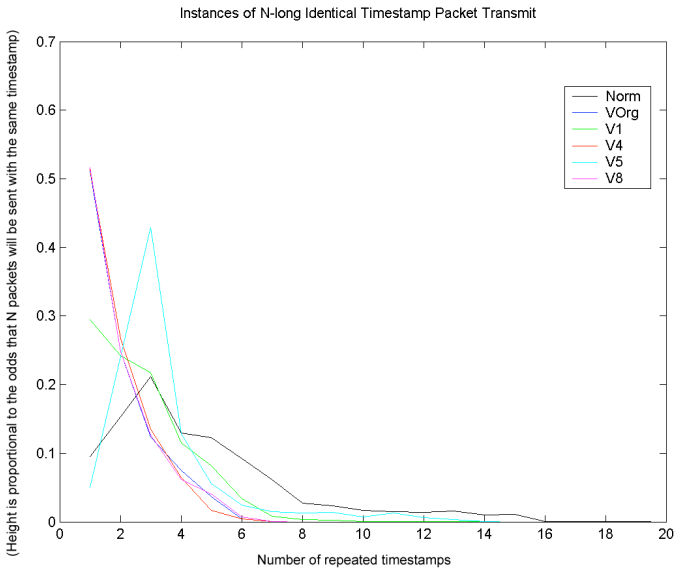

Figure 8: Unique timestamp count binning

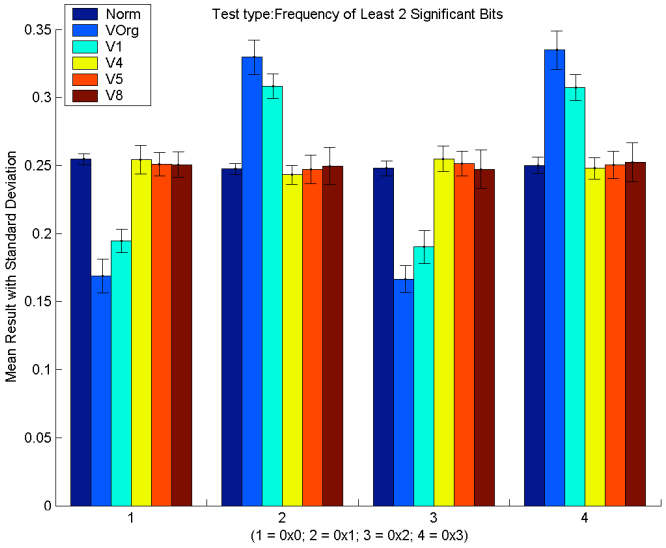

Figure 9: Frequency analysis on the least 2 significant bits

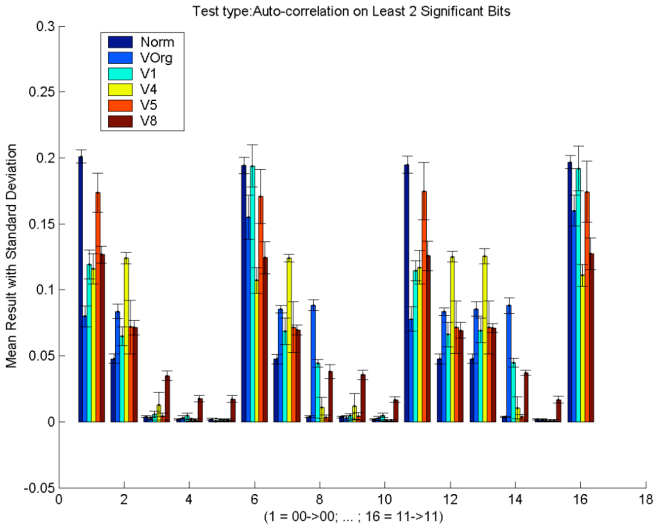

Figure 10: Auto-correlation analysis on the least 2 significant

bits

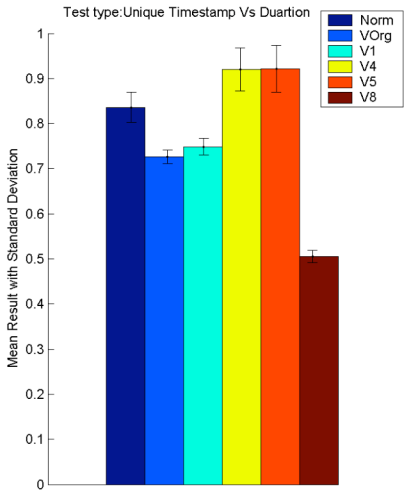

Figure 11: Unique timestamp usage to total timestamp count ratio

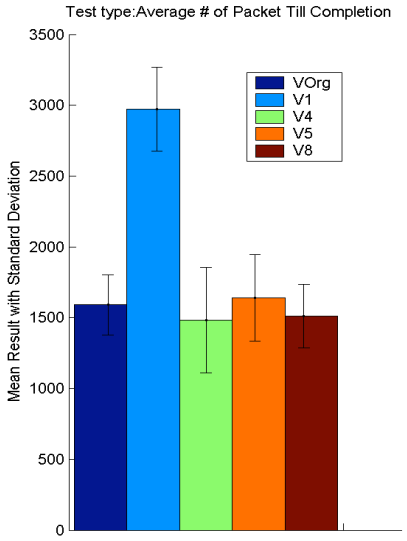

Figure 12: Average number of packets till complete message transmission

VIII. Future Considerations

A. *The RTT Problem*

Unfortunately, all of the algorithms presented in this paper suffer from the same problem; an unusual RTT pattern. This pattern is trimodal for DEVCC and for Half-DEVCC, bimodal for Timestamp-Masked and Timestamp-Independent, and giving a curve instead of a spike for Bernoulli, and in each case it is simple to detect on a relatively stable network given even a moderately small number of packets. While some conditions such as the watermarking scheme described in [5] may obfuscate this, it still exists as a general problem, and how to best solve it without impacting network performance or the number of packets per message transmission is an area for future research.

B. *Send Delaying*

We were constrained by the monotonicity of time and the 10ms granularity of the system clock for each of the algorithms described. Delaying the transmission of each outgoing packet by the duration of one timestamp tick would allow us to send subsequent packets one earlier as well as one later, effectively allowing us to go back in time. This would allow the unique timestamps to duration ratio to move closer to 1:1, and would allow autocorrelation numbers to more closely approach those of real traffic (depending on the algorithm, this may apply solely to two-or-more bit autocorrelation or to all). Unfortunately this would decrease latency on all connections by 10ms, as slowing it on only the connection carrying the covert channel would be trivial to detect.

C. *Bit Density*

We can reduce the rate of usage of a given packet in a variety of ways. This allows us to arbitrarily closely approach real traffic patterns, at the cost of decreased bit/packet density.

D. *Duplicate Bits*

Since each of the algorithms presented in this paper is based on a probabilistic data transfer model, we do not ever know if the entire message has been received, or even which bits have currently been received. Because of this, duplicate bits are frequently sent. This cannot be solved without using a two-way channel, however thankfully TCP allows this, and in fact supports it already; both the ACK number and the optional timestamp reply field allow us to determine whether some bits have been received, without any modification of the receiving system – though this will only work if the intended recipient is the receiver, and not if they are a point in between.

E. *Applications*

It is entirely feasible to transmit data at slightly-slower than typing speed on a fast network using a covert channel like this, especially if the minimum timestamp delta is smaller than 10ms. In particular, a 100Mbps Ethernet connection can easily transmit far more than 100 packets a second, which gives over 99.998% probability of correctly sending a byte during a one second interval. This is not fast enough to transmit streaming audio, but is certainly fast enough for concise instant messaging.

IX. Conclusion

An effective covert channel must not be detectable in any way, therefore a single failed test is sufficient to discard an algorithm. In high speed conditions, only the Timestamp-Independent algorithm is satisfactory. In low speed situations, however, each of the Timestamp-Masked, Timestamp-Independent, and Bernoulli algorithms will approximate normal traffic closely enough as to be statistically indistinguishable. The RTT problem, however, makes any of these detectable in a stable low-load network, so these should not be considered currently useable except in appropriate conditions. Decreased bit density and avoiding resending data

are both promising areas for improvement.

REFERENCES

[1] G. Fisk, M. Fisk, C. Papadopoulos, and J. Neil, "Eliminating Steganogrphy in Internet Traffic with Active Wardens," presented at the 5th International Workshop on Information Hiding, Oct. 2002

[2] S. J. Murdoch, and S. Lewis, "Embedding Covert Channels into TCP/IP," presented at the 7th International Workshop on Information Hiding, June 2005

[3] K. Szczypiorski, "HICCUPS: Hidden Communication System for Corrupted Networks," presented at the International Multi-Conference on Advanced Computer Systems, 2003

[4] J. Giffin, R. Greenstadt, P. Litwack, and T. Tibbetts, "Covert Messaging in TCP," Privacy Enhancing Technologies, vol. 2482, pp. 194-208, 2002

[5] X.Y. Wang, S. Chen, and S. Jajodia, "Tracking Anonymous Peer-to-Peer VoIP Calls on the Internet," in Proc. *12th ACM Conference on Computer Communications Security (CCS 2005)*, November 2005

[6] D. Sanghi, A. K. Agrawala, O. Gudmundsson, and B. N. Jain, "Experimental Assessment of End-to-End Behavior on Internet," *Computer Science Technical Report Series*, vol. CS-TR-2909, 1992

[7] H. Ohaski, M. Murata, and H. Miyahara, "Modeling End-to-End Packet Delay Dynamics of the Internet Using System Identification," in Proc. 17th International Teletraffic Congress, pp. 1027-1038, Dec. 2001