2004-12

Utilization of forward error correction (FEC) techniques with extensible markup language (XML) schema-based binary compression (XSBC) technology

Norbraten, Terry D.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

UTILIZATION OF FORWARD ERROR CORRECTION (FEC) TECHNIQUES WITH EXTENSIBLE MARKUP LANGUAGE (XML) SCHEMA-BASED BINARY COMPRESSION (XSBC) TECHNOLOGY

by

Terry D. Norbraten

December 2004

Thesis Advisor:	Don Brutzman
Co Advisor:	Don McGregor
Second Reader:	Duane Davis

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2004	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Utilization of Forward Error Correction (FEC) Techniques with Extensible Markup Language (XML) Schema-based Binary Compression (XSBC) Technology			**5. FUNDING NUMBERS**
6. AUTHOR Terry D. Norbraten			
7. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES: The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			**12b. DISTRIBUTION CODE**

13. ABSTRACT

In order to plug-in current open sourced, open standard Java programming technology into the building blocks of the US Navy's ForceNet, first, stove-piped systems need to be made extensible to other pertinent applications and then a new paradigm of adopting extensible and cross-platform open technologies will begin to bridge gaps with old and new weapons systems. The battle-space picture in real time and with as much detail, or as little detail needed is now a current vital requirement. Access to this information via wireless laptop technology is here now. Transmission of data to increase the resolution of that battle-space snapshot will invariably be through noisy links. Noisy links such as found in the shallow water littoral regions of interest will be where Autonomous Underwater and Unmanned Underwater Vehicles (AUVs/UUVs) are gathering intelligence for the sea warrior in need of that intelligence.

The battle-space picture built from data transmitted within these noisy and unpredictable acoustic regions demands efficiency and reliability features abstract to the user. To realize this efficiency Extensible Markup Language (XML) Schema-based Binary Compression (XSBC), in combination with Vandermode-based Forward Error Correction (FEC) erasure codes, offer the qualities of efficient streaming of plain text XML documents in a highly compressed form, and a data self-healing capability should there be loss of data during transmission in unpredictable transmission mediums.

Both the XSBC and FEC libraries detailed in this thesis are open sourced Java Application Program Interfaces (APIs) that can be readily adapted for extensible, cross-platform applications that will be enhanced by these desired features to add functional capability to ForceNet for the sea warrior to access on demand, at sea and in real-time. These features will be presented in the Autonomous Underwater Vehicle (AUV) Workbench (AUVW) Java-based application that will become a valuable tool for warriors involved with Undersea Warfare (UW).

14. SUBJECT TERMS: Autonomous Underwater Vehicle (AUV) Workbench (AUVW), Erasure Codes, Extensible Markup Language (XML), Extensible Modeling and Simulation Framework (XMSF), Forward Error Correction (FEC), Scenario Authoring & Visualization for Advanced Graphical Environments (SAVAGE), XML Schema based Binary Compression (XSBC)			**15. NUMBER OF PAGES** 191
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

UTILIZATION OF FORWARD ERROR CORRECTION (FEC) TECHNIQUES WITH EXTENSIBLE MARKUP LANGUAGE (XML) SCHEMA-BASED BINARY COMPRESSION (XSBC) TECHNOLOGY

Terry D. Norbraten
Lieutenant, United States Navy
B.S., Norfolk State University, 1994

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN MODELING, VIRTUAL ENVIRONMENTS AND SIMULATION (MOVES)

from the

NAVAL POSTGRADUATE SCHOOL

December 2004

Author:	Terry D. Norbraten
Approved by:	Dr. Don Brutzman Thesis Advisor
	Don McGregor Thesis Co-Advisor
	CDR Duane T. Davis, USN Thesis Second Reader
	Rudolph P. Darken Chair, MOVES Academic Committee

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

In order to plug-in current open sourced, open standard Java programming technology into the building blocks of the US Navy's ForceNet, first, stove-piped systems need to be made extensible to other pertinent applications and then a new paradigm of adopting extensible and cross-platform open technologies will begin to bridge gaps with old and new weapons systems. The battle-space picture in real time and with as much detail, or as little detail needed is now a current vital requirement. Access to this information via wireless laptop technology is here now. Transmission of data to increase the resolution of that battle-space snapshot will invariably be through noisy links. Noisy links such as found in the shallow water littoral regions of interest will be where Autonomous Underwater and Unmanned Underwater Vehicles (AUVs/UUVs) are gathering intelligence for the sea warrior in need of that intelligence.

The battle-space picture built from data transmitted within these noisy and unpredictable acoustic regions demands efficiency and reliability features abstract to the user. To realize this efficiency Extensible Markup Language (XML) Schema-based Binary Compression (XSBC), in combination with Vandermode-based Forward Error Correction (FEC) erasure codes, offer the qualities of efficient streaming of plain text XML documents in a highly compressed form, and a data self-healing capability should there be loss of data during transmission in unpredictable transmission mediums.

Both the XSBC and FEC libraries detailed in this thesis are open sourced Java Application Program Interfaces (APIs) that can be readily adapted for extensible, cross-platform applications that will be enhanced by these desired features to add functional capability to ForceNet for the sea warrior to access on demand, at sea and in real-time. These features will be presented in the Autonomous Underwater Vehicle (AUV) Workbench (AUVW) Java-based application that will become a valuable tool for warriors involved with Undersea Warfare (UW).

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I am most grateful for and would be entirely remiss for not mentioning the utmost patience, guidance, and inspiration, who with a most reassuring voice and by the gentle manner of my thesis advisor Prof. Donald Brutzman, my time would have been spent in vain. Without his most valuable course, Hamming on Hamming: Learning to Learn, I would have had no idea what FEC was or where to start in this research. His honest enthusiasm to train and increase the skill and situational awareness of the sea warrior efficiently, cost effectively, and while having great fun doing so, was a most sobering and delightful eye opener.

The principal developer of XSBC now lies with Alan D. Hudson, president of Yumetech Inc. (see Yumetech). Alan and his team are contracted by and fellow collaborators with other team research associates of the MOVES Institute at NPS. I would like to thank Alan and his fellow Yumetech associate Justin Couch for their valuable insight into XSBC and Java Swing technologies.

Don McGregor was most valuable in understanding networking theory programming in Java as with Dr. Mark Pullen who introduced me to the world of wireless multicasting through his MV 4774 Networked Virtual Environment (NVE) course held over the internet from George Mason University (GMU) to NPS via the Network Education Ware (NEW).

Capt. James Neushul, USMC opened up a whole new world to me with his implementation of a non-proprietary, open source, open standards solution with his wonderful X-Globe 3D terrain mapping web service application.

CDR Duane Davis for his most valuable mentorship, volunteering as technical sounding board and leadership.

Finally, my friend and confident through over two years of learning struggles, Maj. Glenn Hodges, USA who inspired me to keep moving forward and learn the ever dynamic language of Modeling and Simulation (M&S).

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PROBLEM STATEMENT

Major strides in underwater communications and data networking have taken place in the last century and this. Sound energy sometimes does not propagate very efficiently or reliably under water and especially in shallow water littoral regions. Yet, the US Navy is moving forward with promising results in the research and development (R&D) of Autonomous Underwater and Unmanned Underwater Vehicles (AUVs/UUVs). These devices carry with them the enormous potential of detection, classification and neutralization of sea mines, in 3D mapping of sea floors, in support of salvage operations, exploration, oceanography and meteorological research, and a myriad of other areas of ocean research.

The desire to cheaply setup and maintain an Underwater Acoustic Network (UAN) has been the goal of a plethora of diverse marine scientists who are ever approaching this goal closer than ever before. Technology in underwater modems and data transfer capabilities is to the point where this can be fully realized.

At the Naval Postgraduate School (NPS), one of the main goals of the Modeling, Virtual Environments and Simulation (MOVES) Institute is to collaborate with the Undersea Warfare (UW) curriculum, also at NPS, to develop AUV technology for the above mentioned purposes, plus to aid the at sea warfighter with capabilities of on demand, real-time information exchange so that tactical decisions concerning aspects of USW can be made at the click of a mouse. From accurate sonar prediction models to 3D visualization of AUV collected data, progress is moving forward continually to realize this potential.

Current work to put this powerful capability into the toolkit of the sea warrior is being conducted by continual development of the AUV Workbench (AUVW). This is a powerful open source, open standards Java-based application to plan, execute missions, retrieve data both binary and in 3D for analysis and decision making, and for training in the AUV/UUV area of UW. How that data is transmitted reliably and efficiently underwater remains a formidable challenge, but one that can be overcome well.

The problem statement lies here. How can data be efficiently streamed and reliably retrieved in the often hostile to sound energy propagation conditions of shallow water littoral regions to promote furtherance of AUV/UUV technologies? This thesis proposes an open sourced Java-based solution to integrate into the AUVW application.

B. MOTIVATION

AUV and UUV technology is growing at a rapid pace. In order to leverage this technology for advantage and exploitation by the at-sea warfighter, current technological pace has to be kept and maintained by the appropriate agencies who would place this technology into the hands of that warrior tasked with making critical life and death decisions. With the disadvantage and enormously costly stove-piped proprietary systems that the Department of Defense (DOD) employs today, pace will be further hindered in placing that critical technology into the toolkit of the sea warrior.

Movement towards open standards and open sourced solutions must be integrated into a new paradigm-shift for those agencies that procure war fighting systems. Technological pace can be maintained, costs will makedly decrease and efficiency and security of these systems will increase many fold. No longer would a ship at sea on a critical mission have to rely on outdated equipment and software systems that were once maintained at high cost by companies and contractors that have folded or gone out of business or simply do not see profit in maintaining those systems anymore.

The AUVW is one solution to maintaining pace with the technological sectors of the defense industry as students and other interested collaborators all have valuable ideas and features to offer this application. The main feature of the AUVW is its ability to plan and execute missions on demand as a web service for AUV/UUV employment. Data is collected for a mission's intended purpose and transmitted back to a gateway server for analysis and intelligence gathering purposes. In order to transmit this data efficiently and reliably from a vehicle that possess low and exhaustible power supplies, much like satellites and space probes, data transmission must be quick, efficient and reliable when received by its intended client. This thesis gives background into the efficiency solution by introducing XSBC. The main thrust of this thesis will then be to offer the reliability solution for assuring that the data transmitted will get to its destination intact.

C. OBJECTIVES

This thesis will introduce the feasibility for a software implementation of a Forward Error Correction (FEC) erasure code written in the Java programming language to be used in conjunction with Extensible Markup Language (XML) Schema-based Binary Compression (XSBC) technology as a reliable way to transmit XML messages and data over noisy links such as the shallow water littoral acoustic environments that AUVs and UUVs must operate in to gather data and execute USW missions. This solution will then be integrated into the AUVW application to further is usefulness and potential as a resource for efficient conduct of UW by the sea warrior.

D. THESIS ORGANIZATION

This Chapter offers the problem statement and motivation for this thesis research. Chapter II introduces background information and related work in the field of reliable and efficient data transmission within the realm of UANs. Chapter III gives background into informational theory that led to the invention of FEC codes. Chapter IV gives information of the derivation of a particular kind of FEC in that of erasure codes. Chapter V offers information into the feasibility of combining XSBC technology with FEC to realize the goal of this research. Chapter VI explains in detail how the two technologies were combined within the AUVW. Chapter VII offers information about user interfaces for XSBC and FEC within the AUVW. Chapter VIII covers some pertinent information for security considerations. Chapter IX covers the conclusions and recommendations of this research and offers some ideas for future work to be explored. The appendices, specifically B and C offer source code and familiarity with the XSBC and FEC open source libraries.

II. BACKGROUND AND RELATED WORK

A. INTRODUCTION

ForceNet, as defined by the Chief of Naval Operations (CNO) Admiral (ADM) Vern Clark, is the operational construct and architectural framework for naval warfare in the information age (Clark 2002). Many facets of information superiority and information technology utilized by the warfighter will be incorporated into ForceNet to enable that warfighter to have instantaneous, reliable, on demand or real-time information at his/her fingertips. How that information arrives to that warfighter will be based upon XML formats and those formats will have to be utilized in secure, internet-based environments through a reliable transmission process. The following will present some abstract technologies that pertain to the development of ForceNet and how they will apply to this thesis.

Open Source and Open Standards are keys to reliable and extensible ways to implement information exchange within ForceNet. By enforcing Open Standards compliance with future systems, design add-on products can be developed and readily deployed to the warfighter without delay. XML will be the way to do things here-to-fore were unable to be accomplished. From DOD contracted proprietary and stove-piped software driven systems that are unable to communicate effectively with other pertinent information systems, XML is key to meta-tagging data sets so that cross platform information exchange can take place seamlessly. The Department of the Navy (DON) will be placing more of its data online which will be accomplished by the utilization of XML (French 2002).

The DON Chief Information Officer (DON CIO) chartered the DON XML Work Group to “fully exploit XML as an enabling technology to achieve interoperability in support of maritime information superiority” (Sall 2003).

B. ABSTRACT TECHNOLOGIES HAVING PERTINENCE IN THE FORCENET CONCEPT

1. XSBC

This thesis is primarily based upon the use of FEC erasure codes combined with XSBC as an option for reliable network transmission of binarily compressed XML documents and messages over the noisy links inherent in acoustic environments such as littoral regions around the globe. As development of the AUVW continues at NPS, the need for reliable network transfer of data to and from AUVs was identified prompting research into finding a suitable open source avenue to accomplish this requirement. XSBC [originally called Cross-Format Schema Protocol (XSFP)] was developed as an alternative for networked simulations using hard coded protocols that have to be recompiled at runtime if changes are introduced into the system which detracts from the extensibility and dynamicism of that system (Serin 2003). XSBC is used by the AUVW to compress XML documents and messages for efficient use of limited network capacities available to the at-sea warfighter. A more in depth discussion of XSBC is presented in Chapter V titled Suitability of Combining FEC Encoding with XSBC.

2. Internet Protocol over Sea Water (IP/SW)

The need for reliable transmission of encoded data through noisy acoustic links was presented first in (Reimers 1995). This need is amplified by the fact that repeated retransmission of messages received incorrectly can result in AUV transmitter failure due to limited power supplies in these vehicles. By showing how to enable Internet Protocol (IP) packets to be transmitted reliably the first time over an underwater channel, at great step was taken to provide low-cost reliable transmission of data in the acoustic environment.

The example for FEC in this work was primarily based upon Hamming codes that correct bit errors. Bit error correction was the first type of FEC employed after Dr. Hamming invented them in 1947. This thesis will present an erasure code that is concerned with dropped packet data reconstruction rather that bit error corrections.

3. World Wide Web Consortium (W3C) XML Binary Characterization Working Group

The XML Binary Characterization Working Group is tasked with gathering information about uses cases where the overhead of generating, parsing, transmitting, storing, or accessing XML-based data may be deemed too great for a particular application, characterizing the properties that XML provides as well as those that are required by the use cases, and establishing objective, shared measurements to help judge whether XML 1.x and alternate (binary) encodings provide the required properties (see XML Binary Characterization Working Group). The XML Binary Characterization Working Group was created as a result of the Binary Interchange Workshop, which identified XML particular needs as stated above. In (Binary Interchange Workshop Report 2003) many collaborative efforts are documented with the goal of realizing the utility of compressing XML documents and messages, comparing Infoset level representations with other methods so that a specification of an interoperable transmission format may be realized. The working group is chartered under a royalty-free basis to produce deliverables that enhance XML-based data transfers of which XSBC is one of those deliverables.

4. Sun Microsystems Fast Web Services

Another enhancement to XML-based efficient data transfer is Sun Microsystems' Fast Web Services initiative. Fast Web Services are aimed at the identification of performance problems in existing implementations of Web Services standards (Sandoz and others 2003). The Sun Microsystems group has explored several solutions to these problems. Fast Web Services explores the use of more efficient binary encodings as an alternative to textual XML representations. XML's self-describing nature has significant advantages, but they come at the price of bandwidth and performance. XML-based messages are larger and require more processing than existing protocols such as Remote Method Invocation (RMI), RMI Internet Inter-Object Request Broker Protocol (RMI/IIOP) or Common Object Request Broker Architecture IIOP (CORBA/IIOP): data is represented inefficiently, and binding requires more computation.

Fast Web Services attempts to solve these problems by defining binary-based messages that consume less bandwidth and are faster and require less memory to be processed. Two different solutions have been identified: Fast schema and Fast infoset. As its name suggests, the Fast schema approach uses information from a document's schema to optimize which parts of the infoset to include in a message. Fast infoset, on the other hand, is a pure infoset-based drop-in replacement for XML. Both alternatives have advantages and disadvantages; application requirements will dictate which of the two approaches is more appropriate. Fast infoset is an encoding that is tokenized and thus more compact but also faster to both generate and consume. Fast schema can only be used when both the producer and consumer of the data have access to its schema.

5. Seaweb Underwater Acoustic Network (UAN)

An ongoing project between the National Oceanographic Partnership Program (NOPP), Space and Warfare (SPAWAR) Systems Center San Diego and the Office of Naval Research (ONR), titled US Navy Seaweb, provides for research into a reliably advanced UAN. This network can provide the US Navy with data telemetry and command and control capabilities for a wide set of applications (Rice 2000). This specialized system consists of networked acoustic modems used for wireless real-time delivery of data transmitted in a salt-water acoustic environment from a distributed array of subsurface-networked equipment. This integrated network contains nodes, repeater nodes, gateway nodes and a shore-based control center. Funded experiments with Seaweb consisted of interfacing oceanographic sensing instruments, interfaced with acoustic modems, deployed in trawl-resistant bottom frames along side azimuthally omni-directional acoustic signaling equipment needed for feasible network re-routing (Codiga, Rice and Baxley 2004). Repeaters are individual acoustic modems to relay data so the array covers a larger area. Gateways are buoys with acoustic modems interfaced to cellular telephone modems for communication between the UAN and shore.

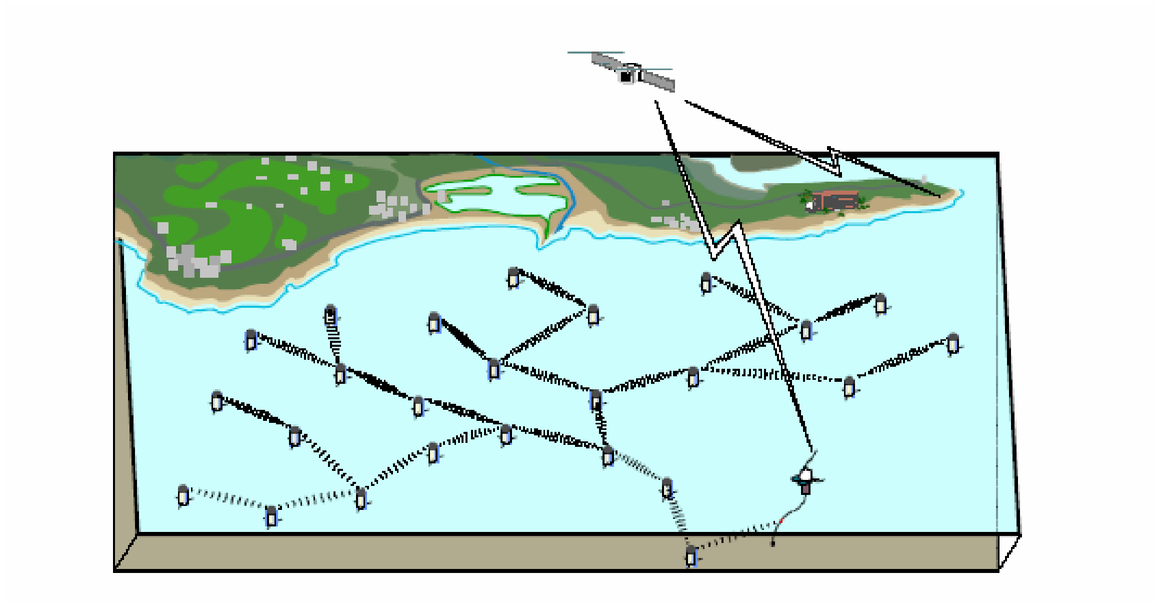


Figure 1. Seaweb Underwater Acoustic Networking Enables Data Telemetry and Remote Control for Undersea Sensor Grids, Vehicles, and other Autonomous Instruments. Gateways to Manned Control Centers Include Radio-acoustic Communications (racom) Nodes with Radio Links to Sky or Shore (From Rice 2000).

The way this system deals with reliable delivery of data packets in this sometimes harsh acoustic network environment is through a protocol designed to facilitate dynamic throughput. In shallow water environments, typically less than 100 meters, surface action wind was the largest contributor of network performance degradations. Other factors such as estuarine flow, varying water temperature columns, upward and downward propagating sound speed profiles, commercial shipping and commercial fishing interferences had noticeable effects on network performance as well.

To mitigate these interferences, this specific UAN protocol first initiates a Request to Send (RTS) from a sensor that has need to transmit data. The nearest network topology repeater then sends a Clear to Send (CTS) message to that sensor. Bit error FEC redundancy may be encoded in the following data packet and if the Bit Error Ratio (BER) was too great for full data recovery by the receiving sensor, an Automatic Repeat Request (ARQ), for the corrupted or dropped data packet to be resent, will be initiated. The process repeats until a gateway has received all data packets per transmission block to be sent via cellular network to a repeater satellite or shore repeater. This full course handshaking and data delivery method ensured that data was reliably delivered during several Seaweb experiments conducted on the west and east coasts of the continental US.

C. SUMMARY

This chapter introduced some technologies that can benefit in increased reliability by using FEC and compressed XML structured documents and data. The goal of this thesis is to present a fusing of these two areas for reliable acoustic networking. By using XML, previous disparate systems can communicate due to the structure of XML and its cross-platform capabilities. Secure, fast and reliable information exchange will always be the goal of past, present and future systems that the warfighter requires to execute successful missions.

III. CLAUDE SHANNON’S INFORMATION THEORY

A. INTRODUCTION

This chapter provides background information regarding work done prior to the derivation of the original Forward Error Correction (FEC) Hamming codes. Claude E. Shannon’s insights and development of Information Theory (Shannon 1948) are synopsized. This theory was the basis for finding utility in error correcting codes by Dr. Hamming. Related work topics in the field of FEC codes are also presented.

B. BACKGROUND IN FORWARD ERROR CORRECTION

1. Impetus that Started it All

Richard W. Hamming, Ph.D. in mathematics, worked at Bell Telephone Laboratories located in Murray Hill, NJ in 1947 when he invented error-correcting codes. He was working on a Model 5 relay computer in New York City in preparation for delivering it to the Aberdeen Proving Grounds along with some mathematical routines programmed in software (Hamming 1997). If an error occurred somewhere in the routines when ran, it was detected by the 2-out-of-5 block codes used then, so, the machine attempted to repeat the step up to three times before dropping the error-riddled routine and then proceeded to pick up the next routine leaving the previous error uncorrected. These routines were run mainly at night and it wasn’t until the next day when the programmer arrived to pick up the results that, indeed, an error in the partially processed data was discovered. Dr. Hamming surmised that if a computer can tell you if there was an error, it should be able to tell where that error occurred and then simply fix the error by changing the state of whatever bit caused the error.

2. Error Correction Theories to Concepts

One theory to combat errors was to build three copies of a machine each with comparator circuits which would then use a “majority vote” (Hamming 1986).

The eventual cost incurred was too great to employ this technique. Dr. Hamming had studied parity checks and their fundamentals and surmised that this would be the route to investigate error correction coding techniques. His first attempt was to arrange message bits in a rectangle, put parity checks on each row and column which when any two parity checks failed they would give the coordinates of the single error causing the parity checks to fail as seen below.

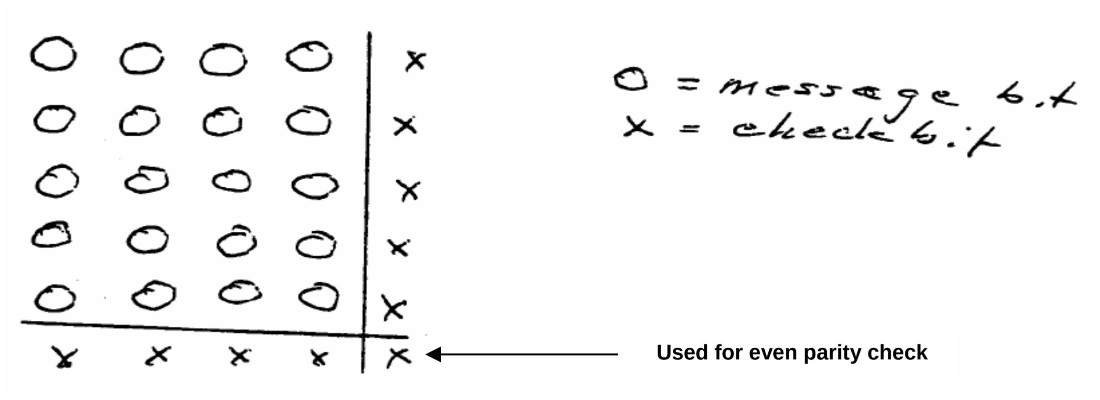


Figure 2. First Error Correcting Codes (After Hamming 1986)

A double error, in this case, would potentially go undetected in this fashion, but he was well on the way to solving the problem of implementing FEC. Some weeks later, as he was riding in the company car to NYC, he thought that if he arranged the message bits in the form of triangle, then he could place the parity check bits along the diagonal, Figure 2, where each parity check bit would identify both the row and the column of the error, so, he thought this would produce more favorable redundancy.

As he journeyed along his route in northern New Jersey he then thought about arranging the message bits in a cube form with parity check bits placed all along the planes of each side, along with a parity check bit placed on the axes, would produce an even better redundant code at a cost of $3n - 2$ parity checks for the n^3 encoded message as seen in Figure 3.

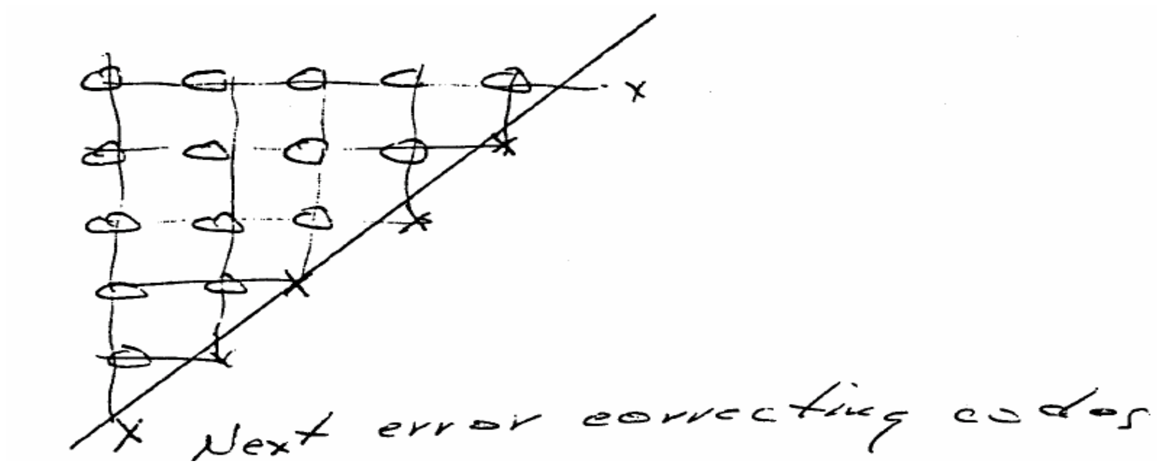


Figure 3. Code Arranged in a Triangular Form with Parity Check Bits Placed on the Diagonal (From Hamming 1997)

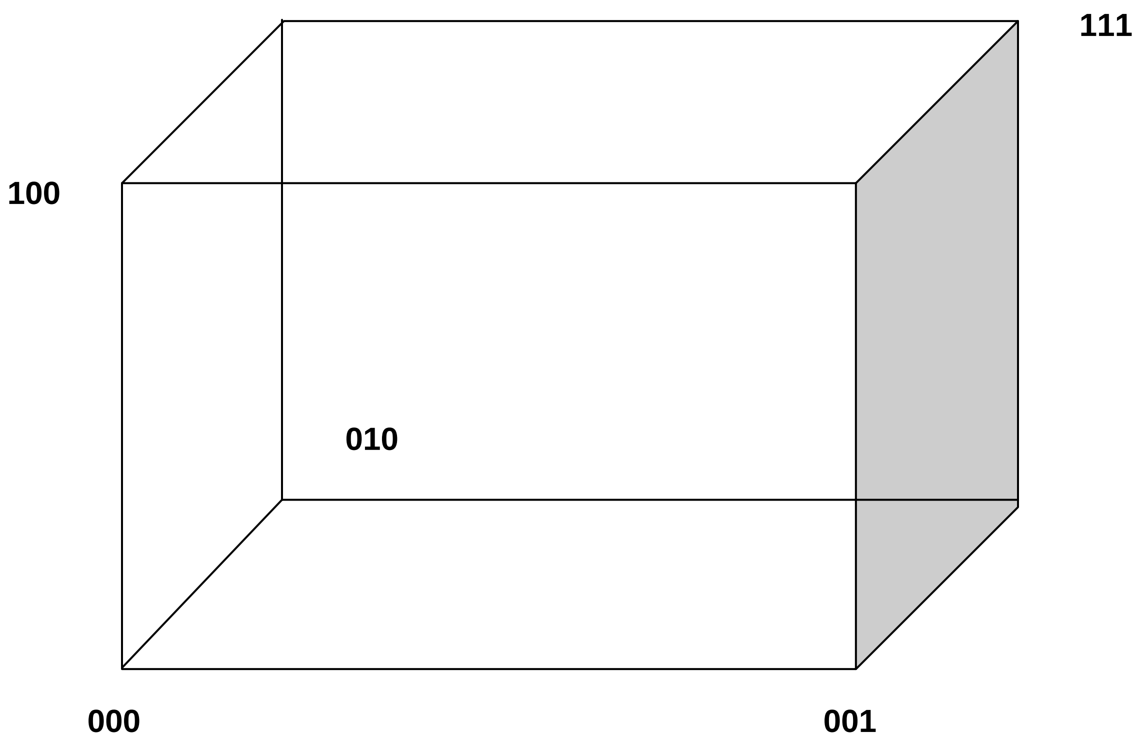
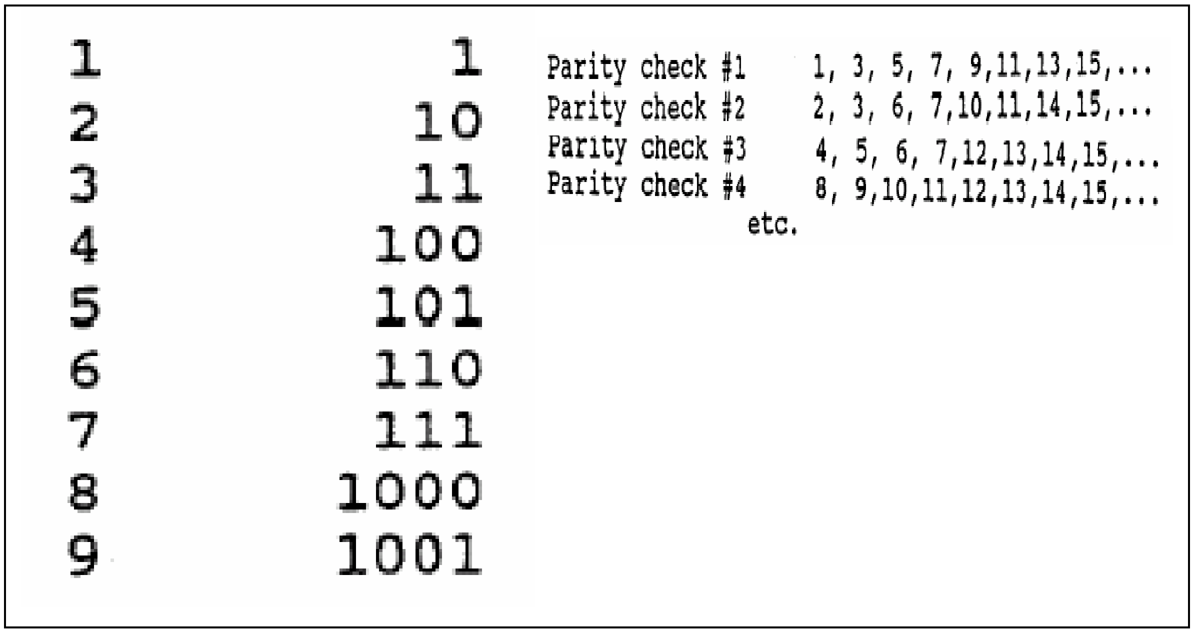


Figure 4. Information Bits Arranged in a Block with Parity Check Bits Placed on the Axes Showing a Maximum Hamming Distance (L_1) of 3 from the Origin

This line of thinking eventually led to Dr. Hamming to ponder the thought of a 2x2x2…x2 dimensional cube with $n + 1$ parity checks. The premise was not to actually build one, but rather to inter-wire a machine that way. These $n + 1$ parity checks could be represented in a string of length $n + 1$ that could represent any of 2^{n+1} things. All that is needed is $2^n + 1$, the 2^n points in the cube and one result that the message was correct. Dr. Hamming figured that he was off by a factor of two in this line of thinking. He eventually set aside this problem for a few days and when he came back to theorizing, he decided a good approach would be to use the syndrome of the error as a binary number that named the place of the error. Figure 5 shows how this might be accomplished. A parity check on the right hand side of a syndrome would involve all positions which have a 1 in the right column and the second digit from the right will involve the numbers which have a 1 in the second column, etc. If any error were to occur in some position, those parity checks would fail producing 1's in the syndrome which will then produce the binary representation of the position of the error.



1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001

```
Parity check #1    1, 3, 5, 7, 9,11,13,15,...
Parity check #2    2, 3, 6, 7,10,11,14,15,...
Parity check #3    4, 5, 6, 7,12,13,14,15,...
Parity check #4    8, 9,10,11,12,13,14,15,...
           etc.
```

Figure 5. Parity Check Bits Naming the Syndrome of an Error (After Hamming 1985)

An example of how this would work follows. Assume a code block with 4 message bit and 3 check bit positions. These numbers satisfy the condition

$$2^3 \geq 7 + 1$$

which is a necessary condition for parity versus message bits. For clarity, chosen are the check positions 1, 2 and 4 as these will scale well to binary representations. The message positions take the other positions of 3, 5, 6 and 7. Let a binary message be 1001. In Figure 6, the message is written on the top line, encoded on the next line and then an error is inserted at position 6 on the next line. On the next lines, parity checks are computed for the three parity check bits chosen. Once the message is received, the parity checks are again applied to the message. In this case, even parity checks will reveal the position of the error once the message in error is decoded. Stripping the parity check bits from the message then flipping the bit in error will produce the original transmitted message.

1	2	3	4	5	6	7	position
		1		0	0	1	message
0	0	1	1	0	0	1	encoded message
0	0	1	1	0	1	1	message with an error
							applied parity checks
1		3		5		7 →	0 passed even parity check
	2	3			6	7 →	1 failed even parity check
			4	5	6	7 →	1 failed even parity check
				Binary number			110 →6 = position of error

Figure 6. Example Decoded Message with the Position of the Error Named (After Hamming 1985)

If one thinks about an all zero message, this will produce an all zero parity check. If a single digit changes, the position of that change will become readily apparent by the syndrome binary number revealing the position of that error exactly. The reader will note that the sum of any two correct messages will always be a correct message, so, these messages will form an additive group modulo two. Correct messages will give all zeros and the sum of a correct message plus any error in a position will give away that position during a decoding process no matter what message is sent. These parity checks focus on the errors and ignore what the content of the message is. Dr. Hamming stated that a good balance is a single error correcting plus a double error detecting code for a message of an appropriate length. If the message is too short, the redundancy cost is too high; however, a message that is too long will run the risk of a double uncorrectable error (Hamming 1997). In single error correcting code terms, this would be a correction that causes a third error.

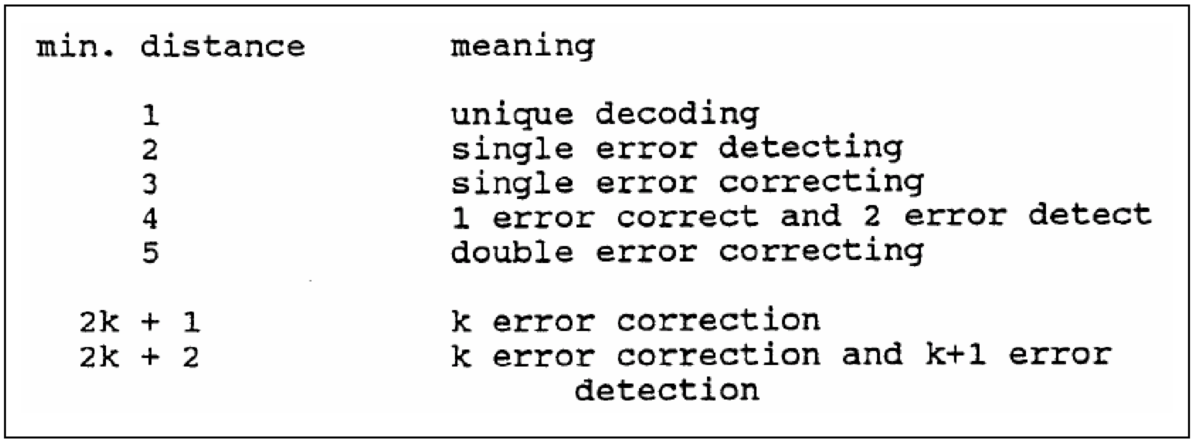
3. N-Dimensional Theory Applied to Error Correcting Codes

In (Hamming 1972), a string of bits can be represented by an n-dimensional cube (see Figure 4) with each string being a vertex of the cube. Any error in a message would then move that string along one edge of an axis, two errors along two edges and so on. Dr. Hamming realized that he would be working in L_1 space where the distance between symbols is the number of positions in which they differ. This metric satisfies these standard conditions for a distance. They are:

1. $D(x, y) \geq 0$ (non negative)
2. $D(x, y) = 0$, iff $x = y$ (identity)
3. $D(x, y) = D(y, x)$ (symmetry)
4. $D(x, y) + D(y, z) \geq D(x, z)$ (triangle inequality)

L_1, or Hamming space, does not use the sum of the square of the distances as in the Pythagorean Theorem, but again, just the sum of the distances much like traveling on a grid in the city. The distance to your destination is the sum of the legs you travel arriving to your destination. Within L_1 space a sphere can be defined as having all points at a fixed distance from its center. Referring back to Figure 4, the origin lies at point (0, 0, 0) and the following points (0, 0, 1), (0, 1, 0) and (1, 0, 0) all lie a unit distance away from the origin. Points (1, 1, 0), (1, 0, 1) and (0, 1, 1) lie two unit distances away and point (1, 1, 1) lies three unit distances away.

In an *n*-dimensional cube, all code points could be thought of as the origin points in spheres of unit radius. These spheres are theoretically packed in this *n*-dimensional cube (a message block), yet do not overlap. A single error in a transmitted message will result in a non-code point which would be recognizable as to where it came from by being somewhere within its unique sphere of radius one about the point from which it was originally transmitted. The minimum distance between code points in this case would be three. If non-overlapping spheres of radius two were used, a double error can be corrected due to the received point being nearer to its original code point than any other point. In this case the minimum distance would be five. Dr. Hamming gives the following table of equivalence minimum distances between code points and the correctability of errors:



min. distance	meaning
1	unique decoding
2	single error detecting
3	single error correcting
4	1 error correct and 2 error detect
5	double error correcting
2k + 1	k error correction
2k + 2	k error correction and k+1 error detection

Table 1. Minimum Distances and Correctability of Errors (From Hamming 1995)

Finding an error correcting code is to find a set of code points in an n-dimensional space that have the required minimum distances between messages. Error correction, in this sense, can be exchanged for more detection by giving up one error correction for two more error detections.

Dr. Hamming gives the upper bound on the number of non-overlapping spheres and code points in the corresponding space by:

$$\frac{2^n}{1 + C(n,1) + C(n,2) + ... + C(n,k)} \geq \#of\, spheres$$

where k = the sphere radius, $C(n, k)$ = the number of points in a sphere of radius k and 2^n = the whole n-dimensional space. To get an extra error detection, one simply adds an overall parity check which increases the minimum distance which before was $2k + 1$ to $2k + 2$.

C. CLAUDE SHANNON'S WORK IN INFORMATIONAL THEORY

Claude E. Shannon, also a Ph.D. in mathematics, worked with Dr. Hamming at Bell Labs in the early 1940s. His original 1948 paper on Information Theory (Shannon 1948) was reprinted with corrections by Bell Labs, now Lucent Technologies (Lucent Technologies), and appropriately titled “A Mathematical Theory of Communication”. At the time of this work, communication was thought of as sending electromagnetic signals down a wire, such as a telegraph, with Morse code. Dr. Shannon proposed a linear schematic model of communications that could possibly send pictures, audio and even video by sending a stream of ones and zeros (binary bits) representing those data types down the same wire. This was a revolutionary vision that we now benefit from as evidenced by advanced multimedia streaming technologies available today.

1. What is Information?

Dr. Shannon identified information with the term “surprise”. He chose the negative of the logarithm of the probability of an event as the amount of information one obtains when the event of probability p happens. As an example, if the reader was told that the ocean looks blue from outer space the with a highest $p \leq 1$, this would equate to not much information. If the reader were told that in certain areas the ocean looks pink from outer space then that would be a surprise and would represent more information. Given that $\log 1 = 0$, the former event contains virtually no information.

Dr. Shannon believed that the measure of the amount of information should be a continuous function of the probability p of an event and for independent events it should be additive, namely, what one would learn from each independent event when added together should be the amount learned from the combined event. To symbolize mathematically, if $I(p)$ is the amount of information one has for an event of probability p then for event x with probability p_1 and for another independent event y with probability p_2 one will obtain for the event of both x and y:

$$I(p_1 p_2) = I(p_1) + I(p_2) \quad (x \, and \; y \, independent \, events)$$

which is the Cauchy functional equation true for all p_1 and p_2. In solving this equation let $p_1 = p_2$ and $p_2 = p$. This then gives:

$$I(p^2) = 2I(p)$$

If $p_1 = p^2$ and $p^2 = p$ then:

$$I(p^3) = 3I(p)$$

By extending this process, one can show by the standard method used for exponents that for all rational numbers m/n:

$$I(p^{m/n}) = (m/n)I(p)$$

From this, it is given that the log is the only continuous solution to the Cauchy functional equation.

Dr. Shannon proposed a customary convention in that to take the base of the log system as 2, so that a binary choice is exactly one bit of information, then information could be measured by the formula:

$$I(p) = -\log_2 p = \log_2(1/p)$$

Information, thus far, has not been defined, but only a formula measuring the amount. This measure, again, depends on the concept of “surprise” which matches well to situations with machines, i.e. telephones, radio, television and computers, but does not represent a normalized human attitude towards information. It should be mentioned that the above formula is only a relative measure depending on one’s state of knowledge. If one were to look at a stream of random numbers from a random source then one could think that each number comes as a surprise, however, if one were given the formula for computing these random numbers then the next number coming down the stream contains no surprise, and consequently, no information.

Dr. Shannon vehemently stuck to his original naming of his work “Information Theory”; however, Bell Labs wanted him to name it Communication Theory (Hamming 1997). This is because, even now, this convention does not fit well with what humans think of as “information”. It gives rise to distortion of the common view of information because it only deals with what Dr. Shannon describes as “surprise”.

2. The Noiseless Coding Theorem of Shannon

Given an alphabet of *q* symbols with probabilities p_i then the average amount of information (the expected value), in the system is:

$$H(P) = \sum_{i=1}^{q} p_i I(p_i) = \sum_{i=1}^{q} p_i \log(1/p_i)$$

This is defined by (Shannon 1948) as the entropy *H* of the system with the probability distribution $\{p_i\}$. The name “entropy” is used because the same mathematical form arises in thermodynamics and in statistical mechanics, which gives rise to an “aura” of importance, which is not justified in this particular case. The same mathematical form does not imply the same interpretation for these *q* symbols; however, a definition for entropy in this case will be given later.

The entropy of a probability distribution does play a central role in coding theory, though, as it represents the number of bits needed for the shortest possible encoding of a message (Hamming 1986 and Wagner 2003). One of the important results for this definition lies in Gibb's inequality for two different probability distributions, p_i and q_i. What had to be proven was:

$$\sum_{i=1}^{q} p_i \log(q_i / p_i) \leq 0$$

It was proven by the following Figure 7. An Equality only occurs when:

$$\log x \leq x-1, \quad (0 \leq x \leq \infty)$$

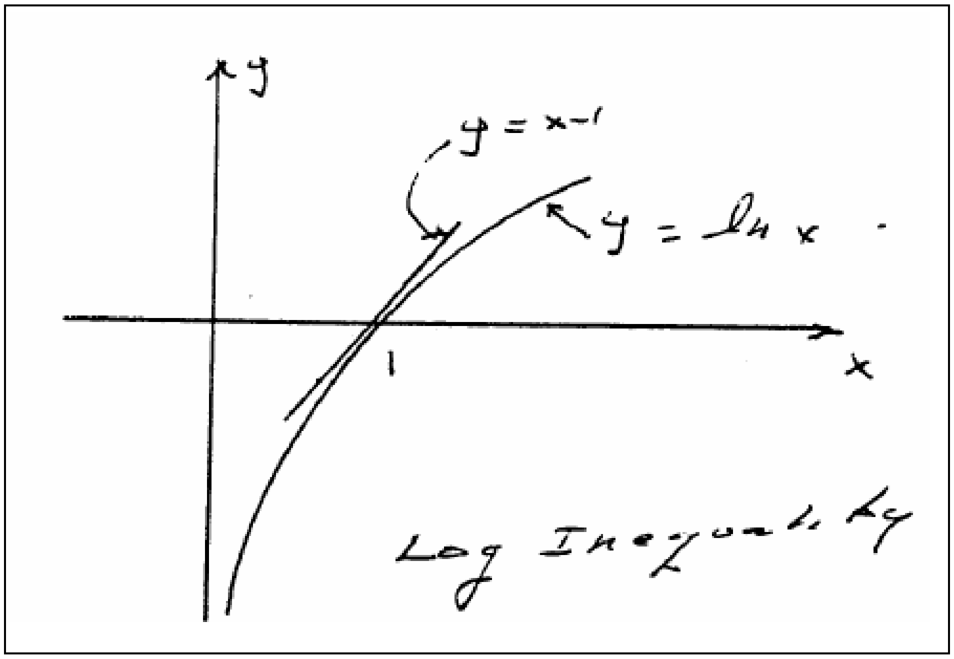


Figure 7. Graphical Representation of a Log Inequality (From Hamming 1995)

Applying the inequality to each term in the sum on the left hand side:

$$\sum p_i (q_i / p_i - 1) = \sum q_i - \sum p_i = 1 - 1 = 0$$

If there are q symbols in the signaling system and choosing the $q_i = 1/q$ then Gibb's inequality is evident. By transposing the q terms:

$$H(P) \le \log q$$

This states that in a probability distribution, if all *q* symbols are of equal probability of 1/*q*, then the maximum entropy is exactly ln *q*, else the inequality holds true.

In (Hamming 1995), his two lectures on Coding Theory gives utility to the Kraft inequality for defining a uniquely decodable code such as (see Kraft's Inequality):

$$K = \sum 1/2^l i \le 1$$

In naming some pseudo probabilities:

$$Q_i = 2^{-l} i / K$$

where $\sum Q_i = 1$, Gibb's inequality will reveal:

$$\sum_{i=1}^{q} p_i \log(1/(Kp_i 2^l i) \le 0$$

and after some further algebraic manipulation:

$$H(p) = \log K + \sum p_i l_i \le L = average\, code\, length$$

Entropy *H*, now, can be further defined as the lower bound for any encoding, symbol to symbol, for the average code length *L* which is the noiseless coding theorem of Dr. Shannon.

3. Signaling in the Presence of Noise

In signaling systems that use the encoding of a bit stream of independent bits, which go symbol to symbol in the presence of noise means that there is a probability that a bit of information is correct with *p* > ½, and the converse probability *Q* = 1 – *P* that the information bit is altered when transmitted. It will be assumed here that errors are independent and are the same for each bit sent, a form of "white noise".

When encoding a long stream of n bits into one encoded message, the nth extension of a one bit code, and where n will be determined later, regard this message of n bits as central points of origin contained within spheres that are theoretically packed together in an n-dimensional space. Since there is an nth extension and assuming that each message has the same probability of occurring and assuming that there are M messages (M to be determined later), the probability of each initial message is $1/M$.

a. Channel Capacity

In examining the above initial message probability, the idea of channel capacity is presented. The greater details will be left out, but it can be thought of as the maximum amount of information that can be sent through a channel reliably maximized over all possible encodings so that no more information can be transmitted reliably as channel capacity will be maxed. For a binary symmetric channel (used by computer controlled communication links) the capacity C, per bit sent, is given by:

$$C = 1 - H(P) = 1 - H(Q)$$

where P is the probability of no error in any bit sent. For the n independent bits sent the channel capacity will be:

$$nC = n\{1 - H(P)\}$$

If near channel capacity is desired then that amount of information (I) for each of the symbols a_i, $i = 1 \dots M$, and of probability $1/M$, then:

$$I(a_i) = n(C - e_1)$$

when any one of the M equally likely messages a_i are sent. Therefore:

$$M = 2^{n(C-e_i)} = 2^{nC} / 2^{ne_1}$$

With n bits one will expect to have nQ errors. In practice, approximately nQ errors will be in the received message. For large n the relative spread (spread = width, √variance) of the distribution of the number of errors will be become narrow as n increases.

From a sender's point of view a message a_i is to be sent in which a sphere is drawn about it of radius:

$$r = (Q + e_2)n, \quad (e_2 > 0, Q + e_2 < 1/2)$$

which is slightly larger by e_2 than the expected number of errors Q as in Figure 8.

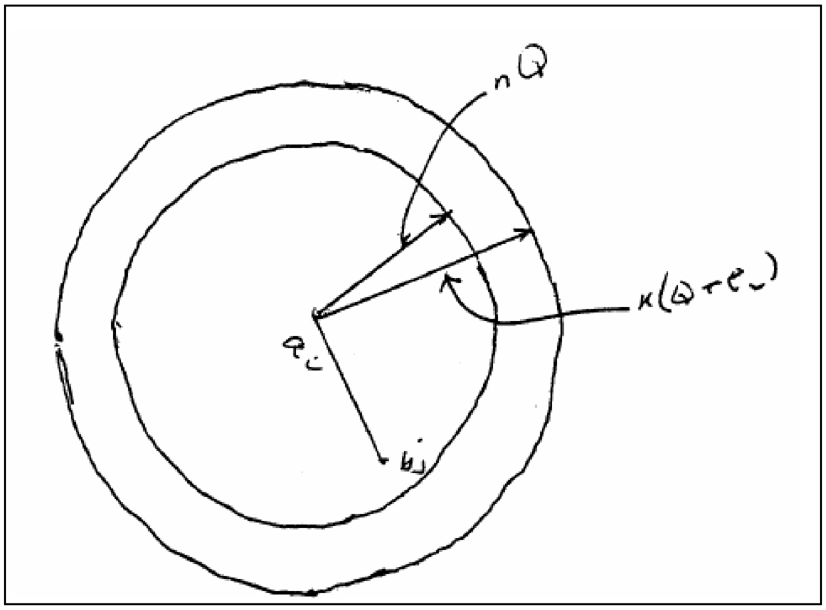


Figure 8. A Sender's Sphere Drawn about a Message Point with Expected Error Probabilities and a Slightly Extended Sphere for the Received Message Point (From Hamming 1995)

If n is large enough then there is a small probability of there occurring a received message point b_j that will fall outside of this sphere. From the sender's point of view, along any radii from the chosen signal a_i to the received message b_j with the probability of an error being within a normal distribution, peaking up at nQ, and with any given e_2, there is an n so large that the probability of the received point b_j falling outside of this sphere will be as small as desired.

Looking from the receiver's point of view as in Figure 9, there is a sphere $S(r)$ of the same radius r about the received point b_j in the space such that if the received message *b_j* is inside the sender's sphere, then the original message *a_i* that was sent is inside the receiving sphere.

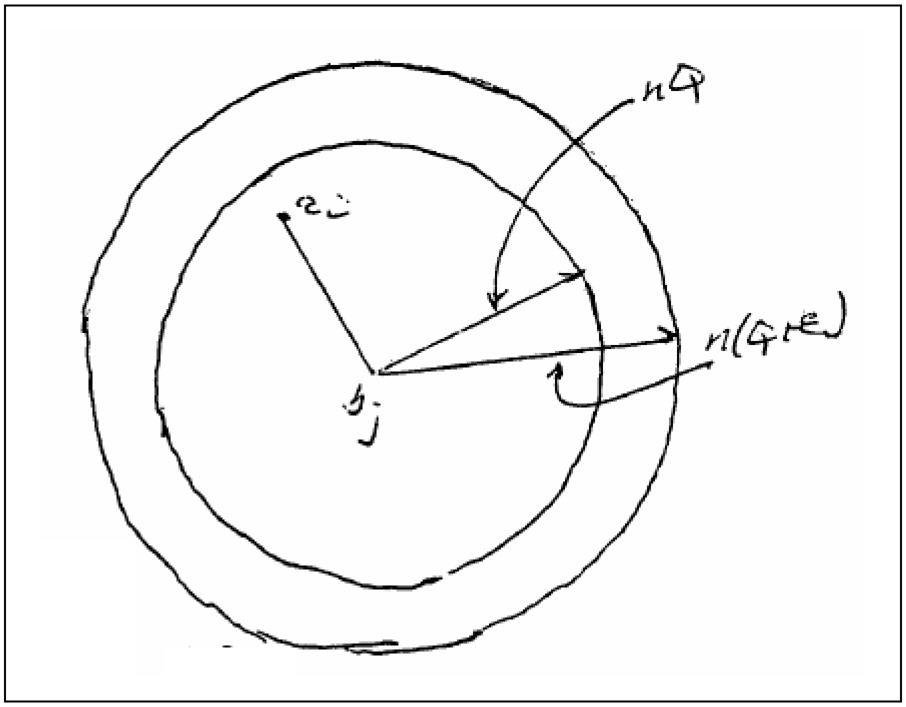


Figure 9. Receiver's Message Sphere with Expected Error Probabilities (From Hamming 1995)

4. Detection of Occurring Errors

How is it that detection of an error occurs? The following Table 2 illustrates this:

case	a_i	another in $S(r)$	meaning
1	yes	yes	error
2	yes	no	no error
3	no	yes	error
4	no	no	error

Table 2. Illustration of Detecting an Occurring Error (From Hamming 1995)

From the above table it can be seen that if there is a least one other original message point in the sphere about the received point then it is an error since one cannot decide which message point is the intended message from the sender. The sent message is correct only if the sent point is in the sphere and there are no other code points within that sphere.

5. Probabilities of Occurring Errors

A mathematical equation for the probability of an occurring error can now be given. If the message sent is *a_i* then:

$$P_E = P\{a_i\, not\, in\, S(r)\} + P\{a_i\ is\, in\, S(r)\} * P\{at\, least\, one\, more\, a_j\ is\, in\, S(r)\}$$

By dropping the first factor in the second term by setting it equal to 1 the following inequality is:

$$P_E \le P\{a_i\, not\, in\, S(r)\} + P\{at\, least\, one\, more\, a_j\ is\, in\, S(r)\}$$

Now, the fact that:

$$P_E(E_1\ and / or\ E_2) \le P(E_1) + P(E_2) - P(E_1 E_2)$$

now gives:

$$P_E(E_1\ and / or\ E_2) \le P(E_1) + P(E_2)$$

and applied repeatedly to the last term on the right:

$$P_E \le P\{a_i\ not\, in\, S(r)\} + \sum_{\forall a_j \neq a_i} P\{a_j\ in\, S(r)\}$$

By making *n* large enough the first term can be made as small as desired, for example, some number *d*:

$$P_E \le d + \sum_{\forall a_j \neq a_i} P\{a_j\, in\, S(r)\}$$

6. Constructing an Encoding Code Book

How one can make a code book for the encoding of *M* messages, each of *n* bits, will now be examined. Not knowing how to encode, as Dr. Hamming's error correcting codes had not been invented yet, Dr. Shannon chose random encoding. By tossing a penny for each bit of the *n* bits of a message in the code book and repeating for all *M* messages, there would be 2^{nM} possible code books with all books having the same probability of $1/2^{nM}$. The random process of making the code book might mean that there would be duplicates and that there may be code points that are close to each other which will be a probable source of errors. With an *n* large enough, it can be shown that this will not occur with a probability above any positive small level of error we choose.

The decisive step is that Dr. Shannon chose to average over all possible code books to find the average error. The symbol *Av* will now be shown to mean the average over the set of all possible random code books. Averaging over the constant *d* gives this constant and since the average of each term is the same as any other term in the sum:

$$Av(P_E) \le d + (M-1)Av\{P(a_j \, in \, S(r)\}$$

which can be increased as *M*-1 goes to *M*:

$$Av(P_E) \le d + M \sum_{\forall a_j \ne a_i} P\{a_j \, in \, S(r)\}$$

For any particular message, when the average is taken over all code books, the encoding runs through all possible values which leads to the average probability that a point is in the sphere is a ratio of the volume of the sphere to the total volume of the *n*-dimensional space. The volume of a coding sphere can be given by:

$$1 + C(n,1) + C(n,2) + \ldots + C(n,ns), \quad where\ s = Q + e_2 < 1/2\ and\ ns\ is\ an\ integer$$

The largest term in this sum is the last. The estimate of its size can be done via Stirling's formula for factorials (see Stirling's Formula). When looking at the rate of fall off to the next term before it, this rate increases the further left one looks. Dr. Shannon states that you can dominate the sum by a geometric progression with this initial rate, extend the geometric progression from *ns* terms to an infinite number and sum the geometric progression to finally arrive at the bound for very large *n*:

$$1 + C(n,1) + C(n,2) + ... + C(n,ns) \le 2^{nH(s)},\ (s < 1/2)$$

The Taylor series expansion of *H*(*s*) = H(*Q* + *e*2) which gives a bound when the first derivative term is evaluated, and with neglecting all others, to arrive at the final expression:

$$Av(P_E) \le d + 2^{-n(e_1 - e_3)}$$

where:

$$e_3 = e_2 \ln\{(1-Q)/Q\},\ (Q < 1/2)$$

By choosing an *e*2 so that *e*3 < *e*1, the last tem will get as small as desired with sufficiently large *n*. The average error P_E can be made as small as desired now while still being as close to channel capacity *C* as desired.

7. Suitability of Dr. Shannon's Code Books

If the average over all codes has a suitably small error then at least one code must be suitable, so, there exists at least one suitable encoding system. This is the main crux of Dr. Shannon's "noisy coding theorem" only lightly explained here for a simple binary symmetric channel. His proofs extend the theory to even greater generalities where the mathematics get extremely difficult, but here, the concern will be with what machines can understand; that being the binary system of bits represented by ones and zeros.

The size of "sufficiently large *n*" is considered to be astronomical in this sense if one desires to be both close to channel capacity and reasonably sure there will no errors. One would have to wait a very long time to accumulate a message of that many bits before encoding it, let alone contemplating the size of the random code books which can not be represented in a shorter form than the complete listing of all *nM* bits both *n* and *M* being very large.

Dr. Hamming's error correcting codes escape this unachievable task due to the fact that they adopt regular computable methods. In theory, error correcting codes tend to lose the ability to come very near to the channel capacity and still keep an arbitrarily low error rate, but when a large number of errors are corrected by the code they can do well. If a capacity for error correction is established at some level, then for efficiency's sake, the ability for correcting many errors must be used or else capacity is wasted; that ability being a high number of errors corrected in each message sent. Dr. Shannon showed that efficient coding schemes must have very elaborate encodings of very long strings of bits of information. This has been accomplished in space probes that have passed the outer planets as they now correct more and more errors per block as they travel farther from both the Earth and the Sun, which supplies some solar power of about 5 watts; other probes using atomic power sources of roughly the same wattage. High error correcting codes had to be devised in order to be effective given their unavoidably low power transmitting sources and their small transmitting dish sizes. Receiving dishes on earth are also of limited size and the distances that the signals have to travel are ever increasing.

D. RELATED FIELDS AND APPLICATIONS FOR FEC

1. Discussion

Since the invention of Hamming FEC codes in 1947, many fields of information processing now use them extensively, especially the field in telecommunications (Potts 1999). From the television signals we receive through the airwaves or through cable, from compact disc (CD) players and digital audio tapes (DAT), to the way our hard drives on our computers store data; all of these technologies use FEC codes extensively. As mentioned before, space probes and communication satellites use FEC to correct many errors within a message block as their transmitting power is very limited and the receivers on earth can only receive partial signals as the earth rotates, therefore receivers located all throughout the world must share the load in retrieving data from these probes and satellites that have very limited beam focusing capabilities across very long distances (many factors here that can introduce errors).

The use of FEC has allowed network planners and design engineers to maximize efficiency and performance of their various products while minimizing cost. FEC can do this by minimizing the BER of signals by effectively eliminating the bit error ratio floor – a necessary consideration if FEC is not used. BERs for digital communication systems typically operate at a BER of 10^{-9}, that is 1 bit error tolerable in 10^{9} bits transmitted. Simply put, FEC facilitates improved data transmission BER's through noisy mediums such as water, deep space and even our terrestrial atmosphere which can be interfered with by lightening, solar flares and temperature differentials over any particular receiving location. Complete erasures or flipping of bits can occur in these mediums during varying environmental conditions causing distortion to intended messages or data being sent through them.

As mentioned before, Dr. Shannon paved the way for Dr. Hamming to invent the first error-correcting codes. Since that time, error-correcting codes have gone through transformations and many improvements in application and utility. In 1960 Irving S. Reed and Gustave Solomon, staff members at the Massachusetts Institute of Technology (MIT) Lincoln Laboratory, developed the Reed-Solomon code, which has become the most widely used algorithm for error correcting. The Reed-Solomon code is well understood, relatively easy to implement, provides a good tolerance to error bursts and is compatible with binary transmission systems. These codes correct many bit errors within large message blocks (see Reed-Solomon Codes). These codes have been implemented in hardware for many years now; however, there is still utility in software implemented FEC as will be discussed later in this thesis.

E. SUMMARY

The need for computers to be able to not only detect the presence of errors, but also identify and be able to correct those errors led Dr. Hamming to make use of Dr. Shannon's theories on Information and Communication in order to understand how efficient encodings could be implemented. Dr. Hamming was able to use the binary "syndrome" of the error to identify and correct the error with an efficiency of $2k + 1$, that is, correct $2k$ errors and be able to detect $2k + 1$ errors. Channel capacity, in Dr. Shannon's terms, was not fully utilized as error correcting codes use specific computational methods to correct errors. However, if many errors are to be corrected, the efficiency in channel capacity utilization does improve by use of efficient error correcting codes, such as used by planetary space probes..

Reed-Solomon codes contain very efficient algorithms for correcting many errors within a message block with remarkable efficiency even during burst errors. These codes have been implemented in hardware and are used virtually anywhere efficient communication, data transfer and data storage is required. Without the use of these codes, invaluable space probe data would not be recoverable and we would not enjoy the efficiency of our communication, internet multimedia streaming and information exchange systems employed today.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. THEORETICAL DERIVATION OF ERASURE CODES

A. INTRODUCTION

Although FEC coding algorithms have been successfully implemented in hardware, there is still utility in software implementations for FEC as will be discussed. The chapter on derivation of the Hamming code introduced a bit error correction code that deals with the detection and correction of single bit errors as they occur in a message block. Another example of this type of FEC is the Reed-Solomon codes. An introduction of a variation on a FEC code, in the form of an erasure code, that deals with dropped packets (Rizzo 1997) is presented here. Packet losses can occur as a result of many reasons such as network congestion, overloaded router and switch buffers which completely drop packets, noisy environments such as found in an acoustic medium and even asymmetric end user equipment which can experience a variety of packet loss rates (PLR) due to differing processing speeds, memory capacities and type of network connection available.

B. WHY SOFTWARE FEC?

In (Rizzo 1997 and Rizzo and Viscisano 1998), three main objectives are given for software implementation of FEC as a tool to combat the problem of dropped packets. First, putting the software implementation of FEC in the right perspective by showing that it is not too computationally expensive; second, to provide the research community with a high-performance C implementation of erasure codes capable of running at speeds in excess of 100 Megabits per Second (Mbs) and third; to present a number of applications where software FEC can be used to improve performance in wireless, fast wired, unicast and even multicast communication protocols without the need for super computing power requirements. Since the principles of operation in finite fields (as will be discussed later) are not directly supported in most processors (Rizzo 1998), the requirement for the proposed type of erasure code can only be implemented with software and will be shown that performance costs can be kept to a minimum.

In using reliable transport protocols such as Transmission Control Protocol (TCP), unavoidable packet losses are dealt with by the technique of using ARQ which is a retransmission request by a receiver to a server to retransmit the missing or lost packet(s) explicitly or after an appropriate timeout. This is a computationally inexpensive process. However, in the presence of these losses, when the bandwidth-delay product approaches the sender's window, ARQ might result in reduced throughput. Data recovery, using ARQ, requires at least one additional round trip time (RTT) which may cause this technique to be unattractive when high latencies cannot be tolerated such as in real time interactive simulations, fast satellite links or deep space communications. This is especially true in multicast communication protocols when a number of clients (groups) start having a growing rate of uncorrelated losses. Another undesirable feature of ARQ is the requirement of a feedback channel from the receiver that may be an expensive consideration in certain transmission environments.

The author of an open source C based implementation of FEC

(Rizzo 1997) reported that by reducing the time needed to recover missing packets, solved by sending them with redundant information, would allow receivers to reconstruct missing data without the need for retransmission techniques. A feedback channel, in this sense, may not be required since in a multicast environment different loss patterns can be recovered from using the same set of transmitted data.

In the realm of computer communications, error detection is generally provided by the lower protocol layers which use checksums such as Cyclic Redundancy Checksums (CRC) to discard corrupted packets, which would be another source of packet loss (see Cyclic Redundancy Check). Error correcting codes are also used in special cases such as modems, wireless and other noisy links in attempts to make the residual error rate comparable to dedicated wired connections. Next in the stack, above the link layered processes, are the upper protocol layers which have to deal with complete erasures of data caused by missing packets in a stream. These types of erasures are easier to deal with as the exact position of the missing data is known.

Multicast communication applications such as video conferencing or audio can tolerate some packet losses causing some performance degradation, but other applications cannot tolerate these losses such as vital secure military communication links and acoustic telemetry from AUVs. These have a much stricter requirement for reliable delivery of all data. Reliability in the transmission of data in the latter would be improved greatly with the techniques of redundant encodings before transmission. Especially in the low powered and limited network capacities of at-sea networks dealing with AUV's, a one time short burst of transmitted telemetry is highly desired. Erasure codes that can operate on packet sized data objects implemented in software using general purpose processors would be a feasible option to handle packet losses in the acoustic medium without elaborate and expensive equipment and with tolerable computational costs.

C. ERASURE CODING THEORY

In (Lin, Costello and Miller 1983) there is an abundance of information in the principle operations of erasure codes. The following discussion will deal with linear block codes as they will be appropriate for this thesis' proposed application and are relatively easy to understand.

1. Encoding and Decoding Process

The key behind erasure codes is that *k* blocks of source data are encoded at the sender to produce *n* blocks of encoded data such that <u>any</u> subset of *k* encoded blocks (and their identity as will be explained later) will be sufficient to reproduce the exact source data. The following figure will graphically represent this. This type of code is called an (*n*, *k*) code which will allow a receiver to recover up to *n* - *k* losses.

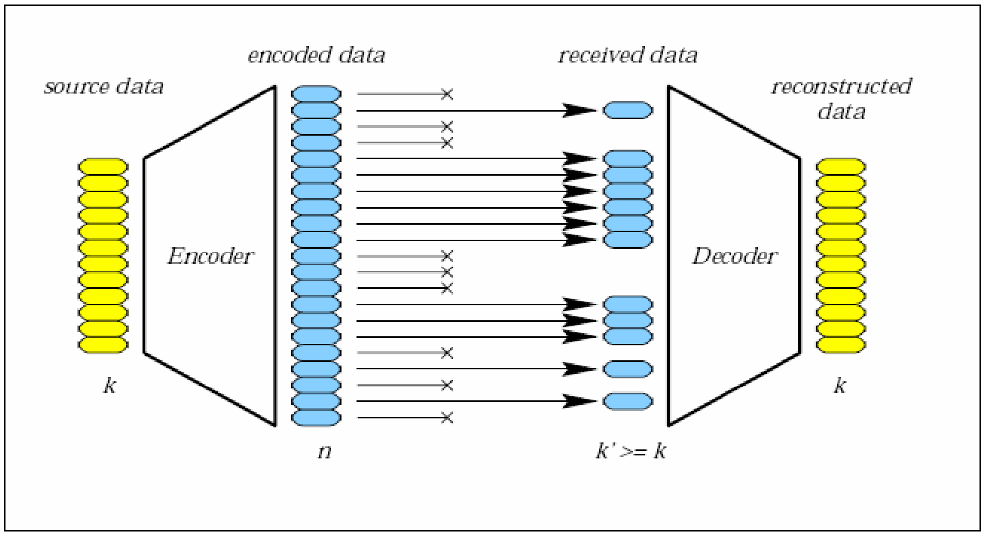


Figure 10. A Graphical Representation of an Encoding/Decoding Process where x's Represent Packet Losses (From Rizzo 1997)

In the telecommunications world a block of data is usually made up of small bits of information. In computer communications the quantum of information is much larger in that a packet of data can contain up to thousands of bits. Large blocks of data may change the way an erasure code can be implemented. The way to deal with encoding large blocks or files is to specify the parameters of (*n*, *k*) and the packet sizes of the data to be encoded. This way, large files can be split or broken up into multiple data items and the encoding/decoding (EN/DE) process can be applied one packet at a time.

The interest lies in a type of erasure code and that is of the linear code class. This is so called because they can be analyzed using properties of linear algebra. Let:

$$\mathbf{x} = x_0 \ldots x_{k-1}$$

be the source data, ***G*** an *n* x *k* matrix, called the encoding matrix, then an (*n*, *k*) linear code can be represented by:

$$\mathbf{y} = \mathbf{Gx}$$

Assuming that *k* components of ***y*** are available at the receiver, source data can be reconstructed by using the *k* equations corresponding to the known components of ***y*** by making ***G'*** a *k* x *k* matrix representing these equations. This can only be possible if these equations are linearly independent which will be the case, generally, if any *k* x *k* matrix extracted from ***G'*** is invertible.

If the encoded blocks include a verbatim copy of the source blocks, the code is called a systematic code. This corresponds to including the identity matrix $\boldsymbol{I}_k$ in ***G***. The advantage of a systematic code is that it will simplify reconstruction of source data if few losses are expected. This is so because the first *k* of the (*n*, *k*) code will be an exact replica of the original source. Again, *n* – *k* will be encoded so that any *k* subset of the *n* encoded packets will reconstruct the data should there be losses. If all *k* original packet replicas were received intact, very few cycles will be spent in reconstruction as the data is already in its source originality. The following figure gives better graphical representation to this concept.

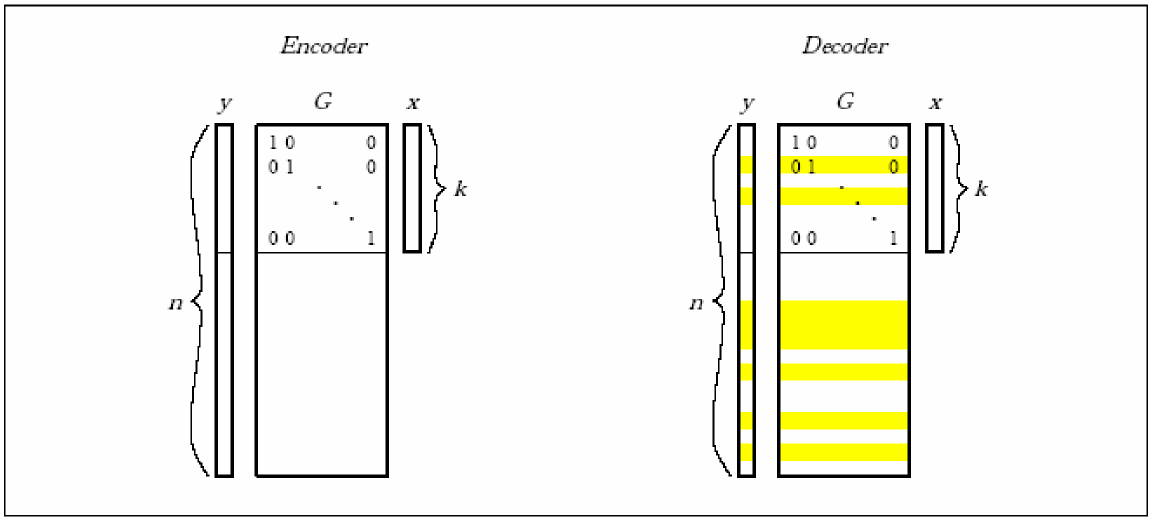


Figure 11. The (EN/DE) Process in Matrix Form for a Systematic Code (the top *k* rows of ***G*** constitute the identity matrix $\boldsymbol{I}_k$). ***y'*** and ***G'*** Correspond to the Grey Areas of the Vector and Matrix on the Right (From Rizzo 1997).

2. The Generator Matrix

G is called the generator matrix of the code because any valid ***y*** is a linear combination of columns of ***G***. With ***G*** being an n x k matrix with rank k, any subset of k encoded blocks will convey information on all of the k source blocks. As a property of this code, each column of ***G*** can have at most k – 1 zero elements. In the case of a systematic code ***G*** will contain the identity matrix $\boldsymbol{I}_k$ which will contain all zero elements. The remaining rows of the matrix must now contain all non-zero elements.

During the reconstruction process, the index of each n packet must be known. These indices are generated during the encoding process and can be transmitted and retrieved ahead of the actual reception of the n encoded packets. The cost in terms of this operation is relatively negligible (compared to the encoded packets) as this will be just single integer information for each index.

There is a cost to be considered for the source encoding operation and that is the precision used for computations. If each x_i is represented using b bits, representing the y_i's require more bits if ordinary arithmetic is used. If each coefficient g_{ij} of ***G*** is represented with b' bits, then the y_i's need:

$$b + b' + (\log_2 k)$$

bits to be represented with out loss of precision. This poses a significant cost in cycles per bit processing as these excess bits must also be transmitted to reconstruct the data exactly. Rounding or truncating the representations of the y_i's would prevent exact reconstruction of data which would lead to disastrous results.

3. Computations in Finite Fields

The potential expansion of the above mentioned data can be dealt with by working in a finite field. In this type of field basic arithmetic can be performed on data much like it is with integers. In (Blahut 1985) the mathematics of finite fields are covered very thoroughly. A field is closed under addition and multiplication meaning that a result of sums and products of field elements are still field elements. A finite field has the property of having a finite number of elements. Most properties of linear algebra will apply to these finite fields.

A desirable advantage of using a finite field lies in the closure property which allows for the ability to calculate precise field elements without requiring extra bits to represent the results as was previously mentioned. In working with finite fields mapping data elements into field elements, operating upon them according to the rules of the field and then applying the inverse mapping to the elements allows for reconstruction of desired results.

a. Prime Fields

Finite fields have been shown to exist with $q = p^r$ elements where p is a prime number. Fields with p elements, with p prime, are called prime fields or $GF(p)$, where GF stands for Galois (pronounced as "gal-wah") Field. Operating in a prime field is relatively simple since $GF(p)$ is the set of integers from 0 to $p - 1$ under the operations of addition and multiplication modulo p. From the point of view of a software implementation, there are two minor difficulties in using a prime field: first, with an exception of $p = 2$, field elements require:

$$\lceil log_2 p \rceil > \log_2 p$$

bits to be represented. This will cause a slight inefficiency in the encoding process and there will be an even larger possibility of inefficiency in operating on these numbers since the operand sizes might not match the word size of a particular processor. The second problem lies in the need of a modulo operation on sums and multiplications. Most of the modulo operations will be on the latter. A modulo calculation is an expensive operation since it requires a division. Both problems are overcome by setting $p = 2^m + 1$.

b. Extension Fields

Fields with $q = p^r$, with p prime and $r > 1$, are called extension fields or $GF(p^r)$. The sum and product in extension fields are not done by taking results modulo q, but rather field elements can be considered as polynomials of degree $r - 1$ with coefficients in $GF(p)$. The sum operation is just the sum between coefficients modulo p; the product is the product between polynomials, computed modulo an irreducible polynomial such as one without divisors in $GF(p^r)$ of degree r and with coefficients reduced modulo p.

Although the apparent complexity presented here, the operations on extension fields can become simple in the case of $p = 2$. With this particular case, elements of $GF(2^r)$ require exactly r bits to be represented. The reader will note that this will greatly simplify the handing of data encoding. Sums and subtractions will become the same operation (bit by bit sum modulo 2) which can simply be executed with an exclusive OR (XOR).

c. Multiplications and Divisions

A noteworthy property of prime or extension fields is that there exists at least one special element, usually denoted by α, whose powers generate all non-zero elements of the field. As an example a generator for $GF(5)$ is 2, whose powers starting from 2^0 are 1, 2, 4, 3, 1, …. Powers of α repeat with a period of length $q - 1$, so, $\alpha^{q-1} = \alpha^0 = 1$. In (Wagner, 2001) there is an excellent Java implementation of how these generators can be determined. This is a variation of Fermat's Theorem which states that:

$$a^{p-1} \bmod p = 1$$

where a is any non-zero number less than p and p is prime.

This property has a direct consequence on the implementation of multiplication and division in that one can express any non-zero field element x

as $x = a^k$. Now k_x can be considered as a logarithm of x and multiplication and division can be computed using logarithms such as:

$$xy = \alpha^{|k_x + k_y|_{q-1}}, \quad \frac{1}{x} = \alpha^{q-1-k_x}$$

where $|a|_b$ stands for "*a* modulo *b*". If the number of field elements is not too large then tables can be built off line to provide the logarithms, exponentials and the multiplicative inverses of each non-zero field element. In some cases it can be convenient to provide a table for multiplications as well. Using the above techniques operations in extension fields with $p = 2$ can be very fast and easy to implement.

4. Data Recovery

Recovery of the original data is possible by solving the linear system:

$$\mathbf{y}' = \mathbf{G}'\mathbf{x} \rightarrow \mathbf{x} = \mathbf{G}'^{-1}\mathbf{y}'$$

where ***x*** is the source data and ***y'*** is a subset of *k* components of ***y*** available at the receiver. Matrix ***G'*** is the subset of rows from ***G*** corresponding to the components of ***y'***.

The problem can be solved in two steps: first ***G'*** is inverted, then $\boldsymbol{x} = \boldsymbol{G'}^{-1}\boldsymbol{y'}$ is computed. This is because the cost of matrix inversion can be amortized over all the elements which are contained in a packet and this can be virtually negligible in most cases. The inversion of ***G'*** can be accomplished by replacing division with multiplication by the inverse field element. The cost of the inversion is reported in (Rizzo 1997) as $O(kl^2)$ where $l \leq \min(k, n - k)$ which is the number of data blocks that must be recovered. Reconstructing the *l* missing data blocks has a total cost of $O(lk)$ operations.

D. AN ERASURE CODE BASED ON VANDERMONDE MATRICES

An effective way to build the generator matrix, ***G,*** will now be discussed. Building this matrix will include coefficients of the form:

$$g_{ij} = x_i^{j-1}$$

where the x_i's are the elements of $GF(p^r)$. Such matrices are commonly known as Vandermonde matrices and their determinant is:

$$\prod_{i,j=1\ldots k, i<k} (x_j - x_i)$$

If all x_i's are different then the matrix has a non-null determinant and it is invertible. Provided $q > k$ and all $x_i \neq 0$, up to $q - 1$ rows can be constructed which satisfy the properties required for ***G***. Such matrices can be extended with the identity matrix $\boldsymbol{I_k}$ to obtain a suitable generator for a systematic code. A good discussion of the polynomial properties of Vandermonde matrices are given in (Lamparter and others 1993).

It should be noted the there will be some special cases of this code which are of trivial implementation. An example would be a (n, 1) code which only requires the same data to be retransmitted several times. This type of code represents a degree-0 polynomial (a constant) which will have the same value at all points, so, ***G*** becomes an n x 1 matrix with unity coefficients. This would be of no cost in overhead. Another example is that of a systematic (k + 1, k) code where the only redundant block is the sum [as defined in $GF(p^r)$] of the k source data blocks. This code would be represented by a degree k – 1 polynomial evaluated at p = 1 making ***G*** a 1 x k matrix with unity coefficients. The sum of all coefficients amount to a simple XOR in the case of p = 2. As a consequence, a (k + 1, k) code is only useful for a small amount of expected losses. In other cases, such as will be presented in this thesis, there will be a need for codes with k > 1 and n – k > 1.

Dr. Rizzo's C implemented FEC code (Rizzo 1997) was written for peak performance within network protocols. This code will support p = 2, any r in the range of 2 … 16 and arbitrary packet sizes. Reported maximum efficiency can be achieved with $r = 8$ since this will allow most operations to be executed using memory manageable lookups for logarithmic and exponential tables and for XOR's. The generator matrix has the form indicated above with $x_i = \alpha^{i-1}$. Up to $2^n - 1$ rows can be built this way which makes it possible to construct codes up to $n = 2(2^r - 1)$, $k = 2^r - 1$. Packet sizes of 1024 bytes were used very efficiently, which is a recommend parameter when specifying this type of erasure FEC encoding. Observing this recommendation will comply with a typical Ethernet Maximum Transmission Unit (MTU) restriction of 1500 bytes.

Dr. Rizzo warns that using erasure codes in a unicast protocol is computationally expensive due to the lack of native support in most processors for work in finite fields, but can be minimized by choosing good values for the parameters *(n, k)* and the size of a data packet to hold encoded data. These choices are available to the implementer of this particular erasure code. In a multicast environment, the benefit will be that erasure coding techniques will remove the effect of independent losses at different receivers. Reconstruction of the original data is possible using the identity of the received packets, the identity being at what index was that packet encoded. Scalability is improved as loss patterns at each receiver are not a main concern as each individual receiver's loss pattern can be dealt with effectively.

E. SUMMARY

Erasure codes based on Vandermonde matrices interprets k source data symbols as the coefficients of a polynomial P of degree $k - 1$. As the polynomial is fully characterized by its values in k different points the desired amount of redundancy can be produced by evaluating P at n different points. Reconstruction of the original data (the coefficients of P) is possible as soon as k of these values are available. In practice, the encoding process requires multiplying the original data by an n x k encoding matrix $\boldsymbol{G}$ which is a Vandermonde matrix. The decoding process requires the inversion of a k x k sub matrix of $\boldsymbol{G'}$ taken from $\boldsymbol{G}$ and the multiplication of the received data by $\boldsymbol{G'^{-1}}$. By simple algebraic manipulation, ***G*** can be transformed to make its top *k* rows constitute the identity matrix making this code a systematic code.

V. SUITABILITY OF COMBINING FEC ENCODING WITH XSBC

A. INTRODUCTION

The previous chapter introduced an open sourced C based implementation of a Vandermonde matrix based FEC code. An open sourced Java implementation of the same type of Vandermonde matrix based FEC code, written by Justin F. Chapweske, founder of Onion Networks, Inc., (see Onion Networks Inc.) based on Dr. Rizzo's code, was selected for this thesis' application for reliable acoustic telemetry transfer in conjunction with XSBC. First, an introduction into XSBC will be given to give the reader a background into this remarkable compression algorithm. Next covered will be the Java based implementation of the erasure code and will conclude with the integration of the two separate libraries and how they interoperate within the AUVW to ensure reliable data and telemetry transfer to and from AUVs.

B. XML SCHEMA BASED BINARY COMPRESSION (XSBC)

The background into XSBC begins in (Serin 2003) with what was formally titled Cross-Format Schema Protocol (XSFP) and Dynamic Behavior Protocol (DBP) before it was later redefined as XSBC (see Extensible Modeling and Simulations (XSMF) projects page). XSFP was originally designed as a further enhancement of the Institute for Electrical and Electronic Engineers (IEEE) Distributive Interactive Simulation (DIS) effort. DIS was designed as a way to exchange state information between entities within a real time distributed interactive simulation.

In order to create this flexible and run-time extensible application layer protocol, XML, due to its terseness, was chosen for describing data structures and as a direct result, an XML schema describing document parameters such as legal elements, attributes and data structures (types) can be used to describe the protocol syntax (Serin 2003). Elements and attributes are replaced via a tokenization scheme which carefully preserves valid XML document structure.

Initial testing showed that XML-izing a document can inflate its size beyond the MTU requirements of some networks (typically 1500 bytes) due its verbose nature in describing data types. Such increases in file size can consume considerable network capacity when transmitting data structures in plain XML text form. The W3C Workshop on Binary XML Interchange (see XML Binary Interchange Workshop Report) is currently working to reduce the overhead in parsing of XML structured text streams. XSBC is offered as a royalty free (RF) exemplar algorithm solution to this cost in transmission overhead.

1. XML Document Serialization

The first principal in the use of XSBC for reliable compression of XML text streams or documents is that the stream or document must be defined by a schema. This is so because during the serialization process a schema parser is employed to create entries in a table for each element and attribute defined in that schema. A replacement algorithm for all element and attribute names of the XML document is then invoked that replaces these names with short tag numbers which constitute the serialization process. It should be noted that data is left intact, but there will still be considerable savings in the serialized XML document over the original uncompressed form file size. Results for showing improved compression over GNU's zip utility (GZIP) and WinZip alone are shown in the following figures with file extension types defined below:

- `x3d`: Extensible 3D (X3D) File Format
- `wrl`: Virtual Reality Modeling Language (VRML) 97 File Format
- `b3d`: Binary X3D File Format
- `zip_x3d`: X3D File Compressed by WinZip Program
- `zip_wrl`: VRML97 File Format Compressed by WinZip Program
- `b3z`: X3D File Compressed by Serializer using GZIP Streams

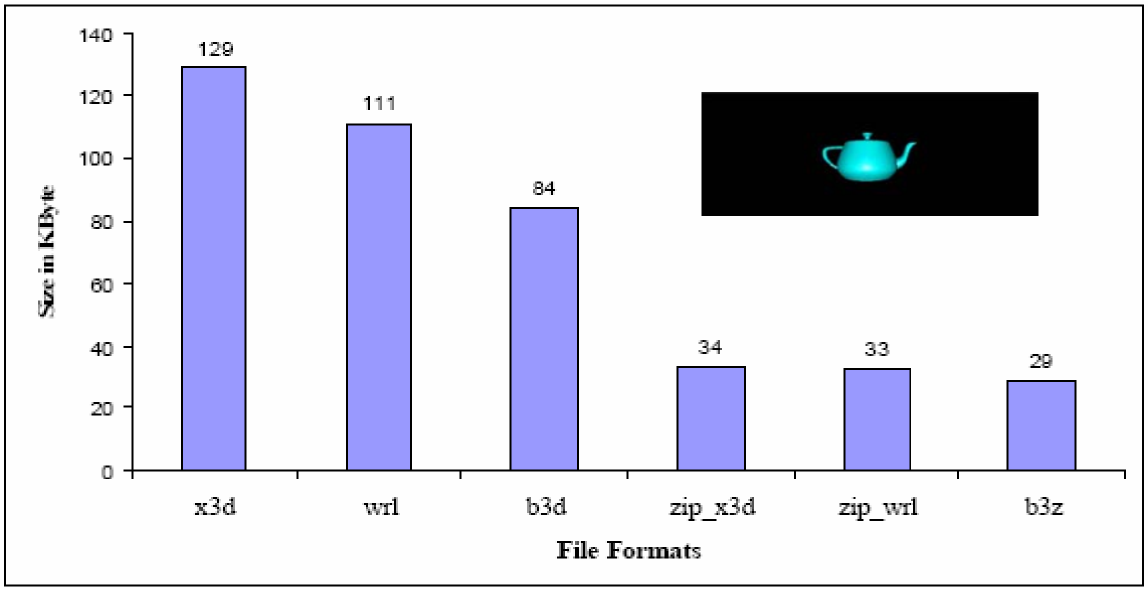
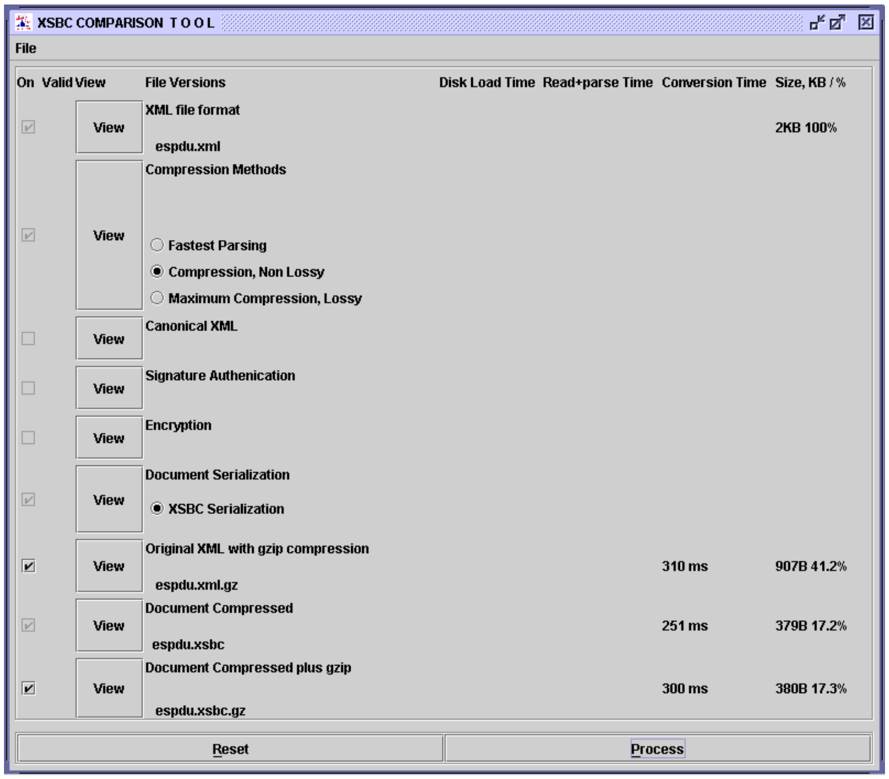


Figure 12. File Format Compression Comparisons for the Teapot Exemplar (After Serin 2003)



On	Valid	View	File Versions	Disk Load Time	Read+parse Time	Conversion Time	Size, KB / %
✓		View	XML file format espdu.xml				2KB 100%
✓		View	Compression Methods: ○ Fastest Parsing [● Compression, Non Lossy] ○ Maximum Compression, Lossy				
☐		View	Canonical XML				
☐		View	Signature Authenication				
☐		View	Encryption				
✓		View	Document Serialization [● XSBC Serialization]				
✓		View	Original XML with gzip compression espdu.xml.gz			310 ms	907B 41.2%
✓		View	Document Compressed espdu.xsbc			251 ms	379B 17.2%
✓		View	Document Compressed plus gzip espdu.xsbc.gz			300 ms	380B 17.3%

Figure 13. Comparison of Filesizes for GZIP Alone, XSBC or combined XSBC/GZip for a Schema Defined XML Document (Consult Appendix C)

Due to a protocol's agreement between user applications via the same schema, whole names are not required to be sent. This is an advantage over DIS in that DIS requires the syntax of the protocol to be static and all of the data must be in their previously specified places so that no subset or selected section of the whole protocol could be transmitted. Therefore, by serializing XML documents, with the advantage of the XSBC attribute and element name replacement algorithms, they can be transmitted in a much more compact way. The two following figures give a better graphical representation as to what happens to the XML document during the serialization process.

```
<?xml version="1.0" encoding="UTF-8"?>
<protocol xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="example.xsd">
        <location x="3.45" y="56.72" z="-10.1"/>
        <header>
                <exerciseID>1</exerciseID>
                <version>1</version>
                <pduType>2</pduType>
        </header>
        <velocity x="1.0" y="0.0" z="-0.7"/>
</protocol>
```

Figure 14. Example XML Document Before the XSBC Serialization Process with Element and Attribute Tagsets Highlighted for Clarity (After Serin 2003)

```
<?xml version="1.0" encoding="UTF-8"?>
<10>
        <14 24="3.45" 25="56.72" 26="-10.1" 15>
        <12>
                <20>1<21>
                <18>1<19>
                <22>2<23>
        <13>
        <16 27="1.0" 28="0.0" 29="-0.7" 17>
<11>
```

Figure 15. Example XML Document Results After the XSBC Serialization Process (After Serin 2003)

2. XML Document De-serialization

The de-serialization process can now be described as the process of recreating the XML tree back from the received binary stream. By using the same schema needed to create a table for the serialization process, the de-serializer uses that schema to create another table to match the tokenized or serialized element and attribute tags with their proper full names and will match any data that belong within those names. To accomplish this the de-serialization process uses stack operations to rebuild the XML tree into its original fully verbose form. For a more detailed description of both the serialization and de-serialization processes the reader is encouraged to consult (Serin 2003).

3. XSBC Instead of Other Types of Binary Compression

XML serialization provides an alternative and compact way to send and receive XML documents over the network. Currently, most XML documents use Unicode Transformation Format – 8 (UTF-8) encoding which corresponds to 8-bit American Standard Code for Information Interchange (ASCII) encoding. With this type of encoding each alphabetical character and number is represented by eight bits. Instead of using 8 bits for each character, the notion agreement is exploited so that element and attribute names are replaced by binary short tags (Serin 2003). The result of this is that data, marked-up by elements and attributes, are serialized to binary form resulting in compact XML.

During the XSBC process XML documents are compressed without using any other types of conventional binary compression algorithms. A list of driving factors of avoiding binary compression algorithms altogether include simplicity of implementation, computational speed performance improvements and ability to bypass bitwise operations.

4. XSBC as a Module within the AUVW

The Scenario Authoring & Visualization for Advanced Graphical Environments (SAVAGE) laboratory of the MOVES Institute located at NPS has a number of ongoing projects dedicated to the improvement of training and war-fighting skill maintenance for military members across all DOD service components and their allies. One such project involves a distributable application to facilitate AUV mission planning and analysis (Lee 2004).

The AUVW is comprised of a set Java-based open source libraries. XSBC is one those libraries used for the compression of telemetry and data to and from since XML is used for data storage and exchange between an AUV and the AUVW Server. One of the most important features of the AUVW is the definition and use of a common AUV mission control script (Hawkins and Van Leuvan 2004). Further development and improvement of this script is currently being conducted by CDR Duane T. Davis, USN, a Ph.D. candidate with the Computer Science Department at NPS. The importance of this common script is that dissimilar AUVs will be able to be controlled and employed through this common language which is XML based and defined by the Autonomous Vehicle Command and Control Language (AVC^{2}L) schema. Further information on AVC^{2}L can be obtained from (see NPS AUV Workbench Flyer).

a. Telemetry from an AUV to the AUVW Server

In the design of the AUVW there was consideration for the need for exchange of information. This is done via a scaled down, yet fully functional multi-threaded web server module within the workbench. Through this server, AUVs are able to uplink information such as current position, list of obstacles encountered, mission findings and such that are publishable and easily accessible by operators and planners of AUV missions (Lee 2004). Since data sent from an AUV to the AUVW server is structured in XML according to the AVC^{2}L schema, this data is a prime candidate for compression by XSBC.

Mission data is compressed in two ways within the AUVW. First, as an on-the-fly routine implemented before data uplink and second, as a manual function selected by planner/operator from one of the workbench's many GUI features. The following figures represent a server/client User Datagram Protocol (UDP) implementation of this process.

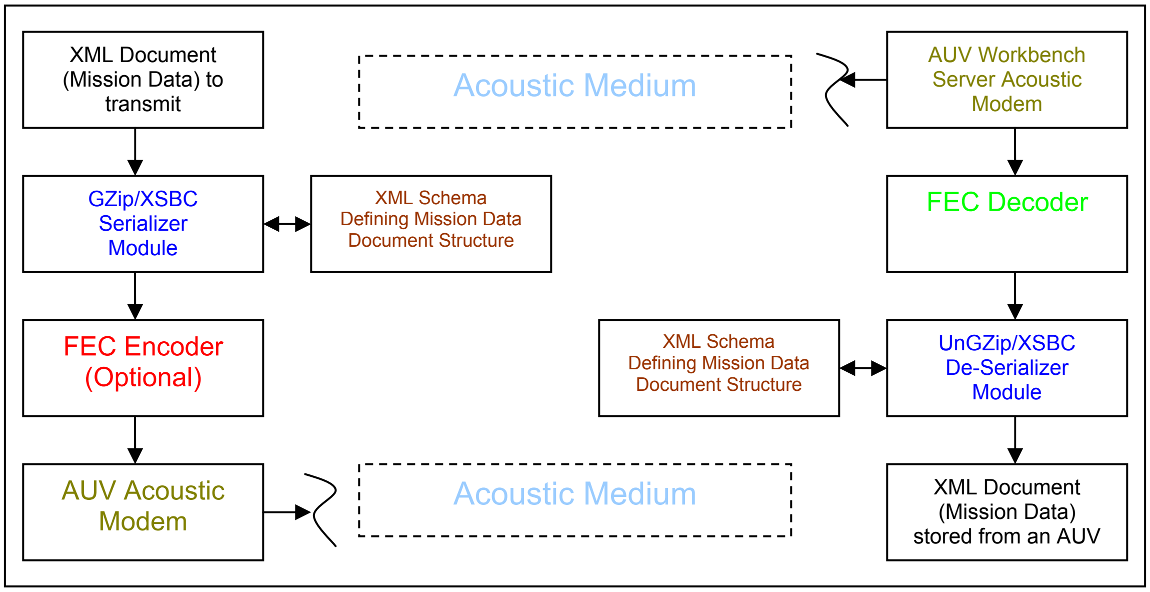
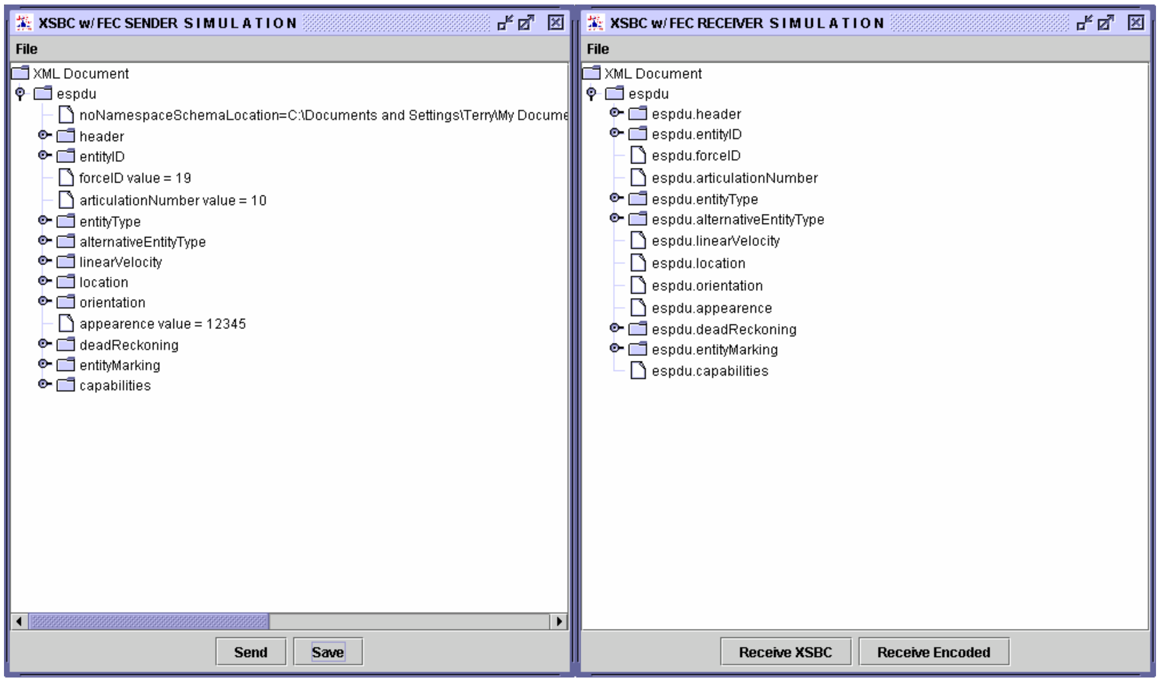

XML Document (Mission Data) to transmit
Acoustic Medium
AUV Workbench Server Acoustic Modem
GZip/XSBC Serializer Module
XML Schema Defining Mission Data Document Structure
FEC Decoder
FEC Encoder (Optional)
XML Schema Defining Mission Data Document Structure
UnGZip/XSBC De-Serializer Module
AUV Acoustic Modem
Acoustic Medium
XML Document (Mission Data) stored from an AUV

Figure 16. Data Flow Diagram for XSBC/FEC Data through an Acoustic Medium (From AUV to AUVW Server)


XSBC w/ FEC SENDER SIMULATION
File
Send
Save
XSBC w/ FEC RECEIVER SIMULATION
File
Receive XSBC
Receive Encoded

```
XML Document
espdu
  noNamespaceSchemaLocation=C:\Documents and Settings\Terry\My Docume
  header
  entityID
  forceID value = 19
  articulationNumber value = 10
  entityType
  alternativeEntityType
  linearVelocity
  location
  orientation
  appearence value = 12345
  deadReckoning
  entityMarking
  capabilities
```

```
XML Document
espdu
  espdu.header
  espdu.entityID
  espdu.forceID
  espdu.articulationNumber
  espdu.entityType
  espdu.alternativeEntityType
  espdu.linearVelocity
  espdu.location
  espdu.orientation
  espdu.appearence
  espdu.deadReckoning
  espdu.entityMarking
  espdu.capabilities
```

Figure 17. Server/Client Simulation of an XML Document Compressed, FEC Encoded, Transmitted, Received, FEC Decoded, Uncompressed and Displayed in JTree Form. Example available within the XSBC 0.91.1 Library.

C. INTRODUCTION OF A JAVA BASED FEC ENCODING LIBRARY

As was mentioned previously an open sourced implementation of a Java based FEC code, based on Dr. Rizzo's C implementation of his Vandermonde based erasures codes, was selected for integration with XSBC within the AUVW's application library. This Java based approach was logical for the fact that Java is cross platform independent as long as that platform recognizes and operates with a Java Virtual Machine (JVM).

An interesting issue to note about the FEC 1.0.3 library from Onion Networks, Inc. is that the designer can implement the C based implementation by including the fec-win32.jar file (available within the FEC 1.0.3 library download) in a platform's JVM classpath. If this option is not desired, as in the case of this thesis work, simply omit this file reference from the classpath which will force a pure Java implementation of the code. The release notes (Chapweske 2000) for FEC 1.0.3 state that the C based implementation is better optimized for speed, but the Java forced implementation is only a few milliseconds behind the C implementation.

The Java based implementation has the same principles of operation as presented in Chapter IV of this thesis and will not be restated here. For further familiarization of the Java FEC 1.0.3 library, and a hyperlink to the source code, the reader is encouraged to consult Appendix B.

D. THE FUTURE OF XSBC

Not only is XSBC used within the AUVW, but it has several other application possibilities. XSBC is a library designed to compress XML documents and messages. It is designed to support both large documents like X3D Graphics and Scalar Vector Graphics (SVG) files as well as short messages such as Simple Object Access Protocol (SOAP) and XML Remote Procedure Calls (XML-RPC). A major feature of this library is the ability to register compressors for an attribute type, an element or document fragment. This allows data-aware compressor algorithms to get much better compression then typical generic routines.

As of the writing of this thesis, XSBC is still under development, however; a working version is available for download and experimentation. XSBC is an open sourced library licensed under the GNU LGPL v2.1 from the Extensible Modeling and Simulation Framework (XMSF) source page at http://sourceforge.net/projects/xmsf. For familiarity in working with XSBC the reader is encouraged to consult Appendix C.

Planned, continued and future work for XSBC technology include:

- Implementation of X3D Compressed Binary Encoding
- Implementation of the International Standards Organization (ISO) Fast Infoset Encoding
- Tracking developments, best practices and eventual W3C recommendation efforts of the XML Binary Characterization Working Group
- Addition of bitwise FEC support
- Provide support for XML Tactical Chat (XTC) and Jabber Chat clients (Brutzman and others 2004)
- Provide support for binary-XML Web Services

E. SUMMARY

After the efficient compression techniques of XSBC are applied to an XML document or message, reliability in transmitting that data can be realized by applying an FEC erasure code to that compressed data. The coding will contain redundant information enabling a receiver to reconstruct that data without retransmission from the sender. In the realm of AUVs and UUVs, this is a prudent technique to apply to preserve the power supplies of their data transmitters that would before employ “try 'til you die” transmission routines. The costs, in terms of computations, are practically negligible compared to the cost in power consumption by having to respond to repeated ARQs from a receiver that received corrupted data or that experienced significant packet losses due to the noisy and lossy characteristics of shallow-water sound propagation.

VI. FEC FILTER COMBINED WITH XSBC

A. INTRODUCTION

This chapter covers how the Java-based FEC codes were integrated with XSBC within the AUVW. For on-the-fly compression with the XSBC utility, the AUVW uses two files which are XsbcSerializer.java and XsbcTransaction.java. The serializer is called when telemetry or other mission data from an AUV is ready to be transmitted uplink. Before this is done, the data, in XML text form, must be compressed, for network capacity considerations, followed by FEC encoding for redundant reliability. Once the compressed/encoded file is received by the AUVW Server, the data is collected in a temporary buffer, decoded (if necessary), written to file form with *.xsbc.gz extension, decompressed and un-serialized resulting in an XML mission results data file for viewing or storing in a local server cache folder. Again, if the file was GZipped and/or FEC encoded (both are options within the AUVW) during the serialization process, the transaction file will decode the incoming GZip stream, uncompress the GZip steam, deserialize the .xsbc file and then create a missionResults(ID#).xml file for storage, later retrieval and evaluation.

1. XSBC Serializer Functionality within the AUVW

The original network functionality was built around a TCP connection. Since, as discussed previously, TCP, with its reliable connectivity issues, may not be the most desirable form of data transmission for this application. An additional constructor was added to the XsbcSerializer.java file to add the desired UDP functionality. Even though UDP only provides “best effort” packet deliveries, this is where the benefit of FEC encoding of XSBC files comes in. If packets are dropped due to the inherent noisy links experienced in an open water acoustic environment, redundant encoding will provide reliable data recovery of potentially lost packets without the need for a feedback channel or without regard to which specific packet were lost or dropped.

a. UDP Functionality

The following will be code snippets from the XsbcSerializer.java file that give the FEC encoding capability option to the workbench planner. XsbcSerializer.java was first developed with TCP/IP functionality originally. UDP functionality was built in as an option to the AUVW Server so as to realize faster data transfer without the need for a feedback channel. Selecting the FEC Server will optimize reliability of UDP transmitted data. Only the major features are covered that may not intuitive to the reader:

```
/**
 * Creates a new instance of XsbcSerializer for a UDP option
 *
 * @param inFilePath (including filename) of input XML file
 * @param udp true if a UDP option is invoked
 * @param encode true if the file is to be FEC encoded, false if not
 * @param p the FECParameters to utilize for FEC file encoding
 * @param host the host address to connect to
 * @param port the port number to bind to
 */
public XsbcSerializer( String inFilePath, boolean udp, boolean
                       encode, FECParameters p, String host, int port ) {

   result = 0;
   udpOption = udp; // tdn
   this.encode = encode;
   params = p;
   setHost( host );
   setPort( port );

   // Check for FEC encoding option
   if ( !encode ) {

      // Serialize without FEC Encoding
      setDocumentWriter( inFilePath ); // tdn

   } else {

      repairBuffer = new Buffer[ p.getN() ];
      stripeOrderArray = new int[ p.getN() ];
      System.out.println( "From Serializer constructor: " + p ); // DEBUG

      doFECEncoding( inFilePath );

   } // end if-else block

} // end UDP option constructor
```

The first thing to note is the FECParameters argument in the constructor. This is where specification of particular parameters to perform encodings are implemented. Mission data or telemetry .xml files are generally

3.5 MBs or more in size. Still, values of k and n can be kept at or under $k = 32$ and $n = 256$ and still provide quick and reliable encodings within this software implemented FEC capability. Using these recommended FECParameter values will force the use of $GF(2^8)$ which allows for much greater (EN/DE) speeds on the fly as all matrix algebraic tables can be generated and kept in memory with a minimal memory footprint observed. The upper limit for n in $GF(2^8)$ is $2^8 = 256$. The speed advantage comes from the fact that the logarithm tables for $GF(2^8)$ are $2^8 \times 2^8 \times 8$ bits = 64 KBs of memory. The code will use a $n \times k$ matrix to perform all encoding calculations which will take $O(nk)$ cycles to complete. Recall that if many losses are expected, specifying a higher k value would be prudent. If lower packet losses are expected, a lower k value of < 16 can be selected. The same reasoning goes for selecting n ($n > k$ must be observed).

The repairBuffer and stripeOrderN data members are key in that they will contain the FEC encoded repair packets and the index number with which those encoded repair packets were created respectively. This is necessary as the proper index information is needed to properly conduct the necessary matrix operations during the decoding process. The software will determine any index < k and systematically reorder the decoding matrix so as to place all repair packets of index < k in the identity matrix. This will save in decoding flops as these < k index repair packets are in their original source symbol form.

The repairBuffer is a byte[] wrapper that holds repair packet encoded data for the entire block size(s) generated by the FEC encoding process. Once an *.xsbc.gz file is ready to be encoded, if the FEC encoding option is selected, that file is automatically broken up according to the FECParameters selected. A file is broken up into separate blocks for encoding by the following convention:

$$Blockcount = filesize/(k * packetsize)$$

where filesize is the size of the .xsbc and/or .xsbc.gz file to be encoded and packetsize is another FECParameter we select. Typical packet sizes of 1024 bytes will work well to maintain within MTU restrictions. The FEC class that performs these automatic file handing capabilities is the FECFile class. The repairBuffer will hold all *n* repair packets each of 1024 bytes in size generated for each block. Each block, if a file is broken up into more than one, will each contain *n* repair packets where index offsets are spaced 1024 bytes apart.

The constructor then calls doFECEncoding() giving the string location of the file to be encoded as an argument:

```
/**
 * Performs the FEC Encoding of an already saved to disc .xsbc file before
 * sending out in a stream
 *
 * @param fileName the .xsbc file to encode
 *
 * @exception IOException if FECFile did not form properly and/or
 * the encoding process failed
 */
private void doFECEncoding( String fileName ) {

   newFile = new File( fileName );

   System.out.println( "File to encode with FEC: " + newFile );

   // Create the FEC file out of the .xsbc file in read only mode with the
   // default parameters K=16, N=32 and repair packet size=1024 B
   try {

      fecF = new FECFile( newFile, "r", params );

   } catch ( IOException ioe ) {

      ioe.printStackTrace();

   } // end try-catch block

   // Create an array of N numbered indices
   for ( int i = 0; i < stripeOrderArray.length; i++ ) {

      stripeOrderArray[ i ] = i;

   } // end for

   // Create an FEC Buffer[] as a wrapper holding N byte[] 1024 B in size.
   // These will be the repair packets of which we only need any K subset
   // per block on the receiving end to decode the orig. file (not to be
   // confused with Datagram Packets) (tdn)
   for ( int i = 0 ;  i < params.getN(); i++ ) {

      repairBuffer[ i ]  = new Buffer( params.getPacketSize() );

   } // end for
```

```
      // Perform a Fisher-Yates shuffle of the stripeOrder[] numbered indices.
      // Repair packets (N of them for each block, last block may contain less)
      // will be encoded according to the stripe ordering produced by this
      // shuffle.
      Util.shuffle( stripeOrderArray );

      for ( int i = 0; i < params.getBlockCount(); i++ ) {

         try {

            // Encode packets in each file partitioned block (i) as repair
            // packets according to the shuffled order of their indices
            // (FEC code)
            fecF.read( repairBuffer, i, stripeOrderArray );

         } catch ( IOException ioe ) {

            ioe.printStackTrace();

         } // end try-catch block

         // Send each block (index information along with each N repair packet)
         // along it's way.  Once decoded by the receiver, the receiver will
         // determine GZipped status and handle accordingly via UDP only.
         sendFECEncoded();

         System.out.println( "Sending block number: " + i );

      } // end for

   } // end doFECEncoding()
```

The fecF data member is an instantiation of FECFile. This method will take in a java File argument to create our encoded repair packets, a parameter argument to make this file a read only file “r” and, finally, the FECParameters object we specify. Before the file begins the encoding process, the file is broken up into blocks if necessary by the above given equation. The FECFile class in the fec 1.0.3 library handles many things in preparing a file for encoding automatically with little impact on the designer. Therefore it was chosen to first have an .xsbc file created, although not efficient for continuous streaming purposes, but other encoding preparations are taken care of such as breaking up a file into encoding blocks if necessary and of padding with zeros in case of the last block not being of size *n*. This padding is necessary in order to keep matrices of size *n x k* even though there may not be enough source code symbols to encode in the last block.

Next, the stripOrderArray is created which will become an integer array containing the indices of each encoded repair packet as they were when encoded within the *GF* Vandermonde matrix. *n* repairBuffer's are then prepared each of size 1024 bytes. This index information is sent out ahead of time and received by the AUVW Server for later incorporation into the decoding process.

In order to simulate packet loss and to optimize the decoding process, the Util.shuffle() method shuffles the order of the indices so that, even though the first *k* packets encoded per block are exactly in their original unencoded source symbol form, these original source packets will get mixed in with encoded repair packets and transmitted. It is prudent to send out source symbols in their original source form, especially for higher values of *(n, k)* so that in the event of successfully receiving a source symbol packet in its original unencoded form, that repair packet will get shuffled back into its < *k* position within the identity matrix portion of the *GF* matrix before decoding. This will save some cycles in decoding time as they do not need to be decoded.

Next, the fecF.read() method is called which gives the arguments of the repairBuffer, block number index and stripeOrder index. This is where each block's worth of *n* encoded repair packets are generated just before transmission. Recall that the first *k* of each block's *n* encoded repair packets are the actual original source code symbols. This is so because if these *k* are received intact and not dropped, then the decoding process is trivial as there will be no decoding process to perform by the AUVW Server. However, if the data transfer process were to experience packet drop losses, then any *k* subset of each block's *n* encoded repair packets will suffice to perform a proper and complete decoding of the original data. Finally, each block's worth of *n* encoded repair packets are transmitted to the AUVW Server for decoding by the XsbcTransaction process via a conventional Java Datagram socket implementation.

2. XSBC Transaction Functionality within the AUVW

The transaction process will be a reverse process of the compression/encoding process. Once the AUVW Server has received all *n* repair packets per message block then *k* of those repair packets from each block will be selected so that the decoding process can begin. This is performed by the fec.decode() method in the XsbcTransaction.java file. Its arguments are the repairBuffer holding the encoded data, in the same manner as the wrapper that contained the encoded repair packets before transmission, and the index information received ahead of time indicating in which index the repair packet was encoded inside of each message block so that proper decoding can take place within the server's decoding *GF* matrix.

```
/**
 * Performs preparations for FEC decoding to enable MissionOutputXX.xml
 * file saving to cache.
 *
 * @exception IOException if the socket couldn't read packets
 */
private void doFECDecoding() {

   //create our fec code
   fec = FECCodeFactory.getDefault().createFECCode( params.getK(),
   params.getN() );

   // Prepare to receive our K * Block Count repair indices.  These
   // are just byte format integers for each Datagram packet.  Again,
   // we will select our K required subset from the N received from
   // each block.
   indicePacket = new DatagramPacket( new byte[ params.getN() ],
   params.getN() );

   // Storage for each block's K subset of N repair indexes that
   // will be received via UDP.
   repairIndexes = new int[ params.getK() * params.getBlockCount() ];

   // Receive each block repair index information
   for ( int ix = 0; ix < params.getBlockCount(); ix++ ) {

      try {

         // Receive our repair index info.
         udpSocket.receive( indicePacket );

      } catch ( IOException ioe ) {

         ioe.printStackTrace();

      } // end try-catch block

      // Feed index information into this BAIS per block
      bais = new ByteArrayInputStream( indicePacket.getData() );
```

```
         System.out.println( "Receiving block number: " + ix );

         // Extract K * Block Count packet elements that make up our
         // repair indices.
         for ( int jx = 0; jx < params.getK(); jx++ ) {

            // Read in our K subset indice information
            repairIndexes[ jx ] = bais.read();

//                  System.out.println( "Received repair packet index: " +
//                  repairIndexes[ jx ] ); // DEBUG

         } // end inner for

      } // end outer for

      /***************** Receive Encoded Repair Packets *****************/

      // Prepare this packet for reception of a repair packet each
      // containing 1024 bytes of encoded data
      fecPacket = new DatagramPacket( new byte[ params.getPacketSize() ],
      params.getPacketSize() );

      // Container for the K required/selected encoded repair packets
      encodedNData = new byte[ params.getN() * params.getPacketSize() *
                               params.getBlockCount() ];
      encodedKData = new byte[ params.getK() * params.getPacketSize() *
                               params.getBlockCount() ];

      // Receive each block's worth of a K subset of N encoded repair packets
      // for storage
      for ( int ix = 0; ix < params.getK() * params.getBlockCount(); ix++ ) {

         // Receive a block of data to encode
         try {

            // Receive our encoded data
            udpSocket.receive( fecPacket );

         } catch ( IOException ioe ) {

            ioe.printStackTrace();

         } // end try-catch block

         // Copy N encoded packets into a byte[] for K selection
         System.arraycopy( fecPacket.getData(), 0, encodedNData, ix *
         fecPacket.getLength(), fecPacket.getLength() );

      } // end for

      System.out.println( "Expanded Block Size: " +
      params.getExpandedBlockSize() ); // DEBUG

      // Extract a K subset of the N encoded repair packets for decoding
      System.arraycopy( encodedNData, 0, encodedKData, 0, params.getK() *
                        params.getPacketSize() );

//      for ( int ix = 0; ix < params.getK(); ix++ ) {
//
//         System.out.println( "Receive FEC encoded data packet " +
//         ix + " " + encodedKData[ ix ] ); // DEBUG
```

```
//
//      } // end for

      // This will hold our K subset of encoded repair packets
      repairBuffer = new Buffer[ params.getK() * params.getBlockCount() ];

      // Put the encoded data into an FEC Buffer wrapper for decoding
      for ( int ix = 0; ix < params.getK() * params.getBlockCount(); ix++ )
         repairBuffer[ ix ] = new Buffer( encodedKData, ix *
         params.getPacketSize(), params.getPacketSize() );



      // Finally, decode the repair packets into the file
      fec.decode( repairBuffer, repairIndexes );

   } // end doFECDecoding()
```

B. SUMMARY

Since the work of this coding is done in *GF(*2^8*)*, efficient use of memory can take place by using lookup tables for logarithmic, inverse and exponential functions needed to encode and decode within the ***G*** matrix. By having an XML file already compressed with XSBC, using the supplied FECFile class within the FEC 1.0.3 library will enhance usefulness by the designer. This is realized by the functionality built into this class of taking a file of any size, breaking it up into appropriate block sizes according to prespecified FECParameters. Encoding and subsequent streaming of the encoded data can then take place with very little overhead in computations.

This chapter concludes the combining of two disparate technologies into a workable implementation where acoustic networking applications can benefit by this combination. By compressing the XML data and adding the required redundancy to that data, it can then be transmitted in a noisy medium such as a shallow water littoral region with assured reliability. The need for a feedback channel can be eliminated by implementing transmission via UDP by the acoustic network and AUV power supplies can be preserved for extended use by not having to “try till you die” in order to transmit critical mission data.

For full text versions of the above source code consult Appendix C.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. USER INTERFACE: AUV WORKBENCH (AUVW) AND XML-BASED TACTICAL CHAT (XTC)

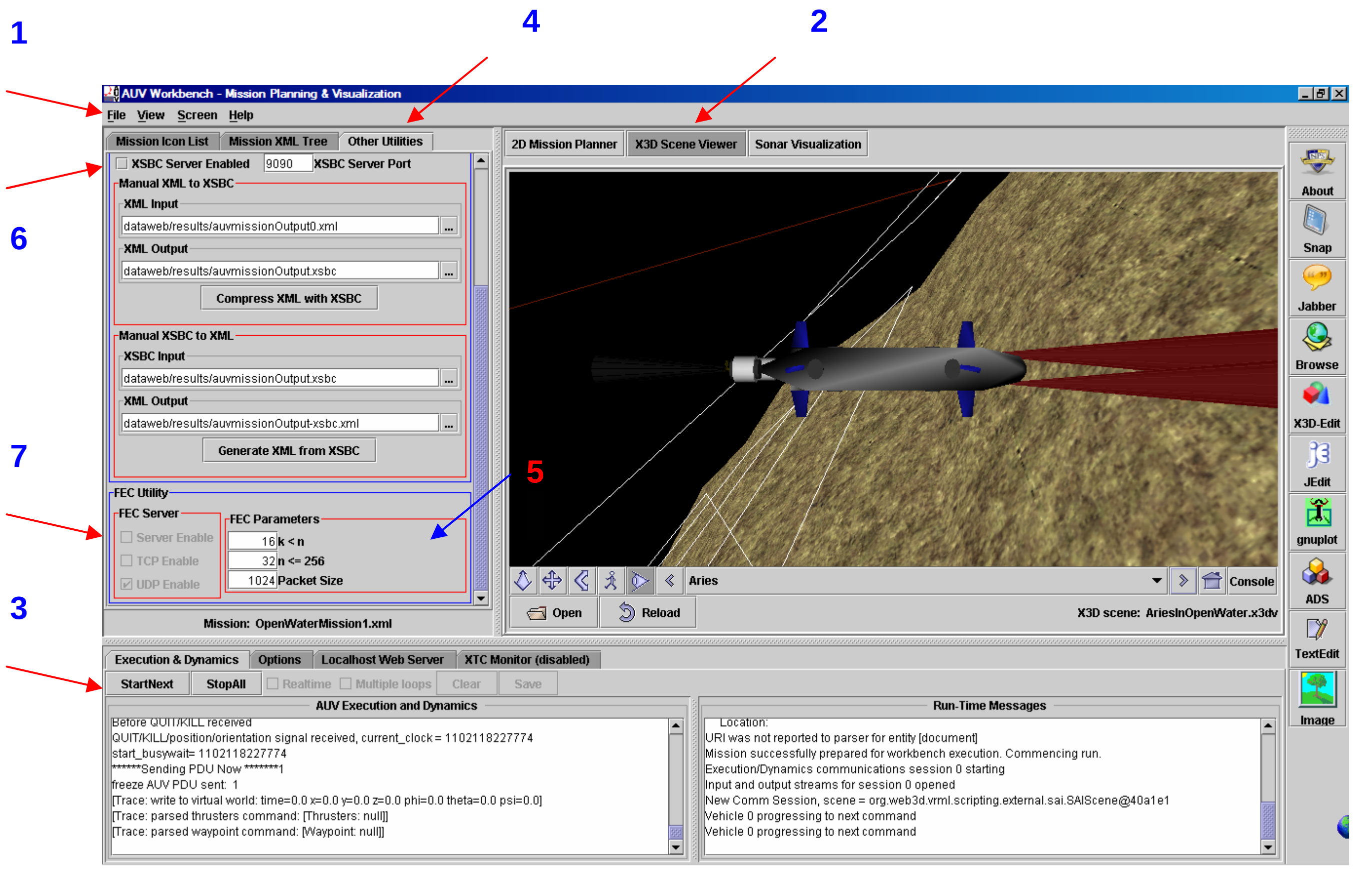
A. INTRODUCTION

This chapter will introduce the Graphical User Interfaces (GUI) associated with the AUVW that enable it to produce mission results and other chat logging files compressed with XSBC, GZipped and encoded with FEC then acoustically transmitted from an AUV to the AUVW Server for archiving and later retrieval.

B. INTERFACE FOR THE XSBC AND FEC SERVERS

Since the AUVW is a Java-based application built for the research and development (R&D) of a common command and control language for dissimilar autonomous vehicles, whether air or sea based (Lee 2004), the integration of FEC into the AUVW was a straightforward process since it is also a Java-based library. The approach was to have FEC functionality built into the XSBC process, as this is where it becomes most useful to aid in reliable transfer and reception of files in the acoustic environment.

The original design process of XSBC transfer is for the AUV to transmit via TCP an XSBC/GZipped mission results files by default. This can now be set false by alteration the AUV/UAV XML based configuration files within the AUVW build package. In order to shed the cost of maintaining a reliable connection such as TCP, the UDP option was chosen to transmit an FEC encoded *.xsbc.gz mission results file. Operation of FEC for archiving mission results files in the AUVW is shown below:



XSBC Server Enabled: 9090 XSBC Server Port

XML Input: dataweb/results/auvmissionOutput0.xml
XML Output: dataweb/results/auvmissionOutput.xsbc

XSBC Input: dataweb/results/auvmissionOutput.xsbc
XML Output: dataweb/results/auvmissionOutput-xsbc.xml

16 k < n
32 n <= 256
1024 Packet Size

Mission: OpenWaterMission1.xml

X3D scene: AriesInOpenWater.x3dv

```
Before QUIT/KILL received
QUIT/KILL/position/orientation signal received, current_clock = 1102118227774
start_busywait= 1102118227774
******Sending PDU Now *******1
freeze AUV PDU sent:  1
[Trace: write to virtual world: time=0.0 x=0.0 y=0.0 z=0.0 phi=0.0 theta=0.0 psi=0.0]
[Trace: parsed thrusters command: [Thrusters: null]]
[Trace: parsed waypoint command: [Waypoint: null]]
```

```
    Location:
URI was not reported to parser for entity [document]
Mission successfully prepared for workbench execution. Commencing run.
Execution/Dynamics communications session 0 starting
Input and output streams for session 0 opened
New Comm Session, scene = org.web3d.vrml.scripting.external.sai.SAIScene@40a1e1
Vehicle 0 progressing to next command
Vehicle 0 progressing to next command
```

Figure 18. AUVW GUI for the XSBC and FEC Server Functionality

1) Once the AUVW is invoked, select mission (i.e. OpenWaterMission1.xml or OpenWaterMission1a.xml) from the File\Open Mission menu. Observe that the mission waypoints are visible in the 2D Mission Viewer.

2) Select the X3D Scene Viewer Panel, then select a scene (i.e. AriesInOpenWater.x3d) to render

3) Start the mission by selecting StartNext

4) Select the Other Utilities Panel

5) The default parameters for FEC are adequate for most applications. Keep the packet size <= 1024 bytes, *n* <= 256 and *k* < *n*. *n* should be a multiple of *k* and *k* should be a factor of *n* for best performance of the FEC Utility.

6) Enable the XSBC Server to enable FEC Server Enabling

7) Select FEC Server Enable

XSBC and GZipping processes will be initiated automatically as per the AUVWorkbenchConfiguration.xml file. To allow reception and archiving of these files, the XSBC Server must be enabled. Once the XSBC Server is enabled, the FEC Server check box will become enabled to allow for selection of FEC Encoding. This will tell the transmitting server to switch sockets to a DatagramSocket to allow for UDP for sending FEC encoded repair packets. Selecting FEC Server Enable will tell the receiving XSBC server to expect these repair packets via UDP and for FEC decoding to take place before un-GZipping and final deserialization of the mission results file. The final step is uploading to a specified folder within the AUVW Server that will archive the file for later retrieval.

C. INTERFACE WITH XML-BASED TACTICAL CHAT (XTC)

As may have been noticed, in the above figure, on the right side of the GUI panel for the AUVW are a vertical column of application access buttons. One of them invokes the Jabber chat client utility. This is an open sourced chat client built into the AUVW for chat and messaging capability. By providing a set of streaming XML protocols and technologies, two entities on the internet are enabled for exchange of messages, presence and other structured information in real time (Lee 2004). Jabber also offers appropriate bridging functionality to similar legacy instant messaging (IM) services such as Yahoo, America online (AOL) instant messaging (AIM).

The advantage of using Jabber for these functions are that is free, open, public and secure because of the adherence to the Extensible Messaging and Presence Protocol (XMPP) that the Jabber protocol is built around. XMPP is supported by the Internet Engineering Task Force (IETF) for its use of Simple Authentication and Security layer (SASL) and Transport Layer Security (TLS). These technologies allow Jabber to interface with the above mentioned proprietary IMs and with others.

The purpose of the Jabber client for use within the AUVW is that two operators can communicate securely about mine contacts reported during an AUV mission deployed to search and classify mine like objects. The Jabber client IM or chat room can be monitored for keywords such as “mine”, “contact”, “mine-like-object” and so forth. These keywords can be monitored because an incoming XML message stream can be parsed for these keywords. This is the function of the XTC Monitor module built into the AUVW. The IMCriteria module of the XTC Monitor causes events to raise alerts within the AUVW triggered by the above mentioned keywords. These messages are parsed by looking for the CDATA element tag of the XML message and then parsed for the appropriate keywords. The XML tagsets that allow for this are shown below.

```
<XTCMonitor>
   <MonitorDefault keywordSubject="mine, bomb, torpedo" keywordBody="nice, mine,
    bomb, torpedo, location, CVN62, SNN12, DDG51">
      <WatchEvent name="Mine" desc="Look out for Mines" expr="^.*(?i)MINE[s|S]? .*[
       (]{1,2}(\d*),[ ]{0,2}(\d*),[ ]{0,2}(\d*)[ ).]?+">
        <Alert type="visual" src="build/image/mine.gif" enabled="true"/>
        <Alert type="sound" src="sound/alert.wav" enabled="true"/>
        <Alert type="url" src="C:/auv/Workbench/javadocs/index.html" enabled="false"/>
      </WatchEvent>
      <WatchEvent name="Ship" desc="Look out for Ships" expr="^.*(?i)SHIP[s|S]? .*[
       (]{1,2}(\d*),[ ]{0,2}(\d*),[ ]{0,2}(\d*)[ ).]?+">
        <Alert type="visual" src="build/image/ship.gif" enabled="true"/>
      </WatchEvent>
      <WatchEvent name="Location" desc="Look out for Locations"
       expr="^.*(?i)LOCATION[s|S]? .*[ (]{1,2}(\d*),[ ]{0,2}(\d*),[ ]{0,2}(\d*)[
       ).]?+" alert=""/>
   </MonitorDefault>
   <Monitor jid="savage@conference.xchat.movesinstitute.org" desc="" datetimeStart=""
    dateTimeEnd="" keywordSubject="mineX, bomb, torpedo" keywordBody="mineX, bomb,
    torpedo, location"/>
   <Monitor jid="auvrobot@surfaris.cs.nps.navy.mil" desc="" datetimeStart=""
    datetimeEnd="" keywordSubject="urgent, problem" keywordBody="damage, sinking,
    surface"/>
</XTCMonitor>
```

Figure 19. XML Tagset Describing Jabber Chat Session Parsing for Mission Critical Keywords (From Lee 2004)

For XSBC to work with these types of XML messages, a schema must be present to define the structure and possible data types that the XML message is comprised of. This schema is not currently designed with the current build of the AUVW, however; if designed to do so, XSBC can be used to compress such messages for transport over other protocols such as TCP/IP or UDP. FEC can still be used to add the desired redundancy to the data symbols since FEC is not concerned about what type of data is being encoded, however the compressed ratios that XSBC offers would not be realized within that particular schema-less XML document. This can be a future developmental capability of the AUVW.

D. SUMMARY

Introduced were the user interfaces to invoke both XSBC and FEC servers. XML mission results files can be transported and archived, for later retrieval, reliably and efficiently through XSBC and with FEC encodings. XTC is a capability of the AUVW to report mission observations and keywords noted and will raise appropriate alarms within AUVW to alert of these mission details. This is accomplished by parsing of XML Jabber server logs and incoming XML messages to Jabber clients.

VIII. SECURITY CONSIDERATIONS

A. INTRODUCTION

The realm of UANs is not without compromise. As information can now be exploited better than ever by computer savvy enemies, protection of underwater network traffic is required to be employed with greater skill and know how. Packet sniffers can be utilized by the enemy to gain insight into protocol usage and enough information could possibly be obtained to introduce a “man in the middle” attack, or, to possibly introduce corrupt packets that would render a data set useless. Protection against these types of intrusions will be discussed as well as some techniques to counter these malicious attempts at exploiting friendly informational assists.

B. CONSIDERATIONS FOR COUNTERING INFORMATIONAL EXPLOITATIONS

Delivery of vital information can be subjected to denial-of-service (DoS) attack by intruders who would send corrupted packets which might be accepted as legitimate by certain receivers (Luby and others 2002). This type of intrusion is a major concern when speaking of multicast delivery schemes due to the possibility of receiving a corrupted packet within a session which could be injected close to the root of a multicast tree. If this were to succeed, there is grave concern because this corrupted packet would now be transmitted to many receivers. When using FEC for encoding information building blocks, the corrupted packet containing encoded data may result in the decoding of an object that is totally unusable by an application depending on the reception of reliable data.

In order to counter this type of denial of service attack, Request for Comments (RFC) 3452 (see Request for Comments) suggests that decoded objects be checked for integrity before delivering objects to applications. An example is using a Message-Digest Algorithm 5 (MD5) hash (Rivest 1992) of an object which could be appended to the protocol header before transmission, and then that hash could be computed and checked after that object is decoded, but before it is delivered to the application needing the object. RFC 3452 also suggests obtaining strong cryptographic integrity protection by use of a digital signature which would be verifiable by the receiver by being computed on top of the MD5 hash value.

A packet authentication tool such as TESLA (Perrig, Canetti, Song and Tygar 2001) can be used to detect and discard corrupted packets upon arrival to a receiver.

Yet another technique to ensure packet reliability is to enable employment of Reverse Path Forwarding (RPF) checks to limit the possibility of a bad agent being successful in injecting a corrupted packet into the multicast tree data path. These are checks that a router can use to forward a multicast datagram if received on the interface used to send unicast datagrams to the source.

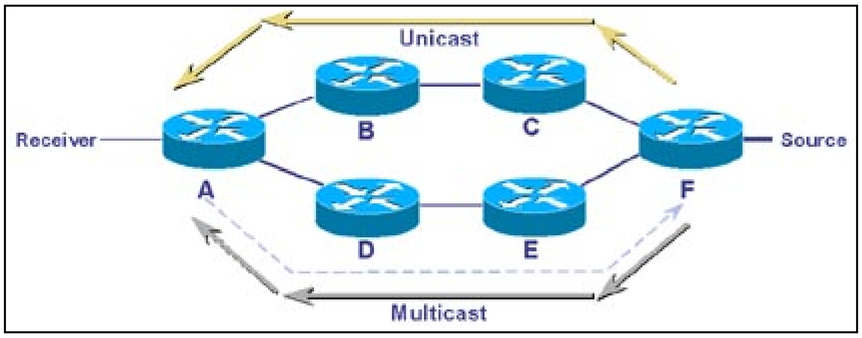


Figure 20.   Illustration of Reverse Path Forwarding

Another possible concern is that some FEC information may be accepted by receivers out-of-band in a session description, and if that session description is forged or corrupted, then the receivers will not use the correct protocol for decoding content received from those packets. Measures to prevent receivers from accepting incorrect session descriptions, such as using source authentication to ensure receivers only accept legitimate session descriptions from authorized senders, should be considered.

C. XML CANONICALIZATION AND SECURITY

Logical equivalence of two XML documents is the premise of XML canonicalization since this equivalence will be determined by comparison of the canonical (or simplified) form, octet-by-octet (Siddiqui 2002). If the form of each document contains the same sequence of octets, it is assumed that the two XML documents are logically equivalent.

This logical equivalence comes into play when digitally signing XML documents for security and authenticity reasons. W3C is the entity for recommending how the canonicalization process should take place and gives guidelines for the process. In order to protect the mission results files generated by an AUV for archival with the AUVW Server, these files can be canonicalized, digested as with the MD5 process mentioned earlier, then digitally signed with a private key before undergoing the FEC encoding process and subsequent UDP transfer of the document to the server. The reason for canonicalization before digesting and signing the mission results XML file is to preclude any signature validation attempts due to variations of an equivalent, but non-cononical version of that XML document.

Canonicaliztion involves defining a standard encoding scheme such as UTF-8, representing line breaks with a standard character string such as #xA, normalizing attributes, encapsulating attribute values by double quotes, handling special characters in attribute values, replacing entity references with actual content, defining default attributes in canonical XML form, removing XML and DTD declarations, proper handling of whitespace, namespace declaration preservation and ordering of namespace declarations. These procedures are detailed with the official specification available at W3C (see Canonocial XML) .

D. SUMMARY

Presented here were issues to consider in order to maintain data security and authenticity when transmitting sensitive mission data files over a UAN such as from an AUV to an AUVW Server for archiving and later retrieval. Techniques such as utilizing MD5 along with public and private key pair digital signatures can be an option to verify integrity. In order to protect against corrupted packets, whether by natural or man-made causes, applications like TESLA can be utilized to detect and drop corrupted packets. By canonicalizing XML documents, digests and digital signatures can be verified due to the logical equivalence factor made possible by XML canonicalization.

IX. CONCLUSIONS AND RECOMMENDATIONS

A. INTRODUCTION

The research conducted in support of this thesis was to introduce an open source solution for reliable XML data transport within noisy links such as a UAN. The UAN environment is dynamic in nature and transmission of data must be enabled to withstand this sometimes data transmitting resistant nature. The process of XSBC and FEC encodings will help in data delivery reliability.

XML compression is realized through the use of XSBC which reads in a schema and tokenizes elements and attributes according to the layout of that schema and replaces with those elements and attributes with serialized integers. In this manner, the verboseness of XML plain text is greatly reduced so that documents can realize compression ratios on average of 17% of their original size over GZipping alone.

By sending the serialized data through a GZip steam, even more compression can be accomplished before adding redundancy through FEC encodings. The process of FEC encoding does not increase the size of the file as the file is broken up into blocks according to file size and the appropriate FEC parameters chosen. Network capacity is somewhat taxed as a trade off to sending many packets with redundant information; however the benefit is much more reliable data transfer and reconstruction. Encodings take place within the encoding *GF* matrix expanded to encode redundant symbols which are *n*-*k*, where *k* is the value of the original source symbology and *n* is the value of desired redundancy thereby producing a systematic erasure code. Data transfer takes place by sending out all *n* repair (FEC encoded) packets via UDP. Transfer via UDP is pertinent choice as the reliable, yet computationally costly, characteristic of TCP is avoided thereby increasing throughput and avoiding the other cost of maintaining a feedback channel.

When any *k* subset of the *n* repair packets are received, first the FEC decoder checks for packets that are indexed < *k* so as to shuffle them back into their original source identity matrix saving cycles in the decoding process. This is the nature of a systematic code. Any *n-k* repair packets received to fulfill the rest of the required *k* subset will be placed in the *GF(2^n)* matrix in the same index as during their encoding process for data recovery (FEC decoding). The data is recovered by using the inverse *GF(2^n)* matrix multiplied by the ***y'*** encoded vector to solve back into the original *x* source symbols.

Un-gzipping and XML de-serialization can take place along with any employed security authentication techniques employed to finally arrive at the ability to archive a verifiably authentic original mission results XML files generated from an autonomous vehicle.

FEC and XSBC are both suited for both uni-cast or multi-cast applications. Both are extensible and scalable to larger applications. By choosing the appropriate FEC parameters, software implementation of FEC was not shown to be a highly computational burden on even simple laptop systems. The added redundancy that produces high reliability of data recovery is of great benefit to systems that have low and quickly exhaustible transmission capabilities for data transfer. By compressing verbose plain text XML data, streaming is efficient is and by utilizing FEC, data recovery and transfer reliability is realized.

B. CONCLUSIONS

By using open sources to accomplish this complex task, the high cost of stove-piped proprietary systems can be avoided. Open sources and open standards promote security, extensibility, cross-platform capability, scalability to larger applications and access to security technologies free and clear of proprietary systems. Both XSBC, maintained and developed by Yumetech, Inc. and FEC, maintained and developed by Onion Networks, Inc., are a desired pair of disparate applications possessing these qualities.

The interfacing methods, to be written by the developer, are intuitive and tailorable to that developer's application needs concerning both the XSBC and FEC libraries.

C. RECOMMENDATIONS

By following RFCs 3695 – Compact Forward Error Correction Schemes, 3452 – Forward Error Correction Building Block and 3453 – The Use of Forward Error Correction in Reliable Multicast, a useful real world application of FEC can be developed. The source code examples contained in the appendices section of this thesis were developed for quick demonstration purposes. Since the AUVW is still under development, the FEC Server implementation exploited the demonstration mode that the AUVW is, at present, used for.

Since there are out-of-band pieces of information to send either ahead, or with the encoded source symbols, a simple, yet effective protocol can be achieved for FEC. As stated earlier, the example source code sidesteps this out-of-band information by simulation methods. When building an FEC code for encoding a file, the parameters *k, n, packetSize* and *fileSize* information must be delivered to the receiving FEC decoding client. An example FEC protocol that can accomplish this could be in similar form to the DIS Entity State Protocol Datagram Unit (ESPDU) format used by other XMSF applications:

Field	Content	Bytes	Value
FEC Server Enabled	Boolean	0	1 or 0
GZip Enabled	Boolean	1	1 or 0
FEC Parameter	*k*	2	1 - <*n*
	n	3-5	< 256
	packetSize	6-9	<=1024
	fileSize	10-max size	KB/MB
MD5	hash value	(offset byte of MD5 hash value)	digest value of XML file
FEC Code	Source Symbols	(offset after MD5 hash value)	1024

Table 3. Proposed Format for an FEC Protocol Containing Data to be Sent Along with Each FEC Encoded Source Symbol Packet

By sending all required field information with each *n* packet sent, this will ensure that any *k* subset of repair packets received can be parsed for this information to initialize FEC parameters in the receiving decoding client. The FECFile class takes an MD5 hash value as an argument during the decoding process and which immediately verifies the integrity of the original source file as before it allows creation of a decoded file. All other field values can be discarded or ignored except the encoded source symbols needed for decoding the entire file in subsequently received repair packets.

If a large file is broken up by the encoding FECFile class, recall that an important parameter "blockSize" is calculated by the fileSize, *k,* and packetSize parameters so that all block specific repair packets can be placed in their appropriate index within a server's decoding matrix, so, the decoder can determine this parameter explicitly through the availability of the other listed parameters within the FEC protocol.

D. FUTURE WORK

Development of a streaming capability feature for FEC would be desirable to realize even faster software implementations of the combined XSBC/FEC utility features. By GZipping, XSBC serialization and FEC encoding with within the same data stream, network capacity requirements can be even further lowered.

An application, much like the afore mentioned Seaweb UAN, which uses selective ARQ to retransmit missing data packets, can be employed to send only those specific redundant packets that were calculated to be missing from the appropriate application. This would involve implementation of a TCP connection should that requirement be desired at the cost of before mentioned TCP characteristics.

Development of a schema for the description and format for Jabber chat sessions or XTC would enable the use of XSBC. XTC logging of data is done in XML, so if these logged chat sessions were desired to be transmitted to other agencies requiring this data, XSBC would add increased efficiency for use in channel capacity limiting characteristics when transmitting this data. Functionality to add a closing “end chat server” tag to the final XTC session log would close the log and enable that log to be a complete and well-formatted XML document.

E. SUMMARY

Open source and open standards allow for scalability and extensibility not available with proprietary software developments. Security is enhanced as source code is readily available for inspection and modification by the designer. XSBC and FEC allow for efficient and reliable transfer of XML documents in any type of noisy environment that would hinder reliable data transfers. This thesis concludes with a solution to the original problem statement. It is the desire of the author that serious consideration be given to the applications presented in this thesis as a low cost and highly reliable solution to other real world applications in support of real time and on demand information for tactical decision-making by the sea warrior.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. LIST OF ABBREVIATIONS

ADM	Admiral
AIM	AOL Instant Messaging
AOL	America Online
ARQ	Automatic Repeat Request
ASCII	American Standard Code for Information Interchange
AVC^{2}L	Autonomous Vehicle Command and Control Language
AUV	Autonomous Underwater Vehicle
AUVW	AUV Workbench
BER	Bit Error Ratio
C	The C Programming Language
CD	Compact Disc
CNO	Chief of Naval Operations
CORBA	Common Object Request Broker Architecture
CRC	Cyclic Redundancy Checksums
DAT	Digital Audio Tape
DBP	Dynamic Behavior Protocol
DIS	Distributive Interactive Simulation
DOD	Department of Defense
DON	Department of the Navy
DoS	Denial-of-Service
EN/DE	Encoding/Decoding Process
ESPDU	Entity State Protocol Datagram Unit
FEC	Forward Error Correction

GL	Galois (pronounced “gal wah”) Field
GMU	George Mason University
GNU	Recursive Acronym for “GNU's Not UNIX”
GZIP	GNU Zip
IEEE	Institute of Electrical and Electronic Engineers
IETF	Internet Engineering Task Force
IIOP	Internet Inter-Object Request Broker Protocol
IM	Instant Messaging
ln	Natural Logarithm (base e)
IP	Internet Protocol
IP/SW	Internet Protocol over Seawater
ISO	International Standards Organization
JNI	Java Native Interface
JVM	Java Virtual Machine
KB	Kilobytes
MB	Megabytes
Mbs	Megabits Per Second
MD5	Message-Digest Algorithm 5
MIT	Massachusetts Institute of Technology
MOVES	Modeling, Virtual Environments and Simulation
M&S	Modeling and Simulation
MTU	Maximum Transmission Unit
NEW	Network Education Ware
NOPP	National Oceanographic Partnership Program

NPS	Naval Postgraduate School
NVE	Networked Virtual Environments
PLR	Packet Loss Rate
RACOM	Radio-acoustic Communications
R&D	Research and Development
RF	Royalty Free
RFC	Request for Comments
RMI	Remote Method Invocation
RPF	Reverse Path Forwarding
RTT	Round Trip Time
SASL	Simple Authentication and Security Layer
SOAP	Simple Object Access Protocol
SPAWAR	Space and Warfare
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UAN	Underwater Acoustic Network
UAV	Unmanned Aerial Vehicle
UDP	User Datagram Protocol
UTF-8	Unicode Transformation Format – 8
UUV	Unmanned Underwater Vehicle
UW	Undersea Warfare
W3C	World Wide Web Consortium
X3D	Extensible 3D
XFSP	Cross Format Schema Protocol

XML	Extensible Markup Language
XML-RPC	XML Remote Procedure Call
XMPP	Extensible Messaging and Presence Protocol
XMSF	Extensible Modeling and Simulation Framework
XOR	Exclusive OR
XSBC	XML Schema based Binary Compression

APPENDIX B. FAMILIARITY NOTES FOR THE FEC 1.0.3 LIBRARY

A. INTRODUCTION

The following paragraphs are presented by the author as a familiarity guide into the methods employed by the FEC 1.0.3 library. These notes were written during trial and error attempts at working with this Java API. This was written with the intent of self-guidance and discovery, and thus it is hoped that the reader will have an understanding into the author's rationale in how to work with this remarkable library.

B. SETUP OF THE FEC 1.0.3 LIBRARY

At this site: http://www.onionnetworks.com/developers/ (Last accessed November 2004) is located the Java open source fec-1.0.3.zip download link.

Upon unzipping, rename the destination end folder. This (your named) end folder contains two folders, fec-1.0.3 and META-INF. The former contains all the goodies. I use the netBeans™ IDE (see netBeans IDE). Mount the fec-1.0.3 folder in netBeans.

This application can be modified / rebuilt using Ant. netBeans™ bundles Ant into its IDE and Ant itself can be updated. Due to dislike of netBeans bundling other application within its IDE the author has the IDE point to a separate (newest) installation Ant 1.6.2 directory. Separation of applications is less confusing to work with. netBeans also bundles current J2SEs.

After inspection of the file structure in the netBeans explorer, locate the fec 1.0.3 build.properties file, right click, select edit and comment out "build.compiler=jikes". This causes many problems. This project builds just fine using the current Java SDKs javac.exe. The Ant script automatically calls this. There are some other modifications commented with the author's initials (tdn) that were found necessary to do within this file and other pieces of source code modified for functionality to work within the AUVW. Prior to building this application with Ant, do this: In the \fec-1.0.3\fec-1.0.3\bin\lib directory is the fec.properties file, comment out all references to “native” and keep all references to “pure”. This is eluded to in the \fec-1.0.3\fec-1.0.3\docs\README file. When building ${app.jar}, this will force pure java implementation of FEC (en/de) routines.

Received a personal email reply, concerning posed implementation questions, dtd 29 JUL 04 from Justin Chapweske, founder of Onion Networks Inc. on the fec-1.0.3 release:

> For native vs. pure, I believe you simply have to have the appropriate native library jar in your class path and it will automatically attempt to load the native library first, then fall back to the pure Java library.
>
> I believe in the 1.0.3 version there are actually some test programs that don’t compile out of the box, so you may need to make some modifications in order to allow them to do so.
>
> After a long hiatus, we are again making new enhancements to the FEC library, so over the next month or so we'll probably post a version 1.1 release that will be much cleaner than the current release in terms of packaging and usage.

There is a fec-win32.jar file that contains these native C routines in 8 and 16 bit implementations, namely fec8.dll and fec16.dll. If one does not include this .jar in the classpath, or, simply not mount this .jar in the netBeans file system, then only pure Java FEC will be implemented.

The build.xml file, used as the Ant Script, can be modified in the “jar target” element to ease the user in creating one jar file that contains embedded classes from other jar files used to compile the source code that come with this build. **However, this is just a convenience method as is not recommended for actual building of applications**. Modified as follows:

```
<!--=========================================================== -->
  <target depends="classes" description="Build the jar files" name="jar">
     <!-- Included the manifest from concurrent-jaxed.jar due to some errors
          that showed up when trying to read this app.jar manifest (tdn) -->
     <jar jarfile="${app.jar}" manifest="manifest/fec.mf" update="true">
        <!-- Include all the ${classpath} jar file classes used to compile the
             classes for creating the application jar.  This produces one happy
             jar for everyone to enjoy.  Found this command on the Java
             Tutorial site (tdn) -->
        <zipgroupfileset dir="${lib}" includes="*.jar"/> ← This is the
        command that embeds classes from other jar files into the ${app.jar}
        <fileset dir="classes">
           <include name="**"/>
        </fileset>
        <fileset dir="bin">
          <include name="lib/**/*"/>
        </fileset>
     </jar>
  </target>
```

Figure 21. Ant Script that Contains the Command to Combine Multiple JAR Files into One for Convenience

Right clicking and executing the javadoc target compiles the source code, builds the onion-fec.jar or ${app.jar} file and produces javadoc all in one swoop. Recommend this be done prior to working with the code. The produced javadoc is missing documentation from the com.onionnetworks.util package. I believe this package is used for Java Native Interface (JNI) use of the native C code that code is originally based on. Here is a link to a more complete online javadoc: http://onionnetworks.com/fec/javadoc/. If the reader mounts this link into netBeans' javadoc manager, methods will be able to point to its documentation by right clicking over a method and selecting "Show Javadoc".

Expand this folder: fec-1.0.3\lib and mount the following jar files in netBeans: concurrent-jaxed.jar, log4j.jar, onion-common.jar and newly produced onion-fec.jar to start using this code as is. This can be done simultaneously by selecting all of these jars, right click and select "Mount JAR". The later two are definitely needed to run the included test programs located in the "tests" folder. The concurrent-jaxed.jar, log4j.jar and onion-common.jar are used for compiling the provided source code and producing the onion-fec.jar by the Ant scripted build.xml file. The fec-win32.jar contains the C native routines and can be omitted if you want to your OS run pure Java FEC only.

C. RUNNING THE INCLUDED TEST FILES

If reader desires to run the test files included in the zip, place: package tests; at the top of the source file before compiling. This is so that netBeans will compile the folder and run the files accordingly. The files: CodeTest, FECOverhead, FECFileTest and IOPerformanceTest work well with the modifications the author has introduced. Modification of the FECFileTest from the fec-1.0.3 library was necessary to be either called from another program or to run as an instantiable main class to show the (EN/DE) process. That file was renamed XSBC_FECFileTest.java so that it could be called from the Comparison Tool of the XSBC 0.91.1 library.

D. TEST FILE DEBUGGING FOR ANALYSIS OF FUNCTIONALITY

Okay, some newer and more interesting stuff. The FECFileTest is an interesting test of the FEC code API. The author experimented with the MOVES Institute logo which can be modified into .jpg format. The file size is 146 KB. The FEC code breaks it up into nine blocks of packet size 1024 B after encoding.

```
FECParameters(k=16,n=32,packetSize=1024,fileSize=146907),
File partition block count: 9,
Hash code count: -287834112, Max stripe size: 9216 B,
Default number of vanilla bytes in a block: 16384 B,
Max num of bytes that a fully encoded block can contain: 32768 B,
Max num stripes (N) that can be created from this file: 32,
Min num packets required to recreate orig. file: 144
Packet size for block: 0 = 1024 B
Num of packets required to send across the original block: 0 = 16
Packet size for block: 1 = 1024 B
Num of packets required to send across the original block: 1 = 16
Packet size for block: 2 = 1024 B
Num of packets required to send across the original block: 2 = 16
Packet size for block: 3 = 1024 B
Num of packets required to send across the original block: 3 = 16
Packet size for block: 4 = 1024 B
Num of packets required to send across the original block: 4 = 16
Packet size for block: 5 = 1024 B
Num of packets required to send across the original block: 5 = 16
Packet size for block: 6 = 1024 B
Num of packets required to send across the original block: 6 = 16
Packet size for block: 7 = 1024 B
Num of packets required to send across the original block: 7 = 16
Packet size for block: 8 = 1024 B
Num of packets required to send across the original block: 8 = 16
Num of packets written to disc: 144
```

Figure 22. Debug Information Obtained by FEC Encoding the MOVESLogo.jpg File

By importing org.apache.log4j.BasicConfigurator and by instantiating the configurator as follows: BasicConfigurator.configure(); detailed debug information, such as in the following figure, can be retrieved.

```
0 [Decoder Thread] DEBUG com.onionnetworks.fec.io.FECFile  - stripeNums=5,17,20,3,16,2,0,4,21,18,15,7,24,29,11,26
0 [Decoder Thread] DEBUG com.onionnetworks.fec.io.FECFile  - trying to decode block : blockNum=0
Packet size for block: 2 = 1024 B
Num of packets required to send across the original block: 2 = 16
10 [Decoder Thread] DEBUG com.onionnetworks.fec.io.FECFile  - stripeNums=7,28,31,15,19,18,17,9,2,11,26,3,22,5,23,27
10 [Decoder Thread] DEBUG com.onionnetworks.fec.io.FECFile  - trying to decode block : blockNum=1
20 [Decoder Thread] DEBUG com.onionnetworks.fec.io.FECFile  - stripeNums=29,2,5,11,13,3,31,26,4,1,8,7,22,23,14,0
20 [Decoder Thread] DEBUG com.onionnetworks.fec.io.FECFile  - trying to decode block : blockNum=2
Packet size for block: 3 = 1024 B
Num of packets required to send across the original block: 3 = 16
Packet size for block: 4 = 1024 B
Num of packets required to send across the original block: 4 = 16
30 [Decoder Thread] DEBUG com.onionnetworks.fec.io.FECFile  - stripeNums=24,6,26,8,27,9,21,31,11,1,14,15,18,13,5,10
30 [Decoder Thread] DEBUG com.onionnetworks.fec.io.FECFile  - trying to decode block : blockNum=3
Packet size for block: 5 = 1024 B
Num of packets required to send across the original block: 5 = 16
30 [Decoder Thread] DEBUG com.onionnetworks.fec.io.FECFile  - stripeNums=12,9,2,18,15,14,6,4,16,27,10,8,20,11,29,1
40 [Decoder Thread] DEBUG com.onionnetworks.fec.io.FECFile  - trying to decode block : blockNum=4
40 [Decoder Thread] DEBUG com.onionnetworks.fec.io.FECFile  - stripeNums=12,13,21,23,26,29,22,18,20,16,7,3,0,9,6,4
40 [Decoder Thread] DEBUG com.onionnetworks.fec.io.FECFile  - trying to decode block : blockNum=5
Packet size for block: 6 = 1024 B
Num of packets required to send across the original block: 6 = 16
Packet size for block: 7 = 1024 B
Num of packets required to send across the original block: 7 = 16
50 [Decoder Thread] DEBUG com.onionnetworks.fec.io.FECFile  - stripeNums=6,13,31,15,25,14,7,10,22,4,18,17,11,5,26,21
50 [Decoder Thread] DEBUG com.onionnetworks.fec.io.FECFile  - trying to decode block : blockNum=6
Packet size for block: 8 = 1024 B
Num of packets required to send across the original block: 8 = 16
60 [Decoder Thread] DEBUG com.onionnetworks.fec.io.FECFile  - stripeNums=3,2,22,19,11,26,28,17,23,18,14,10,12,13,21,31
140 [Decoder Thread] DEBUG com.onionnetworks.fec.io.FECFile  - trying to decode block : blockNum=7
140 [Decoder Thread] DEBUG com.onionnetworks.fec.io.FECFile  - stripeNums=15,24,10,9,31,20,8,16,4,26,14,25,17,28,23,12
140 [Decoder Thread] DEBUG com.onionnetworks.fec.io.FECFile  - trying to decode block : blockNum=8
140 [Decoder Thread] DEBUG com.onionnetworks.fec.io.FECFile  - File Decoded, switching to read-only
Num of packets written to disc: 144
All Good!
```

Figure 23. DEBUG Information Obtained by Instantiating the org.apache.log4j.BasicConfigurator [BasicConfigurator.configure()] within FECFileTest.java

```
Loading: C:\Documents and Settings\Terry\My
Documents\MyFiles\NPS\Courses\MV 0810 Thesis Work\Java
Code\XSBC_FEC\xsbc_fec\examples\espdu.xml

ElementNumber: 137 AttributeNumber: 173
Bytes:  Elements: 1 attributes: 1
ElementNumber: 137 AttributeNumber: 173
Bytes:  Elements: 1 attributes: 1
Unknown attribute: /espdu/@xmlns:xsi
Unknown attribute: /espdu/@xsi:noNamespaceSchemaLocation
Serialized and GZipped: C:\Documents and Settings\Terry\My Documents\My
Files\NPS\Courses\MV 0810 Thesis Work\Java
Code\XSBC_FEC\xsbc_fec\examples\espdu.xsbc.gz
File to encode with FEC: C:\Documents and Settings\Terry\My Documents\My
Files\NPS\Courses\MV 0810 Thesis Work\Java
Code\XSBC_FEC\xsbc_fec\examples\espdu.xsbc.gz
FECParameters(k=16,n=32,packetSize=1024,fileSize=369)
```

Figure 24. DEBUG Information Obtained from Invoking XSBC Compression, GZipping and Encoded with FEC of the espdu.xml File

This is a snapshot of an actual encoded block of the espdu.xml file:

```
Sending FEC encoded data.... Buffer{length: 1024; offset: 0; 0: 125, 1: -127, 2: 109, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0, 10: 63, 11: 11, 12: -110, 13: -
55, 14: -88, 15: -56, 16: 98, 17: 122, 18: 86, 19: 104, 20: 5, 21: 108, 22: -65, 23: -58, 24: 66, 25: 67, 26: 97, 27: 79, 28: -87, 29: -119, 30: -82, 31: 92, 32:
16, 33: 120, 34: 88, 35: -104, 36: 40, 37: 66, 38: 113, 39: -91, 40: -72, 41: 8, 42: -102, 43: -26, 44: -28, 45: 53, 46: 38, 47: 68, 48: -12, 49: 60, 50: -115, 51:
46, 52: 15, 53: -37, 54: -73, 55: 60, 56: 69, 57: -85, 58: 18, 59: 126, 60: 79, 61: 46, 62: -54, 63: 77, 64: -82, 65: 0, 66: 14, 67: -11, 68: -34, 69: -54, 70: 3,
71: -34, 72: -62, 73: 31, 74: 121, 75: -2, 76: 82, 77: -51, 78: -48, 79: 82, 80: 88, 81: 72, 82: 15, 83: -39, 84: 107, 85: 70, 86: -37, 87: 92, 88: 97, 89: 94, 90: -
60, 91: 75, 92: -122, 93: -103, 94: -47, 95: 47, 96: 81, 97: 114, 98: -107, 99: -119, 100: 69, 101: 93, 102: -9, 103: 63, 104: -51, 105: 79, 106: -111, 107: -59,
108: -79, 109: -41, 110: -93, 111: 60, 112: -75, 113: 94, 114: 98, 115: 32, 116: 23, 117: 101, 118: -20, 119: -44, 120: -91, 121: -97, 122: 121, 123: 6, 124:
108, 125: 11, 126: -65, 127: -66, 128: 98, 129: -70, 130: -11, 131: 101, 132: 41, 133: 4, 134: 79, 135: 66, 136: 91, 137: 61, 138: 118, 139: 16, 140: -107,
141: -81, 142: -25, 143: 90, 144: 121, 145: 1, 146: -117, 147: -71, 148: 91, 149: -34, 150: 39, 151: -38, 152: 39, 153: -33, 154: -87, 155: -33, 156: 88, 157:
19, 158: 36, 159: -126, 160: -57, 161: -64, 162: -20, 163: 84, 164: 90, 165: 78, 166: -48, 167: -117, 168: 57, 169: 83, 170: 69, 171: -6, 172: -91, 173: -6,
174: -121, 175: 14, 176: 50, 177: 49, 178: 94, 179: -97, 180: 95, 181: 75, 182: 13, 183: 89, 184: -47, 185: 57, 186: 7, 187: -96, 188: 113, 189: 127, 190: -
99, 191: -71, 192: 121, 193: -57, 194: 45, 195: 70, 196: 43, 197: -35, 198: 26, 199: -49, 200: 58, 201: 103, 202: 2, 203: -119, 204: 13, 205: 101, 206: 74,
207: 27, 208: 8, 209: 9, 210: 54, 211: -79, 212: 118, 213: -94, 214: 45, 215: -7, 216: -63, 217: 101, 218: 53, 219: -15, 220: 123, 221: 5, 222: 16, 223: -125,
224: 33, 225: -5, 226: -36, 227: 32, 228: -124, 229: 13, 230: -51, 231: -70, 232: 127, 233: 70, 234: 64, 235: 100, 236: 62, 237: -29, 238: 26, 239: 118, 240:
23, 241: 14, 242: 88, 243: 6, 244: 40, 245: -17, 246: 85, 247: 73, 248: -75, 249: -37, 250: -84, 251: -104, 252: 76, 253: -50, 254: 108, 255: 52, 256: 22, 257:
-86, 258: -39, 259: -67, 260: -26, 261: -67, 262: -117, 263: -3, 264: 26, 265: 19, 266: -66, 267: -103, 268: 63, 269: -49, 270: -41, 271: -2, 272: 42, 273: 38,
274: 8, 275: -39, 276: -82, 277: -15, 278: -7, 279: 4, 280: -5, 281: 54, 282: -76, 283: -88, 284: 28, 285: -51, 286: 66, 287: 110, 288: -45, 289: -108, 290: -91,
291: -42, 292: -75, 293: 16, 294: -44, 295: -64, 296: -24, 297: 112, 298: -84, 299: -116, 300: -118, 301: 5, 302: 112, 303: 72, 304: -55, 305: 39, 306: -52,
307: 108, 308: 58, 309: -66, 310: 29, 311: 118, 312: 88, 313: 40, 314: 1, 315: 44, 316: 112, 317: 74, 318: -13, 319: -81, 320: -73, 321: 18, 322: -122, 323: -
70, 324: -13, 325: -56, 326: -85, 327: -73, 328: 112, 329: 75, 330: 18, 331: 92, 332: 17, 333: 12, 334: 93, 335: 97, 336: -92, 337: -102, 338: 24, 339: -121,
340: 80, 341: 23, 342: -90, 343: -106, 344: -62, 345: 98, 346: 7, 347: -70, 348: 117, 349: 68, 350: 109, 351: -127, 352: -13, 353: -73, 354: -26, 355: 68,
356: 53, 357: -127, 358: 22, 359: 5, 360: 0, 361: -58, 362: -96, 363: 2, 364: 112, 365: 35, 366: 46, 367: 0, 368: 0, 369: 0, 370: 0, 371: 0, 372: 0, 373: 0,
374: 0, 375: 0, 376: 0, 377: 0, 378: 0, 379: 0, 380: 0, 381: 0, 382: 0, 383: 0, 384: 0, 385: 0, 386: 0, 387: 0, 388: 0, 389: 0, 390: 0, 391: 0, 392: 0, 393: 0,
394: 0, 395: 0, 396: 0, 397: 0, 398: 0, 399: 0, 400: 0, 401: 0, 402: 0, 403: 0, 404: 0, 405: 0, 406: 0, 407: 0, 408: 0, 409: 0, 410: 0, 411: 0, 412: 0, 413: 0,
414: 0, 415: 0, 416: 0, 417: 0, 418: 0, 419: 0, 420: 0, 421: 0, 422: 0, 423: 0, 424: 0, 425: 0, 426: 0, 427: 0, 428: 0, 429: 0, 430: 0, 431: 0, 432: 0, 433: 0,
434: 0, 435: 0, 436: 0, 437: 0, 438: 0, 439: 0, 440: 0, 441: 0, 442: 0, 443: 0, 444: 0, 445: 0, 446: 0, 447: 0, 448: 0, 449: 0, 450: 0, 451: 0, 452: 0, 453: 0,
454: 0, 455: 0, 456: 0, 457: 0, 458: 0, 459: 0, 460: 0, 461: 0, 462: 0, 463: 0, 464: 0, 465: 0, 466: 0, 467: 0, 468: 0, 469: 0, 470: 0, 471: 0, 472: 0, 473: 0,
474: 0, 475: 0, 476: 0, 477: 0, 478: 0, 479: 0, 480: 0, 481: 0, 482: 0, 483: 0, 484: 0, 485: 0, 486: 0, 487: 0, 488: 0, 489: 0, 490: 0, 491: 0, 492: 0, 493: 0,
494: 0, 495: 0, 496: 0, 497: 0, 498: 0, 499: 0, 500: 0, 501: 0, 502: 0, 503: 0, 504: 0, 505: 0, 506: 0, 507: 0, 508: 0, 509: 0, 510: 0, 511: 0, 512: 0, 513: 0,
514: 0, 515: 0, 516: 0, 517: 0, 518: 0, 519: 0, 520: 0, 521: 0, 522: 0, 523: 0, 524: 0, 525: 0, 526: 0, 527: 0, 528: 0, 529: 0, 530: 0, 531: 0, 532: 0, 533: 0,
534: 0, 535: 0, 536: 0, 537: 0, 538: 0, 539: 0, 540: 0, 541: 0, 542: 0, 543: 0, 544: 0, 545: 0, 546: 0, 547: 0, 548: 0, 549: 0, 550: 0, 551: 0, 552: 0, 553: 0,
554: 0, 555: 0, 556: 0, 557: 0, 558: 0, 559: 0, 560: 0, 561: 0, 562: 0, 563: 0, 564: 0, 565: 0, 566: 0, 567: 0, 568: 0, 569: 0, 570: 0, 571: 0, 572: 0, 573: 0,
574: 0, 575: 0, 576: 0, 577: 0, 578: 0, 579: 0, 580: 0, 581: 0, 582: 0, 583: 0, 584: 0, 585: 0, 586: 0, 587: 0, 588: 0, 589: 0, 590: 0, 591: 0, 592: 0, 593: 0,
594: 0, 595: 0, 596: 0, 597: 0, 598: 0, 599: 0, 600: 0, 601: 0, 602: 0, 603: 0, 604: 0, 605: 0, 606: 0, 607: 0, 608: 0, 609: 0, 610: 0, 611: 0, 612: 0, 613: 0,
614: 0, 615: 0, 616: 0, 617: 0, 618: 0, 619: 0, 620: 0, 621: 0, 622: 0, 623: 0, 624: 0, 625: 0, 626: 0, 627: 0, 628: 0, 629: 0, 630: 0, 631: 0, 632: 0, 633: 0,
634: 0, 635: 0, 636: 0, 637: 0, 638: 0, 639: 0, 640: 0, 641: 0, 642: 0, 643: 0, 644: 0, 645: 0, 646: 0, 647: 0, 648: 0, 649: 0, 650: 0, 651: 0, 652: 0, 653: 0,
654: 0, 655: 0, 656: 0, 657: 0, 658: 0, 659: 0, 660: 0, 661: 0, 662: 0, 663: 0, 664: 0, 665: 0, 666: 0, 667: 0, 668: 0, 669: 0, 670: 0, 671: 0, 672: 0, 673: 0,
674: 0, 675: 0, 676: 0, 677: 0, 678: 0, 679: 0, 680: 0, 681: 0, 682: 0, 683: 0, 684: 0, 685: 0, 686: 0, 687: 0, 688: 0, 689: 0, 690: 0, 691: 0, 692: 0, 693: 0,
694: 0, 695: 0, 696: 0, 697: 0, 698: 0, 699: 0, 700: 0, 701: 0, 702: 0, 703: 0, 704: 0, 705: 0, 706: 0, 707: 0, 708: 0, 709: 0, 710: 0, 711: 0, 712: 0, 713: 0,
714: 0, 715: 0, 716: 0, 717: 0, 718: 0, 719: 0, 720: 0, 721: 0, 722: 0, 723: 0, 724: 0, 725: 0, 726: 0, 727: 0, 728: 0, 729: 0, 730: 0, 731: 0, 732: 0, 733: 0,
734: 0, 735: 0, 736: 0, 737: 0, 738: 0, 739: 0, 740: 0, 741: 0, 742: 0, 743: 0, 744: 0, 745: 0, 746: 0, 747: 0, 748: 0, 749: 0, 750: 0, 751: 0, 752: 0, 753: 0,
754: 0, 755: 0, 756: 0, 757: 0, 758: 0, 759: 0, 760: 0, 761: 0, 762: 0, 763: 0, 764: 0, 765: 0, 766: 0, 767: 0, 768: 0, 769: 0, 770: 0, 771: 0, 772: 0, 773: 0,
774: 0, 775: 0, 776: 0, 777: 0, 778: 0, 779: 0, 780: 0, 781: 0, 782: 0, 783: 0, 784: 0, 785: 0, 786: 0, 787: 0, 788: 0, 789: 0, 790: 0, 791: 0, 792: 0, 793: 0,
794: 0, 795: 0, 796: 0, 797: 0, 798: 0, 799: 0, 800: 0, 801: 0, 802: 0, 803: 0, 804: 0, 805: 0, 806: 0, 807: 0, 808: 0, 809: 0, 810: 0, 811: 0, 812: 0, 813: 0,
814: 0, 815: 0, 816: 0, 817: 0, 818: 0, 819: 0, 820: 0, 821: 0, 822: 0, 823: 0, 824: 0, 825: 0, 826: 0, 827: 0, 828: 0, 829: 0, 830: 0, 831: 0, 832: 0, 833: 0,
834: 0, 835: 0, 836: 0, 837: 0, 838: 0, 839: 0, 840: 0, 841: 0, 842: 0, 843: 0, 844: 0, 845: 0, 846: 0, 847: 0, 848: 0, 849: 0, 850: 0, 851: 0, 852: 0, 853: 0,
854: 0, 855: 0, 856: 0, 857: 0, 858: 0, 859: 0, 860: 0, 861: 0, 862: 0, 863: 0, 864: 0, 865: 0, 866: 0, 867: 0, 868: 0, 869: 0, 870: 0, 871: 0, 872: 0, 873: 0,
874: 0, 875: 0, 876: 0, 877: 0, 878: 0, 879: 0, 880: 0, 881: 0, 882: 0, 883: 0, 884: 0, 885: 0, 886: 0, 887: 0, 888: 0, 889: 0, 890: 0, 891: 0, 892: 0, 893: 0,
894: 0, 895: 0, 896: 0, 897: 0, 898: 0, 899: 0, 900: 0, 901: 0, 902: 0, 903: 0, 904: 0, 905: 0, 906: 0, 907: 0, 908: 0, 909: 0, 910: 0, 911: 0, 912: 0, 913: 0,
914: 0, 915: 0, 916: 0, 917: 0, 918: 0, 919: 0, 920: 0, 921: 0, 922: 0, 923: 0, 924: 0, 925: 0, 926: 0, 927: 0, 928: 0, 929: 0, 930: 0, 931: 0, 932: 0, 933: 0,
934: 0, 935: 0, 936: 0, 937: 0, 938: 0, 939: 0, 940: 0, 941: 0, 942: 0, 943: 0, 944: 0, 945: 0, 946: 0, 947: 0, 948: 0, 949: 0, 950: 0, 951: 0, 952: 0, 953: 0,
954: 0, 955: 0, 956: 0, 957: 0, 958: 0, 959: 0, 960: 0, 961: 0, 962: 0, 963: 0, 964: 0, 965: 0, 966: 0, 967: 0, 968: 0, 969: 0, 970: 0, 971: 0, 972: 0, 973: 0,
974: 0, 975: 0, 976: 0, 977: 0, 978: 0, 979: 0, 980: 0, 981: 0, 982: 0, 983: 0, 984: 0, 985: 0, 986: 0, 987: 0, 988: 0, 989: 0, 990: 0, 991: 0, 992: 0, 993: 0,
994: 0, 995: 0, 996: 0, 997: 0, 998: 0, 999: 0, 1000: 0, 1001: 0, 1002: 0, 1003: 0, 1004: 0, 1005: 0, 1006: 0, 1007: 0, 1008: 0, 1009: 0, 1010: 0, 1011: 0,
1012: 0, 1013: 0, 1014: 0, 1015: 0, 1016: 0, 1017: 0, 1018: 0, 1019: 0, 1020: 0, 1021: 0, 1022: 0, 1023: 0}
```

Host to send to: 127.0.0.1 and port to send to: 4040 <- we set these in the Sender so the Datagram Socket can bind to the local host on port 4040.

Figure 25. Example Block Encoding for the espdu.xml File Showing FEC Parameters of k=16, n=32 and a Packet Size of 1024 Bytes and Padding with Zeros to Maintain Parameter Matrix Size *n x k*

As can be plainly seen, the FECParameters set are: (k=16,n=32,packetSize=1024,fileSize=369). The espdu.xml was serialized (XSBC) and then compressed further with GZip. The resulting file size is 369B. When invoking the FECFile with these parameters, block partitioning of the file is determined based on these parameters by taking the file's size as a numerator and dividing by:

(K * PACKET_SIZE).

So, in this case:

$$\begin{aligned} BLOCKCOUNT &= 369/(16*1024) \\ &= 0.02252197265 \end{aligned}$$

and the size of the block is (K * PACKET_SIZE).

This number is rounded up to an integer value, so the block size for the 369 byte espdu.xsbc.gz file is 1. Inside this block will be 16 byte arrays each 1024 bytes in size. These contain the FEC encoding values (symbols) and are designated repair packets. The packets are padded with zeros to maintain *GF* matrix of size *n x k* if there is no further encoding to be done on that particular packet. These zeros are parsed for and discarded during the decoding process.

Using the Vandermonde-based Erasure Correction Codes that this FEC API is based upon, the MOVESLogo.jpg was encoded into a set of 32 original data packets per block which were placed in a set of *n* - *k* encoded packets such that any *k* of the *n* encoded packets is sufficient to reproduce the original data (Chapweske 2000). FEC codes do not care which form the data is in (Hamming 1995), therefore, the author concludes that it is correct to encode a combined .xsbc and binarily compressed file *xsbc.bz before sending it on its way.

E. EXAMPLE OF FEC ENCODING AN XML FILE COMPRESSED WITH XSBC

1. XSBC_FECFileTest.java

```java
/* Program:      XML Schema Binary Compression (XSBC) files encoded with Forward
 *               Error Correction (FEC) coding
 *
 * Author:       LT Terry D. Norbraten, USN
 * Modifier:
 *
 * Created on:   July 11, 2004, 1030
 * Modified on:  November 12, 2004, 2358
 *
 * Course:       MV 0810 (Thesis Research)
 *               Summer 2004
 *
 * File:         XSBC_FECFileTest.java
 *
 * Compiler:     netBeans IDE 3.6 (External), J2SDK 1.4.2_04
 * O/S:          Windows XP Home Ed. (SP 1)
 *
 * Description:  First attempt at encoding an .xsbc file with fec followed by
 *               decoding, then finally uncompressing to observe the magic of it
 *               all.
 *
 * Information:  Using fec-1.0.3 created by Justin F. Chapweske of Onion
 *               Networks and xsbc-0.91.1 created by Alan Hudson of Yumetech
 *               Inc.  Both are Java open source APIs.  Invoke this test after
 *               invoking SimpleExample or a FileNotFoundException will occur.
 */

package xsbc_fec;

// External imports (tdn)
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.io.IOException;

import java.security.DigestInputStream;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

// Local imports (tdn)
import com.onionnetworks.fec.io.FECFile;
import com.onionnetworks.fec.io.FECParameters;

import com.onionnetworks.util.BlockDigestInputStream;
import com.onionnetworks.util.Buffer;
import com.onionnetworks.util.FileIntegrity;
import com.onionnetworks.util.FileIntegrityImpl;
import com.onionnetworks.util.Util;

import org.apache.log4j.BasicConfigurator;

/**
 * Code heavily borrowed from the FECFileTest.java of the fec-1.0.3 API. Encodes
 * an .xsbc file and decodes checking file integrity to show that it is indeed
 * an original reproduction.
 *
 * @author  LT Terry D. Norbraten, USN
 */
public class XSBC_FECFileTest {
```

```
/* DATA MEMBER(s) */

/* Public */

/* Private */

/** Used for file integrity checks */
private BlockDigestInputStream bdis;

/** The buffer array used for FEC decoding */
private Buffer[] repairBuffer;

/** Used for file integrity checks */
private DigestInputStream dis;

/** Our (en/de) pair */
private FECFile server,
                client;

/** Our FEC code parameters */
private FECParameters params;

/** A check on our decoding process */
private FileIntegrity integrity;

/** In steam for file encoding */
private InputStream is;

/** Used for file integrity checks */
private MessageDigest md;

/** File base to apply .xsbc extention to */
private String fileBase;

/** File name to substring into a file base */
private String fileName;

/** Designate a message digest algorithm */
private final String DIGEST_ALGORITHM = "sha";

/** Name our test file with .xsbc extenstion */
private final static String TEST_FILE = "espdu.xsbc";

/** String pointer to my user directory */
private final static String USER_DIR = "C:\\Documents and Settings\\Terry\\"
+ "My Documents\\My Files\\NPS\\Courses\\MV 0810 Thesis Work\\Java Code\\" +
"XSBC_FEC\\xsbc_fec\\examples\\";

/** Our stripe order array N */
private int[] stripeOrder;

/** The random indices of our stripe orders K */
private int[] indexes;

/** Designate 3 out of 4 FEC parameters (tdn) */
private final int K = 16,
                  N = 32,
                  PACKET_SIZE = 1024;

/* CONSTRUCTOR(s) */

/** Creates and new instance of XSBC_FECFileTest */
public XSBC_FECFileTest() {

   md = null;
   dis = null;
   bdis = null;
   stripeOrder = new int[ N ];
   repairBuffer = new Buffer[ K ];
   indexes = new int[ K ];
   // This function, which was originally commented out, produces a DEBUG
   // output that shows Block Counts and the decoding process.  If commented
   // out, it will throw a couple of log4j error about not being initialized
   // properly (tdn)
   BasicConfigurator.configure();

} // end constructor

/* MAIN METHOD */

/**
 * @param args the command line arguments if any (tdn)
 *
 * @exception Exception thrown if (en/de) failed
 */
public static void main( String[] args ) {

   try {

      // Moved everything that was in this main to fileEncode_Decode() so
      // that I could call it from the XSBC ComparisonTool GUI, or run this
      // class as an instantiable main class (tdn)
      new XSBC_FECFileTest().fileEncode_Decode( new File( USER_DIR +
      TEST_FILE ) );

   } catch ( Exception e ) {

      System.out.println( e );

   } // end try-catch block

} // end main()

/**
 * Encodes an .xsbc file, then decodes into another file with a "Decoded"
 * prefix.
 *
 * @param file the .xml file to (en/de)
 *
 * @exception NoSuchAlgorithmException
 */
public void fileEncode_Decode( File file ) throws Exception {

   // Get our file name in String form
   fileName = file.getName();
   System.out.println( "File name: " + fileName ); // DEBUG

   // Cut off the .xml extension and create a new file with an .xsbc
   // extension as the xsbc version of the .xml (tdn)
   fileBase = fileName.substring( 0, fileName.lastIndexOf( "." ) );
   System.out.println( "File base: " + fileBase ); // DEBUG

   System.out.println( "Computing checksums on: " + fileBase
                     + "Decoded.xsbc" );

   // Read in the file
   is = new FileInputStream( file );

   // Build our integrity checker
   try {

      md = MessageDigest.getInstance( DIGEST_ALGORITHM );
      System.out.println( "The message digest object: " + md ); // DEBUG
      dis = new DigestInputStream( is, md );
      System.out.println( "The digest stream: " + dis ); // DEBUG
      bdis = new BlockDigestInputStream( dis, DIGEST_ALGORITHM,
                                         K * PACKET_SIZE );

   } catch ( NoSuchAlgorithmException e ) {

      throw new IOException( e.getMessage() );
      } // end try-catch block

      // The length 8192 of this byte array has something to do with the
      // encMartix size in the code's FEC math functions.  This is a character
      // array of size K * N.  This new byte[] will take orig. file information
      // for the File integrity check fucntion of FECFile writting in rw mode
      // (tdn)
      byte[] b = new byte[ 8192 ];
      long fileLength = 0;
      int c = 0;

      // Read in the data and digest it.
      while ( ( c = bdis.read( b ) ) != -1 ) {

         fileLength += c;

      } // end while

      bdis.finish();
      // Finish must be called first to avoid throwing a
      // java.lang.IllegalStateException (tdn)
      System.out.println( "The BDIS object: " +
                          bdis.getBlockDigests()[ 0 ] + "\n" ); // DEBUG

      // Create our parameters
      params = new FECParameters( K, N, PACKET_SIZE, fileLength );

      System.out.println( params ); // DEBUG

      // Finalize the integrity table
      integrity = new FileIntegrityImpl( DIGEST_ALGORITHM, new Buffer(
      md.digest() ), bdis.getBlockDigests(), params.getFileSize(),
      params.getUnexpandedBlockSize() );

      System.out.println( "The File Integrity hash computation for file: " +
                          file + "\nis " + integrity.getFileHash() ); // DEBUG

      System.out.println( "Unexpanded Block Size: "
                          + params.getUnexpandedBlockSize() ); // DEBUG

      System.out.println( "checksums done doing our thang." );

      // Only need to do this once.  For loop is for testing purposes only
      for ( int i = 0; i < 1; i++ ) { // orig. iteration was 100 (tdn)

         // Encode the file in read mode only
         server = new FECFile( file, "r", params );

         // TODO: Must be able to show a file that has been encoded with FEC
         // as this file is what would be sent over the net.  Must also show,
         // by implementing some kind of random noise generator that would
         // drop packets, that the encoded code will decode to it's orig. state
         // and then my work will be complete (tdn)

         // Attempting with direct pointing to a file with a name and
         // directory I select.  Decode the file in read/write mode and check
         // for integrity against the original file (tdn)
         client = new FECFile( new File( USER_DIR + "\\" + fileBase +
                               "Decoded.xsbc" ), "rw", params, integrity );

         // Perform the (en/de) function
         doIt( client, server, params );

//         System.out.println( i ); // Num interations for DEBUG/File Integrity
                                    // checks

      } // end for

      System.out.println( "All Good!" );

   } // end fileEncode_Decode()
```

```
/**
 * This is the necessary to invoke the principle (en/de) function
 *
 * @param client the FECFile to decode from packet block/byte form back into
 * a file
 * @param server the FECFile to encode into a block/byte form packets
 * @param params the FEC code parameters used for (en/de)
 */
private void doIt( FECFile client, FECFile server, FECParameters params )
                   throws Exception {

   // Create N of these
   for ( int i = 0; i < stripeOrder.length; i++ ) {

      stripeOrder[ i ] = i;

   } // end for

   // K of these Buffer wrappers each which will hold byte[] of size 1024 B
   // which hold the original source packets (tdn)
   for ( int i = 0;  i < K; i++ ) {

      repairBuffer[ i ] = new Buffer( PACKET_SIZE );

   } // end for

   for ( int i = 0; i < params.getBlockCount(); i++ ) {

      // Shuffle the order of the (N) indices using a Fisher-Yates shuffle
      // algorithm (num blocks) times (tdn)
      Util.shuffle( stripeOrder );

      // Copy the order a (K) subset of N stripe indices into index array
      // (tdn)
      System.arraycopy( stripeOrder, 0, indexes, 0, K );

      // Reads K packets (encoding them if necessary) into the  provided
      // buffers (block num) times (tdn)
      server.read( repairBuffer, i, indexes );
      int l = 0;

      System.out.println( "Num of packets required to send across the "
                        + "original block: " + i + " = " +
                        + params.getUnexpandedPacketCount( i ) ); // DEBUG

      for ( int j = 0; l < params.getUnexpandedPacketCount( i ); j++ ) {

         // Prevent writing of decoded packets padded with zeros
         if ( !params.isPaddingPacket( i, indexes[ j ] ) ) {

            // This decodes the packets and write the orig. file back to
            // disc
            client.write( repairBuffer[ j ], i, indexes[ j ] );

            l++;

         } // end if

      } // end inner for

   } // end outer for

   // This halts threads until the file is written to disc
   client.waitForFileDecoded();

   System.out.println( "Num of packets written to disc: "
                     + client.getWrittenCount() ); // DEBUG

   server.close();
   client.close();
   } // end doIt()

} // end class file XSBC_FECFileTest.java
```

F. EXAMPLE FEC ENCODING THE MOVESLOGO.JPG FILE

1. FECFileTest.java

```
/* Program:      XML Schema Binary Compression (XSBC) files encoded with Forward
 *               Error Correction (FEC) coding
 *
 * Author:       Justin F. Chapweske
 * Modifier:     LT Terry D. Norbraten, USN
 *
 * Created on:   July 11, 2004, 1030
 * Modified on:  November 12, 2004, 2358
 *
 * Course:       MV 0810 (Thesis Research)
 *               Summer 2004
 *
 * File:         FECFileTest.java
 *
 * Compiler:     netBeans IDE 3.6 (External), J2SDK 1.4.2_04
 * O/S:          Windows XP Home Ed. (SP 1)
 *
 * Description:  Test file taking a rather large .jpg file, encoding with FEC
 *               which takes the file, breaks it up into encoding blocks
 *               according to the specified FEC parameters, checks the file
 *               integrity utilizing a message digest "sha" algorithm and
 *               reproducing the original file as a decoded copy.
 *
 * Information:  Using fec-1.0.3 created by Justin F. Chapweske of Onion
 *               Networks Inc., an Java open source API.
 */
package xsbc_fec.testFiles;

// Standard library imports
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.io.IOException;

import java.net.URI;

import java.security.*;

// Specific local imports
import com.onionnetworks.fec.io.*;
import com.onionnetworks.util.*;
import org.apache.log4j.BasicConfigurator;

/*
 * Test file taking a rather large .jpg file, encoding with FEC
 * which takes the file, breaks it up into encoding blocks
 * according to the specified FEC parameters, checks the file
 * integrity utilizing a message digest "sha" algorithm and
 * reproducing the original file as a decoded copy.
 *
 * @author Justin F. Chapweske
 * <p> Modified by LT Terry D. Norbraten, USN </p>
 */
public class FECFileTest {
```

```
/* DATA MEMBERS */

public static final int K = 16; // tdn
public static final int N = 32; // tdn
public static final int PACKET_SIZE = 1024; // tdn
public static final String DIGEST_ALGORITHM = "sha";

/* MAIN METHOD */

public static void main( String[] args ) throws Exception {

   // This function, which was originally commented out, produces a DEBUG
   // output that shows Block Counts and the decoding process.  If commented
   // out, it will throw a couple of log4j error about not being initialized
   // properly (tdn)
   BasicConfigurator.configure();

   // Attempting with pseudofiles (tdn)
   File sourceFile = new File( "C:/Documents and Settings/" +
      "Terry/My Documents/My Files/NPS/MOVESLogo.jpg" );

   System.out.println("Computing checksums");
   InputStream is = new FileInputStream(sourceFile);

   MessageDigest md = null;
   DigestInputStream dis = null;
   BlockDigestInputStream bdis = null;

   try {
      md = MessageDigest.getInstance(DIGEST_ALGORITHM);
      dis = new DigestInputStream(is,md);
      bdis = new BlockDigestInputStream(dis,DIGEST_ALGORITHM,
      K * PACKET_SIZE );
   } catch (NoSuchAlgorithmException e) {
      throw new IOException(e.getMessage());

   } // end try-catch block

   // The length 8192 of this byte array has something to do with the
   // encMartix size in the code's FEC math functions.  This is a character
   // array of size K * N.  This new byte[] will take orig. file information
   // and for the File integrity check (tdn)
   byte[] b = new byte[8192];
   long fileLength = 0;
   int c = 0;
   // Read in the data and digest it.
   while ((c = bdis.read(b)) != -1) {
      fileLength += c;

   } // end while

   bdis.finish();

   FECParameters params = new FECParameters( K, N, PACKET_SIZE, fileLength );
   FileIntegrity integrity = new FileIntegrityImpl
   ( DIGEST_ALGORITHM, new Buffer( md.digest() ), bdis.getBlockDigests(),
   params.getFileSize(), params.getUnexpandedBlockSize() );

   System.out.println("checksums done doing our thang.");

   for ( int i = 0; i < 1; i++ ) { // orig. 100 iterations (tdn)

      FECFile server = new FECFile( sourceFile, "r", params );

      // Attemping with direct pointing to a file with a name and
      // directory I select (tdn)
      FECFile client = new FECFile( new File( "C:/Documents and Settings/" +
      "Terry/My Documents/My Files/NPS/Courses/MV 0810 Thesis Work/" +
      "Java Code/XSBC_FEC/xsbc_fec/examples/test.jpg" ), "rw", params,
      integrity );
```

```
         doIt(client,server,params);

      } // end for

      System.out.println("All Good!");
      System.exit(0);

   } // end main()

   /*
    * Encodes and check file integrity to ensure that the decoded copy file is
    * a computed hash verification of the original during the FEC decoding
    * process
    */
   public static void doIt(FECFile client, FECFile server,
                           FECParameters params) throws Exception {

      // So, 256 of these (N) (tdn)
      int[] stripeOrder = new int[ N ];
      for (int i=0;i<stripeOrder.length;i++) {
         stripeOrder[i] = i;

      } // end for

      // K of these Buffer wrappers each which will hold byte[] of size 1024 B
      // which hold the original source packets (tdn)
      Buffer[] bufs = new Buffer[ K ]; // orig K
      for (int i=0;i<K;i++) { // orig. K
         bufs[i] = new Buffer( PACKET_SIZE );

      } // end for

//      System.out.println( params + ", File partition block count: "
//                          + params.getBlockCount()+ ", \nHash code count: "
//                          + params.hashCode() + ", Max stripe size: "
//                          + params.getMaxStripeSize() + " B,"
//                          + " \nDefault number of vanilla bytes in a block: "
//                          + params.getUnexpandedBlockSize() + " B," +
//                          "\nMax num of bytes that a fully encoded block"
//                          + " can contain: " + params.getExpandedBlockSize()
//                          + " B," + "\nMax num stripes (N) that can be created "
//                          + "from this file: " + params.getNumStripes() +
//                          ", \nMin num packets required to recreate orig. file:"
//                          + " " + params.getUnexpandedPacketCount() ); // DEBUG

      for ( int i = 0; i < params.getBlockCount(); i++ ) {

         // Shuffle the order of the (N) indicies using a Fisher-Yates shuffle
         // algorithm (num blocks) times (tdn)
         Util.shuffle(stripeOrder); // <- this is a curious function (tdn)

         // New array of length (N) (tdn)
         int[] indexes = new int[ K ]; // orig. K

         // Copy the order a (K) subset of N stripe indices into index array
         // (tdn)
         System.arraycopy( stripeOrder, 0, indexes, 0, K );

         // Reads N packets (encoding them if necessary) into the  provided
         // buffers (block num) times (tdn)
         server.read( bufs, i, indexes );

         System.out.println( "Packet size for block: " + i + " " + "= " +
         params.getPacketSize( i, indexes[ i ] ) + " B" ); // DEBUG

         System.out.println( "Num of packets required to send across the "
                             + "original block: " + i + " = " +
                             + params.getUnexpandedPacketCount( i ) ); // DEBUG

         int l=0;
         for (int j=0;l<params.getUnexpandedPacketCount(i);j++) {
            // Prevent writing of decoded packets padded with zeros
            if ( !params.isPaddingPacket( i, indexes[ j ] ) ) {

               // This decodes the packets and writes the orig. file back to
               // disc
               client.write(bufs[j],i,indexes[j]);
               l++;

            } // end if

         } // end inner for

      } // end outer for

      // This halts threads until the file is written to disc
      client.waitForFileDecoded();

      int numPackets = client.getWrittenCount();

      System.out.println( "Num of packets written to disc: "
                        + numPackets ); // DEBUG

      server.close();
      client.close();

   } // end doIt()

} // end class file FECFileTest.java
```

G. EXAMPLE SERVER/CLIENT ENCODING A FILE WITH FEC, SENDING, RECEIVING AND DECODING WITH A FILE COMPARISON FEATURE

1. FECSendN.java

```
/* Program:      Test UDP sending of Forward Error Correction (FEC) coded data
 *
 * Author:       LT Terry D. Norbraten, USN
 * Modifier:
 *
 * Created on:   August 29, 2004, 1903
 * Modified on:  November 13 2004, 1657
 *
 * Course:       MV 0810 (Thesis Research)
 *               Summer 2004
 *
 * File:         FECSendN.java
 *
 * Compiler:     netBeans IDE 3.6 (External), J2SDK 1.4.2_04
 * O/S:          Windows XP Home Ed (SP 1).
 *
 * Description:  Attempt at encoding a file with FEC and sending it via UDP to
 *               a client decoder by sending all N repair packet and selecting
 *               only K for data reconstruction.
 *
 * Information:  Using fec-1.0.3 created by Justin F. Chapweske of Onion
 *               Networks which is a Java-based open source API.
 */

package xsbc_fec.testFiles;

// Standard library imports
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;
import java.net.UnknownHostException;
import java.util.Arrays;
import java.util.Random;

// Application specific local imports
import com.onionnetworks.fec.FECCode;
import com.onionnetworks.fec.FECCodeFactory;

import com.onionnetworks.util.Buffer;
import com.onionnetworks.util.Util;

/**
 * Attempt at encoding a file with FEC and sending it via UDP to a client
 * decoder by sending all N repair packet and selecting only K for data
 * reconstruction.
 *
 * @author <a href="mailto:tdnorbra@nps.navy.mil">Terry D. Norbraten</a>
 */
public class FECSendN {

   /* DATA MEMBER(s) */

   /* Public */

   /**
    * This will be our source file that will be compared to the received and
    * decoded file in the main().
    */
   public static byte[] source;

   /* Private */

   /** Will contain our FEC encoded data for transmission */
   private byte[] repair;

   /** Local port designation */
   private final int DEST_PORT = 4040;

   /** FEC parameters */
   private static final int K = 16,
                            N = 32,
                            PACKET_SIZE = 1024;

   /** Index in which a packet was FEC encoded */
   private int[] repairIndex;

   /* Stream for the FEC index information */
   private ByteArrayOutputStream baos;

   /** Wrappers for our source and repair byte[] required for FEC process */
   private Buffer[] sourceBuffer,
                    repairBuffer;

   /* UDP utility */
   private DatagramPacket packet;

   /* UDP channel sender */
   private DatagramSocket socket;

   /** Our FEC code */
   private FECCode fec;

   /** Source file data generator */
   private Random rand;

   /** String name of this local host */
   private final String LOCAL_HOST = "127.0.0.1";
```

```
/* CONSTRUCTOR(s) */

/** Creates a new instance of FECSendN */
public FECSendN() {
   rand = new Random();
   source = new byte[K*PACKET_SIZE];

   // NOTE: The source needs to split into k*packetsize sections, so if your
   // file is not of the write size you need to split it into groups.  The
   // final group may be less than k*packetsize, in which case you must pad
   // it until you read k*packetsize.  Also, send the length of the file so
   // that you know where to cut it once decoded.
   // this is just so we have something to encode
   rand.nextBytes(source);

   //this will hold the encoded file
   repair = new byte[N*PACKET_SIZE];

   // These buffers allow us to put our data in them.  They reference a
   // packet length of the file (or at least will once we fill them)
   sourceBuffer = new Buffer[K];
   repairBuffer = new Buffer[N];

   // These Buffers each contain the whole file they are wrapping, but at
   // different offsets designated by PACKET_SIZE, as each is holding either
   // the unexpanded file size K, or, the expanded file size N (tdn)
   for( int i = 0; i < sourceBuffer.length; i++ )
      sourceBuffer[i] = new Buffer( source, i*PACKET_SIZE, PACKET_SIZE );

   for( int i = 0; i < repairBuffer.length; i++ )
      repairBuffer[i] = new Buffer( repair, i*PACKET_SIZE, PACKET_SIZE );

   //When sending the data you must identify what it's index was when
   //encoded.  Will be shown and explained later.
   repairIndex = new int[N];

   for( int i = 0; i < repairIndex.length; i++ )
      repairIndex[i] = i;

   // We just need any K subset of the orig. N expanded repair packets.  This
   // will shuffle the order of the indicies which will be used to index each
   // repair packet as it's encoded by the FEC process.  This shuffled order
   // will be sent with the encoded repair packets to prove that any K subset
   // of the N expanded repair packets is what's needed to reconstruct the
   // orig. data.
   Util.shuffle( repairIndex );

   //create our fec code
   fec = FECCodeFactory.getDefault().createFECCode(K,N);

   //encode the data
   fec.encode( sourceBuffer, repairBuffer, repairIndex );
   //encoded data is now contained in the repairBuffer's repair byte array

   //From here you can send each 'packet' of the encoded data, along with
   //what repairIndex it was encoded with.  Also include the group number if
   //you had to split the file
   sendFECEncoded();

} // end constructor

/* MAIN METHOD */

/**
 * @param args the command line arguments
 */
public static void main( String[] args ) {

   // Start the receiver as a thread of this class
   FECReceiveN fecrn = new FECReceiveN();
   Thread t = new Thread( fecrn );
```

```
      t.start();

      // Invoke this sender
      new FECSendN();

      // Allow the receiver to process
      try {
         Thread.sleep( 1000 );
      } catch ( InterruptedException ie ) {}

      //check for equality
//      for ( int ix = 0; ix < K * PACKET_SIZE; ix++) {
//         System.out.println( "Source file contents at " + ix + " is: " +
//                             sourceFile[ ix ] );
//         System.out.println( "Decoded file contents at " + ix + " is: " +
//                             fecrn.receivedK[ ix ] );
//      } // end for DEBUG

      if ( Arrays.equals( source, fecrn.receivedK ) ) {

         System.out.println( "Source and decoded files are equal!" );
         System.exit( 0 );

      } else {

         System.out.println( "Opps!  Something went wrong." );
         System.exit( 1 );

      } // end if-else block

   } // end main()

   /* PUBLIC METHOD(s) */

   /* GETTER(s) and SETTER(s) */

   /* PRIVATE METHOD(s) */

   /**
    * Sends only FEC encoded packets down stream time along with the repair
    * packet index of each repair packet encoding order.
    *
    * @exception SocketException if the Datagram Socket didn't open
    * @exception UnknownHostException if host couldn't be determined
    * @exception IOException if the socket couldn't send the packet
    */
   private void sendFECEncoded() {

      // Send repair packet indicie info first
      try {

         // Open our UDP socket
         socket  = new DatagramSocket();

         // Prepare a BAOS for sending repair packet indice info.
         baos = new ByteArrayOutputStream();

      } catch ( SocketException se ) {

         se.printStackTrace();

      } catch ( IOException ioe ) {

         ioe.printStackTrace();

      } // end try-catch block

      // Write the repair index information to a buffer
      for ( int i = 0; i < N; i++ ) {
         // Write an N amount of the shuffled indices to the stream
         baos.write( repairIndex[ i ] );

      } // end for

      try {
         // Flush the bytes out of the buffer into the stream
         baos.flush();

      } catch ( IOException ioe ) {

         ioe.printStackTrace();

      } // end try-catch block

      try {

         // Prepare the packet to send indicie info.
         packet = new DatagramPacket( baos.toByteArray(),
         baos.toByteArray().length, InetAddress.getByName( LOCAL_HOST ),
         DEST_PORT );

      } catch ( UnknownHostException uhe ) {

         uhe.printStackTrace();

      } // end try-catch block

      // Send the N repair index Datagram Packets first
      try {

         // Send repair packet indicie info.
         socket.send( packet );

//          for ( int ix = 0; ix < repairIndex.length; ix++ )
//             System.out.println( "Sending FEC encoded data packet index: " +
//             repairIndex[ ix ] + " in packet size: "
//             + baos.toByteArray().length ); // DEBUG

      } catch ( IOException ioe ) {

         ioe.printStackTrace();

      } // end try-catch block

      try {

         baos.close();

      } catch ( IOException ioe ) {

         ioe.printStackTrace();

      } // end try-catch block

      /****************************** FEC Sending ******************************/

      // Send N repair packets
      for ( int i = 0; i < N; i++ ) {

         try {

            // Load the packet with encoded data of just repair packet size and
            // send each to keep under the 1500 byte MTU as specified by NPS
            // research associate Don McGregor.  I know I'm making new Datagram
            // packets with each iteration of the for loop, but with the FEC
            // Buffer[], this is the only way to extract encoded data into byte
            // array form for the Datagram Packet for each repair packet that
            // is to be transmitted.  So here, we send N packets of encoded data
            // per block.
            packet = new DatagramPacket( repairBuffer[ i ].getBytes(),
            PACKET_SIZE, InetAddress.getByName( LOCAL_HOST ),
            DEST_PORT ); // tdn

         } catch ( UnknownHostException uhe ) {

            uhe.printStackTrace();
         } // end try-catch block

//         System.out.println( "Sending FEC encoded data packet " + i + " " +
//                             repairBuffer[ i ] ); // DEBUG

         // Send the repair encoded packets
         try {

            // Send encoded data
            socket.send( packet );

//            System.out.println( "Successfully sent encoded repair packet " + i +
//            " of size: " + packet.getLength() + " B to: " +
//            packet.getAddress().toString() + " on port: " +
//            packet.getPort() ); // DEBUG

         } catch ( IOException ioe ) {

            ioe.printStackTrace();

         } // end try-catch block

      } // end for

      // Close the socket
      socket.close();

   } // end sendFECEncoded()

} // end class file FECSendN.java
```

2. FECReceiveN.java

```
/* Program:      Test UDP receiving of Forward Error Correction (FEC) coded data
 *
 * Author:       LT Terry D. Norbraten, USN
 * Modifier:
 *
 * Created on:   August 29, 2004, 1908
 * Modified on:  November 13 2004, 1657
 *
 * Course:       MV 0810 (Thesis Research)
 *               Summer 2004
 *
 * File:         FECRecieveN.java
 *
 * Compiler:     netBeans IDE 3.6 (External), J2SDK 1.4.2_04
 * O/S:          Windows XP Home Ed (SP 1).
 *
 * Description:  Attempt at decoding an FEC file after receiving it via UDP from
 *               an encoding server upon reception of all N encoded repair
 *               packets.  K repair packets will be selected for the file
 *               reproduction.
 *
 * Information:  Using fec-1.0.3 created by Justin F. Chapweske of Onion
 *               Networks Inc. which is a Java-based open source API.
 */

package xsbc_fec.testFiles;
```

```
// Standard library imports
import java.io.ByteArrayInputStream;
import java.io.IOException;

import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;
import java.net.UnknownHostException;

// Application specific local imports
import com.onionnetworks.fec.FECCode;
import com.onionnetworks.fec.FECCodeFactory;

import com.onionnetworks.util.Buffer;

/**
 * Attempt at decoding an FEC file after receiving it via UDP from an encoding
 * server upon reception of all N encoded repair packets.  K repair packets will
 * be selected for the file reproduction.
 *
 * @author <a href="mailto:tdnorbra@nps.navy.mil">Terry D. Norbraten</a>
 */
public class FECReceiveN implements Runnable {

   /* DATA MEMBER(s) */

   /* PUBLIC */

   /** Will be accessed by the sender for comparison with the source file */
   public byte[] receivedK;

   /* PRIVATE */

   /** Storage of our N received repair packets */
   private byte[] receivedN;

   /** Local port designation */
   private final int RECEIVE_PORT = 4040;

   /** FEC parameters */
   private static final int K = 16,
                            N = 32,
                            PACKET_SIZE = 1024;

   /** Encoding index ordering */
   private int[] receiverIndex,
                 repairIndexes;

   private ByteArrayInputStream bais;

   /** FEC wrappers used for decoding */
   private Buffer[] repairBuffer;

   /* UDP utilities */
   private DatagramPacket indexPacket,
                          fecPacket;

   /* UDP channel receiver */
   private DatagramSocket socket;

   /** FEC codes used for decoding */
   private FECCode fec;

   /** String name of this local host */
   private final String LOCAL_HOST = "127.0.0.1";

   /* CONSTRUCTOR(s) */

   /** Creates a new instance of FECReceiveN */
   public FECReceiveN() {
      try {
         socket = new DatagramSocket(RECEIVE_PORT);

      } catch ( SocketException se ) {
         se.printStackTrace();

      } // end try-catch block

   } // end constructor

   /* PUBLIC METHOD(s) */

   /* THREAD */

   /** Ran as a thread from the server */
   public void run() {

      doReceiveFECEncoded();

   } // end run

   /* GETTER(s) and SETTER(s) */

   /* PRIVATE METHOD(s) */

   /**
    * Receives only FEC encoded packets along with the repair packet index of
    * each repair packet encoding order.
    *
    * @exception SocketException if the Datagram Socket didn't open
    * @exception UnknownHostException if host couldn't be determined
    * @exception IOException if the socket couldn't send the packet
    */
   private void doReceiveFECEncoded() {

      // Prepare to receive our K repair indicies.  These are just byte format
      // integers for each Datagram packet.  Again, we will require a K subset
      // from the N generated by the encoder.
      indexPacket = new DatagramPacket( new byte[ N ], N );

      //We only need to store k, packets received, but we'll receive N and
      // choose K
      receiverIndex = new int[ N ];

      try {

         // Receive our repair index info.
         socket.receive( indexPacket );

      } catch ( IOException ioe ) {

         ioe.printStackTrace();

      } // end try-catch block

      // Feed index information into this BAIS per block
      bais = new ByteArrayInputStream( indexPacket.getData() );

      // Storage for each block's K subset of N repair indexes that will
      // be copied from receiveIndexes.
      repairIndexes = new int[ K ];

      // Extract K packet elements that make up our repair indices.
      for ( int ix = 0; ix < indexPacket.getLength(); ix++ ) {

         // Collect our repair packet indicies
         receiverIndex[ ix ] = bais.read();

//         System.out.println( "Received repair packet index: " +
//         receiverIndex[ ix ] ); // DEBUG
```

```
      } // end for

      // Extract and store each K subset of N repair packet indexes.
      System.arraycopy( receiverIndex, 0, repairIndexes, 0, K );

//      for ( int ix = 0; ix < K ; ix++ )
//         System.out.println( "Array copied repair index " + ix + ": " +
//         repairIndexes[ ix ] ); // DEBUG

      /**************************** FEC Receiving ***************************/

      // Prepare this packet for reception of an encoded repair packet each
      // containing 1024 bytes of data.
      fecPacket = new DatagramPacket( new byte[ PACKET_SIZE ], PACKET_SIZE );

      // Container for our entire Expanded encoded repair file size
      receivedN = new byte[ N * PACKET_SIZE ];

      // Receive each of N encoded repair packets for storage
      for ( int ix = 0 ;  ix < N; ix++ ) {

         // Receive a block of data to encode
         try {

            // Receive our encoded data
            socket.receive( fecPacket );

         } catch ( IOException ioe ) {

            ioe.printStackTrace();

         } // end try-catch block

         // Copy N repair packets of the encoded file
         System.arraycopy( fecPacket.getData(), 0, receivedN, ix *
         fecPacket.getLength(), fecPacket.getLength() );

      } // end for

      // Close the socket
      socket.close();

      //this will store the received packets to be decoded
      receivedK = new byte[K*PACKET_SIZE];

      // Extract and store a K subset of N repair packet data.
      System.arraycopy( receivedN, 0, receivedK, 0, K * PACKET_SIZE );

      // This will hold our K subset of encoded repair packets
      repairBuffer = new Buffer[ K ];

      //create our Buffers for the encoded data extracting only K encoded repair
      // packets
      for ( int ix = 0; ix < K; ix++ ) {

         // Prepare our decoding buffers with offsets and length
         repairBuffer[ ix ] = new Buffer( receivedK, ix *
         fecPacket.getLength(), fecPacket.getLength()  );

//         System.out.println( "Receive FEC encoded data packet " +
//         ix + " " + repairBuffer[ ix ] ); // DEBUG

      } // end for

      //create our fec code
      fec = FECCodeFactory.getDefault().createFECCode(K,N);
```

```
            // Finally, decode the repair packets into the file
            fec.decode( repairBuffer, repairIndexes );

         } // end doRecieveFECEncoded()

      } // end class file FECReceiveN.java
```

H. SUMMARY

The preceding files were given so as to represent, in a simplistic way, how to employ the Java FEC 1.0.3 library by Onion Networks, Inc. The API is not very well documented, so, one has to “play” with the API and experiment a bit with the mechanics of how to implement the features of this API. The next appendix will feature how this research put together FEC with XSBC so as to enable reliable UAN features that may employed by AUV/UUVs to communicate with surface based gateway servers.

APPENDIX C. FAMILIARITY NOTES FOR THE XSBC 0.91.1 LIBRARY

A. INTRODUCTION

The following paragraphs are presented by the author as a familiarity guide into the methods employed by the XSBC 0.91.1 library. These notes were written during trial and error attempts at working with this Java API. This was written with the intent of self-guidance and discovery, and thus it is hoped that the reader will have an understanding into the author's rationale in how to work with this remarkable library.

B. SETUP OF THE XSBC 0.91.1 LIBRARY

Download and mount the xsbc-0.91.1.jar file from http://sourceforge.net/projects/xmsf. The source code can be retrieved from the CVS link and the same page.

An xsbc.jar file can be created from the source code package by the Ant build process that will be explained later.

The reader must modify the path in each .xml file contained in the examples folder to reflect a user.dir on a local machine, for example: xsi:noNamespaceSchemaLocation="C:\(your path)\(your schema).xsd"

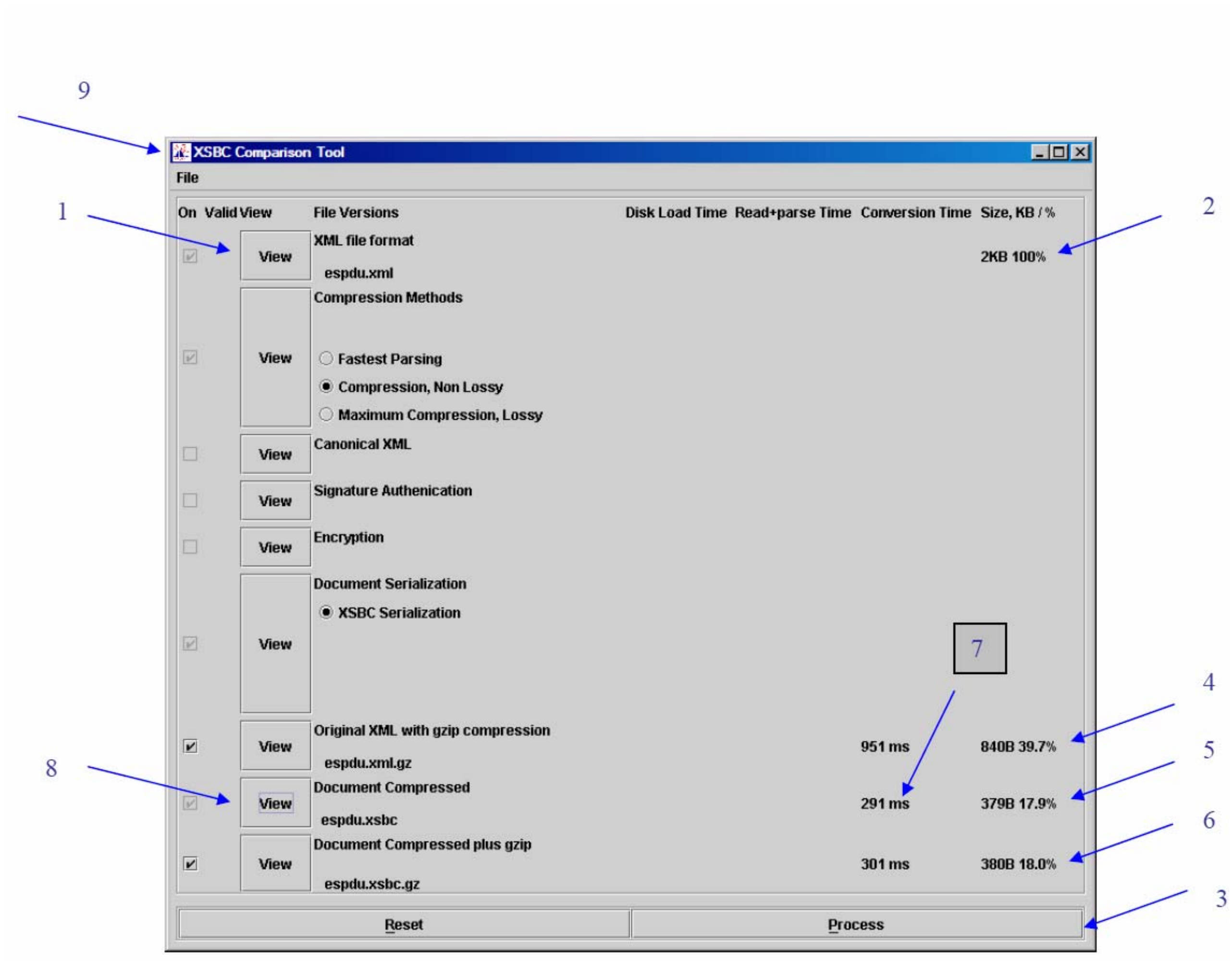
The ComparisonTool will not show gzipped statistics unless you download gzip. The best open source one the reader found is here: http://gnuwin32.sourceforge.net/packages/gzip.htm. To set an Environmental Variable, put GZIP_HOME = C:\Program Files\GnuWin32 and amended the PATH to %GZIP_HOME%\bin and now gzip invokes very well from the command line.

A word on setting the Environment Variables for Windows XP Home Ed. SP1. Set these variables from the System icon in the Control Panel, or right clicking on My Computer on your desktop (PC only) and select Properties, Advanced, Environment Variables you add (your application)_HOME =

%HOMEDRIVE%%HOMEDIR% and then PATH = %(your application)_HOME%\bin. If .exe, .cmd or .bat files are located elsewhere, then use \(folder). Separate entries by a ";" and never any whitespace between entries. If the directory path contains spaces then put the entire path in "", or, just the specific directory between "\"s.

The ComparisonTool was modified slightly to reflect GUI output that makes slightly more sense to (because the author needs to completely understand the slightest detail as much as humanly possible).

Once invoked, select File, Open, examples and click on an example .xml file. Select process and will observe the following:



On	Valid	View	File Versions	Disk Load Time	Read+parse Time	Conversion Time	Size, KB / %
✓		View	XML file format espdu.xml				2KB 100%
✓		View	Compression Methods ○ Fastest Parsing ● Compression, Non Lossy ○ Maximum Compression, Lossy				
☐		View	Canonical XML				
☐		View	Signature Authenication				
☐		View	Encryption				
✓		View	Document Serialization ● XSBC Serialization				
✓		View	Original XML with gzip compression espdu.xml.gz			951 ms	840B 39.7%
✓		View	Document Compressed espdu.xsbc			291 ms	379B 17.9%
✓		View	Document Compressed plus gzip espdu.xsbc.gz			301 ms	380B 18.0%

Figure 26.   Sample Process Output of the Comparison Tool GUI (ComarisonTool.java) included with the XSBC 0.91.1 Library

1) The name of the original document that was selected from the File menu pull-down. Selecting “View” will invoke an XML reader (XMLSpy2004 in this case) and load the file.

2) Observe the original size of the .xml document

3) Selecting Process will serialize the document with the XSBC algorithm which tokenizes attributes and elements thereby compressing the original .xml document, but preserving its tree as it is structured by its own schema.

4) This shows the original .xml binary compression size by just GZipping.

5) This shows the size of the file with XSBC invoked.

6) This shows the combined XSBC / GZipping compression size. In this particular case, GZipping actually made the .xsbc file grow slightly.

7) This shows the time it took to compress a file in either binary (GZip), XSBC, or both.

8) This feature will invoke the FEC encoding and decoding (en/de) utility creating a file of the original named .xml with a “Decoded” prefix, uncompress from the XSBC form and recreate the original .xml file (called foo.xml in this demonstration case) using its schema.

9) Shows the Moves Institute logo.

The reader can select between Fastest Parsing, Compression - Non Lossy or Maximum Compression - Lossy and observe the compression results as compared to the original file size. The reader can then select View / Document Compressed which will be decompressed and renamed into a generic foo.xml, placed in the examples folder and XMLSpy2004 will open for viewing that file.

There are four good working examples. ComparisonTool, SimpleExample, SenderSimulation and ReceiverSimulation. When compressing with the ComparisonTool these errors will show; however, it is not a bug:

Unknown attribute: /espdu/@xmlns:xsi

Unknown attribute: /espdu/@xsi:noNamespaceSchemaLocation

It's because these attributes are not to be reproduced by the Dom4jReader in the uncompressed/deserialized XML version. This fact was verified by meeting with the XSBC developer Alan Hudson on Thurs. Aug. 26, 2004.

The author modified the application's XML build file to pull in the dom4j classes into the xsbc.jar so that one can produce a single jar to mount and work with. **This is a convenience method only and should not used for actual deployable .jar files**. The following modification was made inside of the jar target:

```
<!-- Include the dom4j jar file classes used to compile the
      classes for creating the application jar.  This produces one happy
      jar for everyone to enjoy.  Found this command on the Java
      Tutorial site (tdn) -->
      <zipgroupfileset dir="${lib.dir}" includes="*.jar"/>
```

Figure 27. Example Ant Script written in XML that takes Multiple Jar Files and Combines them into a Single JAR

All licenses and notifications were copied from dom4j into the META-INF\MANIFEST as required by dom4j.

This action of creating a single jar file was confirmed as not a good practice because if any of the "pulled-in" jars from other sources were modified, by Justin Couch, Alan Hudson and Don Brutzman. The entire jar would have to be rebuilt each time another required jar was modified by its developer, so, this is not good development practice.

The author modified the ComparisonTool to grab the FEC decoded file with the prefix "Decoded" to produce the foo.xml to show in XMLSpy.

The author had to go into the GZipProcess.java and modify where .bak files are made for both the original .xml and the resulting .xsbc files before gzipping. This wasn't working well due to problems encountered with the Runtime rt = Runtime.getRuntime() call. The System's command interpreter was never being invoked the way the code was originally written. It was because these method calls were originally Unix commands, not Windows. The GZipProcess.java file is in package org.web3d.xmsf.xsbc.apps.comparison.process. The modification now invokes the system command interpreter to backup, perform gzip and unbackup .xml and .xsbc files which now show correct statistics on the ComparisonTool GUI. Also, the author replaced the icon on this GUI to show the MOVES logo.

Found and even better ExecRunner (for Runtime calls) and StreamGobbler (required to output system errors encountered) at: http://spumoni.sourceforge.net/java2html/. This is under a GNU General Public License and which does not combine well with the GNU LGPL v2.1 that XSBC is open licensed with. With a little modification, annotated with the author's initials (tdn) in the source code, it works like a champ using java.lang.Runtime's exec() with the String cmd, String[] env and File dir arguments. The env argument is set to null in this particular implementation. Must also set a maximum time to allow the ExecRunner to do its thing, because if an application is still open, a while loop will be infinitely running waiting for a System.exit integer return.

Another important point about the original GZipProcess.java: not only was the author having command interpreter problems the way the original code was written, he didn't understand that is was written to invoke the LINUX / UNIX command interpreter. Modifications were made it to work the Windows XP command interpreter. The ExecRunner.java will investigate for all Windows O/S from 95 – XP and use the proper commands for each of these particular O/S builds.

Improved functionality by making the Sender Simulation and Receiver Simulation demos more robust. The Sender Simulation starts the Receiver

Simulation as a thread because the author needed to send to the receiver pre-information on the type of functions to expect such as if the .xml file will be XSBC'd and/or XSBC and GZipped. Played with the JFrame new look and feel features to window dress the GUIs a bit more. Built in a check box option to add FEC encoding of the resulting .xsbc or .xsbc.gz files. GZipping is now done via the GZIPInputStream and GZIPOutputStream method of java.util. Files can be saved to disc in either *.xsbc or *.xsbc.gz form and read into a DOM tree by the Receiver Simulation GUI. Put back in the functionality of producing an .xml DOM tree in the Sender Simulation so that the file can be visually rendered in JTree form before sending via UDP for comparison with the resulting received and decompressed form. This was accomplished by reintroducing Ekrem Serin's DocumentProccesor.java class. Of course, the option to GZip a file was also built into the demo pair.

The XSBC serializing and de-serializing processes, along with the option to add FEC encoding before transmitting, are now accomplished by the AUVW XsbcSerializer and XsbcTransaction classes. These classes have extra functionality that permit TCP or UDP transmission of packets over the net. FEC encoding and decoding functions are also built in if the (en/de) option is desired. So, the Sender serializes and encodes through the XsbcSerializer and the Receiver decodes and de-serializes through the XsbcTransaction.

C. XSBC CODE EXAMPLES

The following are XSBC code examples that show functionality of XSBC and how files from the AUVW were modified to incorporate both XSBC and FEC. There is an option for selection of FEC encoding and for GZipping.

1. SimpleExample.java

```
/* Program:      XML Schema Binary Compression (XSBC) example
 *
 * Author:       Alan Hudson, Yumetech Inc.
 * Modifier:     LT Terry D. Norbraten, USN [comments/changes labeled (tdn)]
 *
 * Created on:   2004-06-08 12:52 PM
 * Modified on:  August 05, 2004, 2224
 *
 * Course:       MV 0810 (Thesis Research)
 *               Summer 2004
 *
 * File:         SimpleExample.java
 *
 * Compiler:     netBeans IDE 3.6 (External), J2SDK 1.4.2_04
 * O/S:          Windows XP Home Ed. (SP 1)
 *
 * Description:  Example of how XSBC works to compress an XMLized file that
 *               is has a readable schema reference
 *
 * Information:  Using xsbc-0.91.1 created by Alan Hudson of Yumetech
 *               Inc.  An Java open source API.  Invoke this test before
 *               invoking XSBC_FECFileTest.java, or a FileNotFoundException
 *               will occur.
 */

/*****************************************************************************
 *                         Web3d.org Copyright (c) 2004
 *                                Java Source
 *
 * This source is licensed under the GNU LGPL v2.1
 * Please read http://www.gnu.org/copyleft/lgpl.html for more information
 *
 * This software comes with the standard NO WARRANTY disclaimer for any
 * purpose. Use it at your own risk. If there"s a problem you get to fix it.
 *
 ****************************************************************************/
package xsbc_fec;

// External Imports
import java.io.BufferedInputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

import java.net.URL;

// Local imports
import org.web3d.xmsf.xsbc.BlockDataInputStream;
import org.web3d.xmsf.xsbc.DocumentWriter;
import org.web3d.xmsf.xsbc.TableManager;
import org.web3d.xmsf.xsbc.XMLWriter;
import org.web3d.xmsf.xsbc.XSBCReader;

import org.web3d.xmsf.xsbc.datatypes.SimpleType;

import org.xml.sax.*;

/**
 * A simple example showing how XSBC can be used to compress and
 * decompress an XML document.
 *
 * @author Alan Hudson
 * @version 0.91.1
 */
public class SimpleExample {
```

```
/** Substring pointer to our schema */
public static final String SCHEMA = "\\espdu.xsd";

/** Substring pointer to our test .xml file */
public static final String TEST_FILE = "\\espdu";

/**
 * String pointer to my user dircectory.  Make this null to force pointing
 * to the system's user directory
 */
private final static String USER_DIR = "C:\\Documents and Settings\\Terry\\"
+ "My Documents\\My Files\\NPS\\Courses\\MV 0810 Thesis Work\\Java Code\\" +
"XSBC_FEC\\xsbc_fec\\examples\\";

/* MAIN METHOD */

/**
 * Entry point for the program
 *
 * @param args the command line entry arguments if any
 */
public static void main( String args[] ) {

   // Write out an XML file using XSBC
   try {

      File file = new File( USER_DIR + TEST_FILE + ".xml" );
      System.out.println( "Compressing: " + file );
      DocumentWriter dw = new DocumentWriter( file.getPath() );

      SimpleType.setCompressionMethod(
      SimpleType.COMPRESSION_METHOD_SMALLEST_NONLOSSY );

      FileOutputStream fos = new FileOutputStream( USER_DIR + TEST_FILE
                                                  + ".xsbc" );
      DataOutputStream dos = new DataOutputStream( fos );

      dw.serialize( dos );
      fos.close();

   } catch( IOException ioe ) {

      ioe.printStackTrace();

   } // end try-catch block

   // Read a file written using XSBC and write it out to foo.xml
   try {

      System.out.println( "Decompressing to foo.xml" );

      TableManager tableManager = new TableManager( new URL( "file:///"
                                + USER_DIR + SCHEMA ) );

      SimpleType.setCompressionMethod(
      SimpleType.COMPRESSION_METHOD_SMALLEST_NONLOSSY );

      File file = new File( USER_DIR + TEST_FILE + ".xsbc" );

      FileInputStream fis = new FileInputStream( file );
      BufferedInputStream bis = new BufferedInputStream( fis );
      BlockDataInputStream bdis = new BlockDataInputStream( bis );

      XSBCReader reader = new XSBCReader( tableManager );

      // Added path so that it goes directly to a folder that I designate
      // (tdn)
      XMLWriter writer = new XMLWriter( new FileOutputStream( new File(
      USER_DIR + "foo.xml")));
      reader.read( bdis, writer );
      } catch( Exception e ) {

         e.printStackTrace();

      } // end try-catch block

   } // end main()

} // end class file SimpleExample.java
```

2. Example UDP Implementation of an XSBC/FEC Server/Client Utility

a. SenderSimulation.java

```
/* Program:      Test UDP sending of XML compressed (XSBC) data encoded with
 *               Forward Error Correction (FEC)
 *
 * Author:       Alan Hudson, Yumetech Inc.
 * Modifier:     LT Terry D. Norbraten, USN [comments/changes labeled (tdn)]
 *
 * Created on:   August 20, 2004, 1903
 * Modified on:  November 13 2004, 1657
 *
 * Course:       MV 0810 (Thesis Research)
 *               Summer 2004
 *
 * File:         SenderSimulation.java
 *
 * Compiler:     netBeans IDE 3.6 (External), J2SDK 1.4.2_04
 * O/S:          Windows XP Home Ed (SP 1).
 *
 * Description:  This is a loopback UDP implementation of a server/client that
 *               takes an XML file, which is definied by an XML schema, is
 *               compressed with XSBC (XSBC 0.91.1 library) and encoded with FEC
 *               (FEC 1.0.3 library).  A Dom4jTree is created in each of the
 *               sever and client GUIs to show the origianl and transmitted XML
 *               files and appropriate XSBC and XML files are generated through
 *               optional GZipping and FEC processes.  The client is started as
 *               a thread from the server, but can run independently to analyze
 *               transmitted files.  This server/client application is supported
 *               by XsbcSerializer.java and XsbcTransaction.java which handle
 *               the XSBC and FEC processes.
 *
 * Information:  Using fec-1.0.3 created by Justin F. Chapweske of Onion
 *               Networks Inc. and xsbc-0.91.1 created by Alan Hudson of
 *               Yumetech Inc.  Both are Java open source APIs.
 */

/*****************************************************************************
 *                         Web3d.org Copyright (c) 2004
 *                                Java Source
 *
 * This source is licensed under the GNU LGPL v2.1
 * Please read http://www.gnu.org/copyleft/lgpl.html for more information
 *
 * This software comes with the standard NO WARRANTY disclaimer for any
 * purpose. Use it at your own risk. If there's a problem you get to fix it.
 *
 ****************************************************************************/
package xsbc_fec;

// External Imports (tdn)
import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.Image;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.io.IOException;

import java.net.InetAddress;
import java.net.UnknownHostException;
import java.net.URL;
import javax.swing.ButtonGroup;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JCheckBoxMenuItem;
import javax.swing.JComponent;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JPanel;
import javax.swing.JRadioButtonMenuItem;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTree;
import javax.swing.tree.TreeSelectionModel;
import javax.swing.filechooser.FileFilter;

// Application specific local imports (tdn)
import com.onionnetworks.fec.io.FECParameters;

import com.onionnetworks.util.Buffer;
import com.onionnetworks.util.Util;

import xsbc_fec.comparison.process.DocumentProcessor;
import xsbc_fec.comparison.process.XsbcSerializer;

/**
 * Class to show a simulation of XML serialization.  Provides a GUI to send XML
 * data over the network or save to local disc.  User loads the XML file and
 * sends it to the local host or saves to disc.  Modified by (tdn) to add a GZip
 * stream option and a FEC option to serialize XML data and/or, encode with
 * FEC before sending or saving to local disc.  This class will automatically
 * start the ReceiverSimulation as a thread of this class.  Modified by (tdn)
 * </p>
 *
 * @author <a href="mailto:giles@yumetech.com">Alan Hudson, Yumetech Inc.</a>
 * Modified by <a href="mailto:tdnorbra@nps.navy.mil">Terry D. Norbraten</a>
 * </p>
 */
public class SenderSimulation extends JComponent implements ActionListener {

   /* DATA MEMBER(s) */

   /** Check select button group */
   private ButtonGroup bGroup;

   /** Used to render and XML tree before processing to an .xsbc document */
   private DocumentProcessor dp;

   /** FEC encoding parameters that need to be sent to the reciever for file
    * decoding
    */
   private FECParameters params;

   /** Container for file information, path, etc */
   private File file,
                newFile;
```

```
/** In steam for file encoding */
private InputStream is;

/** File operation selections (tdn) */
private JButton sendFile,
                saveFile;

/** Option to encode file with FEC (tdn) */
private JCheckBoxMenuItem fecFile;

/** Option to gzip file (tdn) */
private JCheckBoxMenuItem gzipFile;

/** Our file chooser */
private JFileChooser fc;

/** bunch of gui stuff, we'll build actual gui later */
private JMenu fileMenu;
private JMenuBar menuBar;

/** Menu items */
private JMenuItem loadFile,
                  exit;

/** Our content pane */
private JPanel contentPane,
               buttonPanel;

/** The viewing pane for a document tree */
private JScrollPane treeView;

/** Document tree display area */
private JTextArea output;

/** Tree window */
private JTree tree;

/** String name of this local host */
private final String LOCAL_HOST = "127.0.0.1";

/**
 * String pointer to my user dircectory.  Make this null to force pointing
 * to the system's user directory
 */
private final String USER_DIR = "C:\\Documents and Settings\\Terry\\" +
"My Documents\\My Files\\NPS\\Courses\\MV 0810 Thesis Work\\Java Code\\" +
"XSBC_FEC\\xsbc_fec\\examples";

/** Instance of the receiver to notify of GZipping */
private ReceiverSimulation rs;

/** Thread instance to run Receiver Simulation */
private Thread t;

/** Our XML serializer process */
private XsbcSerializerFecEncoder xsfe;

/** Indicate our return value */
private int returnVal;

/** Local port designation */
private final int DEST_PORT = 4040;

/** Designate 3 of 4 FEC file parameters */
private final int K = 16,
                  N = 32,
                  PACKET_SIZE = 1024;
```

```
/* CONSTRUCTOR(s) */

/** Creates a new instance of SenderSimulation */
public SenderSimulation() {

   file = newFile = null;
   returnVal = 0;
   params = null;
   dp = null;
   startReceiverSimulation();

} // end constructor

/* PUBLIC METHOD(s) */

/**
 * Creates the menu bar
 *
 * @return the created menu bar
 */
public JMenuBar createMenuBar() {

   // Create the menu bar
   menuBar = new JMenuBar();

   // Create the first menu
   fileMenu = new JMenu( "File" );
   menuBar.add( fileMenu );

   // A load XML file menu item
   loadFile = new JMenuItem( "Load XML File" );
   loadFile.addActionListener( this );
   fileMenu.add( loadFile );

   // a group of check box menu items
   fileMenu.addSeparator();
   bGroup = new ButtonGroup();

   // Allow for either one or both check boxes to be selected
   gzipFile = new JCheckBoxMenuItem( "GZip File" ); // tdn
   gzipFile.setSelected( true );
   gzipFile.setMnemonic(KeyEvent.VK_C);
   gzipFile.addActionListener( this );
   fileMenu.add( gzipFile );

   fecFile = new JCheckBoxMenuItem( "Encode with FEC" ); // tdn
   fecFile.setMnemonic(KeyEvent.VK_H);
   fecFile.addActionListener( this );
   fileMenu.add( fecFile );

   fileMenu.addSeparator();

   exit = new JMenuItem( "Exit" );
   exit.addActionListener( this );
   fileMenu.add( exit );

   return menuBar;

} // end createMenuBar()

/**
 * Creates the content pane for the GUI
 *
 * @return a container for this content pane
 */
public Container createContentPane() {

   // Create the content-pane-to-be.
   contentPane = new JPanel(new BorderLayout());
   contentPane.setOpaque(true);
   // Add our operating buttons
   contentPane.add( createButtons(), BorderLayout.SOUTH );

   return contentPane;

} // end createContentPane()

/** @return an Image or null */
public Image getFDImage() {

   URL imgURL = SenderSimulation.class.getResource("images/MOVESLogo.gif");
   if (imgURL != null) {
      return new ImageIcon(imgURL).getImage();
   } else {
      System.out.println( "Unable to find icon" );
      return null;
   }

} // end getFDImage()

/* PRIVATE METHOD(s) */

/**
 * Starts the Receiver Simulation at the same time as this simulation to
 * work as a pair
 */
private void startReceiverSimulation() {

   // Run the Receiver Simulation as a thread of this GUI
   rs = new ReceiverSimulation();
   t = new Thread( rs );
   t.start();

} // end startReceiverSimulation()

/**
 * Creates our operating buttons
 *
 * @return a panel for our buttons
 */
private JPanel createButtons() {

   buttonPanel = new JPanel();

   sendFile = new JButton( "Send" );
   sendFile.addActionListener( this );
   saveFile = new JButton( "Save" );
   saveFile.addActionListener( this );

   buttonPanel.add( sendFile );
   buttonPanel.add( saveFile );

   return buttonPanel;

} // end createButtons()

/** Performes the .xml file loading procedure */
private void doLoadFile() {

   fc = new JFileChooser( USER_DIR );
   fc.addChoosableFileFilter( new XMLFileFilter() );

   returnVal = fc.showDialog( this, "Load XML File" );

   if ( returnVal == JFileChooser.APPROVE_OPTION ) {

      file = fc.getSelectedFile();

      System.out.println( "Loading: " + file.getPath() );

   } // end if
   // Cause the selected .xml document to render as a dom4j tree in this
   // GUI (tdn)
   new Thread( new Runnable() {

      public void run() {

         try {

            if ( file != null ) {

               // Using Ekrem Serin's Document Processor from XFSP
               dp = new DocumentProcessor(new URL("file:///"
               + file.getPath()));

            } else {

               System.out.println( "You didn't load a file? " );
               return;

            } // end if-else block

            tree = new JTree(dp.getTree());

            tree.getSelectionModel().setSelectionMode(
            TreeSelectionModel.SINGLE_TREE_SELECTION);

            //Create the scroll pane and add the tree to it.
            treeView = new JScrollPane(tree);

            // Add the scroll pane to the content pane.
            contentPane.add(treeView, BorderLayout.CENTER);

            // XML tree won't render unless we do this (tdn)
            contentPane.setVisible(false);
            contentPane.setVisible(true);

         } catch (Exception ex) {

            ex.printStackTrace();

         } // end try-catch block

      } // end run()

   } ).start();

} // end doLoadFile()

/**
 * Performs the send file procedure.  This is invoked once to render the DOM
 * tree in the receiver and then, when selected again, will generate the
 * mission file .xml in the dataweb/results cache foler.
 */
private void doSendFile() {

   // Make sure we load a file first before we can send one out serialized
   if ( file == null ) {

      System.out.println( "You must load a file before sending!" );
      doLoadFile();

   } // end if

   // Check if we want encode with FEC
   if ( fecFile.getState() ) {

      // Invoke the serializer and send .xsbc file info via the UDP
      // option in GZIP and XSBC form with FEC encoding parameters.  The
      // file was already saved as an .xsbc or .xsbc.gz (tdn)
      xsfe = new XsbcSerializerFecEncoder( newFile.getPath(), true,
      fecFile.getState(), params, LOCAL_HOST, DEST_PORT );
```

```
         // Check if we want to gzip before .xsbc
         } else if ( gzipFile.getState() ) {

            // Invoke the serializer and send .xml file info via the UDP option
            // in GZIP and XSBC form w/o FEC encoding (tdn)
            xsfe = new XsbcSerializerFecEncoder( file.getPath(), true,
            fecFile.getState(), null, LOCAL_HOST, DEST_PORT );

            returnVal = xsfe.writeNetwork( LOCAL_HOST, DEST_PORT,
            gzipFile.getState() );

            // Indicate results of writing to gz.xsbc or just .xsbc
            if ( returnVal == 1 )

               System.out.println( "Sucessfully sent in GZipped and XSBC " +
               "form." );

            else if ( returnVal == 0 )

               System.out.println( "Problem encountered while sending in " +
               "GZipped and XSBC form." );

         // We just want to .xsbc the file
         } else {

            // Invoke the serializer and send .xml file info via the UDP option
            // in XSBC form w/o FEC encoding (tdn)
            xsfe = new XsbcSerializerFecEncoder( file.getPath(), true,
            fecFile.getState(), null, LOCAL_HOST, DEST_PORT );

            returnVal = xsbcs.writeNetwork( LOCAL_HOST, DEST_PORT,
            gzipFile.getState() );

            // Indicate results of writing to gz.xsbc or just .xsbc
            if ( returnVal == 1 )

               System.out.println( "Sucessfully sent in XSBC form." );

            else if ( returnVal == 0 )

               System.out.println( "Problem encountered while sending in " +
               "XSBC form." );

         } // end if-else block

         // Trying to be careful with these
         returnVal = 0;

         // Must select fec encoding again next time around
//       fecFile.setState( false );

      } // end doSendFile()

      /** Performs the save file to disc procedure in .xsbc or .xsbc.gz format */
      private void doSaveFile() {

         // Invoke the serializer second constructor and send file info with
         // write to disc only function w/o FEC encoding (tdn)
         xsfe = new XsbcSerializerFecEncoder( file.getPath(), false, false, null,
                                              LOCAL_HOST, DEST_PORT );

         // Check if we want gzip before .xsbc
         if ( gzipFile.getState() ) {

            // Rename our file to be gzipped & serialized with an xsbc.gz extension
            // to indicate xsbc in gzip format (tdn)
            returnVal = xsbcs.writeFile( file.getPath().substring( 0,
            file.getPath().lastIndexOf( '.' ) ) + ".xsbc.gz",
            gzipFile.getState() );

            // Indicate results of writing to gz.xsbc or just .xsbc
            if ( returnVal == 1 ) {

               // We need access to this file for FEC Encoding
               newFile = new File(file.getPath().substring( 0,
               file.getPath().lastIndexOf( '.' ) ) + ".xsbc.gz" );

               System.out.println( "Serialized and GZipped: " + newFile );

            } else if ( returnVal == 0 ) {

               System.out.println( "Problem encountered while saving: "
               + file.getPath().substring( 0, file.getPath().lastIndexOf( '.' ) )
               + ".xsbc.gz" );

            } // end else-if block

         } else {

            // Rename our serialized file with an .xsbc extension only (tdn)
            returnVal = xsbcs.writeFile( file.getPath().substring( 0,
            file.getPath().lastIndexOf( '.' ) ) + ".xsbc",  gzipFile.getState() );

            // Indicate results of writing to .xsbc
            if ( returnVal == 1 ) {

               // We need access to this file for FEC Encoding
               newFile = new File(file.getPath().substring( 0,
               file.getPath().lastIndexOf( '.' ) ) + ".xsbc") ;

               System.out.println( "Serialized only: " + newFile );

            } else if ( returnVal == 0 ) {

               System.out.println( "Problem encountered while saving: "
               + file.getPath().substring( 0, file.getPath().lastIndexOf( '.' ) )
               + ".xsbc" );

            } // end else-if block

         } // end if-else block

         // Trying to be careful about these
         returnVal = 0;

      } // end doSaveFile()

      /**
       * Creates an FECParameters that the decoding process will need access to
       * for orig. file reconstruction (tdn)
       */
      private void createFECParameters() {

         // Make sure we load a file first before we can FEC encode and send
         if ( file == null ) {

            System.out.println( "You must load a file before sending with FEC" +
            " encoding!" );
            return;

         } // end if

         // We need save a file in .xsbc or .xsbc.gz form so that the FEC
         // encoder has access to the file.  The File will be called newFile when
         // completed.
         doSaveFile();

         params = new FECParameters( K, N, PACKET_SIZE, newFile.length() );
         // Send the parameters to the receiver for file decoding info.  This is a
         // convenience method that actually wouldn't be done in a real server/
         // client application.  The parameters would be coded in the data members
         // of the client for K and N.
         rs.setFECParameters( params );

      } // end createFECParameters()

      /* ACTION LISTENER(s) */

      /**
       * Implememnts action listener
       *
       * @param e the event to handle
       */
      public void actionPerformed( ActionEvent e ) {

         Object evt = e.getSource();

         // Notify the receiver of the GZipped option
         rs.setIsGZipped( gzipFile.getState() );

         // Notify the receiver of the FEC option
         rs.setIsFECEncoded( fecFile.getState() );

         if ( evt == exit ) {

            System.exit( 0 );

         } // end if

         // Load the selected .xml file and show the DOM tree.  This should be
         // done first in all cases (tdn)
         if ( evt == loadFile ) {

            doLoadFile();

         } // end if

         // XSBC selected .xml file and send over the net (tdn)
         if ( evt == sendFile ) {

            doSendFile();
            return;

         } // end if

         // GZip and/or XSBC selected .xml file and save to disc.  Must load the
         // file again before saving.  Sending it twice to produce DOM tree and
         // the mission file closes all the streams.  Loading the file again will
         // open all appropriate streams.
         if ( evt == saveFile ) {

            // This one ignores if FEC Encoding is selected b/c it's just being
            // serialized to disc only.  It can be encoded later if we wish.
            doSaveFile();
            return;

         } // end outer if

         if ( fecFile.getState() ) {

            // If the .xsbc or .xsbc.gz file didn't get saved to disc,
            // FECParameters will be null.  Once another action such as load
            // file is fired, this will not return.
            if ( file == null ) {

               System.out.println( "You must load a file before hitting the" +
               " 'Send' button!" );
               return;
            } else

               // Creates the FECParameter that the decoder will need for file
               // reconstruction.  This method will save a serialized .xsbc or
               // .xsbc.gz file to disc so that the serializer method only has to
               // FEC encode the file (tdn)
               createFECParameters();

         } // end if

      } // end actionPerformed()

      /* MAIN METHOD */

      /**
       * Method called when used as an application.  Borrowed from Java Sun's
       * GUI tutorial.
       *
       * @param args the command line initial argument
       *
       * @exception Exception if GUI didn't create successfully
       */
      public static void main( String args[] ) {

         // Execute a job on the event-dispatching thread creating this GUI
         try {

            javax.swing.SwingUtilities.invokeAndWait( new Runnable() {

               public void run() {

                  createAndShowGUI();

               } // end run()

            } ); // end invokeAndWait()

         } catch ( Exception e ) {

            System.err.println( "createAndShowGUI() didn't successfully complete" +
                                " due to:\n" );

            e.printStackTrace();

         } // end try-catch block

      } // end main()

      /**
       * Create the GUI and show it.  For thread safety, this method is invoked
       * from the event-dispatching thread (Java Sun's Tutorial on GUIs)
       */
      private static void createAndShowGUI() {

         //Make sure we have nice window decorations.
         JFrame.setDefaultLookAndFeelDecorated( true );

         // Create and set up the window
         JFrame f = new JFrame( "XSBC w/ FEC SENDER  S I M U L A T I O N" );
         f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

         //Create and set up the content pane.
         SenderSimulation ss = new SenderSimulation();
         f.setJMenuBar( ss.createMenuBar() );
         f.setContentPane( ss.createContentPane() );

         //Display the window.
         f.setSize(500, 600);
         f.setLocation(200, 100);
         f.setVisible( true );
```

```
      // Show the MOVES Logo as an icon
      f.setIconImage( ss.getFDImage() );

   } // end createAndShowGUI()

   /* INNER CLASS(s) */

   public class XMLFileFilter extends FileFilter {

      /** String representation of a file extension */
      private String extension = null;

      /** String representation of a file name */
      private String fn = "";

      /** Accept all directories and all gif, jpg, or tiff files */
      public boolean accept( File f ) {

         if ( f.isDirectory() )

            return true;

         fn = f.getName();

         int i = fn.lastIndexOf('.');

         if (i > 0 &&  i < fn.length() - 1) {

            extension = fn.substring( i + 1 ).toLowerCase();

         } // end if

         if ( extension != null ) {

            if ( extension.equals( "xml" ) )

               return true;

            else

               return false;

         } // end outer if

         return false;

      } // end accept()

      /**
       * The description of this filter
       *
       * @return a string describing the filter
       */
      public String getDescription() {

         return "Just XML Files";

      } // end getDescription()

   } // end inner class XMLFileFilter

} // end class file SenderSimulation.java
```

3. The Receiver Client Code

a. ReceiverSimulation.java

```
/* Program:      Test UDP sending of XML compressed (XSBC) data encoded with
 *               Forward Error Correction (FEC)
 *
 * Author:       Alan Hudson, Yumetech Inc.
 * Modifier:     LT Terry D. Norbraten, USN [comments/changes labeled (tdn)]
 *
 * Created on:   August 20, 2004, 1903
 * Modified on:  November 13 2004, 1657
 *
 * Course:       MV 0810 (Thesis Research)
 *               Summer 2004
 *
 * File:         ReceiverSimulation.java
 *
 * Compiler:     netBeans IDE 3.6 (External), J2SDK 1.4.2_04
 * O/S:          Windows XP Home Ed (SP 1).
 *
 * Description:  This is a loopback UDP implementation of a server/client that
 *               takes an XML file, which is definied by an XML schema, is
 *               compressed with XSBC (XSBC 0.91.1 library) and encoded with FEC
 *               (FEC 1.0.3 library).  A Dom4jTree is created in each of the
 *               sever and client GUIs to show the origianl and transmitted XML
 *               files and appropriate XSBC and XML files are generated through
 *               optional GZipping and FEC processes.  The client is started as
 *               a thread from the server, but can run independently to analyze
 *               transmitted files.  This server/client application is supported
 *               by XsbcSerializer.java and XsbcTransaction.java which handle
 *               the XSBC and FEC processes.
 *
 * Information:  Using fec-1.0.3 created by Justin F. Chapweske of Onion
 *               Networks Inc. and xsbc-0.91.1 created by Alan Hudson of
 *               Yumetech Inc.  Both are Java open source APIs.
 */

/*****************************************************************************
 *                        Web3d.org Copyright (c) 2004
 *                               Java Source
 *
 * This source is licensed under the GNU LGPL v2.1
 * Please read http://www.gnu.org/copyleft/lgpl.html for more information
 *
 * This software comes with the standard NO WARRANTY disclaimer for any
 * purpose. Use it at your own risk. If there's a problem you get to fix it.
 *
 ****************************************************************************/
package xsbc_fec;

// External imports
import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.Image;

import java.io.BufferedInputStream;
import java.io.ByteArrayInputStream;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PushbackInputStream;

import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.Socket;
import java.net.SocketException;
import java.net.URL;

import java.util.zip.GZIPInputStream;

import javax.swing.ButtonGroup;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JPanel;
import javax.swing.JRadioButtonMenuItem;
import javax.swing.JScrollPane;
import javax.swing.JTree;
import javax.swing.tree.TreeSelectionModel;
import javax.swing.filechooser.FileFilter;

import org.dom4j.Document;

// Application specific local imports (tdn)
import com.onionnetworks.fec.FECCode;
import com.onionnetworks.fec.FECCodeFactory;

import com.onionnetworks.fec.io.FECParameters;

import com.onionnetworks.util.Buffer;
import com.onionnetworks.util.Util;

import org.apache.log4j.BasicConfigurator;

import org.web3d.xmsf.xsbc.BlockDataInputStream;
import org.web3d.xmsf.xsbc.Dom4jReader;
import org.web3d.xmsf.xsbc.Dom4jReaderNew;
import org.web3d.xmsf.xsbc.TableManager;
import org.web3d.xmsf.xsbc.XSBCReader;
import org.web3d.xmsf.xsbc.apps.XMLSwingTree;

import xsbc_fec.comparison.process.XsbcTransaction;

/**
 * Class to show a simulation of XML serialization.  Provides a GUI to receive
 * the XML data.  User must assign schema before actual receive process to
 * define the packet type.  This class is automatically started as a thread from
 * the SenderSimulation.  Modified by (tdn) to ungzip and decode FEC. </p>
 *
 * @author <a href="mailto:giles@yumetech.com">Alan Hudson, Yumetech Inc.</a>
 * Modified by <a href="mailto:tdnorbra@nps.navy.mil">Terry D. Norbraten</a>
 * </p>
 */
public class ReceiverSimulation extends JComponent implements ActionListener,
Runnable {

   /* DATA MEMBER(s) */

   /** Our data input stream */
   private BlockDataInputStream bdis;
```

```
/**
 * An FEC Buffer[] holding byte arrays (packets of decoded data size 1024 B)
 * to be read into the DOM reader for .xml tree rendering
 */
private Buffer[] repairBuffer;

/** A buffered input stream */
private BufferedInputStream bis;

/** Check select button group */
private ButtonGroup bGroup;

/** An output stream carrying bytes */
private ByteArrayInputStream bais;

/** Our UDP packets to receive data */
private DatagramPacket packet,
                       fecPacket,
                       indicePacket;

/** Our UDP socket */
private DatagramSocket udpSocket;

/** DOM4J document */
private Document domDoc;

/** XML document reader */
private Dom4jReader domReader;

/** XSBC document reader */
private Dom4jReaderNew dombuilder;

/** Our decoding code algorithms */
private FECCode fec;

/** The parameters we set for FEC encoding */
private FECParameters params;

/** Container for the schema file information, path, etc */
private File schemaFile;

/** Container for the binary file information, path, etc */
private File binaryFile;

/** Input stream for reading binary files */
private FileInputStream fis;

/** Our inbound GZip stream */
private GZIPInputStream zipStream;

/** File operation selections (tdn) */
private JButton receiveXSBCFile,
receiveEncodedFile;

/** Our file chooser */
private JFileChooser fc;

/** bunch of gui stuff, we'll build actual gui later */
private JMenu fileMenu;
private JMenuBar menuBar;

/** Menu items */
private JMenuItem assignSchema,
                  loadBinary,
                  exit;

/** Our content pane */
private static JPanel contentPane,
                      buttonPanel;
```

```
/** The viewing pane for a document tree */
private JScrollPane treeView;

/** the swing tree, to show the XML document as a swing tree */
private JTree tree;

/** String reference to a file path */
private String path;

/** Point relatively to the x3d schema directory (tdn) */
public final String X3DSCHEMA = "examples\\x3d-3.0.xsd";

/** Thread of the XSBC transaction process */
private Thread x;

/** String pointer to my user directory */
private final String USER_DIR = "C:\\Documents and Settings\\Terry\\" +
"My Documents\\My Files\\NPS\\Courses\\MV 0810 Thesis Work\\Java Code\\" +
"XSBC_FEC\\xsbc_fec\\examples\\";

/** The orig. base name of a now encoded file */
private String origName;

/** look up table */
private TableManager tableManager;

/** Our XML swing tree */
private XMLSwingTree swingTree;

/** XSBC reader */
private XSBCReader reader;

/** Our XML de-serializer process */
private XsbcTransactionFecDecoder xtfd;

/** Buffers to hold FEC data */
private byte[] encodedKData,
               encodedNData;

/** Flag to denote if file is GZipped */
private boolean isGZipped;

/** Flag to denote if file is encoded with FEC */
private boolean isFECEncoded;

/** Container for all block received indices */
private int[] repairIndexes;

/** Indicate our return value */
private int returnVal;

/** the port that we listen */
private final int PORT = 4040;

/* CONSTRUCTOR(s) */

/**
 * Constructs a new instance of ReceiverSimulation
 */
public ReceiverSimulation() {

   returnVal = 0;
   setIsGZipped( false);
   setIsFECEncoded( false );
   openDatagramSocket();

} // end constructor
```

```
/* PUBLIC METHOD(s) */

/**
 * Creates the menu bar
 *
 * @return the created menu bar
 */
public JMenuBar createMenuBar() {

   // Create the menu bar
   menuBar = new JMenuBar();

   // Create the first menu
   fileMenu = new JMenu( "File" );
   menuBar.add( fileMenu );

   // A group of menu items
   assignSchema = new JMenuItem("Assign Schema");
   assignSchema.addActionListener( this );
   fileMenu.add( assignSchema );

   loadBinary = new JMenuItem( "Load Binary File" );
   loadBinary.addActionListener( this );
   fileMenu.add(loadBinary);

   // a button menu item
   fileMenu.addSeparator();
   bGroup = new ButtonGroup();

   exit = new JMenuItem( "Exit" );
   exit.addActionListener( this );
   fileMenu.add( exit );

   return menuBar;

} // end createMenuBar()

/**
 * Creates the content pane for the GUI
 *
 * @return a container for this content pane
 */
public Container createContentPane() {

   // Create the content-pane-to-be.
   contentPane = new JPanel(new BorderLayout());
   contentPane.setOpaque(true);

   // Add our operating buttons
   contentPane.add( createButtons(), BorderLayout.SOUTH );

   return contentPane;

} // end createContentPane()

/** @return an Image or null */
public Image getFDImage() {

   URL imgURL = ReceiverSimulation.class.getResource("images/MOVESLogo.gif");
   if (imgURL != null) {
      return new ImageIcon(imgURL).getImage();
   } else {
      System.out.println( "Unable to find icon" );
      return null;
   }

} // end getFDImage()
```

```
/* GETTER(s) / SETTER(s) */

/**
 * Sets whether received file is GZipped
 *
 * @param f true if file is GZipped
 */
public void setIsGZipped( boolean f ) {

   isGZipped = f;

} // end setIsGZipped()

/**
 * Indicates whether received file is GZipped
 *
 * @return true if file is GZipped
 */
public boolean getIsGZipped() {

   return isGZipped;

} // end getIsGZipped()

/**
 * Sets whether received file is FEC Encoded and set the FEC Parameters
 * if so.
 *
 * @param f true if file is FEC Encoded
 */
public void setIsFECEncoded( boolean f ) {

   isFECEncoded = f;

} // end setFECEncoded()

/**
 * Indicates whether received file is FEC Encoded
 *
 * @return true if file is FEC Encoded
 */
public boolean getIsFECEncoded() {

   return isFECEncoded;

} // end getIsFECEncoded()

/**
 * Sets the FEC Parameters that we need for file decoding.  This is cheating
 * as we need to send this info over the net instead.
 *
 * @param p the FEC Parameters to set
 */
public void setFECParameters( FECParameters p ) {

   params = p;

} // end setFECParameters()

/**
 * Retrieves the FEC Parameters that we need for file decoding.  This is
 * cheating as we need to send this info over the net instead.
 *
 * @return the FEC Parameters to set
 */
public FECParameters getFECParameters() {

   return params;

} // end getFECParameters()
```

```
/* PRIVATE METHOD(s) */

/**
 * Creates our operating buttons
 *
 * @return a panel for our buttons
 */
private JPanel createButtons() {

   buttonPanel = new JPanel();

   receiveXSBCFile = new JButton( "Receive XSBC" );
   receiveXSBCFile.addActionListener( this );

   receiveEncodedFile = new JButton( "Receive Encoded" );
   receiveEncodedFile.addActionListener( this );

   buttonPanel.add( receiveXSBCFile );
   buttonPanel.add( receiveEncodedFile );

   return buttonPanel;

} // end createButtons()

/**
 * Opens the Datagram socket for receiving packets on designated port
 *
 * @exception SocketException if the socket couldn't bind to the port
 */
private void openDatagramSocket() {

   try {

      udpSocket = new DatagramSocket( PORT );

   } catch ( SocketException se ) {

      se.printStackTrace();

   } // end try-catch block

} // end openDatagramSocket()

/** Performs the schema loading procedure */
private void doAssignSchema() {

   fc = new JFileChooser(USER_DIR);
   fc.addChoosableFileFilter(new SchemaFileFilter());

   returnVal = fc.showDialog(this,"Assign Schema");

   if (returnVal == JFileChooser.APPROVE_OPTION) {

      schemaFile = fc.getSelectedFile();
      System.out.println("Loading schema: " + schemaFile.getPath());

      try {

         tableManager = new TableManager(new URL("file:///"
         + schemaFile.getPath()));

      } catch (Exception ex) {

         ex.printStackTrace();

      } // end try-catch block

      // Invoke our receiver .xsbc document reader at let it know if the
      // incoming file is in GZip format and or FEC encoded (tdn)
      xtfd = new XsbcTransactionFecDecoder( udpSocket, schemaFile, 0,
             getIsGZipped(),
      getIsFECEncoded(), getFECParameters() );
      x = new Thread( xtfd );

   } // end inner if

} // end doAssignSchema()

/** Performs the receiving file procedure to build the DOM tree */
private void doReceiveFile() {

   new Thread( new Runnable() {

      /**
       * @exception IOException if there are socket or GZIPInputStream
       * issues
       */
      public void run() {

         // Notify to load the schema first
         if (tableManager == null) {

            System.out.println( "You must load a schema first!" );
            doAssignSchema();;

         } // end if

         // create an .xml document processor
         domReader = new Dom4jReader( tableManager );

         // Prepare the packet for reception of data
         packet = new DatagramPacket( new byte[ 1500 ], 1500 );

         // The first time you hit the send button, this receiver will
         // generate the dom4j .xml document tree.  If you hit this send
         // button a second time, a missionResultsXsbc0.file will
         // be generated in the dataweb/results folder (tdn)
         try  {

            udpSocket.receive( packet );

         } catch ( IOException ioe ) {

            ioe.printStackTrace();

         } // end try-catch block

         System.out.println("Received Packet Size " +
         packet.getLength());

         // Grap the packet contents
         bais = new ByteArrayInputStream( packet.getData() );

         doRenderDOMTree();

         // Start the xsbc transaction thread to make the mission file after
         // the second time the send button is selected from the server
         x.start();

      }  // end run

   } ).start();

} // end doReceiveFile()
```

```
/**
 * Performs preparations for FEC decoding to render an .xml file in the
 * DOM tree.  In this runnable thread, the socket will listen for packets and
 * receive them from the Server Simulation.
 *
 * @exception IOException if the socket couldn't read packets
 */
private void doFECDecoding() {

   new Thread( new Runnable() {

      public void run() {

         if ( getIsFECEncoded() )

            //create our fec code
            fec = FECCodeFactory.getDefault().createFECCode( params.getK(),
            params.getN() );

         else {

            // If we don't return here, a null pointer exception will be
            // thrown.
            System.out.println( "FEC Parameters haven't been set by the"
            + " encoding server! " );
            return;

         } // end if-else block

         // Notify to load the schema first
         if (tableManager == null) {

            System.out.println( "You must load a schema first!" );
            doAssignSchema();

         } // end if

         // create an .xml document processor
         domReader = new Dom4jReader( tableManager );

         // Prepare to receive our K * Block Count repair indices.  These
         // are just byte format integers for each Datagram packet.  Again,
         // we will select our K required subset from the N received from
         // each block.
         indicePacket = new DatagramPacket( new byte[ params.getN() ],
         params.getN() );

         // Storage for each block's K subset of N repair indexes that
         // will be copied from receiveIndexes.
         repairIndexes = new int[ params.getK() * params.getBlockCount() ];

         // Receive each block repair index information
         for ( int ix = 0; ix < params.getBlockCount(); ix++ ) {

            try {

               // Receive our repair index info.
               udpSocket.receive( indicePacket );

            } catch ( IOException ioe ) {

               ioe.printStackTrace();

            } // end try-catch block

            // Feed index information into this BAIS per block
            bais = new ByteArrayInputStream( indicePacket.getData() );

            System.out.println( "Receiving block number: " + ix );
```

```
               // Extract K * Block Count packet elements that make up our
               // repair indices.
               for ( int jx = 0; jx < params.getK(); jx++ ) {

                  // Collect our K selected repair packet indicies
                  repairIndexes[ jx ] = bais.read();

//                  System.out.println( "Received repair packet index: " +
//                  repairIndexes[ jx ] ); // DEBUG

               } // end inner for

            } // end outer for

            /***************** Receive Encoded Repair Packets *****************/

            // Prepare this packet for reception of a repair packet each
            // containing 1024 bytes of encoded data
            fecPacket = new DatagramPacket( new byte[ params.getPacketSize() ],
            params.getPacketSize() );

            // Temp storage for encoded data
            encodedNData = new byte[ params.getN() * params.getPacketSize() *
            params.getBlockCount() ];
            encodedKData = new byte[ params.getK() * params.getPacketSize() *
            params.getBlockCount() ];

            // Receive each block's worth of N encoded repair packets for
            // storage
            for ( int ix = 0; ix < params.getN() * params.getBlockCount();
                  ix++ ) {

               // Receive a block of data to encode
               try {

                  // Receive our encoded data
                  udpSocket.receive( fecPacket );

               } catch ( IOException ioe ) {

                  ioe.printStackTrace();

               } // end try-catch block

               // Copy N encoded packets into a byte[] for K selection
               System.arraycopy( fecPacket.getData(), 0, encodedNData, ix *
               fecPacket.getLength(), fecPacket.getLength() );

            } // end for

            // Extract a K subset of the N encoded repair packets for decoding
            System.arraycopy( encodedNData, 0, encodedKData, 0, params.getK() *
                              params.getPacketSize() );

//            for ( int ix = 0; ix < params.getK(); ix++ ) {

//               System.out.println( "Receive FEC encoded data packet " +
//               ix + " " + encodedKData[ ix ] ); // DEBUG

//            } // end for

            System.out.println( "Expanded Block Size: " +
            params.getExpandedBlockSize() ); // DEBUG

            // This will hold our K subset of encoded repair packets
            repairBuffer = new Buffer[ params.getK() * params.getBlockCount() ];

            // Put the encoded data into an FEC Buffer wrapper for decoding
            for ( int ix = 0; ix < params.getK() * params.getBlockCount();
                  ix++ )
               repairBuffer[ ix ] = new Buffer( encodedKData, ix *
               params.getPacketSize(), params.getPacketSize() );

            // Finally, decode the repair packets into the file
            fec.decode( repairBuffer, repairIndexes );

            // Grab the repairBuffer's contents
            bais = new ByteArrayInputStream( encodedKData );


            // Prompt to assign the schema to be able to render the DOM tree
            if ( schemaFile == null ) {

               doAssignSchema();

            } // end if

            doRenderDOMTree();

            // Start the xsbc transaction thread to make the mission file
            x.start();

         } // end run()

      } ).start();

   } // end doFECDecoding()

   /**
    * Renders the DOM tree in the JScroll Panel after being decoded and/or
    * uncompressed.
    */
   private void doRenderDOMTree() {

       try {

         // Test if the byte array input stream is wrapped in a
         // GZIPInputSteam.  If not, continue normally (tdn)
         if ( getIsGZipped() ) {

             zipStream = new GZIPInputStream(bais);
             System.out.println( "Receiving GZipped data.... " );
             bis = new BufferedInputStream(zipStream);
             bdis = new BlockDataInputStream(bis);

          } else {

            bdis = new BlockDataInputStream(bais);

          } // end if-else block

          // Read the data (throws a null pointer exception at org.web3d.xmsf.
          // xsbc.Dom4jReader.readingAttributes(Dom4jReader.java:317)
          // Impact: Receiver Dom4jTree panel incomplete rendering of full
          // received XSBC file (tdn)
          domReader.parseData( bdis );

          // Build the XML tree, but something is broke.  Not all
          // attributes are reconstructing in the DOM tree (tdn)
          swingTree = new XMLSwingTree( domReader.getDocument() );
          tree = new JTree(swingTree.getSwingTree());
          tree.getSelectionModel().setSelectionMode(
          TreeSelectionModel.SINGLE_TREE_SELECTION);
          treeView = new JScrollPane( tree );

          // Add the scroll pane to the content pane.
          contentPane.add(treeView, BorderLayout.CENTER);

          // XML tree won't render unless we do this (tdn)
          contentPane.setVisible(false);
          contentPane.setVisible(true);
       } catch ( IOException ioe ) {

          ioe.printStackTrace();

       } // end try-catch block

   } // end doRenderDOMTree()


   /** Performes the binary file loading */
   private void doLoadBinary() {

      new Thread( new Runnable() {

         public void run() {

            // Notify to load the schema first
            if (tableManager == null) {

               System.out.println( "You must load a schema first!" );
               doAssignSchema();

            } // end if

            try {

               fc = new JFileChooser(USER_DIR);
               fc.addChoosableFileFilter(new XMLFileFilter());

               returnVal = fc.showDialog(ReceiverSimulation.this,
               "Load XML File");

               if (returnVal != JFileChooser.APPROVE_OPTION)
                  return;

               // created the document processor
               binaryFile = fc.getSelectedFile();

               // Test for a .gz extension
               String extension = null;
               String fn = binaryFile.getName();
               int i = fn.lastIndexOf('.');
               if (i > 0 &&  i < fn.length() - 1) {
                  extension = fn.substring(i+1).toLowerCase();
               } // end if
               if (extension != null) {
                  if (extension.equals("gz"))
                     setIsGZipped( true );
               } // end if

               System.out.println("XSBC file: " + binaryFile.getPath() +
               " \nand schema: " +
               schemaFile.getPath() ); // DEBUG

               fis = new FileInputStream(binaryFile);
               byte[] buffer = new byte[ (int) binaryFile.length() ];
               fis.read( buffer );
               bais = new ByteArrayInputStream( buffer );
               if ( getIsGZipped() ) {
                  zipStream = new GZIPInputStream( bais );
                  bis = new BufferedInputStream(zipStream);
                  bdis = new BlockDataInputStream(bis);
               } else {

                  bdis = new BlockDataInputStream(bais);

               }

               System.out.println("Loading GZipped binary data .... ");

               dombuilder = new Dom4jReaderNew(tableManager);
               reader = new XSBCReader(tableManager);

               reader.read(bdis, dombuilder);

               domDoc = dombuilder.getDocument();

               swingTree = new XMLSwingTree(domDoc);
               tree = new JTree(swingTree.getSwingTree());

               tree.getSelectionModel().setSelectionMode(
               TreeSelectionModel.SINGLE_TREE_SELECTION);


               //Create the scroll pane and add the tree to it.
               treeView = new JScrollPane(tree);

               // Add the scroll pane to the content pane.
               contentPane.add(treeView, BorderLayout.CENTER);

               // XML tree won't render unless we do this (tdn)
               contentPane.setVisible(false);
               contentPane.setVisible(true);

            } catch (Exception ex) {

               ex.printStackTrace();

            } // end try-catch block

         } // end run()

      } ).start();

   } // end doLoadBinary

   /* ACTION EVENT(s) */

   /**
    * Take action appropirate on the events
    *
    * @param e the event to take action on
    */
   public void actionPerformed(ActionEvent e) {

      Object evt = e.getSource();

      if ( evt == exit ) {

         System.exit( 0 );

      } // end if

      if ( evt == assignSchema ) {

         doAssignSchema();

      } // end outer if

      if ( evt == receiveXSBCFile ) {

         doReceiveFile();

      } // end if

      // Load an .xsbc file, decompress and view (tdn)
      if ( evt == loadBinary ) {

         doLoadBinary();

      } // end if
      if ( evt == receiveEncodedFile ) {

         doFECDecoding();

      } // end if

   } // end actionPerformed()


   /* MAIN METHOD */

   /**
    * Method called when used as an application.  Borrowed from Java Sun's
    * GUI tutorial.
    *
    * @param args the command line initial argument
    *
    * @exception Exception if GUI didn't create successfully
    */
   public static void main( String args[] ) {

      // Execute a job on the event-dispatching thread creating this GUI
      try {

         javax.swing.SwingUtilities.invokeAndWait( new Runnable() {

            public void run() {

               createAndShowGUI();

            } // end run()

         } ); // end invokeAndWait()

      } catch ( Exception e ) {

         System.err.println( "createAndShowGUI() didn't successfully complete" +
         " due to:\n" );

         e.printStackTrace();

      } // end try-catch block

   } // end main()

   /* THREAD */

   /** To be ran as a thread from Server Simulation */
   public void run() {

      //Make sure we have nice window decorations.
      JFrame.setDefaultLookAndFeelDecorated( true );

      // Create and set up the window
      JFrame f = new JFrame( "XSBC w/ FEC RECEIVER  S I M U L A T I O N" );
      f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

      //content panes must be opaque
      f.setJMenuBar( createMenuBar() );
      f.setContentPane( createContentPane() );

      // Show the MOVES Logo as an icon
      f.setIconImage( getFDImage() );

      //Display the window.
      f.setSize(500, 600);
      f.setLocation(750, 100);
      f.setVisible( true );

   } // end run()
```

```
/**
 * Create the GUI and show it.  For thread safety, this method is invoked
 * from the event-dispatching thread (Java Sun's Tutorial on GUIs)
 */
private static void createAndShowGUI() {

   //Make sure we have nice window decorations.
   JFrame.setDefaultLookAndFeelDecorated( true );

   // Create and set up the window
   JFrame f = new JFrame( "XSBC w/ FEC RECEIVER  S I M U L A T I O N" );
   f.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

   //Create and set up the content pane.
   ReceiverSimulation rs = new ReceiverSimulation();

   //content panes must be opaque
   f.setJMenuBar( rs.createMenuBar() );
   f.setContentPane( rs.createContentPane() );

   //Display the window.
   f.setSize(500, 600);
   f.setLocation(750, 100);
   f.setVisible( true );

   // Show the MOVES Logo as an icon
   f.setIconImage( rs.getFDImage() );

} // end createAndShowGUI()

/* INNER CLASS(s) */

/**
 * Inner class used to filter schema files
 */
public class SchemaFileFilter extends FileFilter {

   /** String representation of a file extension */
   private String extension = null;

   /** String representation of a file name */
   private String fn = "";

   // Accept all directories and all gif, jpg, or tiff files.
   public boolean accept(File f) {
      if (f.isDirectory())
         return true;

      extension = null;

      fn = f.getName();

      int i = fn.lastIndexOf('.');

      if (i > 0 &&  i < fn.length() - 1) {
         extension = fn.substring(i+1).toLowerCase();

      }

      if (extension != null) {
         if (extension.equals("xsd"))
            return true;

         else
            return false;
      }

      return false;

   } // end accept()
```

```
      /** The description of this filter */
      public String getDescription() {

         return "Just Schema Files";

      } // end getDescription()

   } // end inner class SchemaFileFilter

   /** Inner class used to filter binary files */
   public class XMLFileFilter extends FileFilter {

      /** String representation of a file extension */
      private String extension = null;

      /** String representation of a file name */
      private String fn = "";

      // Accept all directories and all gif, jpg, or tiff files.
      public boolean accept(File f) {
         if (f.isDirectory())
            return true;

         extension = null;

         fn = f.getName();

         int i = fn.lastIndexOf('.');

         if (i > 0 &&  i < fn.length() - 1) {
            extension = fn.substring(i+1).toLowerCase();

         }

         if (extension != null) {
            if (extension.equals("xsbc") || extension.equals("gz"))
               return true;

            else
               return false;
         }

         return false;

      } // end accept()

      /** The description of this filter */
      public String getDescription() {

         return "Just Binary Files";

      } // end getDescription()

   } // end inner class XMLFileFilter

} // end class file ReceiverSimulation.java
```

4. Supporting Code Called from the Server/Client Application

The following two files are supporting files called by the SenderSimulation.java and ReceiverSimulation.java files. They take care of the implementation of compressing XML files by the XSBC process and FEC encoding and the converse of uncompressing and decoding. As noted, the

SenderSimulation invokes the ReceiverSimulation as a thread process, so executing the SenderSimulation readies all four files to process XSBC and FEC functions.

a. XsbcSerializerFecEncoder.java

```
/* Program:      Extensible Markup Language (XML) Schema-based Binary
 *               Compression (XSBC) w/Forward Error Correction (FEC)
 *
 * Author:       Duane T. Davis
 * Modifier:     LT Terry D. Norbraten, USN [comments/changes labeled (tdn)]
 *
 * Created on:   Janurary 30, 2004: 0000
 * Modified on:  December 02, 2004: 1428
 *
 * Course:       MV 0810 (Thesis Research)
 *               Summer 2004
 *
 * File:         XsbcSerializerFecEncoder.java
 *
 * Compiler:     netBeans IDE 3.6 (External), J2SDK 1.4.2_06
 * O/S:          Windows XP Home Ed (SP 1).
 *
 * Description:  Reads and XML document and compresses it with XSBC and/or
 *               GZip and either writes to file or sends over the net in
 *               either a TCP or a UDP option.  Also adds an option to encode
 *               with FEC which is only used in the UDP option only (tdn).
 *
 * Information:  Using fec-1.0.3 created by Justin F. Chapweske of Onion
 *               Networks Inc. and xsbc-0.91.1 created by Alan Hudson of
 *               Yumetech Inc.  Both are Java open source APIs.
 */
package xsbc_fec.comparison.process;

// Standard library imports
import java.io.ByteArrayOutputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.IOException;
import java.io.OutputStream;

import java.net.DatagramSocket;
import java.net.DatagramPacket;
import java.net.InetAddress;
import java.net.Socket;
import java.net.SocketException;
import java.net.UnknownHostException;

import java.util.zip.GZIPOutputStream;

// Application specific local imports
import com.onionnetworks.fec.io.FECFile;
import com.onionnetworks.fec.io.FECParameters;

import com.onionnetworks.util.Buffer;
import com.onionnetworks.util.Util;

import org.web3d.xmsf.xsbc.DocumentWriter;
import org.web3d.xmsf.xsbc.datatypes.SimpleType;

//import execution.*;
```

```
/**
 * Reads and XML document and compresses it with XSBC and/or GZip and either
 * writes to file or sends over the net in either a TCP or a UDP option
 * (tdn). </p>
 *
 * @author <a href="mailto:dtdavis@nps.navy.mil">Duane T. Davis</a>
 * Modified by <a href="mailto:tdnorbra@nps.navy.mil">Terry D. Norbraten</a>
 * </p>
 */
public class XsbcSerializerFecEncoder {

   /* DATA MEMBER(s) */

   /**
    * An FEC wrapper (K of them) to hold encoded data (byte[] packets of size
    * 1024 B) that will be encoded with repair data before for transmission
    */
   private Buffer[] sourceBuffer;

   /** An outstream to contain byte arrays for sending packets */
   private ByteArrayOutputStream baos;

   /** A UDP packet to carry our data out */
   private DatagramPacket packet,
                          repairIndices;

   /** A UDP socket for sending and receiving data over the net */
   private DatagramSocket udpSocket;

   /** Stream for sending binary files to disc */
   private DataOutputStream dos;

   /** The document writer that serializes .xml files */
   private DocumentWriter writer;

   /** Instance of the FEC encoding file object */
   private FECFile fecF;

   /** The parameters we set for FEC encoding */
   private FECParameters params;

   /** Our .xml file object */
   private File inFile;

   /** Our serialized .xml file to .xsbc */
   private File outFile;

   /** Output stream for writing a serialized .xml file */
   private FileOutputStream fos;

   /** GZip stream for send further compressed binary files */
   private GZIPOutputStream zipStream;

   /** Input stream to monitor TCP socket connection */
   private InputStream iStream;

   /** Out stream for sending a binary file */
   private OutputStream oStream;

   /** A TCP socket to transmit data */
   private Socket tcpSocket;

   /** String name of our host */
   private String host;

   /** A flag indicating a UDP option */
   private boolean udpOption;

   /** Flag indicating FEC option */
   private boolean encode;
```

```
/** Indicate the port any socket is connected to */
private int port;

/** Indicate result of serializing an .xml file */
private int result;

/** Our stripe order (repair indice) arrays */
private int[] stripeOrderN;

/* CONSTRUCTOR(s) */

/**
 * Creates a new instance of XsbcSerializerFecEncoder for a TCP option
 *
 * @param inFilePath (including filename) of input XML file
 * @param encode option to use FEC encoding
 * @param p the FEC parameters set if FEC option was selected
 */
public XsbcSerializerFecEncoder( String inFilePath, boolean encode,
                                 FECParameters p ) {

   // TCP option
   this( inFilePath, false, encode, p, null, 0 );

} // XsbcSerializer (default)

/**
 * Creates a new instance of XsbcSerializerFecEncoder for a UDP option
 *
 * @param inFilePath (including filename) of input XML file
 * @param udp true if a UDP option is invoked
 * @param encode true if the file is to be FEC encoded, false if not
 * @param p the FECParameters to utilize for FEC file encoding
 * @param host the host address to connect to
 * @param port the port number to bind to
 */
public XsbcSerializerFecEncoder( String inFilePath, boolean udp,
                                 boolean encode, FECParameters p,
                                 String host, int port ) {

   result = 0;
   udpOption = udp; // tdn
   this.encode = encode;
   params = p;
   setHost( host );
   setPort( port );

   // Check for FEC encoding option
   if ( !encode ) {

      // Serialize without FEC Encoding
      setDocumentWriter( inFilePath ); // tdn

   } else {

      sourceBuffer = new Buffer[ p.getN() ];
      stripeOrderN = new int[ p.getN() ];
      System.out.println( "From Serializer constructor: " + p ); // DEBUG

      doFECEncoding( inFilePath );

   } // end if-else block

} // end UDP option constructor
```

```
   /* PUBLIC METHOD(s) */

   /**
    * Writes the Xsbc data to a specified file
    *
    * @param outFilePath (including name) of the output file
    * @param zipped true if output file is to be compressed (zipped)
    *
    * @exception if outstreams have failed
    */
   public int writeFile(String outFilePath, boolean zipped) {
      try {
         outFile = new File(outFilePath);
         fos = new FileOutputStream(outFile);

         if (zipped)
            result = writeZippedStream( fos );
         else
            result = writeStream( fos );

         fos.close();
         return result;
      }  catch (Exception e) {
         e.printStackTrace();
         return(0);
      } // catch (Exception e)
   } // end writeFile()

   /**
    * Writes the XSBC data to a network address by opening a socket.  Closes the
    * socket once write is complete.  FEC encoded packets will be send via the
    * sendFECEncoded() method (tdn) </p>
    *
    * @param host the string name of the host
    * @param port the port on host to connect to
    * @param zipped true if the file is to be compressed, false if not
    *
    * @exception UnknownHostException
    * @exception SocketException
    * @exception IOException
    */
   public int writeNetwork( String host, int port, boolean zipped ) {

      // The setHostAndPort() method doesn't get called if the file isn't
      // selected for FEC encoding, so we set these parameters here (tdn)
      setHost( host );
      setPort( port );

      // Check for the UDP option
      if ( !udpOption ) {

         try  {

            tcpSocket = new Socket( InetAddress.getByName(host), port );
            iStream = tcpSocket.getInputStream();
            oStream = new DataOutputStream( tcpSocket.getOutputStream() );
            oStream.flush();

            if (zipped)
               result = writeZippedStream(oStream);
            else
               result = writeStream(oStream);

            oStream.close();

            // loop until server closes connection
            while (tcpSocket.isConnected());
         }  catch (Exception e) {
//          Utilities.traceOut("Unable to establish network connection for
//            XSBC-compressed archive transfer");
            e.printStackTrace();
         } // catch (Exception e)

      // Send packet via UDP (tdn)
      } else {

          try {

             udpSocket  = new DatagramSocket(); // tdn
             baos = new ByteArrayOutputStream(); //tdn
             oStream = new DataOutputStream( baos ); // tdn
             oStream.flush();

             if (zipped)
                result = writeZippedStream(oStream);
             else
                result = writeStream(oStream);

             oStream.close();

          } catch ( SocketException se ) {

             se.printStackTrace();

          } catch ( IOException ioe ) {

             ioe.printStackTrace();

          } // end try-catch block

      } // end outer if-else block

      return result;

   } // end writeNetwork()

   /* GETTER(s) / SETTER(s) */

   /**
    * Set the host for sending FEC Encoded files
    *
    * @param host the host to send to
    */
   public void setHost( String host ) {

      this.host = host;

   } // end setHost()

   /**
    * Retrieve the host to send FEC Encoded files to
    *
    * @return the host to send to FEC Encoded files to
    */
   public String getHost() {

      return host;

   } // end getHost()

   /**
    * Set the port number for sending FEC Encoded files
    *
    * @param port the port number to connect to
    */
   public void setPort( int port ) {

      this.port = port;

   } // end setPort()
```

```
/**
 * Retrieve the port # to send FEC Encoded files to
 *
 * @return the port # to send FEC Encoded files to
 */
public int getPort() {

   return port;

} // end getPort()

/* PRIVATE METHOD(s) */

/**
 * Set up the XSBC document writer
 *
 * @param file the incoming .xml file to serialize
 */
private void setDocumentWriter( String file ) {

   try {
      inFile = new File(file);
      writer = new DocumentWriter(inFile.getPath());
   } catch(Exception ioe) {
      ioe.printStackTrace();
   } // catch (IOException)

} // end setDocumentWriter()

/**
 * Performs the FEC Encoding of an already saved to disc .xsbc file before
 * sending out in a stream
 *
 * @param fileName the .xsbc file to encode
 *
 * @exception IOException if FECFile did not form properly and/or
 * the encoding process failed
 */
private void doFECEncoding( String fileName ) {

   System.out.println( "File to encode with FEC: " +
                       new File( fileName ).getName() );

   // Create the FEC file out of the .xsbc file in read only mode with the
   // default parameters K=16, N=32 and repair packet size=1024 B
   try {

      fecF = new FECFile( new File( fileName ), "r", params );

   } catch ( IOException ioe ) {

      ioe.printStackTrace();

   } // end try-catch block

   // Create an array of N numbered indices
   for ( int i = 0; i < stripeOrderN.length; i++ ) {

      stripeOrderN[ i ] = i;

   } // end for

   // Create an FEC Buffer[] as a wrapper holding N byte[] 1024 B in size.
   // These will be the repair packets of which will expand to N encoded
   // repair packets.  We only need any K subset per block on the receiving
   // end to decode the orig. file (not to be confused with Datagram
   // Packets). (tdn)
   for ( int i = 0;  i < params.getN(); i++ ) {

      sourceBuffer[ i ] = new Buffer( params.getPacketSize() );
   } // end for

   // Not sure why we do this save only to mix the repair index ordering
   // to prove that any K of N repair packets per block is all that's needed
   // to restore the orig. file.  Yes, confirmed we just need any K
   // subset of the orig. N expanded repair packets and this method, by
   // shuffling the indices, will prove that any K subset of the N will do
   // the trick.

   // Perform a Fisher-Yates shuffle of the stripeOrder[] numbered indices.
   // Repair packets (N of them for each block, last block may contain less)
   // will be encoded according to the stripe ordering produced by this
   // shuffle.
   Util.shuffle( stripeOrderN );

   for ( int i = 0; i < params.getBlockCount(); i++ ) {

      try {

         // Encode packets in each file partitioned block (i) as repair
         // packets according to the shuffled order of their indices
         // (FEC code)
         fecF.read( sourceBuffer, i, stripeOrderN );

      } catch ( IOException ioe ) {

         ioe.printStackTrace();

      } // end try-catch block

      // TODO: Implement TCP option of sending FEC encoded data

      // Send each block (index information along with each N repair packet)
      // along it's way.  Once decoded by the receiver, the receiver will
      // determine GZipped status and handle accordingly via UDP only.
      sendFECEncoded();

      System.out.println( "Sending block number: " + i );

   } // end for

} // end doFECEncoding()

/**
 * Sends only FEC encoded packets down stream one block at a time along with
 * the repair packet index of each repair packet ordering via UDP only
 *
 * @exception SocketException if the Datagram Socket didn't open
 * @exception UnknownHostException if host couldn't be determined
 * @exception IOException if the socket couldn't send the packet
 */
private void sendFECEncoded() {

   // Send repair packet indicie info first
   try {

      // Open our UDP socket
      udpSocket  = new DatagramSocket();

      // Prepare a BAOS for sending repair packet indice info.
      baos = new ByteArrayOutputStream();

   } catch ( SocketException se ) {

      se.printStackTrace();

   } catch ( IOException ioe ) {

      ioe.printStackTrace();

   } // end try-catch block
```

```
      // Write the repair index information to a buffer
      for ( int i = 0; i < params.getN(); i++ ) {

         // Write an N amount of the shuffled indices to the stream
         baos.write( stripeOrderN[ i ] );

      } // end for

      try {

         // Flush the bytes out of the buffer into the stream
         baos.flush();

      } catch ( IOException ioe ) {

         ioe.printStackTrace();

      } // end try-catch block

      /******************** Send Encoded Repair Packets *********************/

      try {

         // Prepare the packet to send indice info.
         repairIndices = new DatagramPacket( baos.toByteArray(),
         baos.toByteArray().length, InetAddress.getByName( getHost() ),
         getPort() );

      } catch ( UnknownHostException uhe ) {

         uhe.printStackTrace();

      } // end try-catch block

      // Send the N repair index packets first
      try {

         // Send repair packet indice info.
         udpSocket.send( repairIndices );

//          for ( int ix = 0; ix < repairIndices.getLength(); ix++ )
//
//             System.out.println( "Sending FEC encoded data packet index: " +
//             stripeOrderArray[ ix ] + " in packet size: "
//             + baos.toByteArray().length ); // DEBUG

      } catch ( IOException ioe ) {

         ioe.printStackTrace();

      } // end try-catch block

      try {

         baos.close();

      } catch ( IOException ioe ) {

         ioe.printStackTrace();

      } // end try-catch block

      // Send each block's worth of N repair packets
      for ( int i = 0; i < sourceBuffer.length; i++ ) {

         try {
            // Load the packet with encoded data of just repair packet size and
            // send each to keep under the 1500 byte MTU as specified by NPS
            // research associate Don McGregor.  I know I'm making new Datagram
            // packets with each iteration of the for loop, but with the FEC
            // Buffer[], this is the only way to extract encoded data into byte
            // array form for the Datagram Packet for each repair packet that
            // is to be transmitted.  So here, we send N packets of encoded data
            // per block.
            packet = new DatagramPacket( sourceBuffer[ i ].getBytes(),
            params.getPacketSize(), InetAddress.getByName( getHost() ),
            getPort() ); // tdn

         } catch ( UnknownHostException uhe ) {

            uhe.printStackTrace();

         } // end try-catch block

//         System.out.println( "Sending FEC encoded data packet " + i + " " +
//                             sourceBuffer[ i ] ); // DEBUG

         // Send the repair encoded packets
         try {

            // Send encoded data
            udpSocket.send( packet );

//            System.out.println( "Successfully sent encoded repair packet " + i +
//            " of size: " + packet.getLength() + " B to: " +
//            packet.getAddress().toString() + " on port: " +
//            packet.getPort() ); // DEBUG

         } catch ( IOException ioe ) {

            ioe.printStackTrace();

         } // end try-catch block

      } // end for

      // Close the socket
      udpSocket.close();

   } // end sendFECEncoded()

   /**
    * Writes the XSBC data to an output stream via the TCP or the UDP option.
    * Does not open stream. </p>
    *
    * @param stream the outstream to write data to
    *
    * @exception FileNotFoundException
    * @exception IOException
    */
   private int writeStream( OutputStream stream ) {

      try {

         SimpleType.setCompressionMethod(
         SimpleType.COMPRESSION_METHOD_SMALLEST_NONLOSSY);
         dos = new DataOutputStream(stream);
         writer.serialize(dos);
         dos.flush();
         dos.close();

         // Check for the UDP option to send this file.  If the file is being
         // saved to disc, this will not be selected by the caller (tdn)
         if ( udpOption ) {

            // TODO: Implement streaming FEC here
            sendUDP();

            System.out.println( "Sending data non-GZipped.... "
            + baos.toByteArray().length + " B" ); // DEBUG

         } // end if

         // Indicate success (tdn)
         return( 1 );

      }  catch ( Exception e ) {

//        Utilities.traceOut("Unable to write XSBC to stream");
         e.printStackTrace();

         return( 0 );

      } // catch (Exception e)

   } // end writeStream()

   /**
    * Writes the XSBC data to an output stream in zipped form. Does not open
    * the stream. </p>
    *
    * @param stream the output stream to write compressed Xsbc to
    *
    * @exception IOException
    */
   private int writeZippedStream(OutputStream stream) {
      try {
         zipStream = new GZIPOutputStream(stream);
         SimpleType.setCompressionMethod(
         SimpleType.COMPRESSION_METHOD_SMALLEST_NONLOSSY);
         dos = new DataOutputStream(zipStream);
         writer.serialize(dos);
         dos.flush();
         dos.close();

         // Check for the UDP option to send this file.  If the file is being
         // saved to disc, this will not be selected by the caller (tdn)
         if ( udpOption ) {

            // TODO: Implement streaming FEC here

            sendUDP();

            System.out.println( "Sending data GZipped.... " +
            baos.toByteArray().length + " B" ); // DEBUG

         } // end if

         // Indicate success (tdn)
         return(1);
      } catch (Exception e) {
//        Utilities.traceOut("Unable to write XSBC to Zip Stream");
         e.printStackTrace();
         return(0);
      } // catch (Exception e)

   } // end writeStreamZipped()

   /**
    * Sends the packet via UDP best effort delivery
    *
    * @exception UnknownHostException if the packet didn't form
    * @exception IOException if the packet didn't send    *
    */
   private void sendUDP() {

      try {
         // This is able to happen b/c the baos was part of the orig. data
         // output stream that the serializer requires for serializing .xml
         // documents and it's the baos that holds the serialized data.
         packet = new DatagramPacket( baos.toByteArray(),
         baos.toByteArray().length, InetAddress.getByName( getHost() ),
         getPort() );

         udpSocket.send( packet ); // tdn

      } catch ( UnknownHostException uhe ) {

         uhe.printStackTrace();

      } catch ( IOException ioe ) {

         ioe.printStackTrace();

      } // end try-catch block

   } // end sendUDP()

} // end class file XsbcSerializerFecEncoder.java
```

b. XsbcTransactionFecDecoder.java

```
/* Program:      Extensible Markup Language (XML) Schema-based Binary
 *               Compression (XSBC) w/Forward Error Correction (FEC)
 *
 * Author:       Duane T. Davis
 * Modifier:     LT Terry D. Norbraten, USN [comments/changes labeled (tdn)]
 *
 * Created on:   February 01, 2004: 0006
 * Modified on:  December 02, 2004: 1428
 *
 * Course:       MV 0810 (Thesis Research)
 *               Summer 2004
 *
 * File:         XsbcTransactionFecDecoder.java
 *
 * Compiler:     netBeans IDE 3.6 (External), J2SDK 1.4.2_06
 * O/S:          Windows XP Home Ed (SP 1)
 *
 * Description:  Receives GZipped streams of XSBC data, reads the data,
 *               uncompresses the .xsbc file and writes out to cache the
 *               reconstructed .xml document.  FEC Encoded data will only be
 *               received via UDP.
 *
 * Information:  Using fec-1.0.3 created by Justin F. Chapweske of Onion
 *               Networks Inc. and xsbc-0.91.1 created by Alan Hudson of
 *               Yumetech Inc.  Both are Java open source APIs.
 */
package xsbc_fec.comparison.process;

// Standard Library imports
import java.io.ByteArrayInputStream;
import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.IOException;

import java.util.zip.GZIPInputStream;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.Socket;

// Application specific imports
import com.onionnetworks.fec.FECCode;
import com.onionnetworks.fec.FECCodeFactory;

import com.onionnetworks.fec.io.FECParameters;

import com.onionnetworks.util.Buffer;
import com.onionnetworks.util.Util;

import org.apache.log4j.BasicConfigurator;

import org.web3d.xmsf.xsbc.BlockDataInputStream;
import org.web3d.xmsf.xsbc.TableManager;
import org.web3d.xmsf.xsbc.XMLWriter;
import org.web3d.xmsf.xsbc.XSBCReader;
import org.web3d.xmsf.xsbc.datatypes.SimpleType;

/**
 * Receives GZipped streams of XSBC data, reads the data, uncompresses the
 * .xsbc file and writes out to cache the reconstructed .xml document.
 *
 * @author <a href="mailto:dtdavis@nps.navy.mil">Duane T. Davis</a>
 * Modified by <a href="mailto:tdnorbra@nps.navy.mil">Terry D. Norbraten</a>
 * </p>
 */
public class XsbcTransactionFecDecoder implements Runnable {

   /* DATA MEMBER(s) */

   /** Input stream for UDP packets */
   private ByteArrayInputStream bais;

   /** A block data input stream */
   private BlockDataInputStream bdis;

   /** A buffered input stream */
   private BufferedInputStream bis;

   /** An FEC Buffer[] holding byte arrays (packets of decoded data size 1024 B)
    * to be read into the DOM reader for .xml tree rendering
    */
   private Buffer[] repairBuffer;

   /** Our UDP packets to receive data */
   private DatagramPacket packet,
                          fecPacket,
                          indicePacket;

   /** A UDP socket to connect to the net */
   private DatagramSocket udpSocket;

   /** Our FEC Code used for decoding */
   private FECCode fec;

   /** Our FEC parametes for decoding */
   private FECParameters params;

   /** An outfile object to go to cache */
   private File outFile;

   /** File pointer to a schema location */
   private File schemaLoc;

   /** For writting a file to disc */
   private FileOutputStream fos;
   /** Our inbound GZip stream */
   private GZIPInputStream zipStream;

   /** Our TCP input stream */
   private InputStream iStream;

   /** A TCP socket to connect to the net */
   private Socket tcpSocket;

//   /** Point to the data cache folder */
//   private final String DATACACHE = "/dataweb/results/";

   /** Point to the data cache folder (tdn) */
   private final String DATACACHE = "C:\\Documents and Settings\\Terry\\" +
   "My Documents\\My Files\\NPS\\Courses\\MV 0810 Thesis Work\\Java Code\\" +
   "XSBC_FEC\\xsbc_fec\\dataweb\\results\\";

   /** Point to the cache file folder */
   private final String CACHEFILE = "missionResultsXsbc";

   /** Location of AVCL schema for use by XSBC routines */
//    public static final String AVCLSCHEMA = "../build/Scripts/AVCL.xsd";
   public static final String AVCLSCHEMA = "C:/auv/AuvWorkbench/Scripts/AVCL.xsd";

   /** A Table manager to hold the XML schema tree */
   private TableManager tableManager ;

   /** Our XML document writer */
   private XMLWriter writer;

   /** Our XSBC document reader */
   private XSBCReader reader;

   /** Flag to denote if incoming packet is FEC encoded */
   private boolean isEncoded;

   /** Flag to denote if incoming packet is in GZip format */
   private boolean isGZipped;

   /** Buffers to hold FEC data */
   private byte[] encodedKData,
                  encodedNData;

   /** Indentify a transaction by this id */
   private int transactionId;

   /** Container for all block received indices */
   private int[] repairIndexes;

   /* CONSTRUCTOR(s) */

   /**
    * Creates a new instance of XsbcTransaction
    *
    * @param socket the TCP socket over which the transaction will take place
    * @param p the FEC parameters to utilize if FEC encoding was selected
    * @param e true if FEC encoding was selected
    * @param id the trasnaction id of a generated mission output file
    */
   public XsbcTransactionFecDecoder( Socket socket, FECParameters p, boolean e,
                                     int id) {
      this( null, null, id, false, e, p );
      this.tcpSocket = socket;
   } // XsbcTransactionFecDecoder (default)
```

```
/**
 * Creates a new instance of XsbcTransactionFecDecoder for a UDP option
 *
 * @param socket the UDP socket over which the transaction will take place
 * @param file the XML schema used to reconstruct the orig. .xml tree
 * @param id the trasnaction type of this process
 * @param g a flag to denote if incoming packet is in GZip format
 * @param e a flag to denote if incoming packet is FEC encoded
 * @param p our given FEC Parameters
 */
public XsbcTransactionFecDecoder( DatagramSocket socket, File file, int id,
                                  boolean g, boolean e, FECParameters p ) {

   udpSocket = socket;
   schemaLoc = file;
   transactionId = id;
   isGZipped = g;
   isEncoded = e;
   params = p;

   if ( udpSocket != null ) {

      // Sufficient capacity to receive UDP packets
      packet = new DatagramPacket( new byte[ 1500 ], 1500 );

   } // end if

   // This function produces a verbose DEBUG output that shows Block
   // Counts and the decoding process.  If commented out, it will throw a
   // couple of log4j error about not being initialized properly.  They
   // can be ignored (tdn)
   BasicConfigurator.configure();

} // end UDP option constructor

/* THREAD PROCESS */

/**
 * Processes the XSBC transaction
 */
public void run() {

   SimpleType.setCompressionMethod(
   SimpleType.COMPRESSION_METHOD_SMALLEST_NONLOSSY);

   // Check for a TCP or a UDP socket
   if ( tcpSocket != null ) {

      System.out.println("External network connection made. Begin XSBC " +
      "file upload");
      try // IOException
      {
         // TODO: Implement decoding of FEC encoded data through TCP

         iStream = tcpSocket.getInputStream();
         zipStream = new GZIPInputStream(iStream);
         schemaLoc = new File(AVCLSCHEMA);
         System.out.println("XSBC using schema located at " +
         schemaLoc.toURL());
         tableManager = new TableManager(schemaLoc.toURL());
         reader = new XSBCReader(tableManager);

         bis = new BufferedInputStream(zipStream);
         bdis = new BlockDataInputStream(bis);
         outFile = new File(DATACACHE + CACHEFILE + transactionId
         + ".xml");
         fos = new FileOutputStream(outFile);
         writer = new XMLWriter(fos);
         reader.read(bdis, writer);

         fos.flush();
         fos.close();
         tcpSocket.close();
         System.out.println("XSBC file successfully uploaded");
      } catch (Exception e) {
         System.out.println("Exception while attempting to upload file");
         e.printStackTrace();
      } // catch (Exception e)

   } else {

      runUDPProcess();

   } // end outer if-else block

} // end run()

/**
 * Receives a UDP packet and processes the data back into an .xml file from
 * either a GZipped/XSBC format or just from XSBC format.
 *
 * @exception IOException if there are socket reception, GZIPInputStream, and
 * or file writing issues
 * @exception MalformedURLException if there is trouble finding the schema
 */
private void runUDPProcess() {

   System.out.println("Receiving file via UPD.  Begin XSBC file upload");

   // Check for FEC encoding option state
   if ( !isEncoded ) {

      try  {

         System.out.println("Waiting for compressed data....");
         udpSocket.receive( packet );
         System.out.println( "Received packet size: " + packet.getLength()
         + " B" ); // tdn

      } catch (IOException ioe) {

         System.out.println("Exception while attempting to upload file");
         ioe.printStackTrace();

      } // end try-catch block

      bais = new ByteArrayInputStream( packet.getData() );

   } else {

      doFECDecoding();

      // Grab the repairBuffer's decoded contents
      bais = new ByteArrayInputStream( encodedKData );

   } // end if-else block

   // Check for GZip flag
   if ( isGZipped ) {

      try {

         zipStream = new GZIPInputStream(bais);
         bis = new BufferedInputStream(zipStream);

      } catch ( IOException ioe ) {

         ioe.printStackTrace();

      } // end try-catch block

   } else {
      bis = new BufferedInputStream(bais);

   } // end if-else block

   try {

      System.out.println("XSBC using schema located at " +
      schemaLoc.toURL());
      tableManager = new TableManager(schemaLoc.toURL());

   } catch ( MalformedURLException murle ) {

      murle.printStackTrace();

   } // end try_catch block

   reader = new XSBCReader(tableManager);
   bdis = new BlockDataInputStream(bis);
   System.out.println("Receiving compressed data .... ");
   outFile = new File(DATACACHE + CACHEFILE + transactionId
   + ".xml");

   // Open the file output stream
   try {

      fos = new FileOutputStream(outFile);

   } catch ( FileNotFoundException fnfe ) {

      fnfe.printStackTrace();

   } // end try-catch block
   writer = new XMLWriter(fos);

   // Write the missionResults doc
   try {

      reader.read(bdis, writer);
      fos.flush();
      fos.close();

   } catch ( IOException ioe ) {

      ioe.printStackTrace();

   } // end try-catch block
   udpSocket.close();
   System.out.println("XSBC file successfully uploaded");

} // end runUDPProcess()

/**
 * Performs preparations for FEC decoding to enable MissionOutputXX.xml
 * file saving to cache.
 *
 * @exception IOException if the socket couldn't read packets
 */
private void doFECDecoding() {

   //create our fec code
   fec = FECCodeFactory.getDefault().createFECCode( params.getK(),
   params.getN() );

   // Prepare to receive our K * Block Count repair indices.  These
   // are just byte format integers for each Datagram packet.  Again,
   // we will select our K required subset from the N received from
   // each block.
   indicePacket = new DatagramPacket( new byte[ params.getN() ],
   params.getN() );
   // Storage for each block's K subset of N repair indexes that
   // will be recieved via UDP.
   repairIndexes = new int[ params.getK() * params.getBlockCount() ];

   // Receive each block repair index information
   for ( int ix = 0; ix < params.getBlockCount(); ix++ ) {

      try {

         // Receive our repair index info.
         udpSocket.receive( indicePacket );

      } catch ( IOException ioe ) {

         ioe.printStackTrace();

      } // end try-catch block

      // Feed index information into this BAIS per block
      bais = new ByteArrayInputStream( indicePacket.getData() );

      System.out.println( "Receiving block number: " + ix );

      // Extract K * Block Count packet elements that make up our
      // repair indices.
      for ( int jx = 0; jx < params.getK(); jx++ ) {

         // Read in our K subset indice information
         repairIndexes[ jx ] = bais.read();

//                System.out.println( "Received repair packet index: " +
//                repairIndexes[ jx ] ); // DEBUG

      } // end inner for

   } // end outer for

   /***************** Receive Encoded Repair Packets *****************/

   // Prepare this packet for reception of a repair packet each
   // containing 1024 bytes of encoded data
   fecPacket = new DatagramPacket( new byte[ params.getPacketSize() ],
   params.getPacketSize() );

   // Container for the K required/selected encoded repair packets
   encodedNData = new byte[ params.getN() * params.getPacketSize() *
                            params.getBlockCount() ];
   encodedKData = new byte[ params.getK() * params.getPacketSize() *
                            params.getBlockCount() ];

   // Receive each block's worth of a K subset of N encoded repair packets
   // for storage
   for ( int ix = 0; ix < params.getK() * params.getBlockCount(); ix++ ) {

      // Receive a block of data to encode
      try {

         // Receive our encoded data
         udpSocket.receive( fecPacket );

      } catch ( IOException ioe ) {

         ioe.printStackTrace();

      } // end try-catch block

      // Copy N encoded packets into a byte[] for K selection
      System.arraycopy( fecPacket.getData(), 0, encodedNData, ix *
      fecPacket.getLength(), fecPacket.getLength() );

   } // end for
   System.out.println( "Expanded Block Size: " +
   params.getExpandedBlockSize() ); // DEBUG

   // Extract a K subset of the N encoded repair packets for decoding
   System.arraycopy( encodedNData, 0, encodedKData, 0, params.getK() *
                     params.getPacketSize() );

//      for ( int ix = 0; ix < params.getK(); ix++ ) {
//
//         System.out.println( "Receive FEC encoded data packet " +
//         ix + " " + encodedKData[ ix ] ); // DEBUG
//
//      } // end for

   // This will hold our K subset of encoded repair packets
   repairBuffer = new Buffer[ params.getK() * params.getBlockCount() ];

   // Put the encoded data into an FEC Buffer wrapper for decoding
   for ( int ix = 0; ix < params.getK() * params.getBlockCount(); ix++ )
      repairBuffer[ ix ] = new Buffer( encodedKData, ix *
      params.getPacketSize(), params.getPacketSize() );

   // Finally, decode the repair packets into the file
   fec.decode( repairBuffer, repairIndexes );

} // end doFECDecoding()

} // end class file XsbcTransactionFecDecoder.java
```

D. SUMMARY

The previous code examples complete the illustration of one crude way to implement and optional FEC filter along with the XSBC process. More code is supplied with the CD that this research produced. As was mentioned previously, open source and open standards are desirable to be leveraged as it is much cheaper and can be developed freely by others because the source code is readily available. Proprietary developments do not have to be relied upon with their overhead in expense that could be utilized much more efficiently elsewhere in current DOD procurement practices.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Binary Interchange Workshop. Retrieved December 2004 from http://www.w3.org/2003/08/binary-interchange-workshop/Report.html

Blahut, R.E., *Fast Algorithms for Digital Signal Processing*. Reading, MA: Addison Wesley, 1985.

Brutzman, D., McGregor, D., DeVos, D.A., Lee, C.S., Amsden, S., Blais, C., Davis, D.T., Filiagos, D. and Hitner, B., (2004, January 23), “XML-Based Tactical Chat (XTC): Requirements, Capabilities and Preliminary Progress”, Technical Report NPS-MV-2004-001, Naval Postgraduate School, Monterey, CA. Retrieved December 2004 from http://www.movesinstitute.org/xmsf/projects/XTC/XmlTacticalChat2004January28.pdf

Canonical XML, (2001, March 15), Canonical XML version 1.0, W3C Recommendation retrieved December 2004 from: http:/www.w3.org/TR/2001/REC-xml-c14n-20010315

Chapweske, J.F., (2000), “Forward Error Correction Performance”, Paper contained in the docs folder of the binary fec-1.0.3.zip file in .pdf format titled “FECPerformance”.

Clark, V., “Sea Power 21: Projecting Decisive Joint Capabilities,” *Proceedings*, October 2002. Retrieved December 2004 from http://www.chinfo.navy.mil/navpalib/cno/proceedings.html

Codiga, D.L., J.A. Rice, and P.S. Baxley, 2004. “Networked Acoustic Modems for Real-time Data Delivery from Distributed Subsurface Instruments in the Coastal Ocean: Initial System Development and Performance,” Journal of Atmospheric and Oceanic Technology, 21(2), 331-346.

Cyclic Redundancy Check. Retrieved December 2004 from http://en.wikipedia.org/wiki/Cyclic_redundancy_check

Fermat’s Theorem. Retrieved December 2004 from http://mathworld.wolfram.com/FermatsLittleTheorem.html

French, D., “Navy preps XML policy: Policy seeks to drive data interoperability”, *Federal Computer Week*, 09 December 2002. Retrieved December 2004 from http://www.fcw.com/fcw/articles/2002/1209/news-xml-12-09-02.asp

Hamming, R.W., *Computers and Society*. New York: McGraw-Hill, 1972.

Hamming, R.W., *Coding and Information Theory*, 2nd edition. Englewood Cliffs, New Jersey: Prentice-Hall, 1986.

Hamming, R.W., (1995, April 18 and 20), "Lecture: Coding Theory I and II" from the Hamming on Hamming: Learning to Learn lecture series, a distance education course taught online and sponsored by the Naval Postgraduate School's MOVES Institute. Retrieved December 2004 from http://online.cs.nps.navy.mil/DistanceEducation/MovesContent/Courses/HammingLearningToLearn/session.html

Hamming, R.W., (1995, 21 April), "Lecture: Error Correcting Codes", Naval Postgraduate School taped lecture series.

Hamming, R.W., *The Art of Doing SCIENCE and Engineering: Learning to Learn*. Amsterdam B.V., The Netherlands: Gordon and Breach Science Publishers, 1997.

Hawkings, D.L., and Van Leuvan, B.C., "An Xml-based Mission Command Language for Autonomous Underwater Vehicles (AUVs)", Master's Thesis, Naval Postgraduate School, Monterey, CA, June 2003.

Kraft's Inequality. Retrieved December 2004 from http://www.nist.gov/dads/HTML/kraftsinqlty.html

Lamparter, B., Boehrer, O., Effelsberg, W. and Turau, V., "Adaptable Forward Error Correction for Multimedia Data Streams", Technical Report TR-93-909, Department for Mathematics and Computer Science, University of Mannheim, 1993, accessed December 2004 from http://www.informatik.uni-mannheim.de/techberichte/lib/TR-93-009.pdf

Lee, C.S., "NPS AUV Workbench: Collaborative Environment for Autonomous Underwater Vehicles (AUV) Mission Planning and 3D Visualization", Master's Thesis, Naval Postgraduate School, Monterey, CA, March 2004. Retrieved December 2004 from: http://www.movesinstitute.org/Theses/leethesis.pdf

Lin, S., Costello, D.J. and Miller, M., *Error Control Coding: Fundamentals and Applications*. New Jersey: Prentice-Hall, 1983.

Luby, M., Vicisano, L., Gemmell, J., Rizzo, L., Handley, M., and Crowcroft, J., "Forward Error Correction (FEC) Building Block", RFC 3452, December 2002.

Luby, M., Vicisano, L., Gemmell, J., Rizzo, L., Handley, M., and Crowcroft, J., "The Use of Forward Error Correction (FEC) in Reliable Multicast", RFC 3453, December 2002.

Lucent Technologies. Retrieved December 2004 from http://www.bell-labs.com

netBeans™ IDE. Retrieved December 2004 from http://www.netbeans.info/downloads/download.php?a=b&p=1

NPS AUV Workbench Flyer. Retrieved December 2004 from http://www.movesinstitute.org/xmsf/projects/AUV/NpsAuvWorkbenchFlyerFebruary2004.pdf

Onion Networks Inc. Retrieved December 2004 from http://www.onionnetworks.com/about.php

Perrig, A., Canetti, R., Song, D. and Tygar, J., "Efficient and Secure Source Authentication for Multicast", Network and Distributed System Security Symposium, NDSS 2001, pp. 35-46, February 2001. Retrieved December 2004 from http://www.ece.cmu.edu/~adrian/projects/tesla-ndss/ndss.pdf

Potts, M., "Forward-thinking your capacity", (1999, June 7) *Telephony*. Last accessed December 2004 from http://telephonyonline.com/ar/telecom_forwardthinking_capacity/

Reed-Solomon Codes, (n.d.), "An introduction to Reed-Solomon codes: principles, architecture and implementation", Last accessed December 2004 from http://www.4i2i.com/reed_solomon_codes.htm

Reimers, S.P., "Towards Internet Protocol Over Seawater (IP/SW): Forward Error Correction (FEC) Using Hamming Codes For Reliable Acoustic Telemetry", Master's Thesis, Naval Postgraduate School, Monterey, CA, September 1995. Retrieved December 2004 from http://www.cs.nps.navy.mil/research/auv/ipoversw/

Request for Comments (RFC), retrieved December 2004 from http://www.rfc-editor.org/

Rice, J.A., "Seaweb Network for FRONT Oceanographic Sensors", a National Oceanographic Partnership Program FY-00 annual report retrieved December 2004 from http://www.onr.navy.mil/sci_tech/ocean/reports/docs/nopp_funded/00/00_seaweb.pdf

Rivest, R., “The MD5 Message-Digest Algorithm”, RFC 1321, April 1992.

Rizzo, L., “Effective erasure codes for reliable computer communication protocols”, ACM Computer Communication Review, Vol. 27, n. 2, April 1997. Also available as DEIT Technical report LR-970115, retrieved December 2004 from http://www.iet.unipi.it/~luigi/fec.html

Rizzo, L., "On the feasibility of software FEC", DEIT Technical Report LR-970131. Retrieved December 2004 from http://www.iet.unipi.it/~luigi/softfec.ps

Rizzo, L. and Vicisano, L., “A Reliable Multicast data Distribution Protocol based on software FEC Techniques”, Proceedings of the Fourth IEEE Workshop on High Performance Communication Systems (HPCS ’97), 23-25 June 1997, Chalkidiki, Greece, IEEE.

Rizzo, L., Sources for an erasure code based on Vandermonde matrices. Retrieved December 2004 from http://info.iet.unipi.it/~luigi/vdm98/vdm980702.tgz

Rizzo, L. and Vicisano, L., “RMDP: An FEC-based Reliable Multicast Protocol for Wireless Environments”, Mobile Computer and Communication Review, Vol. 2, n. 2, April 1998. Retrieved December 2004 from http://info.iet.unipi.it/~luigi/mccr6.ps

Sall, K., (2003, November 21), “How the US Federal Government is Using XML: An Overview of Selected US Federal Agency Efforts”. Proceedings by deepX Ltd. retrieved December 2004 from http://www.silosmashers.com/pdfDocuments/Sall-How-Gov-Uses-XML.pdf

Sandoz, P., Pericas-Geertsen, S., Kawaguchi, K., Hadley, M., and Pelegri-Llopart E., “Fast Web Services”, Internet article, August 2003. Retrieved December 2004 from http://java.sun.com/developer/technicalArticles/WebServices/fastWS/

Serin, E., “Design and Test of the Cross-Format Schema Protocol (XSFP) for Network Virtual Environments”, Master’s Thesis, Naval Postgraduate School, Monterey, CA, March 2003. Retrieved December 2004 from http://www.movesinstitute.org/Theses/Serinthesis.pdf

Shannon, C.E., “A Mathematical Theory of Communication”. Reprinted with corrections from *The Bell System Technical Journal,* Vol. 27, pp. 379–423, 623–656, July and October 1948. Retrieved December 2004 from http://cm.bell-labs.com/cm/ms/what/shannonday/paper.html

Siddiqui, B., (2002, Sep. 18 and Oct. 09), “XML Canonicalization (Part I and II)”. Published by O’Reilly Media, Inc. Retrieved December 2004 from http://www.xml.com/pub/a/sw/2002/09/08c14n.html

Stirling’s Formula. Retrieved December 2004 from http://www.sosmath.com/calculus/sequence/stirling/stirling.html

Wagner, N.R., (2003, June) “The Laws of Cryptology with Java Code”, Unpublished online manuscript last accessed December 2004 from http://www.cs.utsa.edu/~wagner/lawsbookcolor/laws.pdf

XML Binary Characterization Working Group. Information last retrieved December 2004 from http://www.w3.org/XML/Binary/

XML Binary Interchange Workshop Report, (2003, September 24-26), “Report From the W3C Workshop on Binary Interchange of XML Information Item Sets”, a workshop conducted in Santa Clara, CA, retrieved December 2004 from http://www.w3.org/2003/08/binary-interchange-workshop/Report.html

XMSF Projects Page retrieved December 2004 from http://www.movesinstitute.org/xmsf/xmsf.html

Yumetech Inc. Retrieved December 2004 from http://www.yumetech.com

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, VA

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, CA

3. RDML Patrick Dunne, USN
   Naval Postgraduate School
   Monterey, CA

4. Dr. Donald P. Brutzman
   Naval Postgraduate School
   Monterey, CA

5. LT Terry D. Norbraten, USN
   U.S. Navy
   Aptos, CA

6. Mr. Richard G. Norbraten
   New York Times
   Manhattan, NY

7. Erik Chaum
   NUWC
   Newport, RI

8. David Bellino
   NPRI
   Newport, RI

9. Dick Nadolink
   NUWC
   Newport, RI

10. VADM Roger Bacon (Ret.)
    Naval Postgraduate School
    Monterey, CA

11. K.C. Stangl
    NAVAIR
    Paxtuxent River, MD

12. Margaret Bailey
Sonalysts
Waterford, CT

13. Dr. Dan Boger
Naval Postgraduate School
Monterey, CA

14. RDML Elizabeth Hight, USN
OPNAV 61-C4
Washington DC

15. Doug Backes
COMPACFLT Science Advisor
Pearl Harbor, HI

16. LT Andrew Hurvitz, USN
FNMOC
Monterey, CA

17. ENS Darin Keeter, USN
FNMOC
Monterey, CA

18. CAPT David Titley, USN
FNMOC
Monterey, CA

19. CAPT Scot Miller, USN
NCTSI
San Diego, CA

20. Dr. Peter Denning
Naval Postgraduate School
Monterey, CA

21. Dr. Alan Washburn
Naval Postgraduate School
Monterey, CA

22. Hans Widmer
COMSUBPAC Science Advisor
Pearl Harbor, HI

23. Alan Hudson
Yumetech Inc.
Seattle, WA

24. CDR Susan Matusiak, USN
Naval Postgraduate School
Monterey, CA