GPU-Based Integrated Security System for Minimizing Data Loss in Big Data Transmission

Shiladitya Bhattacharjee, Midhun Chakkaravarthy and Divya Midhun Chakkaravarthy

Abstract In big data transmission, data loss occurs due to excessive data and time overheads, transmission errors, and various security attacks. However, the current literature does not suggest any combinatorial solution to protect data loss by resolving these issues. Henceforth, this research proposes a GPU-based integrated technique. It includes the new dual rounds of error control operations to address every individual error bit. An idle lossless compression has been included to reduce data overhead. Furthermore, it includes a modern audio steganography which comprises a new sample selection scheme for hiding data. Finally, it reduces execution time by involving GPU during implementation. As per the experimental result, it offers higher time and space efficiencies by producing higher compression percentage and throughput. It shows substantial resistivity against various attacks and transmission errors by producing higher SNR and entropy values. It also offers lower percentage of information loss than other existing techniques.

Keywords Data loss · GPU · Compression percentage · Perceptual similarity Throughput · Lower percentage of information loss

S. Bhattacharjee (✉) · M. Chakkaravarthy
Computer Science and Multimedia Department, Lincoln University College, Selangor Darul Ehsan, Malaysia
e-mail: shiladityaju@gmail.com

M. Chakkaravarthy
e-mail: midhun.research@gmail.com

D. Midhun Chakkaravarthy
Faculty of Post Graduate Studies, Lincoln University College, Selangor Darul Ehsan, Malaysia
e-mail: divya.phd.research@gmail.com

© Springer Nature Singapore Pte Ltd. 2019

V. E. Balas et al. (eds.), *Data Management, Analytics and Innovation*,
Advances in Intelligent Systems and Computing 839,
https://doi.org/10.1007/978-981-13-1274-8_32

1 Introduction

Data overhead, the root cause for data loss can be controlled with the application of distinct compression techniques. Conversely, between lossy and lossless compression, lossy type degrades data quality by eliminating the redundant information permanently which is unwanted for this research. Hence, this discussion relates the lossless compression only. Amidst the lossless compressions, fixed length coding (FLC) is inefficient to produce desire compression efficiency [1]. Contrariwise, the variable length coding (VLC) is dependable on prefix codes. These prefix codes can be easily affected by transmission errors which may cause data loss [1, 2].

As per the literature, another important aspect of data loss is transmission errors. Consequently, this issue increases with the increment of file size. However, the present literature fails to present any efficient error control technique which can control more than 8-bit discrete or continuous errors [3, 4]. Only cryptographic hash function can address up to 8-bit errors. Another way to replace these errors is to transmit the original information redundantly [5]. Cryptography and steganography and pattern generation as well as matching using hash function are the mostly used security techniques to protect various security attacks, performed by distinct illicit users to steal or notch information during transmission. However, these techniques are consuming lots of time and space to complete a number of iterations during execution which are not suitable for big data transmission [6].

1.1 Current Problems

The issues regarding various security schemes to protect data loss in big data transmission are as follows:

- A number of iterations are involving in existing steganography and cryptography techniques, used for protecting various security attacks. As the consequence, time and space overheads increase and data loss may occur [6].
- The existing error control techniques are not efficient to protect more than 8-bit discrete or continuous errors. The retransmission of original information increases time and space overheads [7].
- The current literature fails to suggest any integrated technique which can resolve all these root causes of data loss in big data transmission collectively [8].

1.2 Research Objectives

The primary objective is to provide an integrated solution for data loss in big data transmission by addressing the primary issues in combinatorial way. The other objectives are as follows:

- Minimizing data overhead by introducing a new lossless compression technique.
- Minimizing time overhead using single iteration during execution and GPU.
- Enhancing robustness against various security attacks and transmission errors.

2 Literature Study

This section reviews the strengths as well as the shortcomings of these available techniques and helps to develop the proposed integrated technique by analyzing them. In article [3, 4], the authors proposed a combinatorial Ethernet solution by integrating multipathing and congestion control mechanism. The proposed technique improves network throughput because its application-layer flow differentiation can make full consumption of network bandwidth and congestion control can stop unnecessary traffic from incoming network. However, it fails to utilize the buffer properly. According to article [5], the scarce inputs in wireless body sensor network (WBSN) nodes limit their abilities to survive with massive traffic during multiple, concurrent data communications. However, this technique cannot offer efficient routability and minimize congestion and overflow avoidance proficiently. As per article [6], the huge data transmission of ultrasound data becomes a serious issue of real-time transmission for a GPU-based beamformer. Yet, in this technique, phase data were fixed as 6 bits without any compression.

According to article [7, 8], fault tolerance has been observed as serious to the real use of these GPUs. In this article, the author presented an online GPU error detection, location, and correction method to include fault tolerance into matrix multiplication. However, this technique is not efficient for large number of continuous error bits and it enhances data overhead considerably. In article [9, 10], the authors define a parallel execution of a favorable similarity search procedure for an audio fingerprinting system. With simple reforms, the authors obtained up to 4 times higher GPU performance. However, it is not highly secured and robust against various transmission errors and security attacks.

Henceforth, from the above discussion, we have seen that the existing techniques for minimizing the data loss are inefficient for big data applications. These prevailing techniques are either enhancing excessive space as well as space overheads during the execution or cannot offer adequate robustness against the various transmission errors and security attacks [11, 12]. Therefore, there is further scope to develop an integrated technique which can be functioned with low time and space requirements and which can protect data loss by minimizing various data errors during the transmission process as well as protecting the various security attacks at the same time.

3 Proposed Technique

The detailed steps of implementing the proposed integrated technique are shown in Fig. 1.

The implementation of the proposed integrated technique has been performed using parallel processing in GPU environment. At the final part, the description of retrieval information at the receiving end is shown further.

3.1 Generation and Incorporation of Error Control Bits

The proposed dual rounds of error control bit generation and incorporation have been presented in this subsection with the help of text file as input to make it

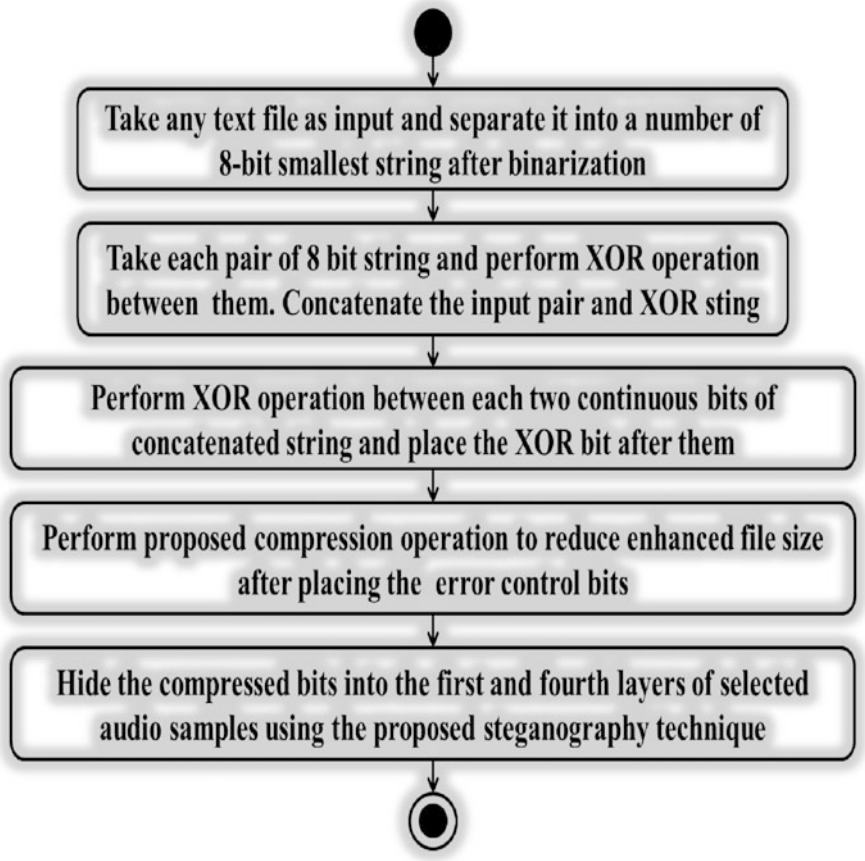

Fig. 1 Proposed integrated model for minimizing data loss using parallel processing with graphics processing unit (GPU)

simplified. Let the input text is denoted as *Text*, and after converting each character of this input text into 8-bit strings, they are stored in the string array *text*. If the total number of characters belonging to the input text is n, then the sequences of converted 8-bit strings are represented as $text_n$. Perform the first round of error control operation by

$$\left.\begin{array}{c} xorStr = text_i \oplus text_{i+1} \\ conStr = \left(text_i \times 10^{l(text_{i+1})}\right) + text_{i+1} \\ concatStr_j = \left(conStr \times 10^{l(xorStr)}\right) + xorStr \\ i = i+2, j = j+1 \end{array}\right\} \quad (1)$$

such that

$$i, j \leq 0 \text{ and } i < (n-2), j < \left(\frac{n}{2}\right),$$

$$l(text_{i+1}) = (\log_{10}(text_{i+1}) + 1) \text{ as well as}$$

$$l(xorStr) = (\log_{10}(xorStr) + 1)$$

After completion of first round of error control bit generation and incorporation, the length of each concatenated string in array *concatStr* becomes 24 bits. Add "0" at most significant bit position of each element in array *concatStr*. Let the order of each string in *concatStr* is denoted by $concatStr_j$, where $0 \leq j < \left(\frac{n}{2}\right)$ and the bit sequence of each string in *concatStr* is signified by $concatStr_{jk}$ where $0 \leq k \leq 24$. Second round of error control bit generation and incorporation are performed to generate string array *concatStr′* of size $\left(\frac{n}{2}\right)$ as

$$\left.\begin{array}{c} xorStr' = concatStr_{pl} \oplus concatStr_{p(l+1)} \\ concatStr'_p = concatStr_{p(l+1)} \\ concatStr'_p = \left(concatStr'_p \times 10^{l(xorStr')}\right) + xorStr' \\ p = p+1 \text{ and } l = l+1 \end{array}\right\} \quad (2)$$

such that

$$p, l \leq 0 \text{ and } p < \frac{n}{2}, l \leq 24 \text{ as well as}$$

$$l(xorStr) = (\log_{10} xorStr + 1)$$

3.2 Application of the Proposed Compression Technique

After adding error control bits, the array *concatStr′* is taken as input. Initially, create a new string array *EleRed* from *concatStr′* by truncating all redundant elements as

$$EleRed = \begin{pmatrix} concatStr'_m \\ \frac{n}{2} \end{pmatrix} \tag{3}$$

After creating of array *EleRed*, calculate the frequencies of each element in array *concatStr′* and put the corresponding frequencies into another array *Freq*. Sort the array *EleRed* according to higher to lower frequencies. As the input file is text and total number of ASCII characters is 128, hence, a maximum of different elements in *EleRed* will be 128. As the consequence of it, the size of *EleRed* which is *m* can be varied from 0 to 128. Now, let the string sequence of *EleRed* is represented by $EleRed_x$, where $0 \leq x < 128$. In the next step, a string array *Comp* of size *x* is created to store the compressed code. If the total size of string array *EleRed* is less than or equal to 32, then generate the compressed codes and store them in *Comp* as

$$\left.\begin{array}{c} s_y = x \; modulo \; 2^y \\ t = \frac{EleRed_x}{2^i}, \text{ where } 0 \leq y \leq 5 \\ z = z \times (10)^{l(s_y)} + s_y, \text{ if } l(s_y) = \log_{10}^{s_y} + 1 \end{array}\right\} \text{if } t \neq 1 \tag{4}$$
$$Comp_x = z$$

where $s, t, y,$ and x are integer variables where $x \geq 0$ and $x < 32$ and z is a string variable. When the size of *EleRed* > 32 and ≥ 62, then create the compressed code for zero to twenty-ninth element using the same process as

$$\left.\begin{array}{c} x = x - 30 \\ s_y = x \; modulo 2^y \\ t = \frac{EleRed_x}{2^i}, \text{ where } 0 \leq y \leq 5 \\ z = z \times (10)^{l(s_y)} + s_y, \text{ if } l(s_y) = \log_{10}^{s_y} + 1 \end{array}\right\} \text{if } t \neq 1$$

Now let the 5-bit binary string, generated from 31 is used as separator (*Sep*). Concatenate separator with the compressed code as

$$z = \left(z \times 10^{l(Sep)}\right) + Sep \tag{5}$$

where $l(Sep) = \log_{10}^{Sep} + 1$.

After the concatenation operation, compressed codes are stored in array *Comp*. When the size of *EleRed* is among 63 and 90, then consider 30 and 31 as separator and repeat same steps. Similarly, when the size of *EleRed* is among 91–116, consider 29, 30, and 31 as separators. Finally, when the length of *EleRed* is more

than 116, then consider 28, 29, 30, and 31 as separator and generate the compressed codes. Store these compressed codes into the *Comp*. Concatenate all strings of *concatStr* into a variable *Comp′*. Concatenate all strings of *EleRed* into a single string variable *Char*. Create the final separator *Sep′* by concatenating 100 bits into 1. Finally, concatenate *Comp′*, *Sep′* and *Char* according to generate compressed string *CompStr*.

3.3 Application of Proposed Steganography Technique

After the creation of complete *CompStr*, the .wav audio file and string *CompStr* are taken as input to create the Stego file. The creation of Stego has four parts, and they are binarization of audio file, selection of samples for incorporation compressed string, incorporation of compressed string into selected samples, and reformation of Stego audio file after incorporation of compressed string. Each part is further described in detail at the following subsections.

3.3.1 Binarization of Input Audio File

The binarization operation has been done by dividing the .wav file into few samples using sampling theory depending upon the amplitude and threshold value. The normalization is done with each sample to get the actual value. 10 is multiplied with normalized value to convert them within the range of 0–254. After multiplication, each value is converted into 8-bit binary string.

3.3.2 Selection of Samples for Incorporating Compressed String

If we hide the compressed code to each sample, then the distortion of sound will be occurred after conversion of this string to Stego sound file. So, we select the particular samples to hide the compressed code. We select the prime position of the samples and then find the sequence of number of each prime position.

3.3.3 Incorporation of Compressed String into Selected Samples

After the selection, two bits are taken in each occasion from the compressed string and incorporated into each selected sample accordingly. Let the selected sample is denoted by *Samp* and the incorporation of compressed bits into *Samp* is shown by Algorithm 1.1 and 1.2.

```
Algorithm-1.1: Audio Steganography (Part-1)
(e1) declare selected samples as ⟨Samp⟩ string array;
(e2)  if ( Samp[3] is modified from 0 to 1)
(e3)     if (Sample[2] == 1 and Sample[4] == 1)
(e4)          for(int i = 0; i ≤ 2; i + +)
(e5)             Sample[i]=0;
(e6)           End for
(e7)     End if
(e8)     if (Sample[2] == 1 and Sample[4] == 0)
(e9)        repeat steps (e4) to (e6)
(e10)    End if
(e11)    if (Sample[2] == 0 and Sample[4] == 1)
(e12)        for(int i = 0; i ≤ 2; i + +)
(e13)               Sample[i]=1;
(e14)         End for
(e15)     End if
(e16)     if (Sample[2] == 0 and Sample[4] == 0)
(e17)          repeat steps (e12) to (e14)
(e18)          for(int j = 4; j ≤ 7; j + +)
(e19)               if(Sample[j]==1)
(e20)                  Sample[j]=0;
(e21)                   break;
(e22)              End if
(e23)              else
(e24)                   Sample[j]=1;
(e25)  End All
```

In Algorithm 1.1, line (e1) declares the 8-bit string variable *Samp*. During the incorporation, if the fourth bit of *Samp* converts from 0 to 1 and if third and fifth bits of *Samp* are 1, then lines (e1) to (e7) describe the required operations. Lines (e8) to (e10) describe the changes in *Samp* if the third bit of it remains 1 and fifth bit is 0. Similarly, lines (e11) to (e15) define the operation when the third bit of *Samp* is 0 and the fifth bit is 1. Lines (e16) to (e25) define the changes in *Samp* if the third and fifth bits are. During embedding operation, if the third bit is changed from 1 to 0, then the required operations are shown by Algorithm 1.2.

```
Algorithm-1.2: Audio Steganography(Part-2)
(e26)  if ( Samp[3] is modified from 1 to 0)
(e27)     if (Sample[2] == 0 and Sample[4] == 0)
(e28)          for(int i = 0; i ≤ 2; i + +)
(e30)              Sample[i]=1;
(e31)           End for
(e32)     End if
(e33)     if (Sample[2] == 0 and Sample[4] == 1)
(e34)        repeat steps (e30) to (e32)
(e35)    End if
(e36)    if (Sample[2] == 1 and Sample[4] == 0)
(e37)        for(int i = 0; i ≤ 2; i + +)
(e38)              Sample[i]=0;
(e39)         End for
(e40)     End if
(e41)     if (Sample[2] == 1 and Sample[4] == 1)
(e42)          repeat steps (e38) to (e40)
(e43)          for(int j = 4; j ≤ 7; j + +)
(e44)               if(Sample[j]= =0)
(e45)                  Sample[j]=1;
(e46)                   break;
(e47)              End if
(e48)              else
(e49)                   Sample[j]=0;
(e50) End All
```

In Algorithm 1.2, we can see the required operation if the fourth bit of the *Samp* changes from 1 to 0 during the incorporation. Line (e26) tells this condition. Lines (e27) to (e31) describe the required changes of *Samp* when the third and fifth bits of *Samp* are 0. If the third bit is 0 and fifth bit is 1 of *Samp*, lines (e32) to (e34) describe the required changes of *Samp*. Lines (e35) to (e39) define the required changes of *Samp* when the third bit is 1 and fifth bit is 0. Finally, lines (e40) to (e50) define the required modification of *Samp* when the third and fifth bits of *Samp* are 1.

Reformation of Stego Audio File After Incorporation of Compressed String
After incorporation of compressed bit in the audio samples *Samp*, convert all the samples into decimal value and then divide all the samples by 10. After the division, de-normalization operation is done, depending upon the predefined threshold values. Then, combine all the samples into single file to construct the final Stego audio file.

3.4 Retrieval of Information at the Receiving End

After receiving complete Stego file, the Stego audio file is split into 8-bit binary samples. After binarization, select the sample by the sample selection technique and extract the incorporated compressed bits from the first and fourth positions of selected samples. After extracting the compressed bits from the each selected sample, they are concatenated into single string to construct final compressed string. After that, the character and compressed code strings are separated based on the separator. After separating them, compressed code array and character array are generated. From the character array, a reference code array is created. After performing first round error control operation, eliminate the entire reference error control bit. If error still exists, then generate original string with the help of reference XOR strings and replace the erroneous string with the original string. Truncate all the reference XOR strings and convert the binary strings to ASCII characters. All characters are then merged into single file to generate the original file.

4 Result Analyses

As per the research objectives, primarily we will examine the capacity of proposed compression technique to reduce file size. As per the literature, *compression percentage (CP)* is the required parameter for measuring the efficiency of any compression technique. The CP can be defined as

$$\mathrm{CP} = \frac{\text{Actual Size} - \text{Compressed Size}}{\text{Actual Size}} \times 100 \tag{6}$$

With the help of Eq. (6), the efficiencies (in terms of CP) of different existing FLCs and VLCs have been compared with the proposed technique in Tables 1 and 2.

Table 1 Comparison of proposed technique with different VLCs

File size (GB)	Arithmetic coding	Huffman BWT	LZSS BWT	Dictionary based	Proposed technique
100	14.0	30.2	34.1	41.6	56.7
200	13.0	30.9	35.2	42.0	57.0
350	12.9	33.7	34.7	43.0	57.9
450	13.9	31.3	34.0	43.7	58.0
600	13.4	30.6	35.1	42.9	58.2
750	13.6	31.5	35.2	44.0	58.8
900	12.9	32.0	34.3	45.0	58.9
1024	12.7	29.9	36.7	44.6	59.7

Table 2 Comparison of proposed technique with different FLCs

File size (GB)	ASCII code	EBCDIC code	Run length coding	Proposed technique
35	0.0002	0.0001	−1.0055	56.7
56	0.0003	0.0002	−1.0026	57.0
90	0.0001	0.0004	0.0001	57.1
120	0.0001	0.0004	−0.998	57.9
200	0.0002	0.0001	−0.762	58.0
350	0.0003	0.0002	−0.985	58.2
450	0.0001	0.0003	−0.976	58.8
600	0.0003	0.0002	−1.112	58.9
1024	0.0002	0.0001	−1.007	59.7

Table 3 Execution speed of proposed technique in CPU and GPU

File size in GB	Throughput offered by CPU (MB/s)	Throughput offered by GPU (MB/s)
100	2.71	74.64
200	2.75	76.27
350	2.72	77.77
450	2.74	78.21
600	2.72	76.78
750	2.75	77.56
900	2.79	79.22
1024	2.76	79.67

From Tables 1 and 2, we have seen that the proposed compression technique offers higher *CP* than the other existing FLC- and VLC-based compression techniques. Hence, the proposed compression technique is efficient to compress the data than other existing which fulfills our first objective.

In the next phase, we will examine the time efficiencies of the proposed technique. As per the literature, *throughput (TP)* is the required parameter for measuring the time efficiency of any technique. *Throughput* is defined as

$$\mathrm{TP} = \left(\frac{\text{Output file size}}{\text{Total execution time to generate output}} \right) \tag{7}$$

With the help of Eq. (7), the *throughput* produced by the proposed technique during the execution in both CPU and GPU has been calculated using distinct file of different sizes. The results have been tabulated in Table 3.

Table 3 shows that the GPU implementation offers much higher execution speed than the CPU implementation in terms of offering higher *throughputs*. As the proposed integrated technique has been implemented in GPU environment,

henceforth, it is much faster than any other existing solution. Thus, we have achieved our second objective.

The third objective of this research is to examine the capacity of preventing data loss by proposed integrated technique. From the literature, we have seen that other than the time and space overheads, transmission errors and various security attacks are the main causes of data loss. The literature also shows that the capacity of any error control technique to control error can be evaluated by calculating *signal-to-noise ratio (SNR).* The SNR is defined as

$$\mathrm{SNR}_{\mathrm{dB}} = 10 \times \mathrm{Log}_{10}\left\{\sum_{n} x^2(n) / \sum_{n}\left[x^2(n) - y^2(n)\right]\right\} \tag{8}$$

In Eq. (8), $x(n)$ denotes mean amplitude values of the cover files, whereas $y(n)$ denotes mean amplitude of Stego sample file, and n is any finite number. The SNR values offered by different error control techniques and the proposed techniques are shown in Fig. 2.

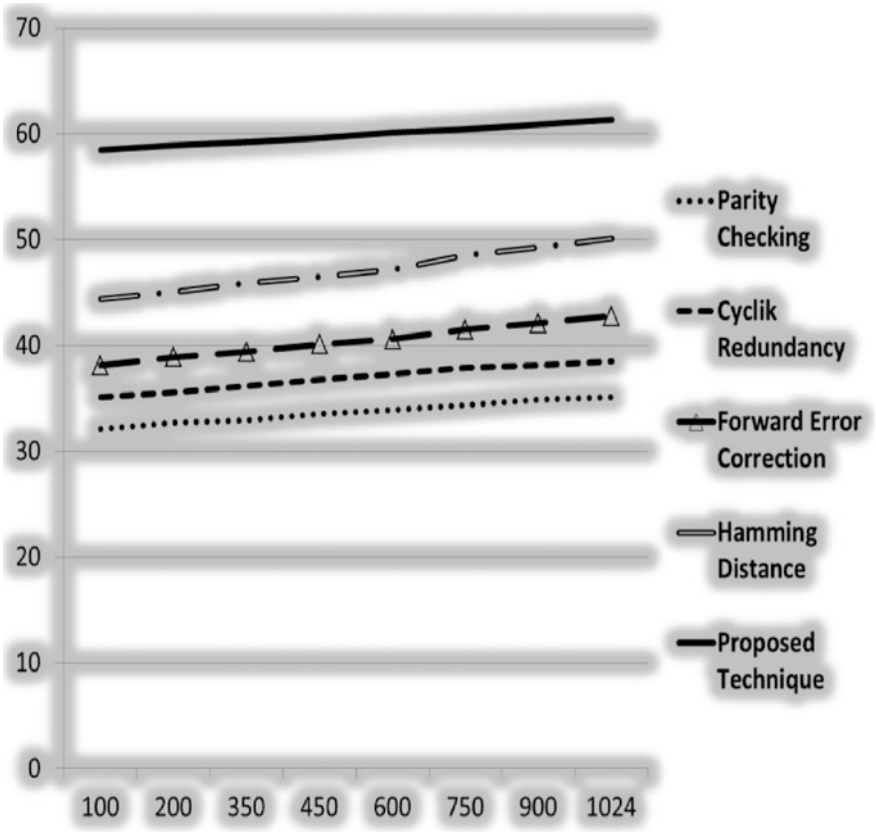

Fig. 2 $\mathrm{SNR}_{\mathrm{dB}}$, offered by different error control techniques

Table 4 Entropy values offered by different security techniques

Security techniques	Entropy values
Existing LSB technique	7.73
Parity coding	7.53
Phase coding	7.67
Spread spectrum	7.52
Echo hiding	7.51
Proposed technique	7.77

As per the definition, if any error control technique is effective to produce higher SNR_{dB}, then it is considered as efficient to reduce transmission errors. Figure 2 shows that proposed error control technique offers higher SNR_{dB} than others. Hence, it is more efficient to reduce transmission errors.

Consequently, the capacity to protect security attacks can be verified by calculating *entropy* values, offered by any security technique. It can be calculated as

$$H(S) = \sum_{i=0}^{2N-1} P(S_i) \log_2 \frac{1}{P(S_i)} \tag{9}$$

Here, $P(S_i)$ represents the probability of symbol S_i. With the help of Eq. (9), the *entropy* values offered by the proposed integrated and other existing technique are shown in Table 4.

From Table 4, we have seen that proposed integrated technique offers higher robustness against the various security attacks as it offers higher entropy values than others. Consequently, Fig. 3 justifies the capacity of protecting information loss by the proposed integrated technique.

Figure 3 shows that the proposed integrated technique offers lower percentage of information loss than the other exiting techniques. Hence, Fig. 2, Table 4, and Fig. 3 shows that the proposed integrated technique offers higher efficiencies to protect information loss by preventing various transmission errors and security attacks which fulfill our third objective.

5 Conclusions and Future Work

From the literature study in Sect. 2, we have seen that the existing security techniques for big data transmission are incapable of protecting data loss, which is a big challenge these days. Henceforth, this research proposed a novel integrated technique for minimizing the data loss in big data transmission and has been designed in GPU-based environment to speed up the execution process. The faster execution process will further reduce the data loss during the transmission. Apart from that, the proposed integrated technique includes a unique error control technique to protect data from various transmission errors, a novel compression technique to

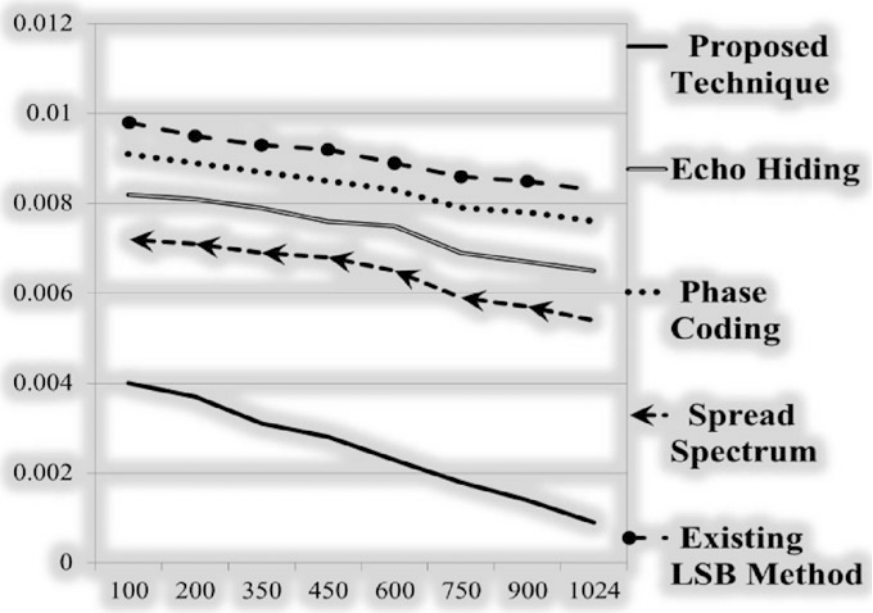

Fig. 3 Capacity of protecting information loss by different security techniques

reduce data load as well as the proposed advanced steganography technique, which helps to protect various security attacks during the transmission. In result section, from Tables 1 and 2, it can be seen that the proposed integrated technique is efficient to reduce space overheads than other existing techniques. At the same time, Table 3 shows that the proposed integrated technique reduces data loss by offering the better time efficiency in terms of providing better execution speed than other existing integrated techniques. Further, Fig. 2 shows that the proposed integrated technique can protect various transmission errors in efficient way than the other corresponding available security techniques by preventing. Table 4 shows that the proposed integrated technique is also efficient to protect various security attacks. Finally, Fig. 3 shows the efficiency of the proposed integrated technique to protect data loss with the help of different input file sizes. However, Figs. 2 and 3 show that the proposed integrated technique can be more improved that it can reduce the information loss further and it can prevent the transmission errors more accurately in the coming future. With the use of GPU environment, the overall time complexity has been reduced dramatically. However, it can be more reduced by reducing the time complexity of each of the individual parts of the proposed integrated technique and by reducing the integration process as well.

References

1. Ding, C., Karlsson, C., Liu, H., Davies, T., & Chen, Z. (2011). Matrix multiplication on GPON-line fault tolerance (pp. 311–317).
2. Lok, U.-W., & Li, P.-C. (2014). Improving performance of GPU-based software beamforming using transform-based channel data compression (pp. 2141–2144).
3. Fang, S., Yu, Y., Foh, C. H., & Aung, K. M. M., A loss-free multipathing solution for data center network using software-defined networking approach (pp. 1–8); Maxwell, J. C. (2013). A treatise on electricity and magnetism (3rd ed., Vol. 2, pp. 68–73). Oxford: Clarendon.
4. Yaakob, N., & Khalil, I. (2016). A novel congestion avoidance technique for simultaneous realtime medical data transmission. *IEEE Journal of Biomedical and Health Informatics, 20* (2), 669–681.
5. Lok, U.-W., Shih, H.-S., & Li, P.-C. (2015). Real-time channel data compression for improved software beamforming using microbeamforming with error compensation (pp. 1–4).
6. Chen, S.-n., Zheng, B.-y., & Zhou, L. (2013). A parity-based error control method for distributed compressive video sensing (pp. 105–110).
7. Xiaoliang, C., Chengshi, Z., Longhua, M., Xiaobin, C., & Xiaodong, L. (2010). Design implementation of MPEG audio layer III decoder using graphics processing units (p. 487).
8. Ouali, C., Dumouchel, P., & Gupta, V. (2015). GPU implementation of an audio fingerprints similarity search algorithm (pp. 1–6).
9. Cheung, N.-M., Au, O. C., Kung, M.-C., Wong, P. H., & Liu, C. H. (2009). Highly paralldistortion optimized intra-mode decision on multicore graphics processors. *Transactions on Circuits and Systems for Video Technology, 19*(11), 169.
10. Rahim, L. B. A., Bhattacharjee, S., & Aziz, I. B. A. (2014). An audio steganography technique to MData hiding capacity along with least modification of host. In *Proceedings of the First InterConference on Advanced Data and Information Engineering*. Singapore: Springer.
11. Bhattacharjee, S., Rahim, L. B. A., & Aziz, I. B. A. (2015). A Lossless compression technique to increase robustness in big data transmission system. *International Journal in Advances in Soft Computing and Its Application*.
12. Bhattacharjee, S., Rahim, L. B. A, & Aziz, I. B. A. (2014). A secure transmission scheme for textual data with least overhead. In *Twentieth National Conference on Communications (NCC)*. New York: IEEE.