Compression Algorithms for Audio-Video Streaming

Tarif Riyad Rahman
Department of Electrical Engineering and Computer Science (EECS), North South University
Dhaka, Bangladesh
tarif@northsouth.edu, tariftalks@gmail.com

Miftahur Rahman, Member, IEEE
Department of Electrical Engineering and Computer Science (EECS), North South University
Dhaka, Bangladesh
mrahman@northsouth.edu, miftahur_05@yahoo.com

***Abstract*— Live audio-video streaming has become popular over the years especially when it comes to communication between people living huge geographical areas apart and at a very cheap rate or even free. However, when it comes to places where bandwidth is low or limited, smooth streaming has become a serious challenge because after all bandwidth is a natural resource and is limited. Therefore, in this paper we present two algorithms or 'network streaming codecs' that are designed for compressing live multimedia streams to considerable levels of 25% and 50%.**

Keywords: Audio-Video streaming, compression, decompression, codecs, network.

I. INTRODUCTION

This paper presents algorithms that use the least amount of resources in terms of computer memory and bandwidth. Even though this is a well studied problem; the objective of the algorithms is intended for those parts of the world that have the least amount of infrastructure. In developing countries such as Bangladesh it has been seen that the only form of internet access people in remote areas have is through the mobile phone network. However, the network traffic is bursty i.e. there are sharp and severe fluctuations in the bandwidth. We are proposing two algorithms that can compress both audio-video and since they have a linear time complexity they will use up the least amount of bandwidth. Hence, even in the most fluctuating network traffic we can achieve the smoothest possible audio-video conferencing environment. The algorithms have been tested over a simple home network using multimedia capture devices such as the web camera and microphone. Two test algorithms were used as references for comparing with the two new techniques. We took one lossless algorithm called Golomb[5,6] coding which is used in a variety of applications including audio and image compression. We have used another algorithm that is lossy in nature and well known in audio compression as the μ-law[1,4,5] algorithm. We were faced with two dilemmas i.e. recovery of original data after the decompression phase just as Golomb coding performs but with a relatively lower compression ratio or achieve a higher compression ratio but only losing more data just like the μ-law algorithm.

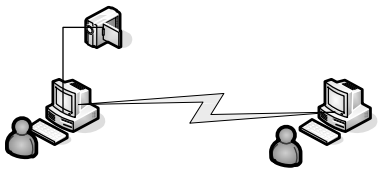

Figure 1 Conferencing showing the described experimental setup.

The experiment was carried out in the following way:

a) The experiment was carried out over a network of two personal computers connected via network cable as depicted in Fig. 1.
b) We assumed that there are only two users; one peer will send the multimedia data and the other will receive it. However, a two way conferencing methodology is also possible as well, using the current setup but was not necessary for experimental purposes.
c) In order for us to capture binary byte data from the devices we employed the [9] Java Media Framework (JMF). JMF allows us to capture binary data from individual frames of a video stream and also from audio streams of time periods that can be set by the programmer.
d) Once the binary data was extracted, we compress the data using the algorithms that form the basis of this paper. The next phase is to send the compressed data over the network to the receiver.
e) Compressed data was sent using socket programming written in [10] Java to the receiver.
f) After the compressed data reaches the receiving peer, the compressed data is decompressed intended for restoration back its original form.
g) The peripheral devices that we used were two types of web-cameras, one having a kilo-pixel resolution and the other having a mega-pixel resolution. In both cases we are capturing RGB frames of varying properties. When using the high resolution camera each frame is of 0.307 megabytes in content size at 15 frames per second. The dimension of the each frame is 640 by 480 pixels. On the other hand when we used a lower resolution camera the configuration was 0.23 megabytes in content size at 15 frames per second. The dimension of each frame is 320 by 240 pixels.

978-0-7695-3973-7/10 $26.00 © 2010 IEEE
DOI 10.1109/ISMS.2010.44

h) We also used a simple microphone for capturing audio samples of 16-bit signed with a sampling frequency of 48 kHz that gives a bit rate of 768 kbps by multiplying the sample size with the sampling frequency as stated in [3].

II. The Compression algorithms

The algorithms share certain properties that make them very similar in nature apart from their differing compression ratios such as they are both asymmetrical i.e. the methods of compression and decompression are not the reverse processes of each other. They are both lossy in nature; losing bits in the process and most importantly both of the algorithms have a linear time complexity. Hence, compressing-decompressing of any sized data would require the minimum possible time. Now let us denote the algorithm that achieves a compression ratio of 50% as ALGO-1, and the other algorithm that has achieved a compression ratio of 25% as ALGO-2.

III. Compression-decompression process of ALGO-1

ALGO-1 is depicted in Fig. 2. It takes two bytes from the multimedia byte streams which are denoted as **Uncompressed Byte-1** and **Uncompressed Byte-2** in the figure. After that the four most significant bits of each of the uncompressed bytes are placed in a compressed byte denoted in the figure as **Compressed Byte**. Initially, we are placing the four most significant bits (a7, a6, a5, a4) of **Uncompressed Byte-1** into the four most significant bit positions (c7,c6,c5,c4) of **Compressed Byte**. We then place the four most significant bits (b7, b6, b5, b4) of **Uncompressed Byte-2** into the four least significant bit positions (c3,c2,c1,c0) of **Compressed Byte-1**. This concludes the compression phase of ALGO-1. The writing manner of the pseudocodes for the procedures of ALGO-1 and ALGO-2 are similar those used in [7].

A compression procedure for ALGO-1 would be:

ALGO-1_COMPRESS(U)

```
C[length[U]/2]
j ←0
for i←0 to length[U]-1
        a←U[i]
        b←U[i+1]
        c←0
        (c7, c6 ,c5,c4) ←(a7, a6, a5, a4)
        (c3, c2 ,c1,c0) ←(b7, b6, b5, b4)
        C[j]←c
j←j+1
i←i+1
return C
```

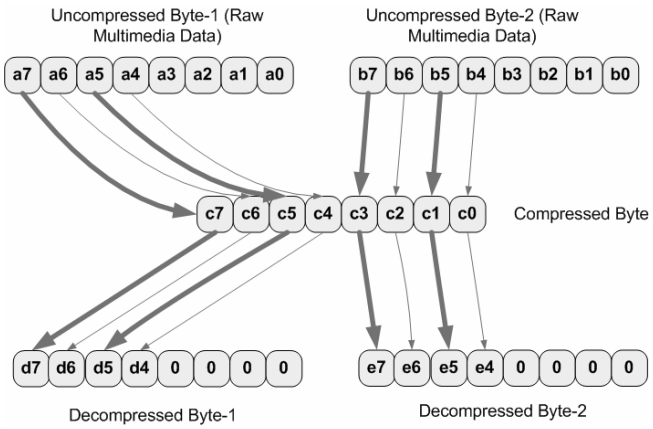

Figure 2 Schematic Diagram of ALGO-1 (Compression and Decompression).

The compressed byte is then sent over the network to the receiver where it is decompressed into two bytes depicted in the figure as **Decompressed Byte-1** and **Decompressed Byte-2**. Bits (c7,c6,c5,c4) are placed into bit positions (d7,d6,d5,d4) of **Decompressed Byte-1** respectively. On the other hand bits (c3,c2,c1,c0) are placed into the byte labeled **Decompressed Byte-2** occupying positions (e7,e6,e5,e4) respectively. Now the dilemma arises about what bit values should be placed in the four least significant bit positions of the decompressed bytes. As far as Fig. 2 is concerned we have padded both the least significant bits with zeros. However, during the phase of the experiment it has been observed that the best performance is achieved in audio conferencing if we add an empirical value of 10 if the decompressed byte value is positive and subtracted 10 if the value is negative. When concerned with video we added 8 for positive values and subtracted 8 for negative values. Even though the justification for adding and subtracting values is purely empirical; a mathematical explanation could be that since the last four bits have the most uncertainty forcing a maximum error of $\pm 2^4$, adding or subtracting an average value would be the wise choice of minimizing error if the data set that we are decompressing follows a Gaussian distribution.

Consequently, a decompression procedure for ALGO-1:

ALGO-1_DECOMPRESS(C)

```
D[length[C]*2]
j ←0
for i←0 to length[C]-1
        c←C[i]
        d←0
        e←0
        (d7, d6 ,d5,d4) ←(c7, c6, c5, c4)
        (e7, e6 ,e5,e4) ←(c0, c1, c2, c3)
        D[j]←d
        D[j+1]←e
    if d>0
       d←d+10
```

```
        else
            d←d-10
          if  e>0
              e←e+10
        else
            e←e-10
    j←j+1
    i←i+1
    return D
```

IV. COMPRESSION-DECOMPRESSION PROCESS OF ALGO-2

This algorithm compresses four consecutive bytes of a multimedia stream to three compressed bytes. The whole compression and decompression processes are depicted in Fig. 3(a) and Fig. 3(b) respectively. As shown in Figure 3(a), the four most significant bits (a7,a6,a5,a4) of **Uncompressed Byte-1** are placed in the four most significant bit positions (e7,e6,e5,e4) of **Compressed Byte-1** respectively. The next step is to place the four most significant bits (b7,b6,b5,b4) of **Uncompressed Byte-2** are placed in (e3,e2,e1,e0) of **Compressed Byte-1**. We repeat the previous step for **Uncompressed Byte-3** and **Uncompressed Byte-4**, in which bits (c7,c6,c5,c4) and (d7,d6,d5,d4) from both the uncompressed bytes are copied to **Compressed Byte-3**, i.e. the bits from **Uncompressed Byte-3** are placed in (g7,g6,g5,g4) and those of **Uncompressed Byte-4** being placed in (g3,g2,g1,g0). The next phase is as follows:

(i) Bits (a3,a2) of **Uncompressed Byte-1** be placed upon bit positions (f7,f6) of **Compressed Byte-2**.
(ii) Bits (b3,b2) of **Uncompressed Byte-2** be placed upon bit positions (f5,f4) of **Compressed Byte-2**.
(iii) Bits (c3,c2) of **Uncompressed Byte-2** be placed upon bit positions (f3,f2) of **Compressed Byte-2**.
(iv) Bits (d3,d2) of **Uncompressed Byte-2** be placed upon bit positions (f1,f0) of **Compressed Byte-2**.

A compression procedure for ALGO-2 as depicted in Fig. 3(a) would be:

ALGO-2_COMPRESS(U)

```
    C[length[U]*3/4]
    j ←0
for  i←0 to length[U]-1
        a←U[i]
        b←U[i+1]
        c← U[i+2]
        d← U[i+3]
        e←0
        f←0
        g←0
        (e7, e6 ,e5,e4) ←(a7, a6, a5, a4)
        (e3, e2 ,e1,e0) ←(b7, b6, b5, b4)
        (f7, f6 ,f5,f4) ←(a3, a2, b3, b2)
        (f3, f2 ,f1,f0) ←(c3, c2, d3, d2)
        (g7, g6 ,g5,g4) ←(c7, c6, c5, c4)
```

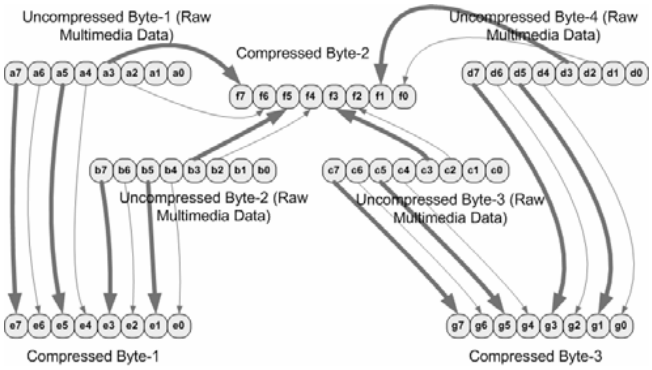

Figure 3(a) Compression Process of ALGO-2.

```
        (g3, g2 ,g1,g0) ←(d7, d6, d5, d4)
         C[j] ←e
        C[j+1] ←f
        C[j+2] ←g
    j←j+3
    i←i+4
return C
```

Basically, **Compressed Byte-2** is playing the role of the container for the bits that occupy the 2nd and 3rd least significant bit positions for all the four uncompressed bytes.

On receiving the three compressed bytes, we map the bits in the following way as illustrated in Fig. 3(b):

(i) (e7,e6,e5,e4) of **Compressed Byte-1** to (h7,h6,h5,h4) of **Decompressed Byte-1** respectively.

(ii) (e3,e2,e1,e0) of **Compressed Byte-1** to (i7,i6,i5,i4) of **Decompressed Byte-2** respectively.

(iii) (g7,g6,g5,g4) of **Compressed Byte-3** to (j7,j6,j5,j4) of **Decompressed Byte-3** respectively.

(iv) (g3,g2,g1,g0) of **Compressed Byte-3** to (k7,k6,k5,k4) of **Decompressed Byte-4** respectively.

(v) (f7,f6) of **Compressed Byte-2** to (h3,h2) of **Decompressed Byte-1** respectively.

(vi) (f5,f4) of **Compressed Byte-2** to (i3,i2) of **Decompressed Byte-2** respectively.

(vii)(f3,f2) of **Compressed Byte-2** to (j3,j2) of **Decompressed Byte-3** respectively.

(viii) (f1,f0) of **Compressed Byte-2** to(k3,k2) of **Decompressed Byte-4** respectively.

For all the decompressed bytes, we have filled out all the bit positions except for two least significant bit positions. However, for all the decompressed bytes we have assigned 1

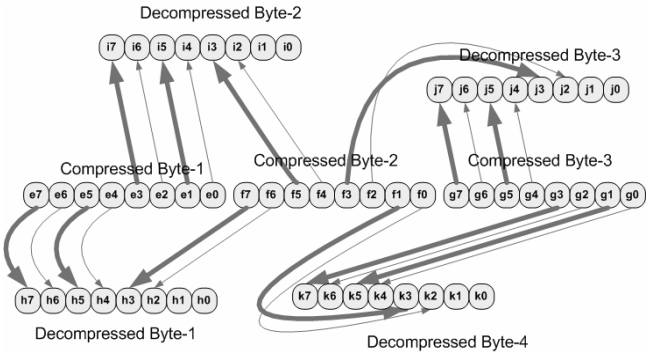

Figure 3(b) Decompression Process of ALGO-2.

for the second least significant bit position, and 0 for the least significant bit position. This was based on the assumption that an average value would yield the least error. Since the maximum likely error is ±3, therefore we have two average values to choose from either 1 or 2. The value 2 was chosen because it consists of a '1' bit that is higher in significance than the '1' bit in the value 1.

Consequently, a decompression procedure for ALGO-2 as shown in Fig. 3(b) would be:

ALGO-2_DECOMPRESS(C)

```
D[length[C]*4/3]
a←0
for b←0 to length[C]-1
        e←C[i]
        f← C[i+1]
        g← C[i+2]
        h←0
        i←0
        j←0
        k←0
        (h7, h6 ,h5,h4) ←(e7, e6, e5, e4)
        (h3, h2) ←(f7, f6)
        (i7, i6 ,i5,i4) ←(e3, e2, e1, e0)
        (i3, i2) ←(f5, f4)
        (j7, j6 ,j5j4) ←(g7, g6, g5, g4)
        (j3, j2) ←(f3, f2)
        (k7, k6 ,k5,k4) ←( g3, g2, g1, g0)
        (k3, k2) ←(f1, f0)
        (h1, h0) ←(1, 0)
        (i1, i0) ←(1, 0)
        (j1, j0) ←(1, 0)
        (k1, k0) ←(1, 0)
        D[j] ←h
        D[j+1] ←i
        D[j+2] ←j
        D[j+3] ←k
a←a+4
b←b+3
  return D
```

V. Quantitative and Qualitative Results from the Experiment when ALGO-1 is applied on Video Streams

Fig. 4 shows two pictures placed in a juxtaposed manner of the same object. The picture on the left hand side is the original picture captured by the web-camera and the picture just placed next to it is the picture extracted from the decompression of the original picture when compressed. From a qualitative point of view we can infer that the pictures are more or less identical. However, we can also see that the decompressed picture is a bit more blur than the original picture, and if seen closely there are also slight non-uniformities in the color which illustrates the loss that it incurs during the process of compression-decompression. Fig. 5 gives us a quantitative perspective of Fig. 4. Fig. 5 is a graph of a small sample of data values taken from the pictures as shown in Fig. 4. The blue lined graph represents the sample taken from the left side picture of Fig. 4 and the green lined graph represents the sample taken from the right side picture of Fig. 4. We can see that the data points of both graphs

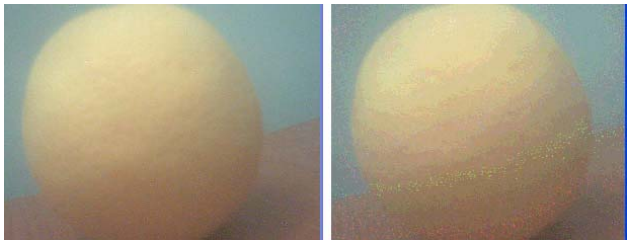

Figure 4 Left Picture Represents Original Uncompressed Picture and the Right Picture is the Decompressed using a Web-Camera of Low Resolution.

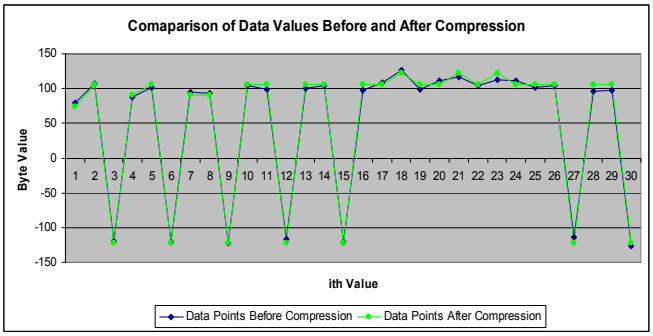

Figure 5 Graph Representing the Data Points of Figure 4 Before and After Compression

Figure 6 Left Picture Represents Original Uncompressed Picture and the Right Picture is the Decompressed using a Web-Camera of High Resolution.

overlap completely or vary very slightly. The web-camera used in this case is low-resolution, thus we observe blurred pictures. Fig. 6 is roughly an illustration of a high resolution version of Fig. 4, but still we see color distortions in the background and on the object.

VI. Quantitative and Qualitative Results from the Experiment when ALGO-2 is applied on Video Streams

The analysis for ALGO-2 will be quite similar to the previous algorithm except the results that we achieved from the above figures are quite unexpected. ALGO-2 has a lower compression ratio compared to ALGO-1, but according to the algorithm it loses less number of bits than ALGO-1 and should produce a higher quality picture. However, keeping that fact in mind if we compare Fig. 4 and Fig. 7, it seems that Fig. 4 has a better picture quality after decompression. To strengthen that claim if we look at the graph in Fig. 8 that is generated by taking sample data from the pictures in Fig. 7; not all the data points before compression and after compression coincide like the graph in Fig. 5. Rather we can see some data points have considerable difference among them. In theoretical terms the opposite should have happened. It is also very important to state that the sample photos taken are still pictures and do not necessarily represent the whole video stream but only just a portion. On the other hand if we compare Fig. 6 and Fig. 9, the latter turns out to be superior in quality as the background color was restored.

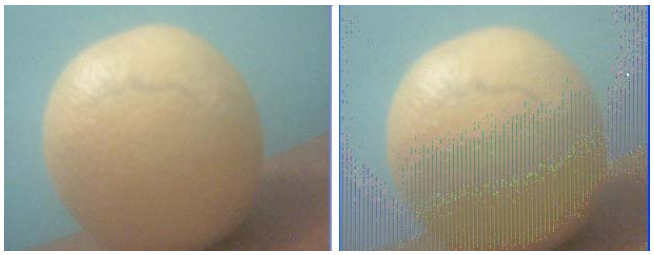

Figure 7: Left Picture Represents Original Uncompressed Picture and the Right Picture is the Decompressed using a Web-Camera of Low Resolution.

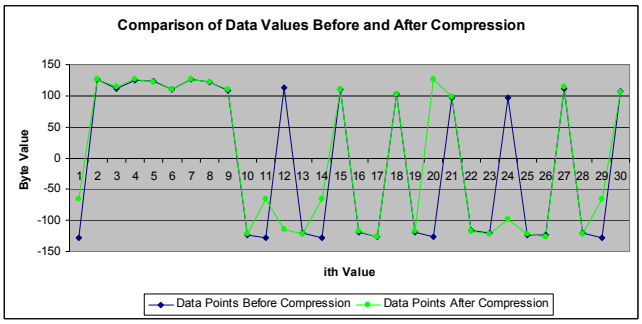

Figure 8 Graph Representing the Data Points of Figure 7 Before and After Compression.

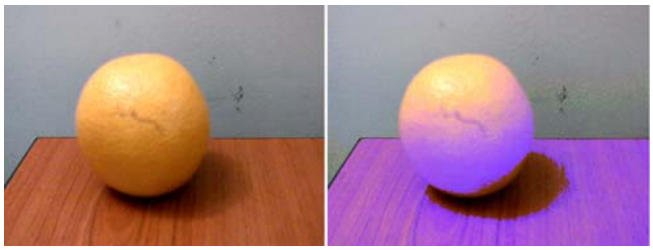

Figure 9 Left Picture Represents Original Uncompressed Picture and the Right Picture is the Decompressed using a Web-Camera of High Resolution.

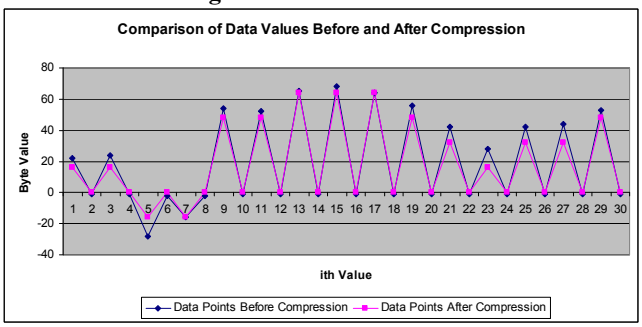

Figure 10 Comparison of Data Points before and after compression when ALGO-1 is applied on Audio Streams.

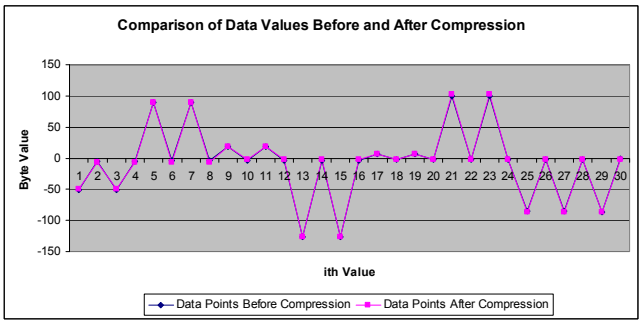

Figure 11 Comparison of Data Points before and after compression when ALGO-2 is applied on Audio Streams.

VII. Quantitative Results from the Experiment when Algo-1 and algo-2 is applied on Audio streams

When the algorithms are tested for their performance on audio streams it can be seen from the graphs depicted in Fig. 10 and Fig. 11 that the byte values before compression and after compression are not quite far apart. In Fig. 10 we see that the data points are close to one another if not overlapping and this difference accounts for the background noise that is produced while the sound is heard after decompression. On the other hand in Fig. 11 we can see that most of the data points are overlapping such that the graph formed from the data points before compression cannot be seen. Hence, the sound heard after decompression is quite clear and almost similar to the original. Basically both algorithms seem to be improvements over the μ-law technique described in [8,11] which is used in Unix [2] as well as in telephone systems. The technique in [8,11] compresses 16-bit audio samples into an 8-bit compressed version keeping the 16-bit sample as a single signed number preserving the sign bit of 16-bit data or the sign bit of the higher order byte. However, accuracy is being lost as the sign bit or the most significant bit of the lower order byte of 16-bit sample is not guaranteed. Both algorithms proposed in this paper compress a 16-bit sample to an 8-bit compressed byte form but here we are splitting the 16-bit to two 8-bit samples and in each case we are at least preserving the four most significant bits of each byte so that maximum accuracy is preserved. In [11] we can see that the comparisons between the values between the data before and after compression there is a data value that has a difference of 129 but in this paper the algorithm ALGO-1 produces a maximum difference of ±16 if we don't add the empirical value to the decompressed byte.

VIII. Conclusion

If we see the Golomb coding process in [5,6] we can see that the Golomb code achieves high compression ratios when the values of input stream are small i.e. because as the values increase the code length increases and thus reduces compression levels even though it is a lossless algorithm. The algorithms that we have proposed here are lossy and achieve rates that remain constant although irrespective of the input data. On the other hand if we look at the μ-law algorithm which achieves a constant 50% compression ratio but at the expense of losing considerable bits. Therefore, the algorithms that we present evens out the drawbacks of both algorithms i.e. both proposed algorithms have considerably high compression ratios at all times and achieve fairly close values after decompression is performed compared to that of μ-law. In [5] there is a parameter p that has to be calculated as soon as the input data arrives but it can take up a lot of calculation time if there is input data of large size whereas our algorithms does not have such an overhead since we are compressing as soon as data arrives and minimum possible time is taken for any sized data. Although the algorithms are very simple in nature as many would argue that it is simply a case of multiple bit manipulations. However, the proposed algorithms in this paper being simple and possessing a linear time complexity makes it ideal for streaming multimedia as it will consume the least of bandwidth resources. In the future, if simple algorithms like the ones developed in this paper which can provide relatively clear audio-video in conditions where bandwidth levels are constrained, it can be a new model for live internet audio-video streaming.

IX. References

[1] Ida Mengyi Pu, Fundamental Data Compression, 2006, Butterworth-Heinemann, Oxford, Great Britain

[2] Gerth, B. J. (2001), Audio on the Web: Enhance On-line Instruction with Digital Audio, TECHNOLOGY AND TEACHER EDUCATION ANNUAL, Vol 1, pages 45-50

[3] Wai C. Chu, Speech Coding Algorithms: Foundation and Evolution of Standardized Coders, 2003, John Wiley & Sons, Inc., Hoboken, New Jersey, USA,

[4] Theodore S. Rappaport, Wireless Communications: Principles and Practice, Second Edition, 2006, Prentice-Hall of India Private Limited, New Delhi, India

[5] David Salomon, Data Compression: The Complete Reference, Third Edition, 2004, Springer, New York, USA

[6] Golomb, S.W. (1966). , Run-length encodings. IEEE Transactions on Information Theory, IT--12(3):399 – 401

[7] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Introduction to Algorithms, First Edition, 1990, MIT Press and McGraw-Hill, Cambridge, MA, USA

[8] http://www.speech.cs.cmu.edu/comp.speech/Section2/Q2.7.html

[9] JMF http://www.java.sun.com/javase/technologies/desktop/media/jmf/

[10] Java http://www.java.sun.com

[11] http://www.developer.com/java/other/article.php/3286861/Java-Sound-Compressing-Audio-with-mu-Law-Encoding.htm