A Reliable Covert Communication Scheme Based on VoIP Steganography

Harrison Neal and Hala ElAarag(✉)

Department of Mathematics and Computer Science, Stetson University, DeLand, FL, USA
{hneal, helaarag}@stetson.edu

Abstract. Steganography is the science of hiding information in such a way that an adversary wouldn't know it existed. A significant amount of research has been done in this field for non-real time mediums. Research on real time mediums, for example Voice over Internet Protocol (VoIP), isn't as mature. In this paper, we propose an algorithm that enables data hiding in G.711, the most commonly used voice codec for VoIP devices, while gracefully handling packet loss. This would allow two telephone users to covertly transfer multiple pieces of arbitrary information between their respective systems in a reliable manner. We use important performance metrics to evaluate our algorithm, namely, throughput, noise-to-signal ratio and the Perceptual Evaluation of Speech Quality algorithm. We demonstrate that our algorithm performs well compared to other algorithms proposed in the literature in real world environments, where packet loss is inevitable, by maintaining high throughput and good speech quality.

Keywords: VoIP · G.711 · Steganography · Covert communication

1 Introduction

Steganography is the science of hiding messages so to appear as if they don't exist. This is usually achieved by either modifying part of a communication that wasn't intended to hold meaningful data for the user, or by subtly modifying meaningful data to contain a second message [1]. Many early papers on steganography focused on hiding secret information in images, videos and audio. With the rapid expansion of the Internet, later research described hiding data in protocol headers [2]. As interest in steganography has increased, research on steganography within VoIP streams and other real-time media has also increased. One example of recent research in this area is a method by Ito, Abe and Suzuki [3], which uses G.726 in tandem with G.711 to determine a tolerable amount of distortion in the G.711 stream. In their paper, Ito et al. [14] neglected to mention scenarios involving packet loss. As G.726 is a stateful codec, the output is dependent on all previous and current inputs. Should a packet be lost, the state and outputs between the sender and receiver are no longer guaranteed to be equivalent. Without this guarantee, hidden data can no longer be reliably retrieved from the stream. We propose an algorithm that uses a similar technique to that of Ito et al. [14], but allowing for continued reliable retrieval from the stream after a packet has been lost. Some preliminary results have been published in [19, 20]. The rest of the paper is organized as follows. In Sect. 2, we give a background about the G.711 codec. Section 3 presents the related work in the literature, while Sect. 4 explains our proposed packet loss tolerant algorithm. In Sect. 5 we demonstrate the evaluation of our algorithm. Finally, we conclude our paper in Sect. 6.

2 Background

G.711 is the oldest voice codec in use not only by today's VoIP systems but also domestic public switching telephone networks (PSTNs) [5]. Some standardization authorities consider G.711 mandatory for VoIP devices as a codec common to all systems for interoperability [6]. In regards to steganography, there appears to be a much greater wealth of available research on G.711 as opposed to other voice codecs. For these reasons, we opt to have our VoIP client use only G.711.

$$\begin{gathered}
\text{s1abcdefghijkl} \rightarrow \text{s111abcd}\\
\text{s01abcdefghijk} \rightarrow \text{s110abcd}\\
\text{s001abcdefghij} \rightarrow \text{s101abcd}\\
\cdots\\
\text{s0000001abcdef} \rightarrow \text{s001abcd}\\
\text{s0000000abcdef} \rightarrow \text{s000abcd}
\end{gathered}$$

Fig. 1. Output of G.711 codec from signed linear audio input, prior to inversion of every other bit

As illustrated in Fig. 1, the G.711 codec takes signed linear audio samples at 8 kHz [7]. For each sample, the codec outputs one byte (eight bits); this results in a 64 kbits/s bit rate. There are three components in the output: the sign (positive or negative, represented by s in Fig. 1), the magnitude on a logarithmic scale and a sub-magnitude on a linear scale (represented by abcd in the figure). The first bit of output is the sign bit and is identical to the sign bit from the input. The second through fourth output bits represent the magnitude of the input on a logarithmic scale. The codec checks the linear sample input for how many "0" bits follow the sign bit before a "1" bit is found, with more "0" bits before the first "1" implying a lesser magnitude. If a "1" bit immediately follows the sign bit, the codec adds "111" to the output. If "01" immediately follows the sign bit, the codec adds "110" to the output, signifying a lesser magnitude than if "1" immediately followed the sign bit. This pattern continues until the case where at least seven consecutive "0" bits follow the sign bit, which would result in an output of "000" for logarithmic magnitude. The remaining 4 bits of output are set to the first four bits in the input that follow the bits used to determine logarithmic magnitude. In transit, systems typically invert every other bit – xor 01010101 (0 × 55) – this is not shown in the figure for simplicity.

3 Related Work

Much work has been done in regards to steganography. They could be classified into two categories. One hides data within TCP/IP packets and the other hides data within the application. The first category allows great flexibility in trying to secretly transmit data.

Ahsan and Kundur [8] showed some simple approaches to manipulating headers in IP packets and reordering packets in such a way that data could be encoded secretly. Reordering packets, while certainly acceptable in many scenarios, would likely create too much jitter for real time communications. Murdoch and Lewis [2] noted that prior work had evaluated the ability for one to use a given TCP or IP field for steganographic purposes, but had neglected to properly evaluate if such manipulation might be more obvious to active wardens. These are people who are familiar with typical TCP traffic on their network, have reason to suspect steganography is taking place and are actively inspecting traffic to look for deviations from the expected typical traffic. Murdoch and Lewis [2] then showed a steganographic approach using initial sequence numbers that takes into consideration methods used by popular operating systems and that would be indistinguishable from packets normally generated by those hosts. This, however, lacks suitability for our purposes, as VoIP typically utilizes UDP. Further, the throughput that could be achieved by only modifying a single packet over the lifetime of a connection is, while incredibly covert, insufficient for our intentions.

In the second category, research has been proposed to hide information on the application level with a focus on audio applications such as those running VoIP. Mazurczyk and Lubacz [9] suggested a method they dubbed Lost Audio paCKets steganography, or LACK. LACK functions by intercepting audio packets at a given interval, replacing the audio payload with data to be transmitted covertly, holding the packet for an interval long enough for a receiver to consider it lost, optionally adding additional jitter to avoid being caught by a simple statistical analysis, and finally sending the modified packet. The interval at which LACK intercepts packets should be based on which audio codec is in use and what rate of packet loss for that codec would result in unacceptable quality; that is to say, LACK is adjustable for any given codec. Optionally, this interval can also be adjusted based on the expected call duration (which can be estimated based on statistics and refined as the call progresses) and amount of data needed to be sent. Mazurczyk [10] thereafter tested the method, showing that G.711 appeared best suited for use with LACK, both due to it having the highest bit rate amongst the codecs tested and it encountering the least degradation of quality as the ratio of lost to total packets increased. At a packet interception rate of 5 %, which would probably lure the attention of someone observing traffic, G.711 still maintained fair mean opinion scores on listening quality objective (MOS-LQO) while achieving 3.2 kbits/s of covert traffic. There is a concern that a mechanism should be in place to ensure packets arriving late on purpose and packets arriving late due to other reasons beyond our control can be differentiated; the jitter introduced in the interval at which packets will be delayed and modified could be produced by a pseudorandom number generator (PRNG) with a seed known to both parties. There is also a concern that a mechanism needs to exist for recovering lost steganographic packets. Additionally, there is an expectation that there is a single static message of known length, which is used during the optional step of regulating the interception interval based on expected call duration and message length. Hamdaqa and Tahvildari [11] suggest ReLACK, which operates in very similar fashion to LACK, but uses a modified version of Shamir’s Secret Sharing Scheme [12] on the message being transmitted, increasing the message size to the degree necessary for a specified fault-tolerance, which could be the amount of data that can be comfortably transmitted with LACK without unacceptable quality loss. With this scheme applied, if the receiver obtains at least as much data as was in the original message, the original message can be reconstructed, which addresses the possible issue of lost data. However, if the receiver doesn't obtain at least as much data as was in the original message, the entire message is lost.

Other methods in the literature attempt to subtly manipulate audio data as opposed to manipulating the flow of that audio data across the network and outright replacing the audio. One of the simplest approaches to steganography for many mediums is to tamper with the least significant bit (LSB) of each unit of data [21]. The sender simply replaces the LSB of a unit with the desired hidden bit to send, and the receiver reads the sent LSB. When using the LSB method, you expect that the modification of the least significant portion of the data will be negligible and go unnoticed by any observing parties.

An approach by Aoki [16] works in similar fashion to LSB in a special case. G.711 can transmit both a +0 and −0 signal, and Aoki's [16] algorithm takes advantage of this by using the sign bit as least significant when the magnitude is 0. This technique is virtually lossless in terms of audio quality, but quickly becomes ineffective in areas with moderate background noise. Additionally, Aoki [16] created a semi-lossless method, which works by increasing the absolute magnitude of all non-zero samples by a variable amount, j, allowing zero-magnitude samples to have both their sign bit and true LSBs manipulated for storing hidden data.

Among recent literature, both the method by Miao and Huang [13] and the method by Ito et al. [14] appear promising. While throughput of secret data would be dependent on many factors, including the nature of the audio being used as cover traffic, both of these methods advertise fairly good throughput. Miao and Huang [13] created an approach that isn't related to LSB-tampering. Their approach works by embedding more data in groups of samples that vary wildly (rapid fluctuations in sound samples) and less data in groups of samples that remain consistent and where distortions might be more apparent. For each group of N samples, the approach treats each G.711 sample as a sign bit followed by a seven bit magnitude integer, and obtains the average for the N samples. It then, for all but one sample, determines the difference between that sample and the average, and plans to embed a number of hidden bits equal to the log of the absolute value of the difference in that sample, rounded down. The sample will then be modified to equal the average plus an altered difference that will contain the hidden bits. The altered difference will be the sum of two numbers: some integer with a bit length equal to the number of bits to hide, plus two raised to the number of bits to hide. Finally, the one sample that was originally excluded will be manipulated to restore the average signed magnitude of the group back to the average originally calculated before anything was modified. This allows for a receiver to properly determine the number of bits that would be hidden in each sample, and subtract the correct average to recover the hidden integer in each sample.

The approach by Ito [14] uses a lower bit rate codec, G.726, in conjunction with G.711. The algorithm tests how many least significant bits of output from G.711 can be freely manipulated before transcoding to the lower bit rate codec begins producing different results. This assures the quality of the G.711 samples will at least match that of the lower bit rate codec. As both the sender and receiver for a given audio stream can use the same method to determine how many bits can be tampered with freely before unacceptable degradation would result, both the sender and receiver would come to the same conclusion and retrieve the correct tampered bits.

4 Packet Loss Tolerant Algorithm

In this section, we propose a new data hiding algorithm that is based on the algorithm proposed by Ito et al. [14] but has two main improvements in terms of reliability and throughput. The method suggested by Ito et al. [14] neglected to account for packet loss. As the lower bit rate codec Ito [14] used maintained state (that is, all inputs prior to the current sample being processed can affect the current output), it assumed that state would always match at the sender and receiver. If a scenario included packet loss as a feasible possibility, the sender and receiver would no longer have a guarantee of identical states if the entire audio stream was being considered. That is to say, should any packet loss occur when using the algorithm by Ito et al. [14], any hidden data starting with the first lost packet and extending to the end of the stream would be lost. As such, our proposed algorithm performs the information hiding on a packet-per-packet basis (resetting the state of the codec with each packet) as opposed to the entire stream of audio.

Our packet loss tolerant algorithm consists of three main functions. A function, tamperableBits, that determines the number of bits that could be tampered with without significantly affecting the quality of the audio, a function for embedding secret information into an outgoing audio stream and another for retrieving embedded secret information from an incoming audio stream. The pseudo code for determining a reasonable number of tamperable bits without significant distortion for a given sample is shown in Fig. 2. tamperableBits takes in a G.711 sample and a G.726 state. It outputs an updated G.726 state and the number of bits that can be tampered in the G.711 sample provided. The function named processCodec takes in a G.711 sample, along with the state of the G.726 codec, and outputs both an updated G.726 state and a G.726 output sample. In tamperableBits, the sample is first transcoded to the lower-bit rate codec without alteration. If the absolute value of the transcoding output is as high as possible (given the signed integers that can be represented with the number of bits used per sample by the lower-bit rate codec), this may suggest an overflow has occurred; that is, that the distortion between the expected value and the actual value is too large to be represented. If this is the case, the output from the lower-bit rate codec can't be used to judge what samples are similar, and the safe option is to assume no bits should be tampered. If this isn't the case, two copies of the sample are made. The least significant bit is set to 1 on the first copy and cleared (set to 0) on the second copy, we then test if transcoding both altered copies produce the same output. If so, the least significant bit can be tampered, and this check can be repeated for more significant bits in the linear sub-magnitude.

Pseudocode for embedding secret information into an outgoing audio stream is shown in Fig. 3. Prior to sending a VoIP packet with audio, it should be intercepted. First, a new state should be constructed for the lower-bitrate codec. Then, for every sample, the number of bits that can be freely tampered is determined by the tamperableBits algorithm explained above, and the bits are replaced.

```
// This function should take in a g711 sample prepared for transmission
// (every other bit inverted – 0x55) and the state for a lower-bit rate codec.
// It should output a sample transcoded with the lower-bit rate codec, and a
// modified state for the lower-bit rate codec.
function **processCodec**: IN g711, INOUT state, OUT codecOutput

// This function should take in a g711 sample not prepared for transmission
// and the state for a lower-bit rate codec. It should output the number of bits
// that can be freely modified before it would affect the output from transcoding
// to the lower-bit rate codec, along with the new state of the codec after
// processing a sample representing the lowest modified sample possible.

function tamperableBits: IN sample, INOUT codecState, OUT bits {
        bits ← 0
        lowTamper ← bitwise sample xor 0x55
        highTamper ← bitwise sample xor 0x55
        mask ← 1
        lastLowState ← codecState
        checkMaxDistort ← call processCodec: g711 ← lowTamper, state ↔ lastLowState
        if (absolute value of checkMaxDistort is less than the highest possible) {
                do {
                        lowTamper ← bitwise lowTamper xor 0x55
                        highTamper ← bitwise highTamper xor 0x55
                        lowTamper ← bitwise lowTamper and not mask
                        highTamper ← bitwise highTamper or mask
                        lowTamper ← bitwise lowTamper xor 0x55
                        highTamper ← bitwise highTamper xor 0x55
                        mask ← mask bitwise shifted left 1
                        stateCopy ← codecState
                        highResult ← call processCodec: g711 ← highTamper, state ↔ stateCopy
                        stateCopy ← codecState
                        lowResult ← call processCodec: g711 ← lowTamper, state ↔ stateCopy
                        if (lowResult doesn't equal highResult) {
                                break do-while
                        }
                        lastLowState ← stateCopy
                        bits ← bits + 1
                } while (bits is less than 4)
        }
        codecState ← lastLowState
}
```

Fig. 2. Method to determine number of bits that can be manipulated

The pseudo code for retrieving embedded secret information from an incoming audio stream is similar. For each sample the number of bits that could be freely tampered is determined then extracted as illustrated in Fig. 4.

Using the steganography technique we proposed, we can embed secret data of our choosing into the G.711 stream. Once the audio samples have been modified to contain our information, multiple samples can be bundled into each Real-Time Protocol (RTP)

```
// Should return the next bit of data to be secretly embedded.
function **nextBitToInsert**: OUT *bit*

// To be run when packets are available:
while (a new packet is available) {
        *samples* ← G.711 samples from the packet (every other bit inverted)
        *state* ← new state for low bitrate codec
        for (each *sample* in *samples*) {
                *noxmit* ← bitwise *sample* xor 0x55
                *bits* ← call **tamperableBits**: *sample* ← *noxmit*, *codecState* ↔ *state*
                *mask* ← 1
                for (*i* from 1 to *bits*) {
                        *noxmit* ← bitwise *noxmit* and not *mask*
                        *insert* ← call **nextBitToInsert**
                        *insert* ← *insert* * *mask*
                        *noxmit* ← bitwise *noxmit* or *insert*
                        *mask* ← *mask* bitwise shifted left 1
                }
                *sample* ← bitwise *noxmit* xor 0x55
        }
}
```

Fig. 3. Method to embed secret data into an audio packet

packet as normal by the VoIP software and sent to their destination. RTP typically uses UDP, so we risk not receiving a packet either due to a packet not arriving or a packet arriving with an invalid checksum.

5 Evaluation

We collected several audio recordings for testing. These audio recordings were grouped into three categories: no voice with low background noise, high volume voice with low background noise and high volume voice with moderate background noise. The no voice with low background noise category had two recordings. The first recording was a muffled thunderstorm recorded in a sealed building, with volume comparable to light background noise. The second was computer-generated silence. The high volume voice with low background noise category included automated voice mail and operator prompts (VM prompts), and English as a second language study material (ESL). Finally, the high volume voice with moderate background noise category included our peers speaking a short story (Peers) and recordings of telephone calls considered of historical significance and publically available (Historic).

To test the algorithms, we used three performance metrics:

1. **Throughput**: which is the number of bits of the secret data embedded per second.
2. **Noise-to-Signal ratio**: which is calculated according to the following equation,

```
// Should receive the next bit secretly transmitted for processing.
function **nextBitInserted**: IN *bit*

// To be run when packets are available:
while (a new packet is available) {
    samples ← G.711 samples from the packet
    state ← new state for low bitrate codec
    for (each sample in samples) {
        noxmit ← bitwise sample xor 0x55
        bits ← call tamperableBits: sample ← noxmit, codecState ↔ state
        mask ← 1
        for (i from 1 to bits) {
            recovered ← bitwise noxmit and mask
            if (recovered equals mask) {
                call nextBitInserted: bit ← 1
            } else {
                call nextBitInserted: bit ← 0
            }
            mask ← mask bitwise shifted left 1
        }
    }
}
```

Fig. 4. Method to retrieve secret data embedded into an audio packet

$$\frac{\sum_{i=1}^{n}\left\{\left(\frac{A_N}{A_S}\right)^2\right\}}{n} \tag{1}$$

Where:

- S denotes the Signal
- N denotes the Noise
- A_S is the original linear amplitude of a given sound sample (G.711 has 8,000 samples per second)
- A_N is the maximum difference between the amplitude of the original sample and the sample after embedding the data on a linear scale
- n is the number of samples

3. **Perceptual Evaluation of Speech Quality (PESQ)**: PESQ algorithm [15] compares unmodified and degraded audio in a way that aims to report how degraded a human would perceive the audio to be. Higher scores are better, with scores of at least 4 considered good and scores of at least 3 considered acceptable but with noticeable degradation.

A higher throughput, a higher PESQ and a lower Noise-to-Signal ratio means better performance.

We first studied the performance of the algorithms in the worst case scenario, where embedding data would result in the greatest possible noise-to-signal ratio.

For Aoki's [16] approach, we evaluate it in lossless (LL) and semi-lossless (SLL) modes. In the former, only the sign bit is manipulated when the magnitude is 0; in the latter, the sign bit and the least significant bit of the magnitude is manipulated i.e. the variable j in [16] is set to 1.

For the algorithm by Ito et al. [14], G.726 could operate at four different bitrates, namely, 16 kbits/s, 24 kbits/s, 32 kbits/s and 40 kbits/s. Using the 32 kbits/s mode offered more throughput than the 40 kbits/s mode without much additional noise. Using 24 kbits/s and 16 kbits/s compared to 32 kbits/s again offered additional throughput, but with a substantial noise increase. We will show results for 32 kbits/s and 24 kbits/s for both the algorithm by Ito et al. [14] and our algorithm.

For the approach by Miao and Huang [13], we use values of 5 and 13 for N, and a value of 96 for the maximum lambda. 13 was the N value used in [13] when presenting results of their algorithm. Their paper suggested that lowering the value of N would increase hidden throughput at the cost of noise; we chose an N of 5 to see this effect. The maximum lambda serves as a mechanism to prevent overflow – should any embedding operation have the potential to cause a sample to vary from the average magnitude more than the maximum lambda, no bits will be embedded in that sample. Should the maximum lambda be exceeded when adjusting the sample used to restore the average, all embedding operations for the entire group will be canceled, and the audio for the entire group will be unmodified.

5.1 Throughput

Figures 5, 6 and 7 show the average throughput of secret data for no voice with low background noise recordings, the high volume voice with low background noise and the high volume voice with moderate background noise categories, respectively. The Figures assume a no packet loss scenario. Aoki's [16] algorithms perform well under

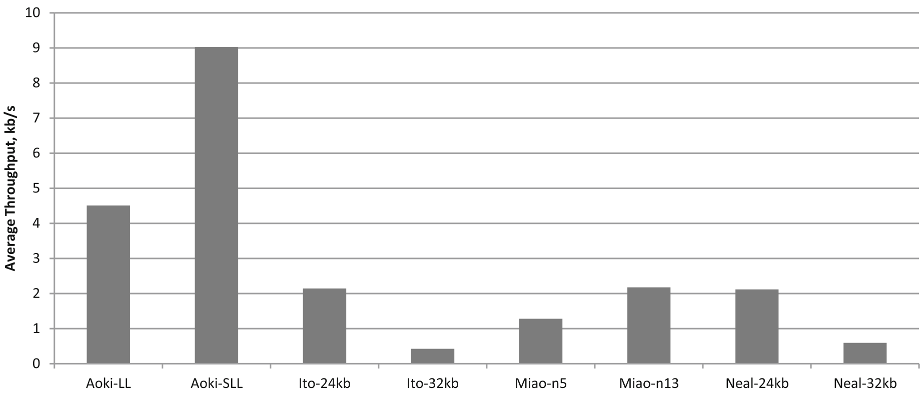

Fig. 5. Average throughput for no voice low background voice recordings

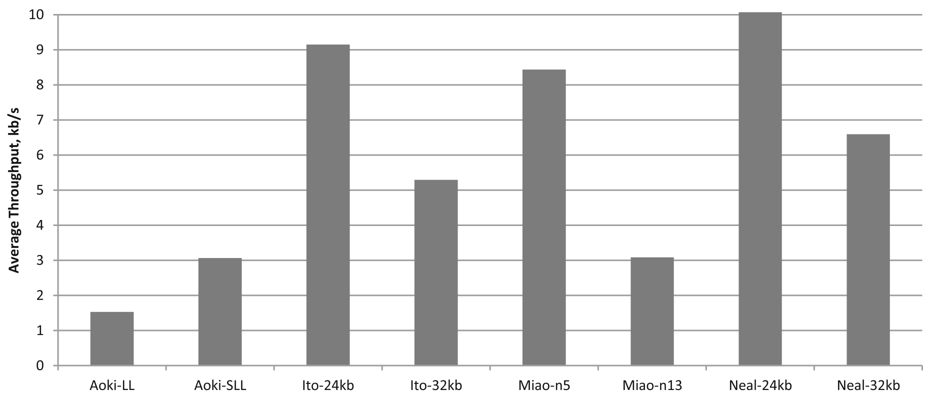

Fig. 6. Average throughput for high voice volume, low background noise recordings

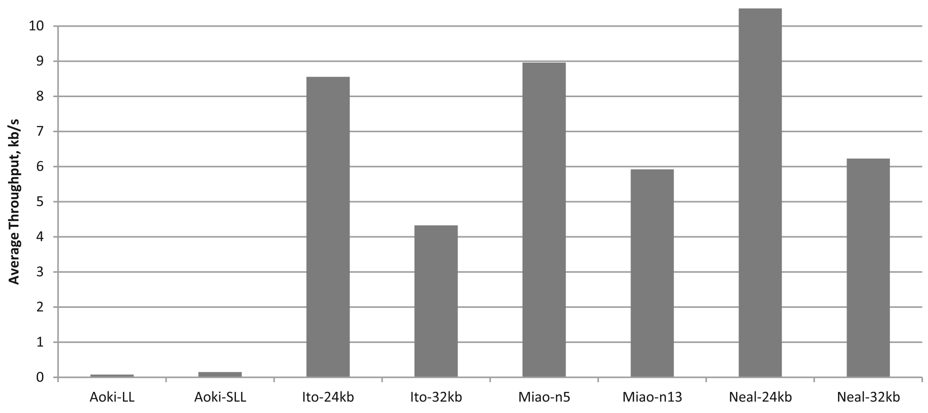

Fig. 7. Average throughput for high voice volume, moderate background noise recordings

no voice and low background noise conditions but poorer under high volume voice conditions. Our algorithm operating at 24 kb/s, outperforms other algorithms under high volume voice conditions, but poorly with generated silence. For Miao and Huang's [13] algorithm, a group of five samples (that is, N = 5) provided better performance than N = 13 for all recordings with the exception of Thunderstorm; for the Thunderstorm recording, N = 13 provided much better performance compared to N = 5.

5.2 Noise to Signal Ratio

In Fig. 8, we show average Noise-to-Signal ratios on modified audio files. The method by Miao and Huang [13] generated the most noise in every instance, with N = 13 generating more noise than N = 5. As N = 5 also provides better throughput in all but one case as well as less noise in all cases, the suggestion by Miao and Huang [13] in their paper to keep N small appears valid. In both recordings from the high volume voice and moderate background noise category (peers and historic), the algorithm by Ito et al. [14] and our algorithm generates more noise than Aoki's [16], but, as shown in Fig. 7, Aoki's generates substantially less throughput. In the high volume voice and low background noise category (VM prompts and ESL), our algorithm generated more noise than Aoki's for VM recording and less noise in the ESL recording, but for both recordings Aoki's [16] algorithm generated less throughput (see Fig. 6). In the no voice with low background noise category, Aoki's [16] algorithm generated more noise than ours, but generated substantially more throughput (see Fig. 5). Neither Ito et al.'s algorithm [14] nor our algorithm made any changes for the silence recording, hence not having a data point in Fig. 8.

5.3 Audio Quality

In Fig. 9, we show PESQ results. Aoki's [16] algorithms score at least 3.0 in all cases and above 4.0 in recordings from the high voice volume with moderate background noise category, suggesting acceptable to good audio quality. Both our and Ito's algorithm consistently score above 4, suggesting consistent good quality. Miao and Huang [13] 's method produced modified audio that scored less than 3.0 for multiple (though not all) recordings, suggesting poor to adequate quality.

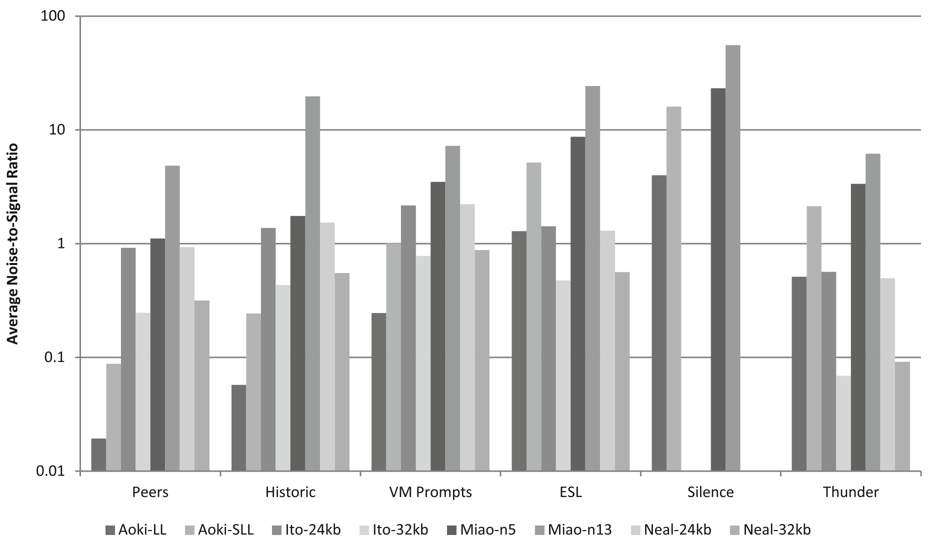

Fig. 8. Average noise-to-signal ratio for recordings

5.4 Effects of Packet Loss

In a more practical scenario where packet loss can occur, the algorithm by Ito et al. [14] quickly levels off and fails to reliably transmit any further data, while our algorithm continue transmitting with graceful degradation as shown in Fig. 10.

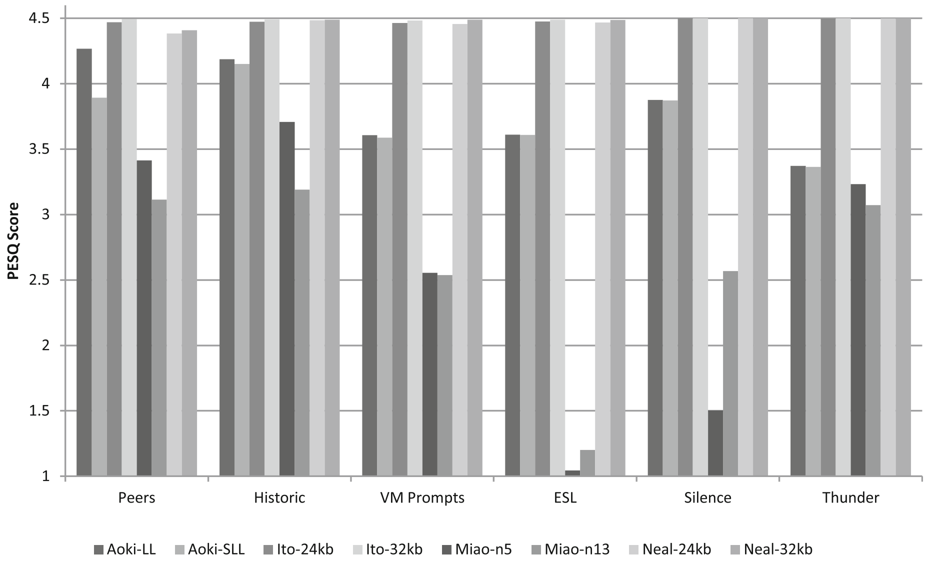

Fig. 9. PESQ scores for recordings

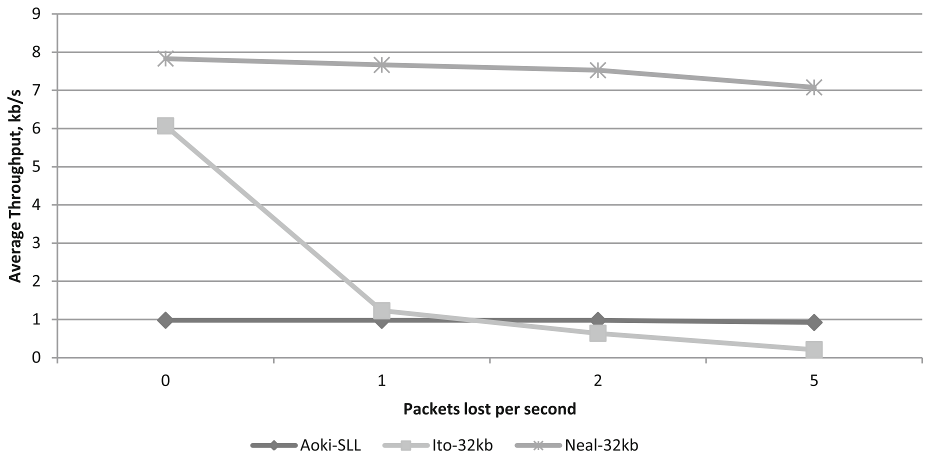

Fig. 10. VM Prompts Sample considering packet loss for 5 s

6 Conclusion and Future Work

In this paper we not only proposed a new algorithm to hide data in a Voice over IP stream but also evaluated several suggested algorithms in this field from the literature, namely those by Aoki [16], Miao and Huang [13] and Ito et al. [14]. As far as our knowledge, this research offers the most comprehensive performance analysis of VoIP steganography algorithms as far as the number of algorithms considered and the performance metrics used. Our evaluation shows that the algorithm by Aoki [16] worked nicely in no voice conditions but far less favorably otherwise compared to other algorithms. On the other hand, other methods do not produce any throughput with artificial silence, while Aoki's [16] method excels. The method by Miao and Huang [13] tends to perform better with a lower N value but generates excessive noise.

We proposed an algorithm based on the work of Ito et al. [14] that dealt with packet loss more appropriately. Our experiments show that our proposed algorithm has several advantages over algorithms found in the literature. The most important advantage is that, unlike other algorithms, it can be used in a practical environment where packet loss is inevitable. It maintains high throughput, low noise levels and high PESQ scores. That is, it maintains a good audio quality on par with or superior to other algorithms found in the literature, in addition, it gracefully degrades as packet loss rate increases.

One drawback of the proposed method is that, similar to all other LSB based steganography techniques, it can be detected by a good steganalysis program.

As VoIP conversations can involve both voice and video, one could extend this research to embed the data not only into the audio stream but also the video stream for a given call, if applicable.

References

1. Artz, D.: Digital steganography: hiding data within data. IEEE Internet Comput. **5**, 75–80 (2001)
2. Murdoch, S.J., Lewis, S.: Embedding covert channels into TCP/IP. In: Barni, M., Herrera-Joancomarti, J., Katzenbeisser, S., Pérez-González, F. (eds.) IH 2005. LNCS, vol. 3727, pp. 247–261. Springer, Heidelberg (2005)
3. Tian, H., et al.: An adaptive steganography scheme for voice over IP. Huazhong University of Science & Technology, University of Nebraska, Tsinghua University (2009) doi:10.1109/ISCAS.2009.5118414
4. Yongfeng, H., Bo, X., Honghua, X.: Implementation of covert communication based on steganography. Department of Electronic Engineering, Tsinghua University, Beijing (2008). doi:10.1109/IIH-MSP.2008.174
5. Karapantazis, S., Pavlidou, F.-N.: VoIP: a comprehensive survey on a promising technology. Thessaloniki (2009). doi:10.1016/j.comnet.2009.03.010
6. International Telecommunication Union: Packet-based multimedia communications (Recommendation ITU-T H.323) (2009)
7. International Telecommunication Union: Pulse Code Modulation (PCM) of Voice Frequencies (ITU-T Recommendation G.711) (1993)
8. Ahsan, K., Kundur, D.: Practical data hiding in TCP/IP. University of Toronto, Toronto (2002). 1-58113-000-0/00/0000
9. Mazurczyk, W., Lubacz, J.: LACK - a VoIP steganographic method. Institute of Telecommunications, Warsaw University, Warsaw (2009). doi:10.1007/s11235-009-9245-y
10. Mazurczyk, W.: Lost audio packets steganography: the first practical evaluation. Warsaw University of Technology, Institute of Telecommunications, Warsaw, arXiv:1107.4076v1 (2011)
11. Hamdaqa, M., Tahvildari, L.: ReLACK: a reliable VoIP steganography approach. In: IEEE Fifth International Conference on Secure Software Integration and Reliability Improvement, Jeju Island, pp. 189–197 (2011)

12. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)
13. Miao, R., Huang, Y.: An approach of covert communication based on the adaptive steganography scheme on voice over IP. Department of Electronic Engineering, Tsinghua University, Beijing (2011). ISBN: 978-1-61284-231-8
14. Ito, A., Abe, S., Suzuki, Y.: Information hiding for G.711 speech based on substitution of least significant bits and estimation of tolerable distortion. Tohoku University, Sendai (2009). ISBN: 978-1-4244-2354-5
15. International Telecommunication Union: Perceptual evaluation of speech quality (Recommendation ITU-T P.862) (2001)
16. Aoki, N.: A band extension technique for G.711 speech using steganography. IEICE Trans. Commun. **E89-B**(6), 1896–1898 (2006)
17. International Telecommunication Union: Perceptual objective listening quality assessment (ITU-T Recommendation P.863) (2011)
18. CenturyLink: CenturyLink IP Network Statistics, December 2011. https://kai02.centurylink.com/PtapRpts/Public/BackboneReport.aspx
19. Neal, H., ElAarag, H.: A packet loss tolerant algorithm for information hiding in voice over IP. In: Proceedings of IEEE Southeast Conference, Orlando, FL, 15–18 March 2012
20. ElAarag, H., Neal, H.: Performance analysis of current data hiding algorithms for VoIP. In: Proceedings of the Communication and Networking Simulation Symposium, Spring Simulation Multiconference, San Diego, CA, 7–10 April 2013
21. Latzenbeisser, S.: Information Hiding Techniques for Steganography and Digital Watermarking. Artech House, New York (2000)