PERFORMANCE ANALYSIS OF MULTIMEDIA COMPRESSION ALGORITHMS

Eman Abdelfattah[1] and Asif Mohiuddin[2]

Department of Computer Science and Engineering, University of Bridgeport, Bridgeport, Connecticut, USA

[1] eman@bridgeport.edu

[2] asifm@bridgeport.edu

ABSTRACT

In this paper, we are evaluating the performance of Huffman and Run Length Encoding compression algorithms with multimedia data. We have used different types of multimedia formats such as images and text. Extensive experimentation with different file sizes was used to compare both algorithms evaluating the compression ratio and compression time. Huffman algorithm showed consistent performance compared to Run Length encoding.

KEYWORDS

Huffman algorithm, Run Length Encoding algorithm

1. INTRODUCTION

Multimedia is a technology used to process textual data, audios and videos, images and animations. Multimedia applications are used in different fields of life like entertainment, video conferencing, application sharing, distance learning, remote working and online games. An example of multimedia applications is WebEx [1] video conferencing. WebEx is an application used across Internet to share data, images and voice. In WebEx, an invitation is sent to all attendees so that all participants can click WebEx link, enter username, password as authentication and start watching presenter's slides. At the same time, all participants will be dialling a common phone number provided during the invitation to hear the presenter's voice.

The challenge in multimedia applications is the transport services to both discrete media such as text and digital images and continuous media such as audio and video with limited bandwidth and huge data size. Unlike text media, multimedia applications are loss tolerant but delay sensitive [2]. Today's multimedia networks strive for "(I) seamless connection management between the network and multimedia devices, (II) multimedia abstractions with QoS guarantees and (III) the integration of service, traffic control and network management architectures" [3].

With the huge demand for bandwidth due to the large data transmitted in multimedia applications, it becomes necessary to apply compression algorithms on transmitted data. The topic of compression algorithms is one of the issues that Multimedia researchers and product developers have worked on to deliver low-bandwidth high-quality audio coding and high-quality video coding. Thus, audio coding with these characteristics can support telephony systems. Moreover, high-quality video coding supports entertainment and broadcasting applications [4].

DOI : 10.5121/ijcsit.2010.2501

Huffman coder is popular in modern multimedia compression standards because of its low computational cost [5]. In [6], a technique based on variation of Huffman coding is presented to offer better compression. Run Length Encoding is a simple data compression algorithm and is based on replacing a long sequence of the same symbol by a shorter sequence [7]. Although, Huffman and run-length coding were introduced years ago, they are utilized recently to develop a technique that achieves superior JPEG compression results with efficient computation [8].

In this paper we evaluate both Huffman compression algorithm and Run Length Encoding (RLE) algorithm used with multimedia applications and inspect their effect on the quality of the transmitted data.

2. MULTIMEDIA COMPRESSION

General JPEG transformation would transform an image to get discrete cosine. Then it quantizes the discrete cosines to differential quantization. Finally, the run length encoding is used to get compressed output with lossy data.

The algorithm works as follows (Figure 1) [9]:

1. Block Preparation: From RGB to Y, CR, CB planes.
2. Transform: Two-dimensional Discrete Cosine Transform (DCT) on 8x8 blocks.
3. Quantization: Compute Quantized DCT Coefficients.
4. Encoding of Quantized Coefficients using Zigzag Scan.

For decompression, the order of the previous steps is reversed.

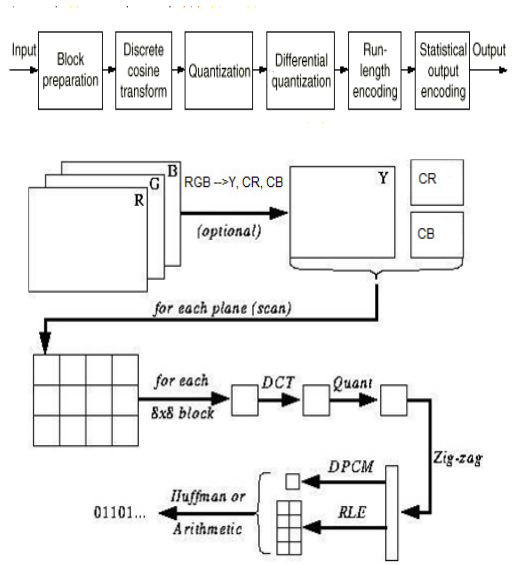

Figure 1. JPEG Overview [4]

2.1. Huffman Algorithm

This algorithm is used to compress data. An example of Huffman encoding is given in [10]. In this technique a tree is constructed to represent the data based on the frequency of characters or numerals. At the end of the procedure the tree is traversed to the character or numeral we want, outputting a 0 every time we take a left branch, and a 1 every time we take a right branch to produce the Huffman code which is compressed compared to original data.

2.2. Run Length Encoding Algorithm (RLE)

The code in RLE is constructed by giving the number of repetitions of a value then followed by the value to be repeated. Thus, as an example the sequence "bbbbbDDDDDDDDDDDDDD" when compressed yields "5b14D". Thus, the compression gain is (19-5) /19 which equals 73.7%. On contrast, the sequence "efficient" when compressed yields "1e2f1i1c1i1e1n1t". In this case, a negative compression gain of (9-16)/9 which equals -78% can be calculated and the reason of this large negative gain is that the sequence contains little character repetition [11].

2.3. Transmission technology

There are many ways of sending and receiving multimedia data in the Internet, but they depend on type of data being sent and received over the Internet. If many clients are requesting data over the Internet simultaneously, efficient allocation of bandwidth to every client and satisfying the QoS requirements of every application is a challenging task. The type of transmission techniques chosen can make difference in the outcome of whole network architecture [12].

3. DATA AND IMAGE COMPRESSION SERVICE USING HTTP-SOAP

The Image Web Service provides clients to receive compressed data and images via Simple Object Access Protocol (SOAP) request and response model. The Web Services Description Language (WSDL) offers three methods or services in SOAP to retrieve data and images:

- Image and Data with no compression
- Using Huffman compression
- Using RLE (Run Length Encoding) compression

3.1. Architecture Diagram

The architecture presented in this paper is built on J2EE framework with Apache AXIS for Web Services and compression algorithm logic implemented on the Web Service Server side and decompression algorithm implemented on the client.

The client in this implementation can be a browser or any application that can communicate with HTTP SOAP using the published ImageServiceWSDL. The API returns static map image or text files.

The architecture contains the following core elements that trigger the image/data file exchange between the client and the server that take part in a transaction as shown in Figure 2.

- WSDL (ImageService) Server Side
- Server side code implementation of the Web Service
- Implementation of Huffman Compression Algorithm
- Implementation of RLE Algorithm
- Implementation of Huffman decompressing utility

3.2. Development Environment

The development is based on distributed n-tier architecture with two Enterprise Archive (EAR) files where the first file is deployed at the server side and the second file at the client side. A full development cycle has been followed from designing the classes using IBM Rational XDE to developing code for Web Services, Huffman and RLE Algorithms.

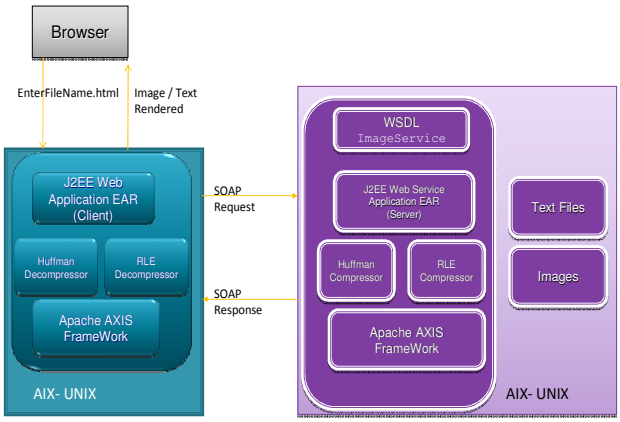

Figure 2. Image / Data Service J2EE Architecture

3.2.1. Development details

The following is a list of the different components used in the development:

- IBM Development Environment (RAD 7.x)
- IBM WebSphere Application Server Version 6.x
- JDK version 5.x
- Full code has been written on server side and client side for the following
  - WebSerivce WSDL
  - WebService Server side implementation
  - WebService Server side Huffman Compressor
  - WebService Server side RLE Compressor
  - Client Side Implementation
    - WebService Client
    - WebService Huffman DeCompressor
    - WebService RLE DeCompressor

3.3. Challenges in Implementing the Algorithms in J2EE Web Services

Various challenges are encountered dealing with compression algorithms in J2EE web services. These challenges include challenges implementing Huffman Algorithm, limitation of RLE algorithm, and overhead of Web Service.

3.3.1. Huffman Algorithm

The following challenges were encountered while implementing Huffman Algorithm:

1. Block Preparation: From RGB to Y, CR, CB planes.
2. Transform: Two-dimensional Discrete Cosine Transform (DCT) on 8x8 blocks.
3. Quantization: Compute Quantized DCT Coefficients.

3.3.2. RLE algorithm

RLE uses simple compression of repeated characters in sequence. Such a pattern rarely happens in an image.

3.3.3. Web Service

Each algorithm requires extra parameters to be able to decode the compressed image. All these parameters are transmitted in bytes.

3.4. Class Diagram for the Server Side

Figure 3 shows the class diagram for the server side.

3.5. Class Diagram for the Client Side

Figure 4 shows the class diagram for the client side.

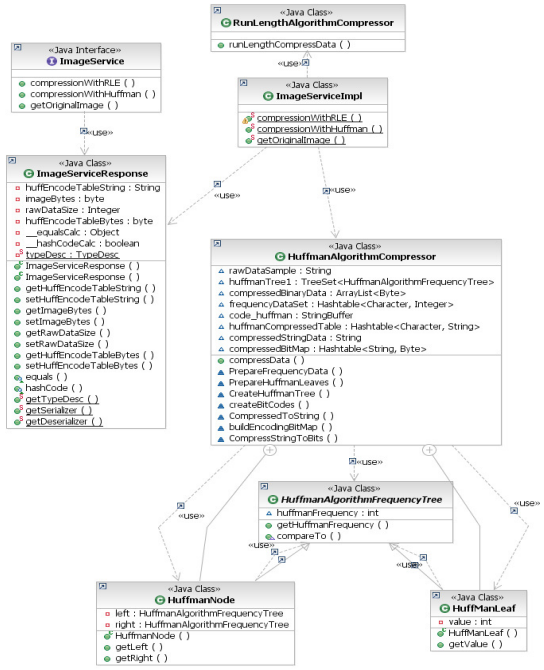

Figure 3. Class diagram for the server side

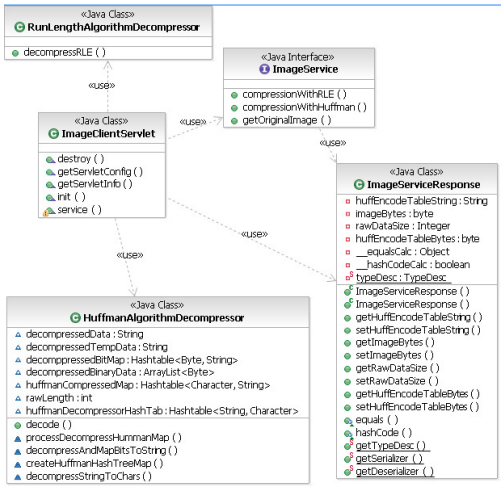

Figure 4. Class diagram for the client side

3.6. Application

The web application can be accessed using any web browser. Figure 5 shows the web page that will be displayed to the client.

Any of the options can be selected to view the file. In our example, we will choose a text file and have Huffman compression applied. The following is the sequence of steps that are triggered to complete Round Trip Time (RTT):

1. User request is submitted to the client application.
2. Client application creates a SOAP request with type of compression and chooses appropriate method call in the WSDL.
3. Request is sent to the server.
4. Server receives the SOAP request and processes the input parameters.
5. Requested file is read and compressed using either Huffman or RLE based on the compression method selected by the client.
6. SOAP response is created.
7. SOAP response is sent to the client.
8. Client receives the request.
9. Client applies decompression algorithm.
10. Client sets the content on the browser and flushes the data.
11. User sees the data.

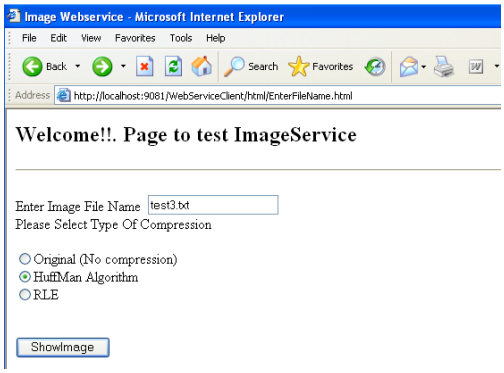

Figure 5. Web page displayed when the application is accessed

4. RESULTS

The following sub-sections summarize the results obtained.

4.1. Huffman Compression on Images

We have tested the Huffman algorithm with ten small size image files. The size of the image files varies from 750 K bytes to about 5.7 M bytes. Table 1 shows the test results for Huffman compression algorithm.

Figure 6 shows the compression ratios between the original and the compressed files. Figure 6 shows higher compression ratio with large files. There are two exception though (Test Run 4 and 6) as these samples have more colors and more patterns. There is not much advantage of using color vs. monochrome images, the compression is very much consistent with the notion of higher compression is achieved with larger image files.

Table 1. Test results for Huffman compression algorithm

Test Run	Original Size in KB	Compressed Size in KB	Huffman Compression Ratio	Time for RTT in Seconds Huffman	Time for RTT in Seconds Original	Image Type
Run 1	756.42	161.97	4.67	1.02	0.02	Color
Run 2	1492.92	286.23	5.13	1.02	0.02	Color
Run 3	1559.67	296.36	5.26	1.01	0.02	Color
Run 4	2006.36	562.63	3.57	1.02	0.02	Color
Run 5	3244.36	566.67	5.72	1.01	0.02	Color
Run 6	3454.36	836.88	4.12	1.01	0.02	Color
Run 7	3578.30	661.09	5.41	1.01	0.02	Monochrome
Run 8	4646.67	854.90	5.44	1.01	0.02	Color
Run 9	4765.36	884.39	5.39	2.02	0.02	Monochrome
Run 10	5773.42	987.22	5.85	2.02	0.02	Monochrome

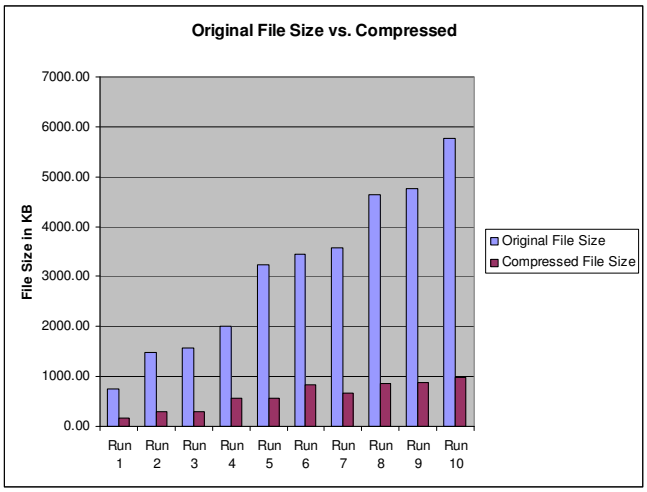

Figure 6. Compression ratios between the original and the compressed files

Figure 7 shows the RTT time comparison between the original file and the compressed file using Huffman encoding.

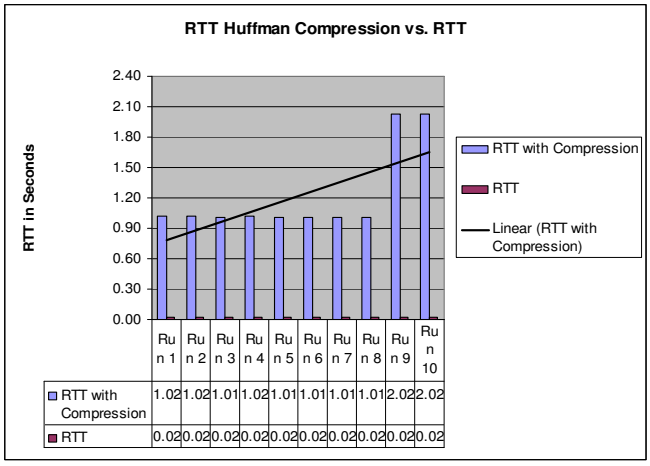

Figure 7. RTT time comparison between the original file and the compressed file using Huffman encoding.

In the RTT comparison, we see a common theme in response time from the server to the client. The uncompressed images have a consistent response time of 0.02 seconds through out the

sample data regardless of size. This may be due to the case the client and the server is running on the same machine, however when compression is applied, it remains in a consistent pattern and changes from 1.02 seconds to 2.02 seconds when the file size's change from about 4 Mb to 5 Mb.

4.2. Huffman Compression vs. RLE on Data Files

We have tested both Huffman and RLE on small text files. Table 2 shows the results of testing using these two algorithms.

To compare Huffman with RLE in text data, various samples have been chosen as RLE is known for its best performance when repeatable sequenced characters are given. We tested Huffman and RLE with sample data representation from News articles, Google search result and stream of sequential characters and combination. Figure 8 shows the original file size and the compressed file size using both Huffman and RLE.

According to the test runs with Huffman and RLE, it's worth noting that based on the content of data, RLE Algorithm can work against compression as RLE is primarily based upon consecutive occurrences. When the data is unique, i.e. (not in a repeatable sequence of characters) RLE will work the opposite in case of Run 1, 2, 3, 5 and 9. The best performance can be observed in RLE is in Run 4, 6, 7, 8 and 10 as the data sample has multiple occurrences of the same character.

Table 2. Huffman and RLE test results on small text files

Test Run	Original Size in KB	Huffman Compressed Size in KB	Huffman Compression Ratio	RLE Compressed Size in KB	RLE Compression Ratio	Data Type
Run 1	7.08	4.14	1.71	6.16	0.512	BBC News
Run 2	11.35	3.48	1.82	13.89	0.509	BBC News
Run 3	12.67	6.97	1.82	24.72	0.512	BBC News
Run 4	18.05	9.52	1.90	8.94	2.01	Stream of Sequential Characters and Combination
Run 5	21.47	13.34	1.61	41.31	0.519	Google Search Result
Run 6	36.09	19.02	1.90	17.88	2.01	Stream of Sequential Characters and Combination
Run 7	56.45	30.14	1.87	29.01	1.94	Stream of Sequential Characters and Combination
Run 8	65.87	35.16	1.87	33.30	1.97	Stream of Sequential Characters and Combination
Run 9	71.08	43.20	1.64	123.59	0.575	Microsoft MSDN XML File
Run 10	82.63	43.53	1.90	40.77	2.02	Stream of Sequential Characters and Combination

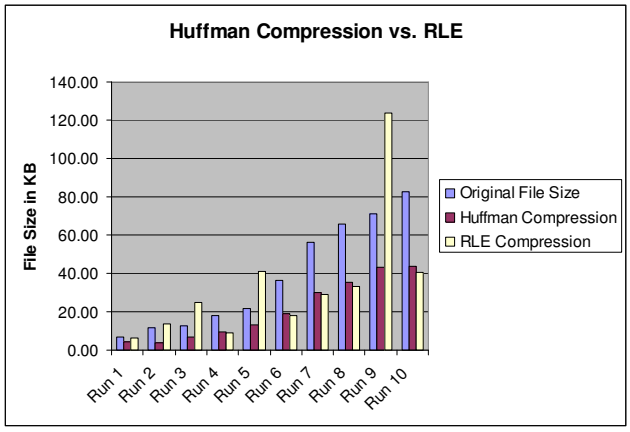

Figure 8. Original file size and the compressed file size using both Huffman and RLE.

Huffman algorithm for text compression can be seen as a consistent compression regardless of the text data. As in our sample test data, we have text from NEWS, Google search result and repeatable sequence of characters. In Figure 8, we compare compression ratio between Huffman and RLE.

Three unique patterns can be identified in comparison of Huffman and RLE.

1. RLE performs better when repeatable sequences of patterns are found in data.
2. RLE performs opposite and increases the file size to around double its original size when it does not encounter repeatable pattern.
3. Huffman on the other hand is very consistent in compression ratio and can compress the file up to 200% regardless of data sample.

4.3. Impact on Images and Data based on error rate

In this paper we are dealing with compression at the application layer. The data link layer is responsible to verify and fix any error that can result in transmission. To study the effect of transmission errors on the quality of images we are introducing error(s) in the range of 5% to 50%. Figure 9 shows the resulted images at different error rates.

5. CONCLUSION

Currently web services are becoming a de facto standard in the corporate world. The idea of reusability and write once and use any where, fits the ideology by adapting to Service Oriented Architecture. With images and large sets of data being used in the corporate world, it's worthy to mention when proper data compression techniques are applied, it can support not only larger data transmission but also efficient usage of bandwidth and response time.

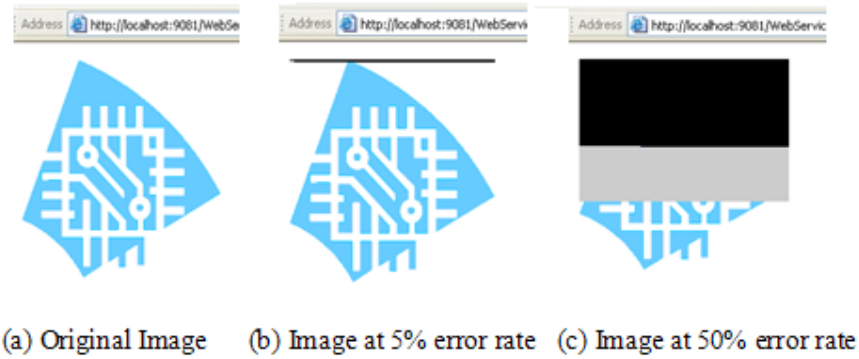

Figure 9. Original image versus corrupted image at 5% and 50% error rates.

Both Huffman and RLE algorithm's have their advantages and limitations. Due to the recursive nature to create the Huffman frequency, large amount of image manipulation can result in delayed transactions and complex coding procedures.

We have also seen how RTT can be affected with the introduction of compression; in most cases of our sample data an introduction of about 1 second has been seen. In real world scenarios, one second can make a big difference as real time applications in many Claims Adjudication and Banking systems need response time less than one second. However, compression can be effectively utilized in batch processing systems in similar banking or healthcare industries, where large amount of data can be reduced up to 200% when effective compression techniques are applied e.g. Huffman Algorithm, thus saving large network traffic and effective utilization of resources.

On the other hand, the implementation of RLE compression algorithm is not difficult. However, it is inefficient except for data with many consecutive repeated elements such as in case of images with large uniform areas.

REFERENCES

[1] Webex Video Conferencing: http://www.webex.com/.

[2] Kuros, J. F. & Ross, K. W., (2004) "Computer Networking: A Top-Down Approach Featuring the Internet", 3rd Edition.

[3] Lazar, A. A., (1994) "Challenges in Multimedia Networking," Retrieved from - www.ee.columbia.edu/~aurel/papers/programmable_networks/high_tech_forum_osaka94.pdf

[4] Rowe, Lawrence A. & Jain , Ramesh, (2005) "ACM SIGMM retreat report on future directions in multimedia research," ACM Transactions on Multimedia Computing, Communications, and Applications TOMCCAP) Vol. 1 , No. 1, pp 3 – 13.

[5] Wu, Chung-Ping & Kuo, C.-C.J.,(2005) "Design of integrated multimedia compression and encryption systems," IEEE Transactions on Multimedia, Vol. 7, No. 5, pp828 – 839.

[6] Kavousianos, X.; Kalligeros, E.& Nikolos, D ., (2007) "Optimal Selective Huffman Coding for Test-Data Compression," IEEE Transactions on Computers, Vol. 56 ,No. 8, pp. 1146-1152.

[7] "The Data Compression Resource on the Internet "Retrieved from - http://www.data-compression.info/Algorithms/RLE/index.htm

[8] Yang, En-hui & Wang, Longji, (2009) "Joint Optimization of Run-Length Coding, Huffman Coding, and Quantization Table With Complete Baseline JPEG Decoder Compatibility," IEEE Transactions on Image Processing, Vol. 18, No. 1, pp 63 – 74.

[9] Shaaban, E., (2000) "Data Compression Basics," Retrieved from - http://meseec.ce.rit.edu/eecc694-spring2000/694-5-9-2000.pdf

[10] Wiseman, J., "A quick tutorial on generating a Huffman tree," Retrieved from - http://www.siggraph.org/education/materials/HyperGraph/video/mpeg/mpegfaq/huffman_tutorial.html

[11] "RLE Compression" Retrieved from - http://en.kioskea.net/contents/video/compimg.php3

[12] Shyu, M. L., Chen, S. C., Luo, H., (2002) "Self-Adjusted Network Transmission for Multimedia Data," Retrieved from - http://www.eng.miami.edu/~shyu/Paper/2002/ITCC02.pdf

Authors

Eman Abdelfattah had received the MS Degree in Computer Science from the University of Bridgeport in 2002. She worked as a programmer and computer teacher in several places in the period from 1983 to 2000. She worked as a C++ and Java instructor in the Continuing Education Department, Housatonic Community College, Bridgeport, Connecticut. Currently, she is working as an adjunct instructor at the University of Bridgeport.

She has research interests in the areas of networking and communications. Her research results were published in prestigious international conferences in circuits and VLSI design. She actively participated as a committee member of the International Conferences on Engineering Education, Instructional Technology, Assessment, and E-learning EIAE 05, EIAE 06, EIAE 07, EIAE 08, and EIAE 09.