2015-03

Evaluation of efficient XML interchange (EXI) for large datasets and as an alternative to binary JSON encodings

Hill, Bruce W.

Monterey, California: Naval Postgraduate School

https://hdl.handle.net/10945/45196

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

EVALUATION OF EFFICIENT XML INTERCHANGE (EXI) FOR LARGE DATASETS AND AS AN ALTERNATIVE TO BINARY JSON ENCODINGS

by

Bruce W. Hill

March 2015

Thesis Advisor:	Don Brutzman
Co-Advisor:	Don McGregor

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

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** March 2015	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE EVALUATION OF EFFICIENT XML INTERCHANGE (EXI) FOR LARGE DATASETS AND AS AN ALTERNATIVE TO BINARY JSON ENCODINGS			**5. FUNDING NUMBERS** W4V02
6. AUTHOR Bruce W. Hill			
7. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME AND ADDRESS Commander Navy Information Dominance Forces (COMNAVIDFOR) 115 Lake View Parkway Suffolk, VA 23435			**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. IRB Protocol number: N/A.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE** A

13. ABSTRACT

Current and emerging Navy information concepts, including network-centric warfare and Navy Tactical Cloud, presume high network throughput and interoperability. The Extensible Markup Language (XML) addresses the latter requirement, but its verbosity is problematic for afloat networks. JavaScript Object Notation (JSON) is an alternative to XML common in web applications and some non-relational databases.

Compact, binary encodings exist for both formats. Efficient XML Interchange (EXI) is a standardized, binary encoding of XML. Binary JSON (BSON) and Compact Binary Object Representation (CBOR) are JSON-compatible encodings. This work evaluates EXI compaction against both encodings, and extends evaluations of EXI for datasets up to 4 gigabytes. Generally, a configuration of EXI exists that produces a more compact encoding than BSON or CBOR. Tests show EXI compacts structured, non-multimedia data in Microsoft Office files better than the default format.

The Navy needs to immediately consider EXI for use in web, sensor, and office document applications to improve throughput over constrained networks. To maximize EXI benefits, future work needs to evaluate EXI's parameters, as well as tune XML schema documents, on a case-by-case basis prior to EXI deployment. A suite of test examples and an evaluation framework also need to be developed to support this process.

14. SUBJECT TERMS Extensible Markup Language (XML), Efficient XML Interchange (EXI), JavaScript Object Notation (JSON), Compact Binary Object Representation (CBOR), Binary JSON (BSON), data serialization, data interoperability			**15. NUMBER OF PAGES** 135
			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** UU

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

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

EVALUATION OF EFFICIENT XML INTERCHANGE (EXI) FOR LARGE DATASETS AND AS AN ALTERNATIVE TO BINARY JSON ENCODINGS

Bruce W. Hill
Lieutenant, United States Navy
B.S., University of Notre Dame, 2008

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN NETWORK OPERATIONS AND TECHNOLOGY

from the

NAVAL POSTGRADUATE SCHOOL
March 2015

Author: Bruce W. Hill

Approved by: Don Brutzman
Thesis Advisor

Don McGregor
Co-Advisor

Dan Boger
Chair, Department of Information Sciences

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Current and emerging Navy information concepts, including network-centric warfare and Navy Tactical Cloud, presume high network throughput and interoperability. The Extensible Markup Language (XML) addresses the latter requirement, but its verbosity is problematic for afloat networks. JavaScript Object Notation (JSON) is an alternative to XML common in web applications and some non-relational databases.

Compact, binary encodings exist for both formats. Efficient XML Interchange (EXI) is a standardized, binary encoding of XML. Binary JSON (BSON) and Compact Binary Object Representation (CBOR) are JSON-compatible encodings. This work evaluates EXI compaction against both encodings, and extends evaluations of EXI for datasets up to 4 gigabytes. Generally, a configuration of EXI exists that produces a more compact encoding than BSON or CBOR. Tests show EXI compacts structured, non-multimedia data in Microsoft Office files better than the default format.

The Navy needs to immediately consider EXI for use in web, sensor, and office document applications to improve throughput over constrained networks. To maximize EXI benefits, future work needs to evaluate EXI's parameters, as well as tune XML schema documents, on a case-by-case basis prior to EXI deployment. A suite of test examples and an evaluation framework also need to be developed to support this process.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

A2/AD	Anti-access/Area Denial
AIS	Automated Identification System
Ajax	Asynchronous JavaScript and XML
API	Application Programming Interface
BSON	Binary JSON
CBOR	Concise Binary Object Representation
C2	Command and Control
DTD	Document Type Definition
EXI	Efficient XML Interchange
FI	Fast Infoset
GB	Gigabyte
GPS	Global Positioning System
GPX	Global Positioning System Exchange
HTTP	Hypertext Transfer Protocol
IANA	Internet Assigned Numbers Authority
IETF	Internet Engineering Task Force
IIS	Internet Information Services
IoT	Internet of Things
IT	Information Technology
JSON	JavaScript Object Notation
KB	Kilobyte
LZ77	Lempel-Ziv 1977 Algorithm
LZMA	Lempel-Ziv Markov-chain Algorithm
MB	Megabyte
NCW	Network-Centric Warfare
NoSQL	Not only Structured Query Language
NOW	Network-Optional Warfare
ODF	Open Document Format
OOXML	Office Open Extensible Markup Language
RF	Radio Frequency

RSS	Really Simple Syndication
SATCOM	Satellite Communications
SGML	Standard Generalized Markup Language
SOA	Service Oriented Architecture
SQL	Structured Query Language
UBJSON	Universal Binary JSON
UTF-8	Universal Transformation Format 8-bit
WAP	Wireless Application Protocol
WBXML	WAP Binary XML
W3C	World Wide Web Consortium
WML	Wireless Markup Language
WoT	Web of Things
X3D	Extensible 3D Graphics Standard
XBC	XML Binary Characterization
XML	Extensible Markup Language
XML-RPC	XML Remote Procedure Call
XMPP	XML Messaging and Presence Protocol

ACKNOWLEDGMENTS

I would like to thank the following individuals for their support through the many stages of this undertaking: Don Brutzman—I thought a quick response code would say it best, and I truly hope your work with EXI continues to infect the system! Don McGregor—you are a man of few words, but when you do speak, you have an uncanny tendency to make difficult concepts seem very, very simple. I couldn't have asked for a better sounding-board throughout the process. Takuki Kamiya, John Schneider, Rumen Kyusakov, Daniel Peintner and the rest of the EXI working group—thanks for letting a rookie join in the process and teaching me a great deal about computers, algorithms, and standards along the way. Steve Debich—it's been great working and commiserating with you through this process. And Terra—thanks for putting up with my stress!

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Network-centric warfare hinges on the dual criteria of interoperable systems and timely information flows, but oftentimes these criteria are at odds. The Extensible Markup Language (XML) is a common data format for systems communicating across the web, and its structure and ubiquity enable both syntactic and semantic interoperability. However, it is verbose by design (Bos, 2001). In network environments with low bandwidth and intermittent connectivity, verbose file formats are undesirable. File compression before transmission alleviates this issue, but until recently, there has been no open standard for an XML-specific compression algorithm. In 2004, the World Wide Web Consortium (W3C) began addressing this issue, and in 2014 it released the Efficient XML Interchange (EXI) Format Recommendation (Schneider, Kamiya, Peintner, & Kyusakov, 2014). EXI is an alternate encoding of XML data, which, in some cases, results in files that are less than 10 percent, the size of the original XML file (Bournez, 2009). The potential performance and impacts of EXI in limited-throughput network environments, such as those regularly encountered by afloat naval units, is significant and warrants investigation through use-case analysis.

A. PROBLEM STATEMENT

Networks with intermittent uptime and limited throughput must maximize utilization of network resources. These constraints are problematic for applications where large, monolithic data sets must be transmitted. In such scenarios, file compression prior to transmission can decrease transfer time. EXI compaction performance has been measured for XML file sizes up to 100 megabytes (MB), and results indicate it offers higher compaction rates than the Zip and Gzip algorithms. However, some applications require larger amounts of data to be transferred, and research is necessary to evaluate the performance of EXI compression in such cases.

Similarly, XML is commonly used by web-based applications and web services to transmit small data exchanges between a client and server as requested by the user. The performance of EXI for these cases is well documented (Bournez, 2009; Kyusakov, 2014;

D. Peintner, Kosch, & Heuer, 2009; J. Schneider, personal communication, October 20, 2014). However, many web applications use the JavaScript Object Notation (JSON) format as an alternative to XML, and compact binary encodings of JSON also exist. Given that these binary encodings are analogous to EXI, research is necessary to evaluate the performance and tradeoffs between the two formats.

The EXI standard defines multiple configuration options that can dramatically affect the performance of an EXI encoding, both in speed and compactness. Though much research and time has been invested in developing the EXI standard and its descriptions of the various configurations, there is little open documentation of best practices or empirical results that can inform developer decisions for configuring an EXI codec.

B. PURPOSE AND MOTIVATION

This research re-evaluates and extends previous work of the W3C EXI Working Group and NPS researchers to explore the use of EXI in transmitting large XML files and database information over networks with intermittent connectivity (Bournez, 2009; Snyder, McGregor, & Brutzman, 2009; Snyder, 2010). It extends the NPS test-corpus of XML documents to include additional datasets representative of real-world naval operations. First, it measures EXI compaction performance for large native XML files from 100 MB to 4 gigabytes (GB) and compares the results to conventional compression algorithms. Second, it measures compaction of both EXI and binary JSON encodings for small data transfers representative of those found in web applications. During both of these threads of inquiry, the methodology explores various configurations of the EXI codec and XML schemas to assess their impact on EXI encodings with the goal of identifying best practices supported by real-world data.

C. RESEARCH QUESTIONS

Four research questions form the basis for inquiry and organization in this thesis. They are:

- For large XML files between 100MB and 4GB, does the EXI format offer compaction rates superior to Gzip?
- Does the EXI format offer compaction rates superior to Gzip-compressed JSON?
- Does the EXI format offer compaction rates superior to BSON and CBOR?
- What impact do the various EXI configurations have on compaction rates?

D. THESIS ORGANIZATION

This thesis is comprised of five chapters. Chapter I provides an introduction, purpose and outline of research questions. Chapter II is a review of relevant scholarly and professional literature on: the United States Navy's information and communications environments; data serialization formats and interoperability; and binary and compressed encodings for compact representation of serialized data. Chapter III describes the methodology for data collection and format conversions for comparing binary and compressed encodings of XML and JSON. Chapter IV presents and discusses empirical test results. Chapter V lists major findings of this research and recommendations for future work.

A portion of Chapter II (specifically, Chapter II, Section D) addressing XML Verbosity and binary XML encodings is a collaborative, co-written product and also appears in *The Role of Efficient XML Interchange in Navy Wide Area Network Optimization* (Debich, 2015). For additional information on EXI and its role in wide area network (WAN) optimization, refer to that document.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND AND RELATED WORK

To meet increased demands and network-centric and cloud-based operating models for pushing real-time information to forward-deployed tactical and afloat units, the Navy must meet the twin criteria of data interoperability and efficient delivery. The Extensible Markup Language (XML) provides the former at the expense of the latter, but the Efficient XML Interchange (EXI) standard allows for compact binary representations of XML data. An alternative to XML commonly used in web-based applications is JavaScript Object Notation (JSON), which is commonly used for data interchange in web applications and in some modern database systems, and also has spurred development of multiple binary encodings. This chapter reviews the literature concerning the Navy's information landscape and technical limitations for high-bandwidth applications, as well as the tradeoffs between and performance of XML, JSON and their derivative binary encodings.

A. THE NAVY INFORMATION LANDSCAPE

Several operational and architectural concepts are defining and changing the road ahead for U.S. Navy communications. Each presents potential benefits, as well as distinct challenges in afloat environments.

1. Network-Centric Warfare

The network-centric warfare (NCW) concept is, and has been for several years, fundamental to the U.S. Navy's operations and information technology (IT) strategy. As originally proposed, NCW derives value "from the content, quality, and timeliness of information moving between nodes on the network" (Cebrowski & Garstka, 1998, "The Business of America", para. 1). "Speed of Command" (Cebrowski & Garstka, 1998, "How Can the Military," para. 4) is one such benefit of modern information flows, and stems directly from timeliness of information. With rapid access to networked information, military commanders can make more effective decisions.

Though modern network speeds continually increase, information timeliness is a problematic and ever-advancing goal for afloat units. First, the capacity of network links available to U.S. Navy ships pales in comparison to terrestrial lines. Second, advances in distributed sensor networks and unmanned vehicles will likely continue to drive exponential growth in data transiting U.S. Navy networks from ship to shore (Chief of Naval Operations for Information Dominance, 2013, p. 10). Large-scale data transfer paired with significant time delay over limited-bandwidth links makes information dominance at sea a challenge.

2. Cloud-Based Architecture

Cloud-based architectures are gaining ground in the public sector, and the U.S. Navy, along with the Department of Defense, is following suit and incorporating the cloud in current policy and IT planning (Department of the Navy Chief Information Officer, 2013). Navy IT leadership expects to reap benefits in performance, security and life cycle costs through a cloud architecture compatible with other military branches and government agencies (Deputy Chief of Naval Operations for Information Dominance [DCNO(ID)], 2014). However, the same document addresses the unique needs of afloat units in a cloud architecture, particularly complications associated with anti-access/area denial (A2/AD) threats.

3. Sensor Networks

The U.S. Navy's Task Force Cloud Charter identifies sensor and combat systems data as initial focus points for the U.S. Navy cloud (DCNO[ID], 2014). These systems tend to generate small, repetitive transmissions, which, in aggregate, offer commanders unprecedented situational awareness. Such awareness, however, comes at a price: as the U.S. Navy deploys more systems to its networks, the collective data load stresses network capacity (Porche, Wilson, Johnson, Tierney, & Saltzman, 2014). Porche et al. (2014, p. 14) note that afloat analysts are crippled by slow transfer speeds when working with sensor data—download limits often prevent them from ever seeing the information. Current cloud approaches may be insufficient to meet the challenging topologies and performance variability facing afloat networks.

4. Network-Optional Warfare

Though network-centric warfare is the basis for much of the fleet's current command and control (C2), A2/AD threats mean that the electromagnetic spectrum will likely be contested in conflicts with technologically advanced opponents (Rowden, Gumataotao, & Fanta, 2015). Alternate C2 models, not predicated on high-throughput radio frequency (RF), are possible. Network Optional Warfare seeks to achieve stealth and surprise through the use of data transmission via visible light, acoustic, free-space optics and other unconventional media (Goff, 2014; Hughes, 2014). As these media are likely low-bandwidth in comparison to existing RF-based channels, efficient, compressed communications technologies are a key enabler (Brutzman, Hughes, Kline, Buettner, & Ekelund, 2014).

5. Network Limitations

The Navy draws heavily on commercially developed technologies to run its information systems. Often, those technologies are designed for use on terrestrial networks with far higher bandwidth than is available for afloat communications channels. As such, terse data exchange formats are not always first priority in commercial technology development. For perspective, a one-terabyte download over a terrestrial fiber line takes a matter of minutes, while the same download over Wideband Global SATCOM link requires a few days (Porche et al., 2014). Similarly, commercial technologies are designed for low-latency networks. Round-trip delays for transcontinental and inter-continental links are approximately 200 to 400ms, but for a ship-to-shore transmission via a satellite in geosynchronous orbit, the delay can be closer to 900ms (Bentrup, Otte, Chan, Vavrichek, & Gingras, 2012). With this degree of disparity, many commercial technologies are not viable for afloat applications.

Intermittent network availability is another issue facing Navy communications. For some applications, such as undersea and remote sensors or unmanned vehicles, continuous data transmission is not feasible or desirable. In these cases, nodes collect and store sensor data for a period of time and transmit once connected to the larger network. Fall (2003) discusses alternate techniques for improving performance in these environments, including military ad-hoc networks, through a delay-tolerant network architecture. Other work explores data mules, or mobile nodes in a static sensor network, to periodically harvest data from nodes and move it to a wide-area network access point (Anastasi, Conti, & Di Francesco, 2008). In either scenario, consolidated bursts of sensor data may spike network traffic, reflecting a tradeoff space between bandwidth consumption and sensor granularity.

6. Solution Approaches

Given these challenges facing afloat network and system designers, there are several general approaches to the Navy's limited network problem, none of which are mutually exclusive.

a. Say Less

Though a trivial solution, reducing or eliminating data transmission is a simple way to conserve network resources, though it is problematic. The approach appears in fleet procedures such as RIVER CITY and EMCON with different motivations, such as operational security and detection avoidance. However, it is incompatible with the fleet's preference for network-centric operations and shift to cloud architecture. If the American tactical advantage comes from fast information flows as suggested by Cebrowski and Garstka (1998), reducing transmissions means ceding that advantage—information ceases to be timely, the crux of NCW. Though useful in certain situations, saying less is generally not the preferred solution.

b. Buy More

Expanding the available network capacity for afloat units is a viable solution, though not without its drawbacks. Adding capacity invariably incurs major costs. The cost may come in systems development and acquisition satellite constellations, measured in billions of dollars (Chaplain, 2009). It can come from using more energy at the cost of fuel, and for battery-constrained systems such as sensor networks, additional power may be technically infeasible. Regardless of cost, additional satellite capacity may not be available when needed, and in fiscally constrained times, less costly solutions are desirable, particularly if they work in conjunction with new system acquisitions.

c. *Use It All*

Network throughput, or the amount of data passing over a network in a unit of time, is dependent on time (Kurose & Ross, 2013, p. 35). Systems that send network traffic in intermittent bursts leave the networks underutilized during downtimes and over-utilized during others, making time the critical resource. This leads to another approach to the issue: traffic shaping and network optimization, which seek to smooth out the flow of network traffic, holding excess transmissions until later, and sending them during lulls. Navy afloat units are working to adopt these technologies.

d. *Say the Same Thing in Fewer Words*

Similar to human languages and communication, digital communications provide many ways to send the same message, some more concise than others. At the simplest level, this category of approaches seeks the digital equivalents of acronyms and contractions, implemented through alternate data encodings and compression. The fundamental drawback to this approach is that the receiver of a message must be able to decode, or make sense of, the message, and they should be able to do so quickly. This work focuses on this area and expands the underlying concepts in subsequent sections.

7. Interoperability Considerations

Network-centric information sharing in a tactical environment hinges also on hundreds or thousands of networked systems communicating. Just as humans must communicate in a shared language, so must the computer systems they use. In broad terms, there are two avenues toward this interoperability: deploy homogeneous systems at all network nodes, or focus on interfaces and data formats.

Homogeneous systems are unviable for the fleet for several reasons. The Navy constantly designs and acquires new systems, which means that synergy requires forward, reverse and sideways compatibility. This is an ongoing challenge for the Navy. The challenge stems in part from human factors such as bureaucracy, acquisition procedures and the sheer size of the organization (Wong & Gonzales, 2014). Another pain point lies in technical issues—operational systems in the Navy's massive IT portfolio span decades of technology. A full technology refresh to a homogeneous baseline is prohibitively expensive and time consuming.

For the Navy, interoperability is best achieved through well-defined data formats and system interfaces. In this paradigm, modular, black-box functionality is acceptable as long as developers incorporate open specifications for the system's data format and interface. Efficient XML Interchange, the subject of this research, maintains XML's lingua franca functionality alongside significant benefits to network throughput, by providing a compact way to transmit XML.

B. DATA SERIALIZATION

When a computer exchanges data over a network, it converts the data from a non-linear form in main memory to a sequential series of bits to send over the communication medium in a process called serialization (Eck, 2011, p. 533). The receiver deserializes the bits back into a data structure in main memory, which it can then process further (Eck, 2011). It is important to note both that the in-memory data structure at either end of the communication may differ while the serialization over the medium remains constant, and that there are multiple possible serializations of any data structure. Serialization ties closely to interoperability and compression.

1. Semantic Interoperability

At the core of interoperability is the notion that an idea needs to move from one human mind to another, and that, regardless of transfer mechanism, it is semantically identical at both ends. In practice, the transfer is rarely precise; people may not express themselves clearly, and two listeners might come to different understandings of the same message. When the communication between people occurs over a network, the message is often distorted in the conversion from idea in mind to serialized bits on a wire.

In computing, semantic interoperability is the ability to clearly convey meaning regardless of its exact data format (Sheth, 1999). Semantically interoperable systems facilitate clear conveyance of ideas between the humans using them—in a network-centric operating paradigm, digital semantics enhance mission capability through higher-quality information. As Shannon (1948) notes, semantics in communication is a separate issue from the engineering issue of syntax. Though clear semantic exchange is a cornerstone of the NCW and information dominance concepts, this work focuses on its prerequisite: syntactic interoperability.

2. Syntactic Interoperability

Syntactic interoperability, analogous to human language, is the process of taking two semantically identical messages, and converting them into a compatible set of bits that can be ingested and exported by heterogeneous computers (Sheth, 1999). The in-memory format on each computer may be different, as long as the computer can make the necessary conversions.

3. Coding

Encoding is the process of transforming one message into another, semantically identical yet syntactically different, message, and doing so in a predefined manner (Salomon, 2008). Decoding reverses the processing into the original message, and a device, system or program that enacts either change is a codec (Salomon, 2008). This work compares two broad categories of encodings: plain text and binary.

Plain-text encodings store data as a sequence of human-readable characters, including numbers ("3" or "9"), letters ("x" or "Z") and special symbols (">" or "%"). Computers, however, can only store binary 0s and 1s, so a preset sequence of bits represents each character inside the computer. For example, in a plain-text encoding scheme, a computer stores the number "13" as the character "1" followed by the character "3." Under the common Universal Transformation Format 8-bit (UTF-8) scheme, this is 8 bits per character, for 16 total: 00110001 00110011 (Asanov, Oleg, Palashina, Krivonosova, & Namjittrong, 2014). As binary arithmetic differs from decimal arithmetic, the computer cannot immediately perform math operations on these bits; they must be converted to a native computer format first.

The alternative to a plain-text encoding is a binary encoding, which represents data in a format more natural to computers. “13” encoded as a basic unsigned binary value uses four bits: 1101. Though simplistic, this example illustrates that data (here the number thirteen) can be represented, or encoded, in multiple formats, and those formats are not all of identical size. Bormann & Hoffman (2013) note there are hundreds of binary encoding formats used in various computing systems. This diversity contributes greatly to the challenges of syntactic interoperability.

4. Data Types

A data type is a set of distinct values with related properties (International Organization for Standardization [ISO], 2007). Data types are a basic method that programmers use to describe their data, and help define which operations the computer performs on those bits (ISO, 2007). For example, if the number 13 is stored as a positive integer data type, a program can deduce that only mathematical functions are valid, that subtracting 14 produces an invalid negative integer, and that 13 cannot be multiplied by the letter “z.” Programming languages use data types to decide how exactly to encode data as 1s and 0s. In many cases, there are multiple data types syntaxes for the same abstract idea. A timestamp in some systems is stored as the number of seconds since January 1, 1970; in others, it is stored as year, month, day, hour, minute and second according to the Gregorian calendar; another system may store a timestamp using the Julian date system. As the above discussion of encoding suggests, some data types are more compact than others carrying the same semantic value. For many data serialization scenarios, selection of data type can significantly impact the size of message transmitted.

5. Compression

Network throughput benefits from more concise encodings—fewer bits require less time to send. To this end, data compression algorithms leverage the principle that a message can be encoded in many ways, turning a given message into a semantically identical but syntactically different message of smaller size before storage or network transmission.

Since truly random data has no semantic value, or meaning, in the human mind, data useful to humans often has patterns and redundancy. Compression algorithms find those redundancies and select shorter encodings to express them (Salomon, 2008, p. 7). To do so, they need prior knowledge of the data, meaning that most specialize in certain applications, and no existing compression system is a panacea (Salomon, 2008, p. 7). This work addresses such a specific compression scheme, targeted at data formatted in the Extensible Markup Language (XML).

For some media types, such as audio or video, the receiving end need not receive exactly the message sent. Lossy compression, in which the encoding sacrifices details for conciseness, is acceptable for these applications. Much research has addressed compression methods for imagery and audio recordings that incorporate various degrees of data loss, and common file formats such as JPEG and MP3 fall in the lossy category (Gonzalez, Woods, & Eddins, 2009, p. 420). For the textual or numeric data in XML, however, lossy compression is unacceptable.

Lossless compression algorithms, however, allow perfect recreation of the original message at the receiver. Methods used in lossless compression for plain-text encodings include variable-length codes and dictionaries. Variable-length codes use shorter series of bits to represent more common characters, and dictionaries replace reoccurring sequences of characters, or words, with shorter codes (Salomon, 2008, p. 21,47).

Of particular relevance to this research is the LZ77 universal compression algorithm proposed by Lempel and Ziv (1977). The algorithm sequentially scans a data stream and uses a dictionary technique to select short representations for previously-scanned sequences of symbols (Sayood, 2005). Based on its sliding window, or the portion of the data being scanned, it favors data streams where repeated sections are relatively close together (Salomon, 2008, p. 50). LZ77, combined with variable-length Huffman coding, is the basis for the DEFLATE compression standard (Deutsch, 1996; Salomon, 2008, p. 50). DEFLATE is implemented by the common Zip and Gzip compression tools (Sayood, 2005).

Many compression algorithms, called composite algorithms, incorporate multiple techniques in series to transform data in multiple steps, giving an overall improvement in compaction (Kattan, 2010). An example is the Bzip2 algorithm, which first rearranges strings of characters into a new string in which identical characters are close together, and afterwards applies variable-length Huffman coding for compression (Salomon, 2008; Seward, 2000). The Lempel-Ziv Markov-chain Algorithm (LZMA) algorithm implemented by 7-Zip is another example of a composite compression scheme; it applies the LZ77 algorithm followed by range encoding (Kattan, 2010; Pavlov, 2014).

Compression algorithm selection for any application involves a broad tradeoff space. Compactness, or the relationship of compressed to uncompressed message size, is the simplest to measure. Another factor is compression speed, measured in the time required to compress a message, which varies between algorithms, and also between implementations and platforms for the same algorithm. Some algorithms are, speed wise, asymmetric. For example, Bzip2 decompresses much faster than it compresses (Kattan, 2010). Parallel processing and hardware implementations can also improve the speed of compression algorithms for some systems (Gilchrist, 2003; Szecówka & Mandrysz, 2009). Choice of algorithm, configuration, and implementation affect energy efficiency, a concern in mobile devices operating on batteries (Dzhagaryan, Milenkovic, & Burtscher, 2013). In general, the choice of compression algorithm is application and scenario dependent, and developers must understand their data, the available compression schemes, and what factor they wish to optimize.

C. XML AND DATA INTEROPERABILITY

The development of networked computer systems, and later the web, precipitated a need for a common, syntactically interoperable data exchange format. Without one, programmers spend significant time implementing data input and output capabilities, rather than core system function. In the early 1990s, available options included proprietary binary formats and the powerful, but complicated and little used, Standard Generalized Markup Language (Fawcett, Ayers, & Quin, 2012). To address this need, a working group of the World Wide Web Consortium (W3C) developed a simpler rendition of SGML, the Extensible Markup Language, and finalized the recommendation in 1998 (Fawcett et al., 2012). Today, XML is the W3C-endorsed lingua franca of the web.

1. Key Attributes of XML

Two core features of XML that lend it to both syntactic and semantic interoperability, are metadata and structure. Metadata adds semantic value, or meaning, to raw data, which means that with metadata, a file describes itself. In XML, metadata is added to a file via tags (simply words enclosed with < and > symbols, such as <tag>) surrounding the data they describe. An XML document arranges the tags in a hierarchical, tree based structure. To the human reader, this indicates how different pieces of data within the document relate to one another, even without associated metadata. Also, computers can easily processing XML's tree structure without prior data descriptions. Figure 1 and Figure 2, both adapted from a similar example by Fawcett et al. (2012), illustrate the difference that metadata and structure add to data. In the first figure, there is no context for the four numbers and two words to suggest what they mean or how they are connected. In the second, with metadata in XML format added, it is clear that the final three numbers are attributes of the Vandegrift, which is a frigate in the 3rd Fleet.

```
3
Vandegrift
Frigate
445
4100
22
```

Figure 1. Unstructured data about a ship without metadata. It is unclear what the purposes of the values are, and how the relate to one another.

```
<fleet number="3">
  <ship name="Vandegrift" class="Frigate">
    <length>445</length>
    <displacement>4100</displacement>
    <draft>22</draft>
  </ship>
  <!-- Other ship elements may follow ... -->
</fleet>
```

Figure 2. XML structured data about the same ship, with metadata describing the purpose of each value and how they are related.

2. Why XML Supports Interoperability

XML is the lingua franca of the web for several reasons. First, it is a proven, stable technology with broad software ecosystem supporting it. In the 16 years since the XML recommendation was first published, the W3C has created numerous supporting standards, each adding new capabilities to the core technology. Every major programming language has commonly available modules for ingesting, validating, processing and exporting XML. Second, as a result, XML is platform independent and well suited for heterogeneous environments across the web (Bos, 2001; Maeda, 2012, p. 177). By default XML includes metadata that adds semantics to the raw data, and through the XML Schema standard, it can use a common set of data types. The metadata can be scoped through namespaces to add context. Third, the modular, extensible nature of XML means that nearly any data or information can be expressed in an XML document (Bos, 2001). Finally, it is an open standard published by an international organization with many major organizations worldwide invested in continuing it at a global scale.

3. Relevant Applications of XML

XML can express an immense spectrum of information types. The following section outlines general categories of XML applications relevant to this research.

a. Web Applications and Services

XML is a key technology in the computing paradigm of web services and service-oriented architectures (SOA). In such an architecture, effectively a client-server model, a client requests data or processing from a server, and receives a response. The server is a central data repository and performs much of the processing. Since it is platform-independent, XML enables this architecture to function for nearly any combination of devices and applications.

A common technique for implementing a service architecture in web applications is Asynchronous JavaScript and XML, or Ajax. Ajax uses JavaScript code on a client device to send and receive small exchanges of XML data between the client and server, then updates the web page on the client end with the new information (Garret, 2005). The result is an interactive web application capable of many things once left to desktop applications, partly because the XML transfers encompass only the changing data, rather than refreshing the entire web page (Paulson, 2005). However, as the technology improves and network throughput increases, application developers push its limits and increase the number of transfers.

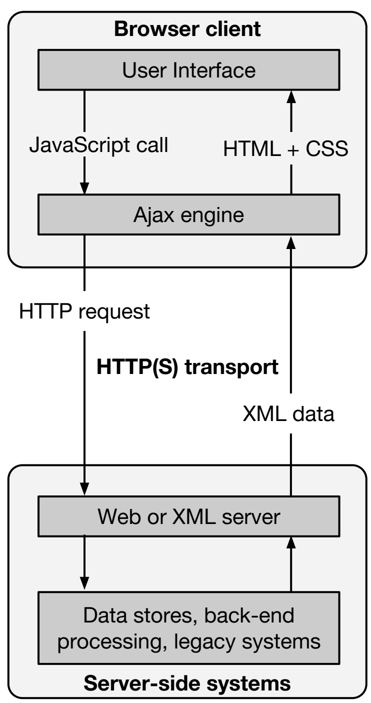

Figure 3. Ajax system process diagram showing movement of XML in a web application (after Garret, 2005 and Paulson, 2005).

The XML-based SOAP format and associated Web Services group of specifications offer a standardized protocol for moving information amongst decentralized systems (Cerami, 2002, p. 49; Gudgin et al., 2007). The specification, currently maintained by the W3C, evolved from Winer's (1999) XML-Remote Procedure Call (XML-RPC) standard, adding extended capability and conformance to the XML Schema specification (Jepsen, 2001). A SOAP message, or envelope, is generic and extensible, meaning it can hold pure data, remote procedure calls, or nearly any message. In a SOAP architecture, shown in Figure 4, both clients and servers format their messages as SOAP XML envelopes.

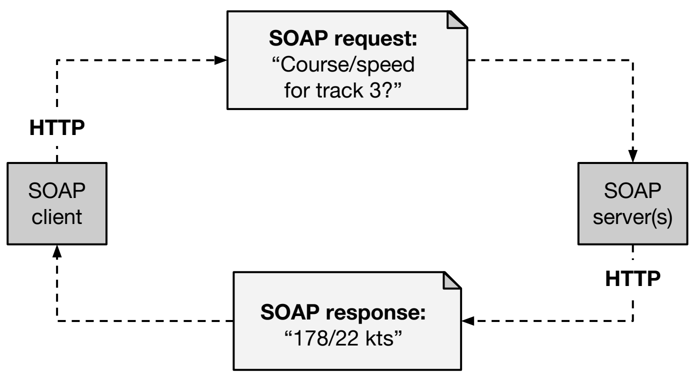

Figure 4. SOAP request and response cycle (after Cerami, 2002, p. 51).

The Web Service Description Language (WSDL) is an XML format describing what information a server provides, and how another system should request it (Cerami, 2002). In practice, a server responding to a SOAP request often builds part of the response, then passes the SOAP message to another server for additional processing prior to transmitting back to the client. Through this mechanism, a client can receive data from multiple sources via a single request. Proper WSDL descriptions can automate the process of establishing connections between computers in a SOAP environment.

Publish-subscribe systems present an alternative to the request-response model implemented in Ajax and SOAP. In a publish-subscribe environment, a client subscribes to a publisher, often via an intermediary. Afterwards, whenever the publisher has new information to disperse, it transmits the message to all subscribers without waiting for a request (Baldoni, Querzoni, & Virgillito, 2005). Prevalent in news syndications, workflow systems, and other event-driven applications, publishers often transmit in XML formats such as Really Simple Syndication (RSS) or Atom (Winer, 2003). XML's platform-independence lends itself to such systems: subscribers and publishers are often heterogeneous and connected over the web.

b. Sensor Networks

As the World Wide Web (WWW) grows in ubiquity and mobile technologies expand, new classes of devices such as remote sensors are moving onto networks, creating the Internet of Things (IoT). Applications range from radiation and biochemical detection to meteorology and oceanography (Ling, Durbha, & King, 2006; Sheth, Henson, & Sahoo, 2008). While the IoT facilitates low-level connections between devices, it does not necessarily convey the semantics of data exchanged. Data spread across isolated sensor networks is problematic to combine and process meaningfully (Sheth et al., 2008).

The Web of Things (WoT) concept seeks to address the issue of interoperability in the IoT by applying proven technologies and techniques from the Web to mobile and micro-scale devices. XML will likely be a core data exchange format in the distributed systems populating the WoT (Kangasharju, 2008, pp. 3–4). Since the metadata in XML documents merges semantics and context with raw data in a standard way, XML can bridge stovepipes of single-purpose micro devices. Combining these existing information sources in new ways may offer value beyond the original design (Raggett, 2010). Furthermore, the EXI standard, discussed in the following section, is a key enabling technology for transforming the IoT into the WoT by allowing resource-constrained devices to communicate and interoperate on the WWW (Brutzman, personal communication, March 12, 2015).

D. XML VERBOSITY

For networked systems with limited throughput, memory or battery power, XML has an Achilles heel: it is not, and was never designed to be, a compact encoding. This section summarizes the design decisions leading to XML's verbosity, and previous working addressing the issue. As described in Chapter I, subsection D, this section was co-written with S. Debich (2015) and therefore also appears in his thesis.

1. Verbose by Design

XML's designers aimed largely to streamline data transfer on the web and to eliminate the complexities of its predecessor, SGML (Kangasharju, 2008, pp. 13–14). They built XML to be simple for humans to use, write and read, implying that it must be a plain-text format - reading and debugging binary documents is near impossible without computer assistance (Bos, 2001; Bray, Paoli, & Sperberg-McQueen, 1998). The specification met its goals, and the plain-text encoding of XML is part of the reason behind it success.

A drawback to the simplicity of plain-text encoding is an associated size increase. The original XML specification states that "terseness in XML markup is of minimal importance" (Bray et al., 1998). In practice, however, computers are often the only entities processing XML messages, so for many applications, terseness is more desirable than human-readability. To clarify the issue, consider the word "Efficient." Encoded in plain text using UTF-8, each of its nine letters uses 1 byte, or 8 bits, for a total of 72 bits. In a binary encoding, those 72 bits can represent 4,722,366,482,869,645,213,696 (2^{72}) different words, which is approximately 7.9 trillion times as many words as there are in the English language (Oxford University Press, 2013). Considering this overhead, there certainly are more efficient ways to communicate the word "Efficient."

2. Generic Compression Approaches

A common approach to reducing XML transfer sizes is to apply a generic, lossless compression algorithm for which both the sender and receiver have a codec. Nearly all operating systems include software implementations of the DEFLATE algorithm and can process file formats such as Zip and Gzip, so those are common in practice. In general, Gzip compression offers significant compaction over plain-text XML, decreasing it to 50% or less of original size, though the compaction rate varies based on the XML contents (Bournez, 2009). In a few cases, however, Gzip encodings increase the file size (Bournez, 2009).

When clients on the web send hypertext transfer protocol (HTTP) requests to servers, they may specify that they accept responses in specific encodings (Fielding & Reschke, 2014, p. 40). If the server supports that encoding, it applies the requested compression prior to sending its response. The Internet Assigned Numbers Authority (IANA) maintains an official list of formats, though clients may choose to request others (Internet Assigned Numbers Authority, 2014). Both the Apache HTTP server and Microsoft Internet Information Services (IIS) server support Gzip compression without extensions (Microsoft, 2014; The Apache Software Foundation, 2014). If the server does not store compressed copies of requested resources, it must spend time applying compression before responding, which presents a series of tradeoffs based on network speed, server capabilities, and traffic load (Morse, 2005).

3. Binary Encoding Approaches

Binary encodings are another group of techniques for compacting XML data. Generic compression algorithms such as Gzip make no assumptions about the data being compressed - they work on any stream of bytes. XML documents, however, have a well-defined tree structure. Binary encoding algorithms designed specifically for XML documents leverage use a priori knowledge of this structure to achieve greater compaction than generic algorithms (Sakr, 2009). Between the publishing of the XML recommendation in 1998 and 2014, multiple standards and formats emerged, each specifying such a binary representation of XML. The following paragraphs briefly describe select methods illustrating techniques relevant to this research. Sakr (2009) provides a broader survey of formats along with comparative test results.

One category of XML compression methods works by substituting short binary tokens for longer plain-text elements in a process called tokenization. The Wireless Application Protocol (WAP), developed by the WAP Forum, was an early such solution in this group. WAP is a group of standards that define a complete architecture optimized for low-memory mobile devices connected by low-capacity wireless networks (The WAP Forum, 2000). A WAP system routes client requests through a gateway device between a web server and end client. The WAP gateway transforms the requested web page or information into a Wireless Markup Language (WML) document, an XML format. In a process called tokenization, the gateway converts longer plain-text elements from the WML document into short, binary tokens that consume less space on the final wireless hop - a format called WAP Binary XML (WBXML) (Martin & Jano, 1999). The Fast Infoset (FI) specification takes a similar approach to WBXML. It uses binary tokens to encode XML structural elements, and builds dynamic vocabulary tables that hold recurring strings of characters (ISO, 2007b).

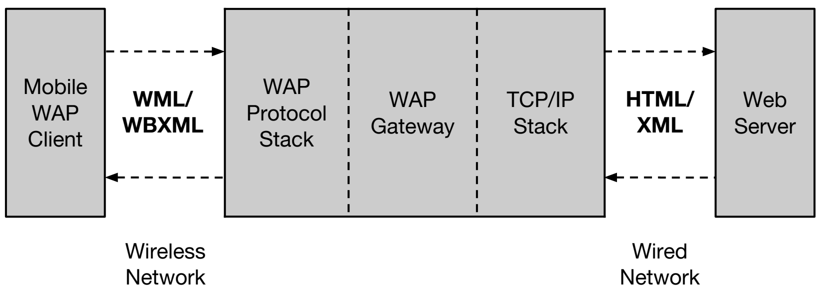

Figure 5. WAP system diagram showing XML transformation into WML via a gateway architecture (after Saha, Jamtgaard, & Villasenor, 2001).

Another common approach to XML compression is to use knowledge of both the XML structure and the mechanics of one or more generic compression algorithms to pre-process the XML document so that conventional compression algorithms such as Gzip perform better as a final processing step (Sakr, 2009). An early example of this method is the XMill compression method, developed by Liefke and Suciu (2000). It separates data from structure, reorganizes the XML document to group similar items together, and finally applies customizable semantic compression if the specific contents of an element are known (Liefke & Suciu, 2000). A possible limitation of this approach is poor compaction for small files—in Liefke and Suciu's (2000) work, files below approximately 20 kilobytes suffered from pre-processing overhead and were better compressed with only Gzip.

4. Efficient XML Interchange

Between 2000 and 2011, a wide variety of XML-specific compression techniques were developed; different methods optimized for memory footprint, difficulty of implementation, encoding speed, decoding speed, random access and compactness (Kangasharju, 2008; Sakr, 2009). In light of this complexity, two W3C working groups evaluated a variety of binary XML encodings for adoption as the consortium's recommended standard. The XML Binary Characterization (XBC) Working Group enumerated a list of target use-cases and an associated list of more than 25 requisite properties for a binary XML standard (Cokus & Pericas-Geertsen, 2005a, 2005b). Beginning in 2005, the successor Efficient XML Interchange Working Group compared a variety of candidate compression algorithms against the properties, ultimately settling on AgileDelta's Efficient XML (Le Hegaret, 2005; Schneider & Kamiya, 2011). In March 2011, the Efficient XML Interchange Format (EXI) reached official W3C recommendations status (Schneider & Kamiya, 2011).

Table 1. Minimum requirements for a binary XML format as determined by XBC Working Group (after Goldman & Lenkov, 2005).

MUST Support	MUST NOT Prevent
Directly Readable and Writable	Processing Efficiency
Transport Independence	Small Footprint
Compactness	Widespread Adoption
Human Language Neutral	Space Efficiency
Platform Neutrality	Implementation Cost
Integratable into XML Stack	Forward Compatibility
Royalty Free	
Fragmentable	
Streamable	
Roundtrip Support	
Generality	
Schema Extensions and Deviations	
Format Version Identifier	
Content Type Management	
Self Contained	

EXI capitalizes on multiple techniques to improve compression, and they vary depending on a set of EXI options passed to the codec. This section briefly discusses core compression techniques available in the EXI specification, while a discussion of various option configurations is included in the methods chapter. Kyusakov (2014) and Peintner and Pericas-Geertsen (2007) offer concise explanations of the EXI algorithm, and the official EXI Specification presents a normative description (Schneider et al., 2014).

a. Grammar-Based Encoding

XML documents have a set of rules governing their structure. While processing an XML document, an XML parser interprets the various tags as events, which can be considered as state transitions in a grammar. At any given point in a document, an XML parser can expect the next event to be one of a finite set of possibilities with different probabilities. An EXI encoder builds an internal model of the document as a set of such grammars, and assigns short, variable-length, numeric codes to each event (Schneider et al., 2014). The short codes replace longer tag structures in the encoded stream.

b. String Table

While parsing an XML document, an EXI encoder builds a table of character strings, which can be thought of as words. The first time the encoder encounters a word, it writes it in full and assigns it a short numeric code and every time afterwards, the encoder replaces the full word with the shorter code. This technique addresses the repetition of opening and closing tags in XML, as well as documents where the data itself is repetitive.

c. Data Types

If an XML document has a corresponding XML Schema describing the data type of its elements, an EXI encoder can use that information to encode data in a compact format. Without a schema, the EXI encoder treats all element contents as plain-text character strings. For example, consider an XML attribute with a value of "false." As a plain-text string, its five characters each fill 8 bits, for a total of 40 bits. However, with a schema identifying the attribute as a Boolean, the EXI encoder knows that the attribute can only take one of two logical alternative values: "true" or "false." Since 1 bit can encode 2 different values, the encoder uses that information to shorten "false" from 40 bits to 1 (Schneider et al., 2014).

d. Range Restrictions

In addition to specifying an element's data type, an XML Schema can define a set of restrictions on its possible values. For example, consider an XML attribute defined as an integer, with a value of 15,009. An EXI encoder by default encodes it with 2 bytes, or 16 bits. However, if the schema restricts the attributes values to the set of integers between 15,000 and 16,000, the EXI encoder simply writes its offset from 15,000 (i.e., the value 9), which uses only 4 bits.

e. Channelization

An EXI encoder can reorganize the contents of an XML document such that similar elements are close together in the output stream, a process called channelization. Since many compression algorithms identify recurring strings of characters, and perform better when the recurrences are localized rather than scattered throughout a document, the reorganization optimizes the document for later compression (Salomon, 2008; Schneider et al., 2014). By default, EXI applies the DEFLATE algorithm for the final compression step, although others can be used.

E. JAVASCRIPT OBJECT NOTATION

JavaScript Object Notation (JSON) is a widely used alternative to XML, often referred to as a lightweight data exchange format (G. Wang, 2011; P. Wang, Wu, & Yang, 2011). The JSON syntax stems from the syntax for defining objects in JavaScript. In 2001, Douglas Crockford self-published a standard for JSON after finding it useful in developing client-server exchanges in JavaScript web application (Crockford, n.d.). Both Ecma International and the Internet Engineering Task Force (IETF) have since published standards documents describing the format (Crockford & Bray, 2014; Ecma International, 2013). Like XML, JSON is plain-text encoded and independent of any specific programming language and thus platform independent.

1. Relationship between JSON and XML

In many situations, the JSON interchange format is functionally equivalent to XML. XML parsers build in-memory representations as a tree, whereas JSON parses into a series of key-value pairs. Wang (2011) notes that, because both formats are similar and widely used, web service developers may need to incorporate both into their systems. Some web services maintain dual application programming interfaces (API) that return data in either JSON or XML.

Given the shared purpose of data interchange and application domain of data exchange over networks, several conversion techniques between XML and JSON exist (Lee, 2011; G. Wang, 2011). In general, the conversion is not a one-to-one process; there are multiple semantically identical and syntactically correct methods for converting one to the other. Some are unidirectional conversions; others can be converted in both directions with no fidelity loss. Some conversions preserve all data while others discard information. Examples include dropping namespaces from an XML document because JSON has no concept of namespaces, or discarding unique keys for child elements. Lee (2011) provides a survey of various translation methods, none of which are standardized.

2. Tradeoff Space

The decision between XML and JSON as a data interchange format presents a multidimensional tradeoff space. This section reviews the key variables and with support from the literature where possible.

a. Expressive Differences

Due to differences in their standards and their underlying structural differences as tree and key-value pairs, JSON and XML express information differently. JSON does not use comments, while XML does (Crockford & Bray, 2014). In general, JSON is well-suited for transmitting atomic values and lists of data, but it cannot handle mixed-content messages where semantic markup intermingles with data content (Walsh, 2010). As there is no standardized method for commuting data between XML and JSON formats, interoperability between the two is not guaranteed—work by the Web3D Consortium on the X3D graphics standard has underscored this challenge in development of parallel standards supporting both formats. (Brutzman & X3D Working Group, 2014). Through XSDs, developers have more options for defining the data types using in their XML, and those data types can be further delineated through range restrictions (Evjen et al., 2007). JSON, by contrast, uses only objects, arrays, numbers, strings and Boolean values (Crockford, 2008). The post-schema-validation-infoset is a useful reference for delineating the descriptive power of XML and EXI documents (Thompson, Beech, Maloney & Mendelsohn, 2004). JSON's descriptive capability is a subset of XML and EXI (Brutzman, personal communication, March 12, 2015).

b. Quantitative Differences

JSON is generally considered to be faster than XML for data interchange, though speed in networking is multivariate in itself. Wang, Wu, and Yang (2011) measured parse times for batches of objects in both JSON and XML format, with batch sizes increasing from 100 to 10,000. In all batches, the average parsing time per object for JSON was between 24 and 34% less than for XML (P. Wang et al., 2011). Maeda (2012) measured object serialization and deserialization times to JSON, XML and other binary formats using a variety of Java libraries. JSON serialized between 3 and 19 times faster than XML, while JSON deserialization ranged from 34 times faster to 30 times slower, depending on the serialization library (Maeda, 2012). Lee (2013) investigated round-trip times for interactions in a client-server application, both in XML and JSON. The test incorporated a variety of client devices and transmission mediums (mobile through wired), and results found negligible difference in speed between the two formats (Lee, 2013). Overall, the inconsistency in results suggests that a wide array of variables affect the speed tradeoff, including programming language, hardware specification, network bandwidth and latency, and software implementation.

For applications concerned with storage, with minimizing lower-layer protocol overhead, or with per-byte pricing plans (e.g., mobile devices), file size can be an important factor. Sumaray and Makki (2012) and Maeda (2012) reported size comparisons with JSON ranging from less than 50% up to 89% the size of XML. Lee

(2013) created multiple JSON and XML files from the same abstract document, and noted cases where JSON was both larger and smaller than XML, depending on formatting conventions. Also, considering that network interactions often apply compression, a document's compressed size must be considered when comparing formats. In Lee's (2013) research, the size difference between XML and JSON, post-compression, was negligible.

c. *Qualitative Differences*

An array of unquantifiable traits also differentiates XML from JSON. Perhaps the most salient of these is the large ecosystem of standards built on the XML specification, along with their corresponding software implementations. Common data functions such as security, processing, querying and format translations are well defined and implemented. JSON, by contrast, aims for overall simplicity and processing of JSON documents is generally not standardized.

W3C standards such as XML Signature, XML Encryption, and XML Key Management Specification (XKMS) define procedures for encrypting and digitally signing XML documents to provide the confidentiality, integrity and authentication components of security (Bartel et al., 2013; Hallam-Baker & Mysore, 2005; Imamura et al., 2013). JSON has no analogs to these standards, and security functions are left for developers to implement outside of the JSON standard.

Of particular interest in this research is the notion of interoperability. For heterogeneous systems to communicate, the message exchange format must be well defined, and be capable of validation. Technologies such as XML Schema, Document Type Definition (DTD), RELAX NG and Schematron are all schema languages that allow a developer to formally describe a family of XML documents, nominally those documents that available for import or export by that system. With this definition, another system can be certain of how it should communicate, or interoperate. At present, there are no widespread technologies for formally describing a JSON document, though the language is capable of such (Nurseitov, Paulson, Reynolds, & Izurieta, 2009). Nogatz and Fruhwirth (Nogatz & Frühwirth, 2013) developed a prototype tool for converting an

XML Schema Document (XSD) into a corresponding JSON Schema. Their work built upon the unfinalized IETF JSON Schema specification, which has multiple implementations despite its draft status (Galiegue, Zyp, & Court, 2013). Robust validation capabilities may become increasingly available for JSON.

Another XML capability critical for interoperable systems is namespaces. A namespace is a method for disambiguating XML metadata tags by breaking them into groups (Fawcett et al., 2012). As an example, consider a supply system that combines inventory information between branches of the military. A tank in an army system could be a rather different item than in a navy system. Figure 6 and Figure 7 demonstrate the concept of XML namespaces for the tank example. By design, JSON does not use namespaces, relying instead on developers to use a unique set of keys in JSON documents (Crockford, 2006). XML namespaces are thus advantageous for complex system-of-system scenarios where a developer only controls a small portion. However, the W3C JSON-Linked Data (JSON-LD) recommendation adds a similar capability for adding context to JSON documents (Sporny, Longley, Kellog, Lanthaler, & Lindstrom, 2014).

```
<lineitem>
  <tank>
    <quantity>3</quantity>
  </tank>
</lineitem>
```

Figure 6. Ambiguous tank line item in a cross-service stock system. It is unclear whether the tank is a fuel tank or an armored vehicle.

```
<lineitem>
  <tank xmlns="http://navy.mil/supply">
    <quantity>3</quantity>
  </tank>
</lineitem>
```

Figure 7. Unambiguous tank line item in the http://navy.mil/supply namespace. Adding the Navy namespace suggests what sort of tank it is.

3. Binary Encoding Approaches

Though JSON is typically smaller than XML, just like XML it is plain-text encoded, which leaves room for additional compaction efforts. Just as with XML, networked systems commonly compress JSON messages with Gzip before transmitting. Studies by Lee (2013) and Gil and Trezentos (2011) show that Gzip performs well on most JSON files, compressing them to between 1% and 30% of plain-text size. One file sample in Lee's (2013) test corpus increased to 148% of original size, likely because it was less than 100 bytes as plain text and Gzip's overhead outweighed the compression benefit. The variation in compression also derives from the contents of each file, because Gzip performs well on repeating character strings. Compression and decompression with Gzip also adds processing time, which may outweigh the benefits realized from smaller file sizes, depending on performance metrics.

Just as EXI addresses the verbosity inherent to plain-text XML, multiple standards have emerged to represent plain-text JSON in a binary format, with varying objectives, strengths, and limitations (Bormann & Hoffman, 2013). Tiller and Harman (2014) researched binary serializations for JSON, including Smile, Universal Binary JSON (UBJSON) and Binary JSON (BSON) and noted that the JSON community lacks consensus on the issue. The Concise Binary Object Representation (CBOR) format is an IETF standard for serializing generic data objects, and is designed for full compatibility with the JSON format (Bormann & Hoffman, 2013). This work focuses on BSON and CBOR.

A comparison of the BSON and CBOR formats highlights a complicated set of engineering challenges. BSON is a byproduct of the MongoDB project, which uses it as the internal storage format for its Not only Structured Query Language (NoSQL) database (MongoDB Documentation Project, 2014). Its stated design goals are to be lightweight or compact, quickly traversed by computers, and efficiently encoded and decoded ("BSON," 2014). CBOR's design objectives are to: (1) be capable of encoding most standardized data formats on the network; (2) allow for compact, memory-inexpensive codec implementations; (3) require no schema for decoding; (4) serialize compactly; (5) minimize CPU usage; (6) convert to and from JSON; and (7) be extensible

(Bormann & Hoffman, 2013). Overall, the two formats emphasize processing speed and resource usage, with compaction a secondary goal.

F. CHAPTER SUMMARY

To maintain information superiority with network-centric operating concepts over constrained networks, the Navy must seek to use its limited bandwidth as efficiently as possible while maintaining interoperability between systems. XML and JSON are both commonly used data interchange formats, but neither is particularly compact. Binary representations of each exist: EXI for XML, and CBOR and BSON among others for JSON. Previous work has shown that binary data encodings can significantly improve compactness of data payloads for networked applications, thereby improving overall system throughput.

THIS PAGE INTENTIONALLY LEFT BLANK

III. METHODS

This research investigated the performance of EXI, in terms of compaction, for two broad categories: small and large files. The small-file category included applications with frequent data transactions or where JSON would likely be used as an alternative to XML, and data collection focused on comparing EXI to the binary JSON encodings CBOR and BSON for files smaller than 1MB. For the large-file category, the datasets came from applications that store large amounts of data in XML format, with the primary intent of validating and extending past performance measurements of EXI to include files larger than 100MB, up to 4GB. This chapter outlines experimentation methods for this research.

A. SINGLE-APPLICATION FOCUS

Previous evaluations of EXI compression cover a broad spectrum of use cases and applications. Some focus on EXI performance in general (across multiple applications), and some on various aspects of EXI performance for a specific application. The W3C EXI Working Group compiled a corpus of over 10,000 XML files, and culled it down to a representative subset of 88 documents (White, Kangasharju, Brutzman, & Williams, 2007). They aimed to select a candidate implementation for adoption and adaptation as the EXI standard, so the corpus reflected the standards wide scope. Snyder (2010) extended the corpus to include DOD-relevant sample files, and explored several characteristics of the XML documents to develop a generalized, predictive model for EXI compaction. Both result sets suggest that EXI never increases the size of XML files, and in practically all cases produces smaller files than Gzip.

Other research addresses EXI performance for specific applications. Kyusakov, Makitaavola, Delsing, and Eliasson (2011) and Sundin (2013) each selected a specific embedded-systems application, collected several files for that application, and measured EXI compaction. Liefke and Suciu (2000) used a similar methodology for assessing XMill performance. This single use case perspective on XML compression provides deeper, but less general, insight. The testing method used in this work falls into the single use case focus category, and drew on the methods from Liefke and Suciu (2000) and Sundin (2013) for assembling sample file sets and presenting results. For the purposes of this work, two files were considered to be from the same use case if they validated against the same XSD, and the following sections use the term *use case* interchangeably with *application*.

The author selected sample files from real-world applications to avoid two potential pitfalls in compression testing. First is the copy-paste issue, in which the tester creates a large file by repeatedly copying and appending a smaller file to itself. In most cases, the repeated data adds little information, but dramatically increases compaction. Second is the random issue, in which a researcher generates a file using a pseudo-random number generation program. The resulting file compacts poorly because it has little repetition and may not represent realistic application traffic, though in cases such as transmitting cryptographic hashes, it may. Either of the copy-paste or random issue tend to misrepresent an application's data, and therefore misrepresent the compression algorithm's performance (J. Schneider, personal communication, October 20, 2014).

B. CONFIGURATION FOCUS

The EXI standard defines many options users can set when encoding XML documents into EXI (Schneider et al., 2014). Though the default option-set works for all XML documents, it does not necessarily produce the smallest possible encoding. Other configurations offer greater compaction, faster processing, smaller implementation footprint and other advantages and disadvantages desirable in different scenarios. Also, the impact of various options depends on the nature of the XML data and the XML schema describing that data. Both the W3C's draft EXI Best Practices document and the EXI specification itself mention the impacts of the various configurations, but neither provides in-depth empirical results for those impacts (Schneider et al., 2014; Cokus & Vogelheim, 2007). Such results could offer insight for developers optimizing applications to use EXI.

This research uses a systematic approach to explore many of the EXI configuration permutations, as applied to each of six use cases. In all cases, the final recorded metric is file size, or compactness, generally achieved at the cost of additional processing time. Table 2 presents a list, with non-normative descriptions, of the EXI options explored in this research.

Table 2. A summary of the EXI options explored in this research (after Schneider et al., 2014).

Option	Description
Alignment	The alignment of grammar event codes and data content in the EXI stream. Can be either *bitpacked*, *byte-aligned* or *precompress*. In conjunction with the 'Compression' option, this field indicates which of the 4 EXI modes is to be used.
Compression	Whether or not EXI should apply DEFLATE compression to the stream, i.e., whether *compress* mode should be used. Mutually exclusive with the Alignment option, and in conjunction with the Alignment option, indicates which of the 4 primary EXI modes is to be used.
Strict	Indicates if deviations from the given schema document are acceptable. Only available for schema-informed encodings.
Schema ID	Indentifies the XML schema used to inform the encoding. If not specified, the encoding will be schemaless.
Preserve	A series of Boolean flags indicating whether or not comments, processing instructions, DTDs, namespace events/prefixes, and the lexical form of element and attribute values should be preserved.

EXI encodings may use either one of three alignment options or a compression option. Together, these effectively create a single set of four mutually exclusive 'modes' that an EXI encoder may use. In bitpacked mode, the encoder writes each event code and value to the stream, in the original document order, with the fewest possible bits rather than adding padding bits to align EXI events on byte boundaries. Since compression algorithms process data streams by bytes, bitpacked encodings do not lend themselves to post-compression (Schneider et al., 2014). A byte-aligned encoding writes each event code and value to the stream, in the original document order, adding padding as necessary to align events on byte boundaries. The EXI specification includes the byte- aligned mode primarily for troubleshooting and debugging purposes (Schneider et al., 2014). In precompress mode, the EXI encoder breaks the stream of EXI events into blocks, and then rearranges the events within each block into channels such that similar events are close together. This step optimizes the stream for additional compression via algorithms such as Zip, Gzip, Bzip2 or 7zip (Schneider et al., 2014). The encoder aligns events in precompress EXI streams on byte boundaries. An EXI encoder in compress mode performs the set of transformations defined in precompress, then applies the DEFLATE algorithm to further reduce the stream size at the expense of additional processing time. This research captured results for each of these four modes (bitpacked, byte-aligned, precompress, compress), as they tend to have the greatest impact on compactness.

If an XML document conforms to a corresponding XSD, the EXI encoder may use information from that XSD to increase compactness. This is called a schema-informed encoding, whereas an EXI encoding that uses no XSD is schemaless (Schneider et al., 2014). Schema-informed EXI streams may additionally use the strict option, which prohibits XML events not conforming to the XSD and results in more compact encodings. This research includes results for schemaless, schema-informed and schema-informed with strict option set (hereafter referred to as strict) encodings for all file samples.

For schema-informed and strict encodings, an EXI encoder can use any XSD that the XML document validates against. However, many XSDs may describe the same XML document, and the specific characteristics of the information in an XSD can affect the compactness of an EXI encoding. If an XSD indicates that an element has a data type, the EXI encoder uses that information to write the element's value in a data type-specific binary format. If the XSD does not specify a data type, the EXI encoder treats the element as a string. Also, XSDs may set maximum and minimum values that an element may take, and if that information is available, an EXI encoder uses it to write the value in fewer bits. For one application, this research presents results of schema-informed encodings for various XSDs.

C. SMALL-FILE CATEGORY

To compare compactness of plain text and binary XML encodings to corresponding JSON-based encodings, this researcher compiled a collection of files, or file sets, from each of three use cases. Each file set comprised a series of semantically equivalent JSON and XML files of varying sizes representing abstract messages from that application. Each plain-text file in a file set was encoded to multiple binary formats using various permutations of encoding options and post-compression algorithms. This section summarizes the applications, describes the XML to JSON conversion process, and lists the derivative file formats. As JSON is commonly used to transfer 'small' messages, plain-text file sizes range from 303B to 584KB in this category, hereafter referred to as the small-file category.

1. Subsetting

In web applications, a client often requests or transmits a group of objects from or to a server where all objects have the same type. Examples could be a list of server-log entries or a group of stock orders. Depending on the application and the client's needs, the group size may range from a single object to hundreds of objects or more. Sundin (2013), as well as Liefke and Suciu (2000) explore the impact of this variation on compression of a related set of files. Their results indicate that compaction varies with file size, and that larger files tend to compact more than smaller files. Given that compression algorithms work by eliminating redundant data, it makes sense that a file with many similar objects and thus high repetition would compress more than one with few objects.

To explore the impact of varying file sizes for applications in the small-file category, this researcher used a master file containing between 750 and 1,000 objects, in XML format, from each application. From the master set, XSLT transformations were used to generate progressively larger files containing a subset of the master file. The n-th subset file contained the XML header, the root element and root-level metadata, and the first n objects. For $n < 10$, every subset file was created, and for $n \geq 10$, every 10th subset file was created. Thus, for a master file with 1,000 objects, subset files were created for $n = \{1, 2, 3 ... 8, 9\} \cup \{10, 20, 30 ... 980, 990, 1000\}$. This resulted in between 84 and 109 subset files per use case.

2. Format Conversions

All encodings tested in the small-file category derived from the n-object subset files in XML format, requiring a series of software conversions to produce syntactically different files containing, as much as was practical, semantically equivalent information.

This section addresses the methodology and software used for each conversion. A series of Bash shell scripts performed all task automation functions.

(1) XML to JSON

Since the XML to JSON conversion process is not one-to-one, this work incorporated a generic, automated methodology to facilitate repeatability, speed up data collection, and to create a "fair" comparison between applications. XML to JSON conversions used the XSLTJSON stylesheet processed with the Saxon9 XSLT processor (Kay, 2009; Stein, 2014) XSLTJSON allows the user to select one of four possible transformation conventions (Stein, 2014). This research used the default, which is the most compact but neither preserves namespaces nor allows for lossless round-trip conversions from XML to JSON and back to XML (Stein, 2014). Though conventions such as the BADGERFISH convention support do support lossless round-trip conversions, this work favored the more compact method. Some degree of information is lost in the process, but including content such as namespaces in a JSON document does not reflect its use in practice and adds a bias toward XML by saddling JSON with irrelevant information. Instead, this approach aims to inform development decisions of whether to use JSON or EXI, or client decisions in situations where a web service offers both JSON and XML API responses.

(2) XML to EXI

All XML to EXI encodings used the EXIficient library (Daniel Peintner & Heuer, 2014). To invoke various options in EXIficient, the ExiProcessor interface was used (Garrett, 2012)(Garrett, 2012). Both EXIficient and ExiProcessor are written in Java.

(3) JSON to BSON

Unlike the above conversions, JSON to BSON is a one-to- one mapping. The BSON codec included with Pythomnic3k framework performed all JSON to BSON conversions (Dvoinikov, 2014). Pythomnic3k is written in Python. The conversion process reads a JSON file into a Python dictionary object, then serializes the object in binary format as BSON.

(4) JSON to CBOR

The process for converting from JSON to CBOR was similar to that in (3) above, but used a CBOR codec written in Python (Olson, 2014). JSON to CBOR conversions are a one-to-one mapping.

(5) Zip, Gzip and Bzip2 Compression

Three BSD command-line utilities, Zip, Gzip and Bzip2, performed all conventional compression encodings where applicable. For each, the version was the default one included in a base installation of OS X 10.9 with Apple Developer Tools. The encodings used default settings in all cases.

0 is a structured diagram of all format conversions for the large-file category. The root node on the left represents an abstract piece of data, for which all other encodings are semantically equivalent. Each leaf node on the right is a final encoding format written as a string of filename extensions indicating the sequence of conversions. For example, ".xml.strict_precompress_exi.bz2" denotes an XML file encoded first with EXI in precompress mode with strict schema adherence, then encoded with the Bzip2 compression algorithm.

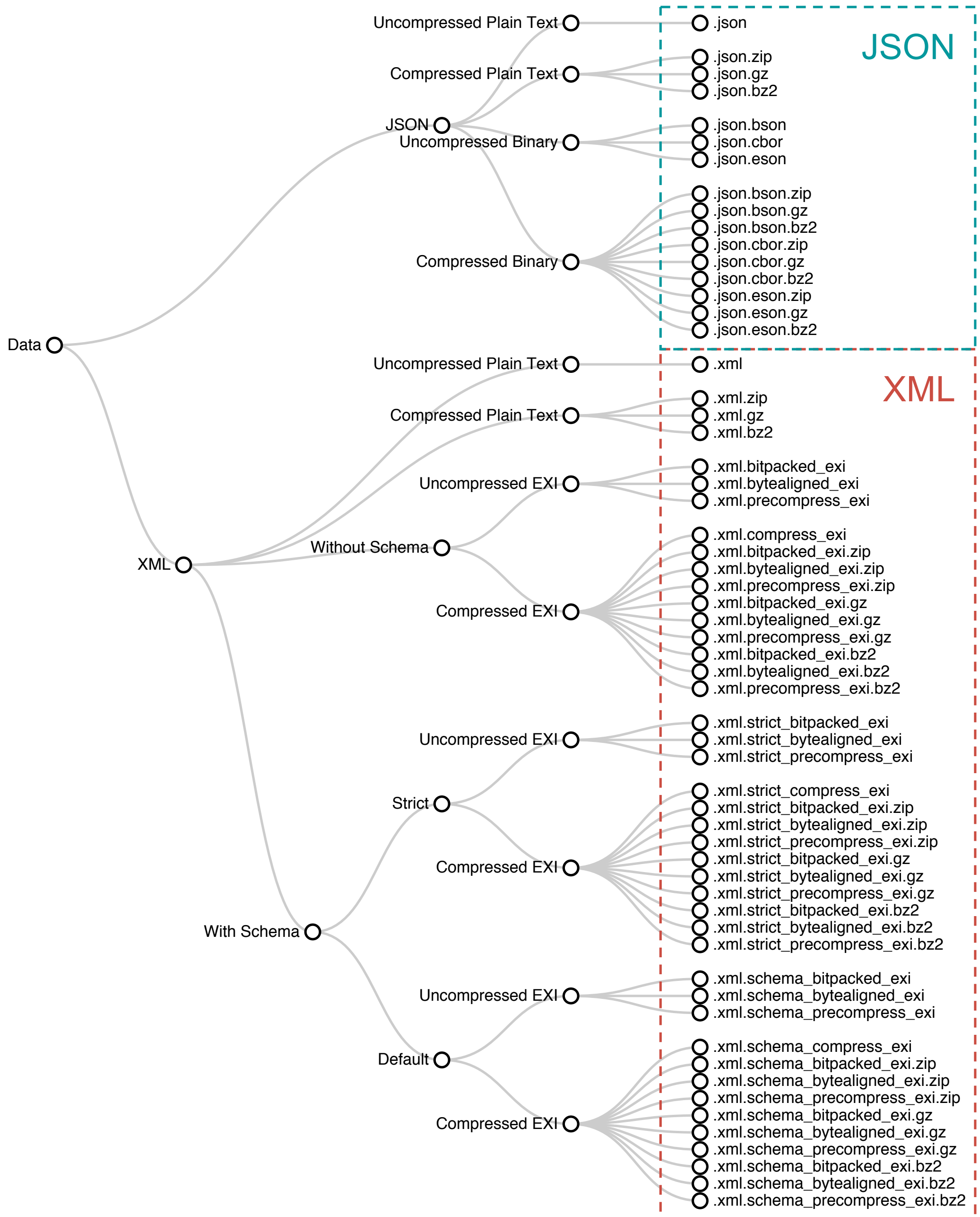

Figure 8. A categorized representation of all small-file category encodings derived from the same data.

3. Focus Questions

For each of the approximately 109 files in a small-file application file set, the test suite generated 55 different encodings, resulting in a total of 5,995 different file size records per file set. Though preliminary review of compression literature and early test results indicated that many of the encodings were not ideal, this researcher conducted the tests for thoroughness. With such a large number of choices, a winnowing method was critical. To facilitate analysis and presentation of results, this research incorporated a series of six focus questions addressing various facets of the XML to JSON comparison space, as well as the various configurations of EXI. The questions derive both from the researchers course of inquiry into the capabilities and traits of EXI, and well as potential questions from developers considering EXI integration. In Chapter IV, a plot relating compaction to original file size for a group of encodings formats answers each of the focus questions.

All plots display compaction as a percentage, calculated as Compaction = original size (in bytes) / compressed size (in bytes). Using this formula, a value of 100% on the y-axis indicates no change in file size. Values greater than 100% on the y-axis indicate the file size increased, and values less than 100% indicate the file size decreased. In other words, a compaction rate of 25% means that the compressed file is 1/4 the size of the original. The original size, or baseline, for most questions is plain-text XML size, though some questions require alternate baseline formats. For example, though an EXI encoding of plain-text XML may be 10% of original size, the result does not reflect the impact of EXI for a network already using Gzip compression. In that case, the Gzip encoding of XML is a more realistic baseline. Table 3 outlines the group of encodings associated with each focus question and the baseline format for comparison. Chapter IV presents the plots for all these questions, with the results grouped by focus question, and in the same question order presented here.

Table 3. Focus questions for small-file category, relevant encodings, and baseline for comparison.

Question	Baseline	Encodings Compared
A. Is JSON more compact than XML either when both are plain-text encoded or when both are compressed with conventional compression algorithms?	.xml	.json .json.gz .json.bz2 .xml.gz .xml.bz2
B. Does post-compression with conventional algorithms increase the compactness of BSON or CBOR?	.json	.json.bson .json.bson.gz .json.bson.bz2 .json.cbor .json.cbor.gz .json.cbor.bz2
C. How do the primary EXI modes compare for schemaless and schema-informed encodings?	.xml	.xml.bitpacked_exi xml.schema_bitpacked_exi xml.compress_exi xml.schema_compress_exi
D. Is Bitpacked-mode EXI more compact than BSON or CBOR?	.xml	.xml.bitpacked_exi .xml.schema_bitpacked_exi .xml.strict_bitpacked_exi .json.cbor .json.bson
E. Is Compress-mode EXI more compact than BSON or CBOR post-compressed with conventional compression algorithms?	.xml	.xml.compress_exi .xml.schema_compress_exi .xml.strict_compress_exi .json.cbor.gz .json.bson.gz
F. For a network already using Gzip compression, do any of the tested binary encodings offer better compactness?	.xml.gz	.xml.strict_compress_exi .xml.strict_bitpacked_exi .xml.compress_exi .json.gz .json.cbor.gz .json.bson.gz
G. Do restrictions on data types and range restrictions in an XSD significantly impact the compaction of schema-informed encodings? Only addressed for 1 use case.	.xml	.xml.schema_compress_exi .xml.strict_compress_exi .xml.schema_bitpacked_exi .xml.strict_bitpacked_exi

4. Use Cases

The following sections describe the use cases included in the small-file category, including data collection and sample code for both XML and JSON file formats. Full file sets are included in the digital appendix.

a. Global Positioning System XML

The Global Positioning System (GPS) Exchange (GPX) format is an XML format for representing GPS data, including waypoints, routes and tracks (Foster, n.d.). For this application, the master file is a GPS watch trace from a 10-mile trail run over 1 hour and 40 minutes, with GPX fixes at 6-second intervals. It includes 870 total fix points (each with a latitude, longitude, elevation and timestamp), as well as general information about the application producing the GPX file. After subsetting, the *n*-th file includes the metadata and the first *n* GPS fixes. All subset files validated against the GPX 1.1 XSD (TopoGrafix, 2004). Figure 9 and Figure 10 are samples from corresponding XML and JSON files, respectively. Both code samples include whitespace and indentations for clarity, but compression tests use versions with unnecessary whitespace removed. Figure 11 is a visual depiction of the GPX master file in Google Earth, with each yellow circle representing a single GPS fix.

```
<?xml version="1.0" encoding="UTF-8"?>
<gpx version="1.1" creator="Garmin Connect"
     xmlns="http://www.topografix.com/GPX/1/1">
  <metadata>
    <link href="connect.garmin.com">
      <text>Garmin Connect</text>
    </link>
    <time>2014-09-05T18:07:03.000Z</time>
  </metadata>
  <trk>
    <name>Untitled</name>
    <trkseg>
      <trkpt lon="-121.69120400212705" lat="36.62711977958679">
        <ele>29.799999237060547</ele>
        <time>2014-09-05T18:07:03.000Z</time>
      </trkpt>
      <!-- ... Series of trkpt's continues ... -->
    </trkseg>
  </trk>
</gpx>
```

Figure 9. Sample code for GPX file, in XML format.

```
{ “gpx”: {
    “creator”: “Garmin Connect,”
    “metadata”: {
        “link”: {
            “href”: “connect.garmin.com,”
            “text”: “Garmin Connect”
        },
        “time”: “2014-09-05T18:07:03.000Z”
    },
    “trk”: {
        “name”: “Untitled,”
        “trkseg”: {
            “trkpt”: [
                {
                    “ele”: 29.799999237060547,
                    “lat”: 36.627119779586792,
                    “lon”: -121.69120400212705,
                    “time”: “2014-09-05T18:07:03.000Z”
                }
                /* ... Series of trkpt’s continues ... */
            ] } } } }
```

Figure 10. Sample code for GPX file, in JSON format.

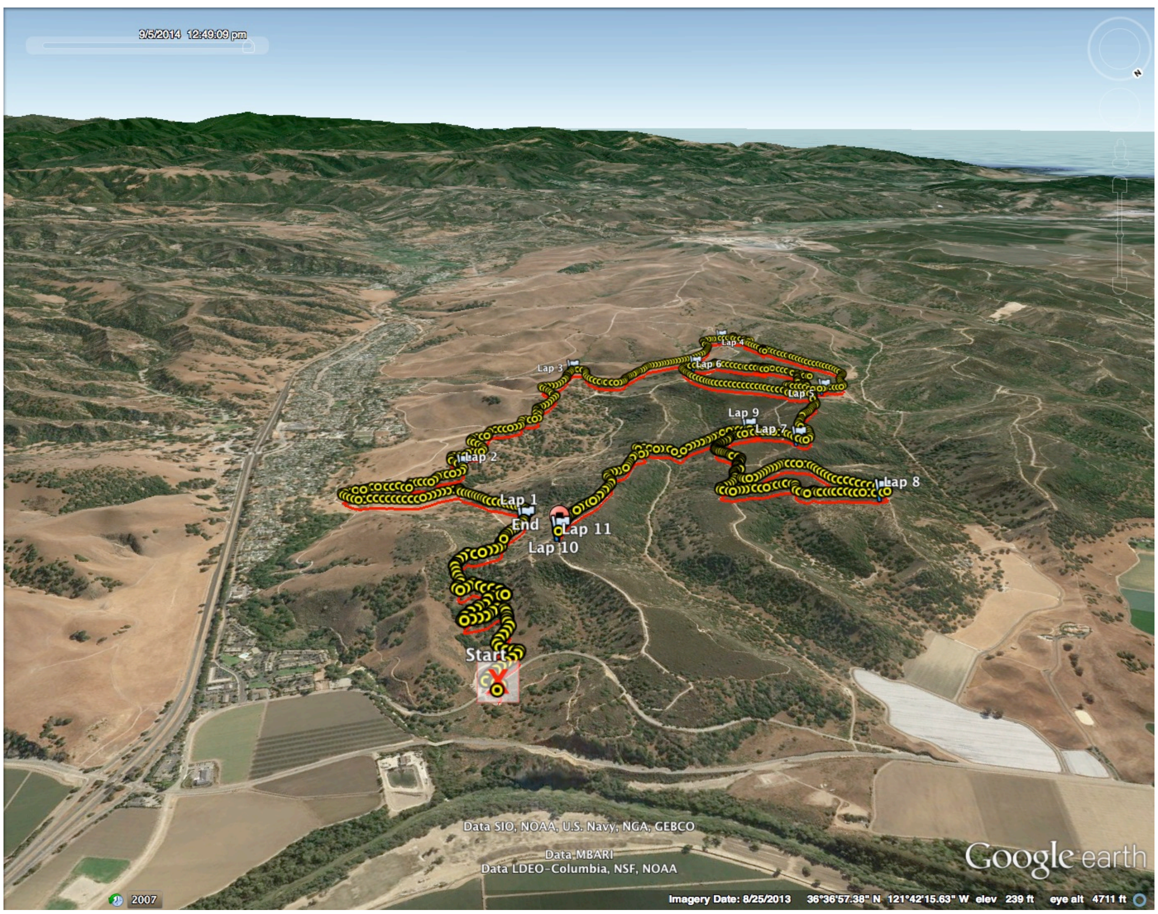

Figure 11. Visualization of GPX master file in Google Earth.

b. OpenWeatherMap XML

OpenWeatherMap aggregates weather data from a variety of sources worldwide and uses it for big-data analysis and weather modeling (OpenWeatherMap Inc., 2015b). The OpenWeatherMap API offers current, forecasted and historical data in both XML and JSON response formats (OpenWeatherMap Inc., 2015a). To assemble the master file for this application, the author selected 1,000 cities from the list cities available through the OpenWeatherMap API, made API queries for the current weather in those cities, and concatenated the results into a single XML file. After subsetting, the *n*-th file included the weather conditions for the first *n* cities in the master file. As OpenWeatherMap does not publish an XSD describing the XML responses, the author generated a schema to include reasonable data type information. Figure 12 and Figure 13 are samples from corresponding XML and JSON files, respectively. Both code samples include whitespace and indentations for clarity, but compression tests use versions with unnecessary whitespace removed.

```
<?xml version="1.0" encoding="utf-8"?>
<group>
  <current>
    <city id="688532" name="Yalta">
      <coord lon="37.27" lat="46.96"/>
      <country>UA</country>
      <sun rise="2014-09-12T03:05:08" set="2014-09-12T15:49:00"/>
    </city>
    <temperature value="292.04"
      min="292.04" max="292.04" unit="kelvin"/>
    <humidity value="78" unit="%"/>
    <pressure value="1014" unit="hPa"/>
    <wind>
      <speed value="1.54" name=""/>
      <direction value="306" code="NW" name="Northwest"/>
    </wind>
    <clouds value="0" name="clear sky"/>
    <visibility/>
    <precipitation mode="no"/>
    <weather number="800" value="Sky is Clear" icon="01n"/>
    <lastupdate value="2014-09-12T21:54:18"/>
  </current>
  <!-- ... Series of cities continues ... -->
</group>
```

Figure 12. Sample code for OpenWeatherMap file, in XML format.

```
{ "group": { "current":
  { "city": {
    "coord": { "lat": 46.960000000000001, "lon":
37.270000000000003 },
    "country": "UA," "id": 688532, "name": "Yalta,"
    "sun": {
      "rise": "2014-09-12T03:05:08,"
      "set": "2014-09-12T15:49:00" } },
  "clouds": { "name": "clear sky," "value": 0 },
  "humidity": { "unit": "%," "value": 78 },
  "lastupdate": { "value": "2014-09-12T21:54:18" },
  "precipitation": { "mode": "no" },
  "pressure": { "unit": "hPa," "value": 1014 },
  "temperature": {
    "max": 292.04000000000002, "min": 292.04000000000002,
    "unit": "kelvin," "value": 292.04000000000002 },
  "visibility": null,
  "weather": { "icon": "01n," "number": 800, "value": "Sky is
Clear" },
  "wind": {
    "direction": { "code": "NW," "name": "Northwest," "value": 306
},
    "speed": { "name": null, "value": 1.54 }
  } }
  /* … Series of cities continues … */
} }
```


Figure 13. Sample code for OpenWeatherMap file, in JSON format.

The collection method for this application introduces potential issues affecting compression. First, the API calls happened in rapid succession, so timestamp information in the master file reflecting last update all reflects the same date. Such repetition improves compaction for many compress algorithms. Second, a pseudorandom function selected the cities to include in the master file, meaning that the data set is not logically related in a geographic sense. Some fields in the API response, such as country code or wind direction, may express less redundancy than for a data set including only weather stations in a certain region, and thus compressing poorly. Though these issues do not invalidate the results, they suggest that a portion of compression performance depends on the underlying semantics of the information.

c. *Automated Identification System*

The Automated Identification System (AIS) is a distributed system for communicating maritime safety information over Very High Frequency (VHF) line-of-sight (LOS) links. Every two to ten seconds while underway, ship-borne transceivers emit messages including position, course, speed and identification data (United States Coast

Guard, 2014a). Nearby ships receive the messages and, using specialized charting software, overlay the data on radar plots or electronic navigation charts to build situational awareness. Systems such as National AIS (NAIS) and MSSIS collect feeds from ships via large networks of shore-based stations, and aggregate them for global-scale intelligence and Maritime Domain Awareness (MDA) purposes (United States Coast Guard, 2015; United States Department of Transportation Volpe Center, n.d.). Though AIS encodes messages in a compact binary format, formats such as XML can assist in software analysis of large AIS datasets, as well as data interchange between heterogeneous storage databases and Geographic Information System (GIS) software.

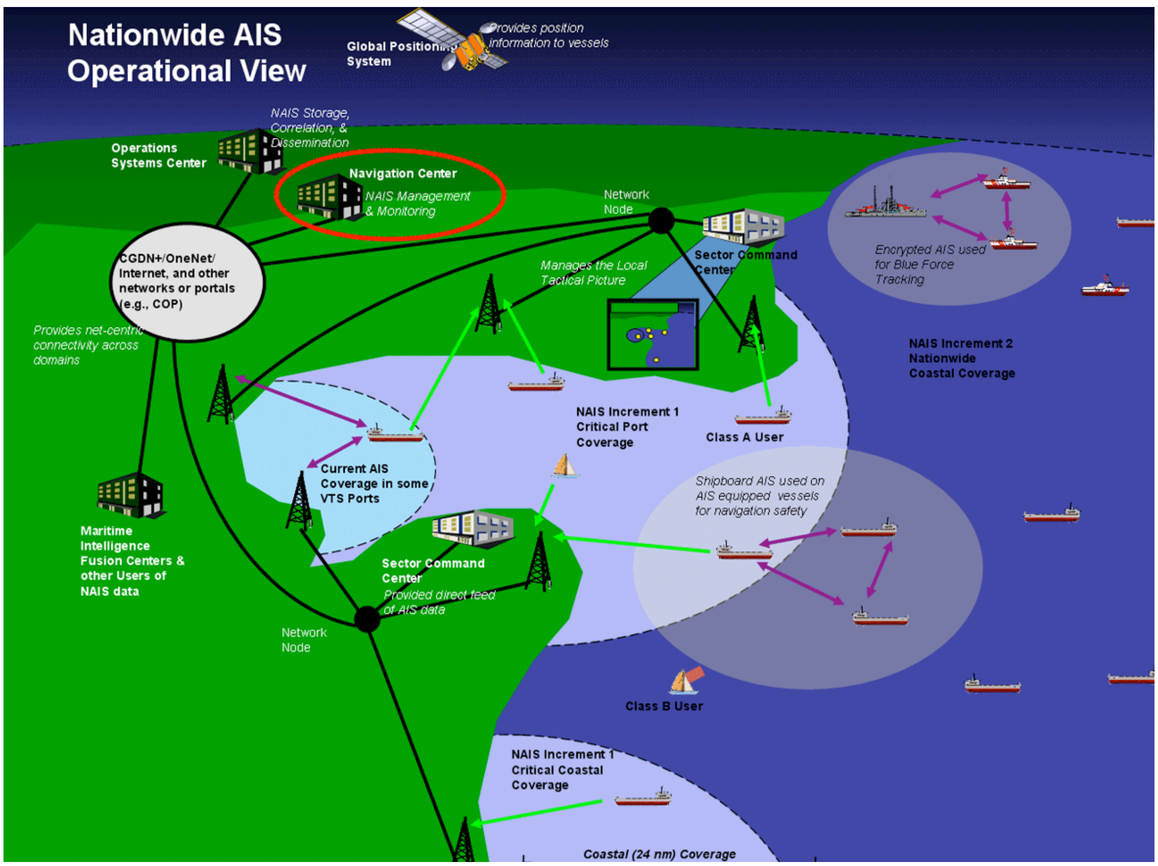

Figure 14. An operational view diagram of the NAIS system including information flows from ship-borne transceivers to intelligence fusion centers (United States Coast Guard, 2014b).

The AIS data in this test case came from the MSSIS AIS feed at the Naval Postgraduate School. To process the raw messages to XML, the author used EsiAisParser, a Java library for AIS feed parsing, to read individual messages into in-memory Java objects, then write them to a file in XML format. Only Type 3 position report messages were included in the master file, which comprised 1,000 AIS position reports. In the absence of an openly available XSD describing AIS messages, the authored developed a simple XML format and a corresponding XSD with data type information based on the U.S. Coast Guard's AIS specification (United States Coast Guard, 2014a). After subsetting, the *n*-th file included the first *n* position reports in the master file. Figure 15 and Figure 16 are samples from corresponding XML and JSON files, respectively. Both code samples include whitespace and indentations for clarity, but compression tests use versions with unnecessary whitespace removed.

```
<?xml version="1.0" encoding="UTF-8"?>
<positionReports>
  <positionReport
    messageId="3" repeatIndicator="0"
    userId="248859000" navStatus="1"
    rateOfTurn="0" speedOverGround="0"
    positionAccuracy="false"
    longitude="117.3649766666666" latitude="-20.534186666666667"
    courseOverGround="3100" trueHeading="310"
    timeStamp="53" specialManoeuvre="0" />
  <!-- ... Series of positionReport's continues ... -->
</positionReports>
```

Figure 15. Sample code for AIS file, in XML format.

```
{ "positionReports": {
    "positionReport": [
        { "courseOverGround": 3100,
          "latitude": -20.534186666666667,
          "longitude": 117.3649766666666,
          "messageId": 3, "navStatus": 1,
          "positionAccuracy": false,
          "rateOfTurn": 0, "repeatIndicator": 0,
          "specialManoeuvre": 0, "speedOverGround": 0,
          "timeStamp": 53, "trueHeading": 310,
          "userId": 248859000 },
    /* Series of positionReports continues */
    ] } }
```

Figure 16. Sample code for AIS file, in JSON format.

D. LARGE-FILE CATEGORY

To measure the compactness of EXI encodings of XML files larger than 100MB, this researcher compiled a collection of files, or file sets, from each of three use cases. Each plain-text file in a file set was encoded to EXI using various permutations of EXI options and post-compression algorithms. Plain-text file sizes in this collection range from 1KB to 3.34GB in this category, hereafter referred to as the large-file category. This section lists the tested file formats, introduces focus questions used to analyze the results, and summarizes the use cases.

1. Format Conversions

Encodings for the large-file category were performed using the same software tools as for the small-file category. However, the purpose of the large-file category was not to compare EXI compactness to binary JSON encodings, and thus no JSON-derivative encodings were measured. That is, the large-file category only measured conversions of XML with conventional compression algorithms, XML to EXI, and EXI with conventional compression algorithms. Figure 17 is a structured diagram of all format conversions for the large-file category. The root node on the left represents the plain-text XML document for which all other encodings are semantically equivalent. Each leaf node on the right is a final encoding format written as a string of filename extensions indicating the sequence of conversions. For example, ".xml.strict_precompress_exi.bz2" denotes an XML file encoded first with EXI in precompress mode with strict schema adherence, then encoded with the Bzip2 compression algorithm.

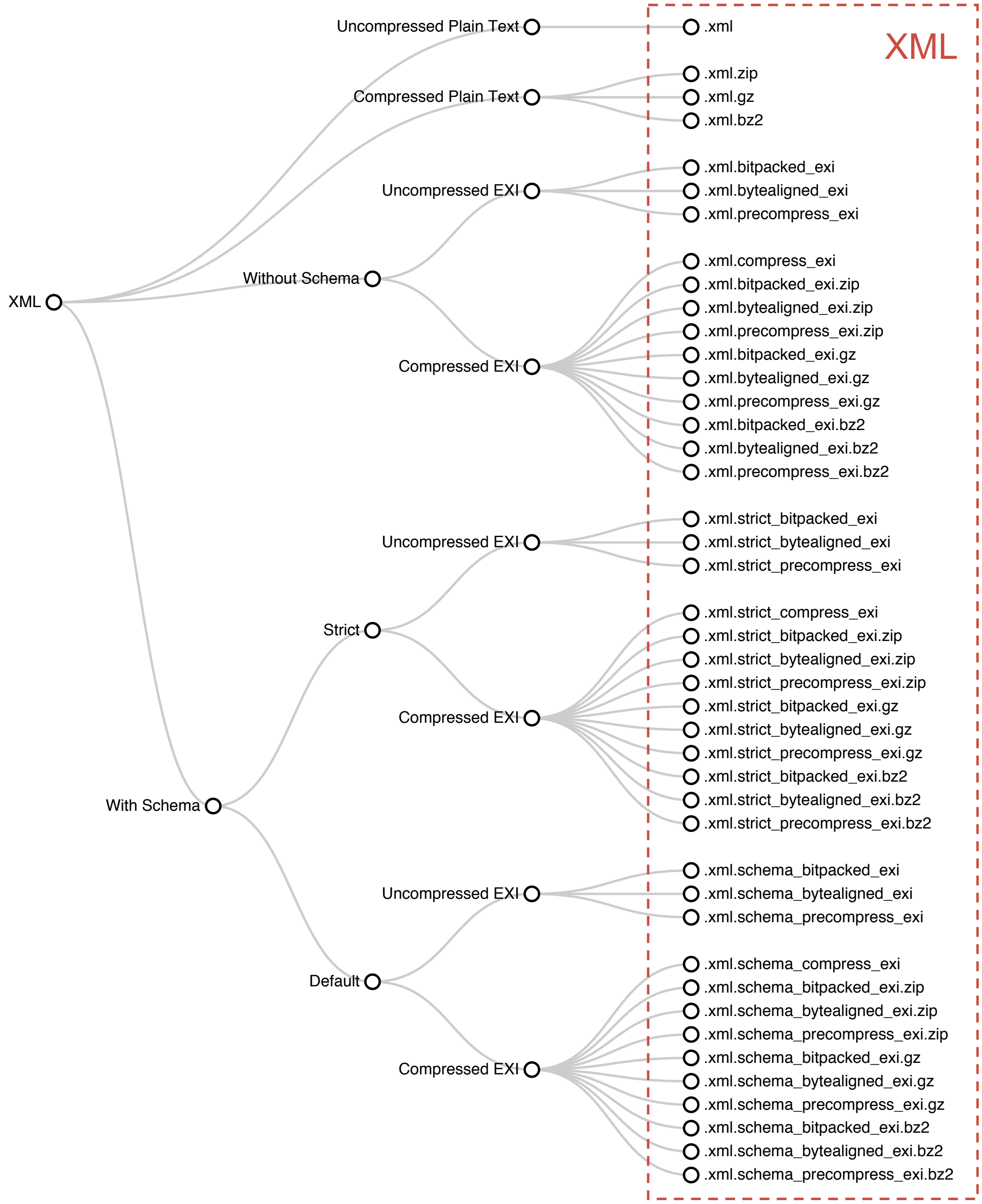

Figure 17. A categorized representation of all large-file category encodings derived from the same plain-text XML document.

2. Focus Questions

For each file in a large-file use case, the test suite generated 36 different encodings. As in the analyses for small-file applications, a series of six focus questions drove the data visualization process. A plot relating compaction rate to original file size for a group of encoding formats answers each focus question. In general, the focus questions for large-file use cases are a subset of those asked for small-file cases; since the large-file tests examined only XML and EXI encodings, the large-file focus questions exclude all JSON comparisons. Table 4 outlines the group of encodings associated with each focus question and the baseline format for comparison. Chapter IV presents the plots for all these questions, with the results grouped by focus question, and in the same question order presented here.

Table 4. Focus questions for large-file category, relevant encodings, and baseline for comparison.

Question	Baseline	Encodings Compared
A. How do the primary EXI modes compare for schemaless and schema-informed encodings?	.xml	.xml.bitpacked_exi xml.schema_bitpacked_exi xml.compress_exi xml.schema_compress_exi
B. Does the ‘strict’ option significantly improve compaction for schema-informed encodings?	.xml	.xml.schema_bitpacked_exi .xml.strict_bitpacked_exi .xml.schema_compress_exi .xml.strict_compress_exi
C. Do any of the tested conventional compression algorithms better compact a schemaless, precompress EXI document than EXI’s default DEFLATE?	.xml.compress_exi	.xml.precompress_exi.zip .xml.precompress_exi.gz .xml.precompress_exi.bz2
D. Do any of the tested conventional compression algorithms better compact a schema-informed, precompress EXI document than EXI’s default DEFLATE?	.xml.schema_compress_exi	.xml.schema_precompress_exi.zip .xml.schema_precompress_exi.gz .xml.schema_precompress_exi.bz2
E. Which EXI encoding is the most compact?	.xml	.xml.strict_compress_exi .xml.strict_bitpacked_exi .xml.compress_exi
F. For a network already using Gzip compression, do any of the EXI encodings offer better compactness?	.xml.gz	.xml.strict_compress_exi .xml.strict_bitpacked_exi .xml.compress_exi

3. Use Cases

The following sections describe the use cases included in the large-file category, including data collection and sample code. Full file sets are included in the digital appendix.

a. Digital Forensics XML

Digital Forensics XML (DFXML) is an XML file format describing the results of forensic scans of digital media. The digital forensics field uses a many tools to automate various parts of the process, from data extraction to data fusion and correlation (Garfinkel, 2012). However, in many cases, these tools use disparate file formats to store results, which makes automating the process as a whole challenging (Garfinkel, 2011). DFXML is an interoperable, machine-readable format that addresses this issue of tool integration.

A DFXML file describing a disk image includes information about the disk sectors, about each of the disk's partitions, about each file on the disk, and the provenance of the DFXML file itself, such as information about the tool that created it and timestamp (Garfinkel, 2009). The file set for this application includes DFXML outputs from 51 disk images created by the program fiwalk. In plain-text XML format, they range from 1 KB to 563 MB. After minor manual editing, all files validated against the Digital Forensics XML Schema (Nelson & Digital Forensics XML Working Group, 2014).

```
<fileobject>
  <filename>casper/filesystem.manifest-desktop</filename>
  <filesize>32672</filesize>
  <inode>651</inode>
  <meta_type>1</meta_type>
  <mode>511</mode>
  <nlink>1</nlink>
  <uid>0</uid>
  <gid>0</gid>
  <mtime>2008-12-29T01:33:32Z</mtime>
  <atime>2008-12-28T05:00:00Z</atime>
  <crtime>2008-12-29T01:33:32Z</crtime>
  <byte_runs>
    <byte_run file_offset="0" fs_offset="5577728"
      img_offset="5609984" len="32672"/>
  </byte_runs>
  <hashdigest type="md5">
    bd1b0831fcba1f22eff2238da96055b6</hashdigest>
  <hashdigest type="sha1">
    7e072af67f8d989cc85978487b948048ac3c7234</hashdigest>
</fileobject>
```

Figure 18. A DFXML element representing a single file, including information about the file name, size, hash codes, location on disk, and provenance (after Garfinkel, 2011).

b. Packet Details Markup Language

The Packet Details Markup Language is an XML-based format for describing the details of decoded network packets (Risso, 2010). It combines XML elements for packets, protocols, and protocol fields, offering full details of a packet and all encapsulated protocols, sans the payload. Wireshark was used to decode two large .pcap-formatted packet traces from online repositories and export portions of them into PDML format, following a subsetting approach similar to that in the small-file category. After subsetting, the PDML files contained between 1 and 300,000 packet descriptions, and ranged from 9 KB to 3.1 Gigabytes (GB) in plain-text encoding. All files validated against a slightly modified version of the PDML schema (NetBee.org, n.d.).

```
<pdml>
  <packet caplen="74" len="74" num="4"
    timestamp="963585313.991300">
    <proto longname="Ethernet 802.3"
      name="Ethernet" pos="1" size="14">
      <field longname="MAC Destination" name="dst" pos="1"
        showdtl="000629-992da3 Unicast address (Vendor
        IBM RISC6000 system)"
        showmap="IBM RISC6000 system" showvalue="000629-992da3"
        size="6" value="000629992da3"/>
      <field longname="MAC Source" name="src" pos="7"
        showdtl="00e01e-ec3c84 Unicast address (Vendor Cisco)"
        showmap="Cisco" showvalue="00e01e-ec3c84"
        size="6" value="00e01eec3c84"/>
      <field longname="Ethertype" name="type-length" pos="13"
        showvalue="0800" size="2" value="0800"/>
    </proto>
    <proto longname="IPv4 (Internet Protocol version 4)"
      name="IP" pos="15" size="20">
      <!-- Additional field elements -->
    </proto>
    <proto longname="ICMP (Internet Control Message Protocol)"
      name="ICMP" pos="35" size="40">
      <!-- Additional field elements -->
    </proto>
  </packet>
</pdml>
```

Figure 19. A sample PDML element representing a single ICMP packet. Adapted from Risso (2010).

c. OpenStreetMap XML

The OpenStreetMap project aims to create a free database of geographic data for the every feature on the planet, leveraging crowdsourced inputs from hundreds of thousands of mappers to populate data (Bennett, 2010). After collecting raw GPS data, or traces, mappers upload them to the OpenStreetMap servers and use them to mark locations of, and metadata about, streets, waterways, paths, structures, signs and sundry other features. Commercial and government entities also provide geospatial data to OpenStreetMap under its license (Bennett, 2010).

OpenStreetMap uses an XML format based around nodes (single points), ways (ordered lists of points), and relations (lists of nodes and ways that create complex features) for data exchange between applications that process OpenStreetMap data. As a single-file export, the entire OpenStreetMap database is over 498 GB in plain-text XML format. This research used 20 smaller extracts covering single countries with file sizes from 50 MB to 927 MB (Geofabrik GmbH, 2014). All files validated against a modified version of the OpenStreetMap XSD (OpenStreetMap contributors, 2012).

```
<node id="25270672"
  lat="39.6735751" lon="-31.1131917"
  version="4" timestamp="2014-08-08T22:35:03Z"
  changeset="24625291" uid="1963239" user="UserX">
  <tag k="name" v="Vila do Corvo"/>
  <tag k="place" v="village"/>
  <tag k="population" v="430"/>
  <tag k="population:date" v="2011"/>
</node>
<way id="185364122" version="1" timestamp="2012-10-11T22:17:36Z"
  changeset="13460710" uid="179581" user="UserY">
  <nd ref="1863929971"/>
  <nd ref="1959512759"/>
  <tag k="highway" v="path"/>
</way>
```

Figure 20. Sample XML fragments, in OpenStreetMap format, for node and way elements.

E. CHAPTER SUMMARY

To extend and build upon previous research on EXI performance, this research methodology focused on measuring the impact of various EXI and conventional compression configurations on overall file size. It adopted an intra-use case focus, rather than comparing results across multiple use cases, and examined the relationship between file size and compaction rate within a single use case. This research identified three small-file use cases: GPX, OpenWeatherMap and AIS; as well as three large-file use cases: DFXML, PDML and OpenStreetMap. A systematic method, centered on a series of focus questions, for comparing the myriad configurations of EXI, BSON and CBOR was developed.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. EXPERIMENTAL RESULTS AND ANALYSIS

This chapter presents experimental test results collected during compression testing. The first section explains the compaction metric used and comments on interpreting the plots and results. The second section presents results for the small-file category of use cases, grouped by focus question. A summary analysis follows each focus question that identifies common or generalizable trends across use cases. Finally, the third section presents results for the large-file category of use cases, group by focus question. Again, a summary analysis follows each focus question.

A. INTERPRETING THE NUMBERS

The following sections include compression test results for each focus question organized first by category (small-file and large-file) and then by use-case. Each figure is a scatterplot comparing the compaction rate of multiple encodings for all sample files in a use-case. Colors denote the various encodings. Solid lines connect the points in a series for visual clarity, but do not represent a parametric model of the trend.

Compaction rate, shown as a percentage on the y-axis in each figure, is compressed size divided by original size, or more intuitively, percent of original size.

Compaction = compressed size / original size * 100%

With this measurement, smaller values indicate greater compaction (smaller is better). A compaction rate of 100% indicates that the compressed size is identical to the original size. Values less than 100% indicate the compressed size is less than the original size, and values greater than 100% indicate that compression increased the file's size.

In most figures, the baseline encoding for original size refers to the plain-text XML encoding, though some figures use different baselines. When a figure compares a mix of XML-based and JSON-based encodings, the baseline is plain-text XML. When a figure compares only JSON-based encodings, the baseline is plain-text JSON. Both of these baselines are appropriate for objectively comparing compression algorithms, they do not suggest how much benefit a compression algorithm will provide for a network already using Zip or Gzip by default. Thus, some figures display compaction using Gzip-compressed plain-text XML as a baseline.

For all figures, the x-axis shows the original size of the file in the baseline encoding, in bytes, on a logarithmic scale. Though plotted identically, the file-size values differ slightly between small-file and large-file use cases. Since the sample files in the small-file use cases are all subsets of a master file, the points on the x-axis are subsets of the same file. For large-file use cases, the sample files vary in size, but are not necessarily subsets of one another. Thus, the plots in small-file use cases tend to create strictly increasing or strictly decreasing series, while the plots of large-file use cases do not.

B. RESULTS BY FOCUS QUESTION—SMALL-FILE CATEGORY

This section presents empirical data for each focus question in the small-file category. The focus question is reiterated, followed by corresponding plots from each of the three small-file use cases: GPX, OpenWeatherMap and AIS, in that order. A brief discussion follows, addressing results across the use cases in response to the focus question.

1. Focus Question A: Base Comparison of JSON and XML

Is JSON more compact than XML either when both are plain-text encoded or when both are compressed with conventional compression algorithms?

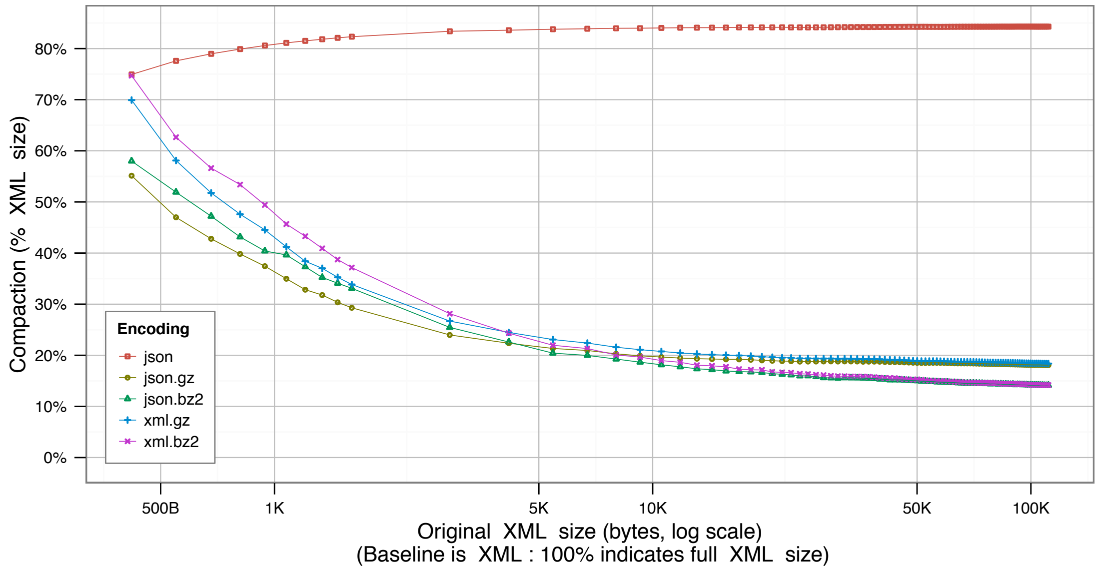

Figure 21. Plot for GPX use case, focus question A.

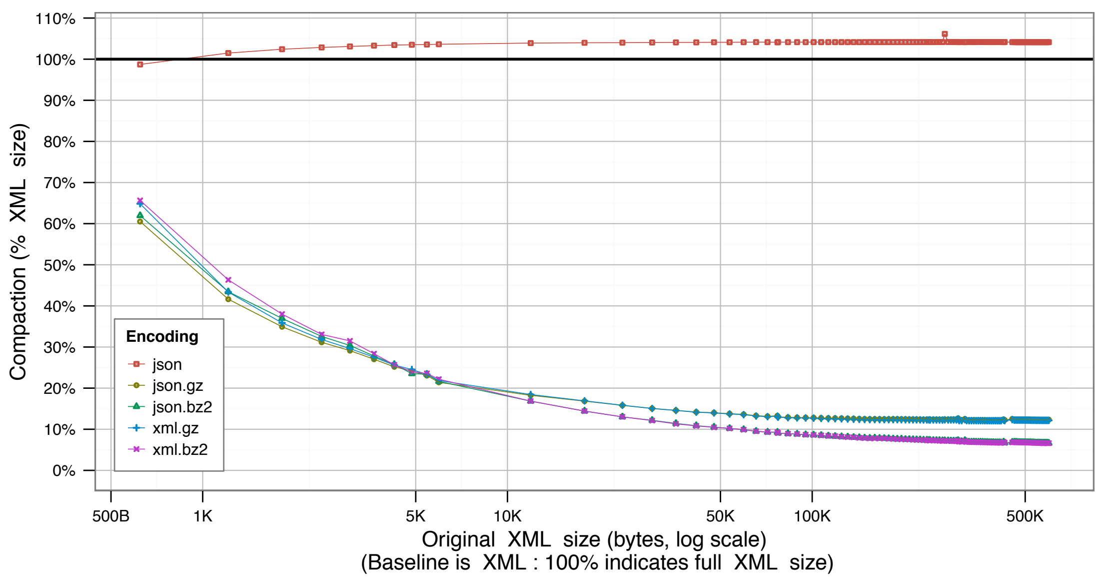

Figure 22. Plot for OpenWeatherMap use case, focus question A.

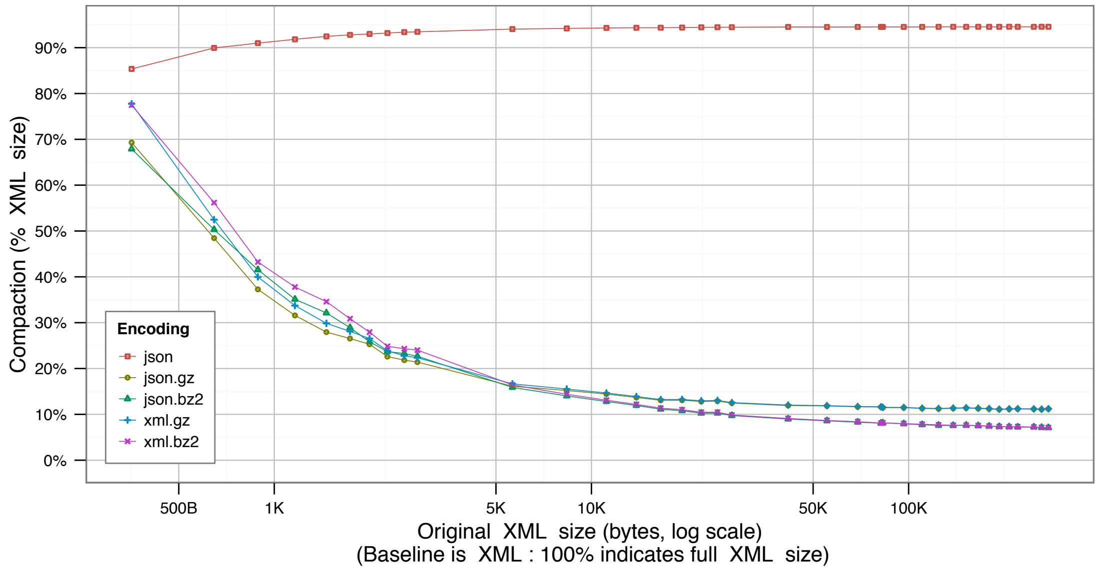

Figure 23. Plot for AIS use case, focus question A.

For the GPX and AIS use cases, plain-text JSON was ~75-95% of XML size. Visual inspection of the files revealed the decrease occurred when identically named XML elements collapsed into a JSON array, thereby mitigating the penalty of repetitive opening and closing tags for repeated elements in XML format. In the OpenWeatherMap use case, JSON was approximately 98–106% of XML size. The increase in size came from conversion of XML elements with multiple attributes into several key-value pairs in JSON, as shown in Figure 24 and Figure 25.

```
<temperature value=“292.04” min=“292.04” max=“292.04”
unit=“kelvin”/>
```

Figure 24. OpenWeatherMap temperature element in XML format, using 69 characters.

```
{“temperature”:{“value”:292.04,”min”:292.04,”max”:292.04,”unit”:”k
elvin”}
```

Figure 25. Semantically equivalent data as JSON, using 73 characters.

For smaller subset sizes, JSON compressed with Gzip and Bzip2 produced more compact encodings than XML compressed with the same algorithm. For subsets larger than ~7KB, XML compressed with Gzip and Bzip2 was more compact than JSON compressed with the same algorithm. The Gzip and Bzip2 trend lines for all three use cases show that in general, as the size of the plain-text file increases, compaction improves. This is intuitive considering that Gzip and Bzip2 look for repeated strings of characters to compress a file. When a file is small, there are few, if any, repeated strings. As the subset size increase, there are more repeated strings, so compaction improves. However, the trend is asymptotic, and compaction rates flatten. This pattern of diminishing returns recurred in nearly all tests during experimentation.

2. Focus Question B: Post-Compression of BSON and CBOR

Does post-compression with conventional algorithms increase the compactness of BSON or CBOR?

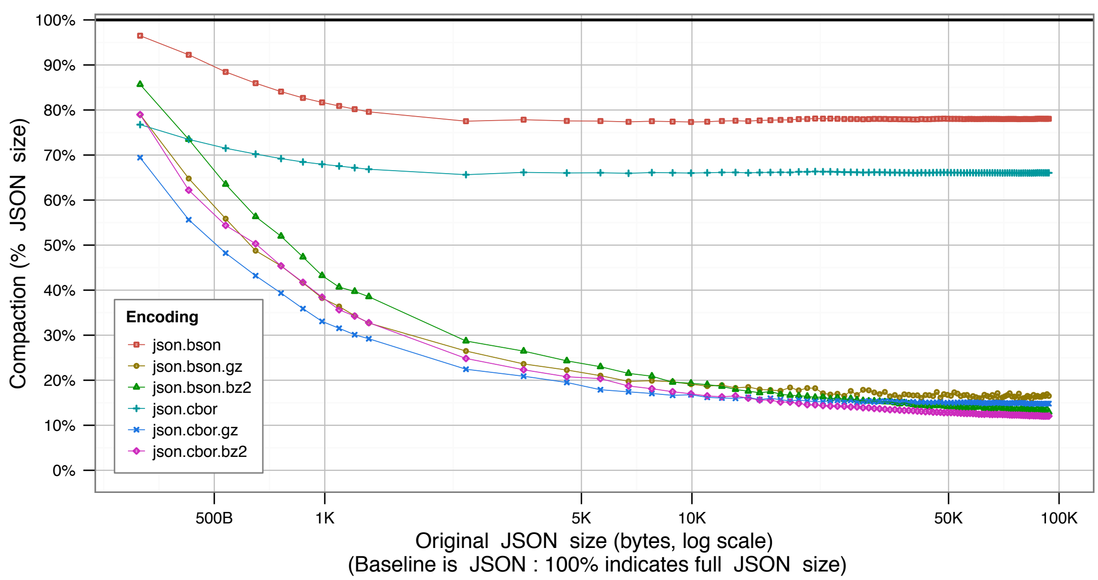

Figure 26. Plot for GPX use case, focus question B.

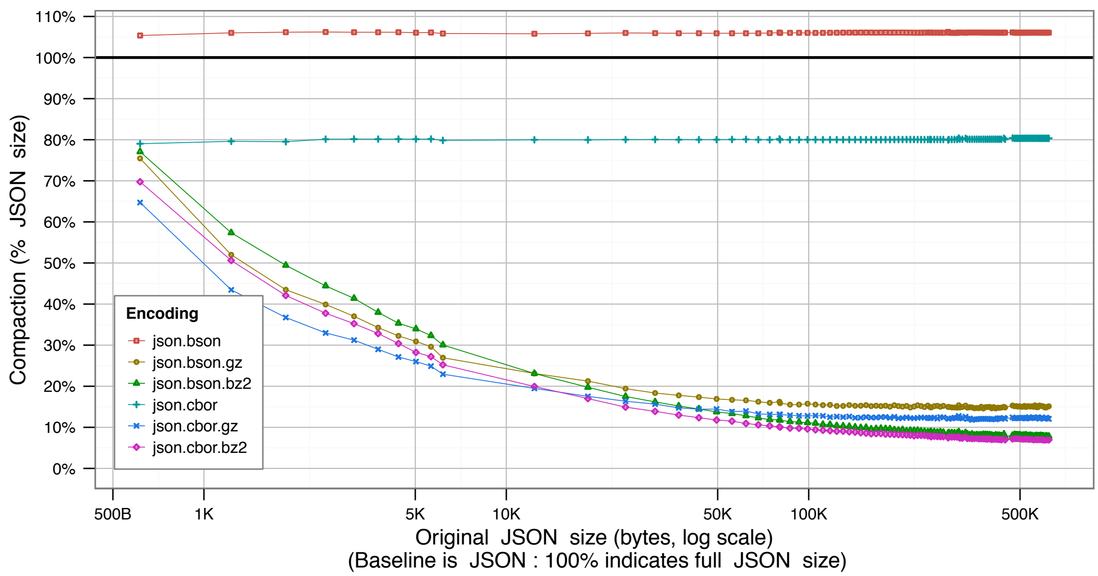

Figure 27. Plot for OpenWeatherMap use case, focus question B.

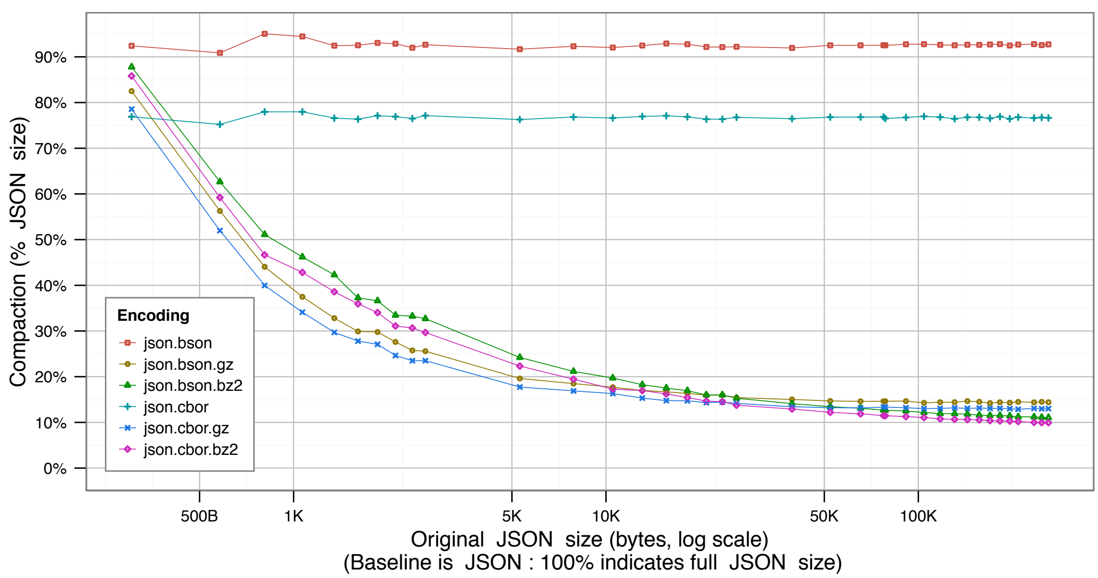

Figure 28. Plot for AIS use case, focus question B.

In all three use cases, conventional compression algorithms significantly increased the compactness of both BSON and CBOR encodings, with the exception of the smallest file in the GPX and AIS use cases, where Gzip and Bzip2 increased the file size. Compaction demonstrated a curve of diminishing returns. After compression, compaction leveled out at ~11-21% of the corresponding BSON or CBOR size.

3. Focus Question C: Primary EXI Modes

How do the primary EXI modes compare for schemaless and schema-informed encodings?

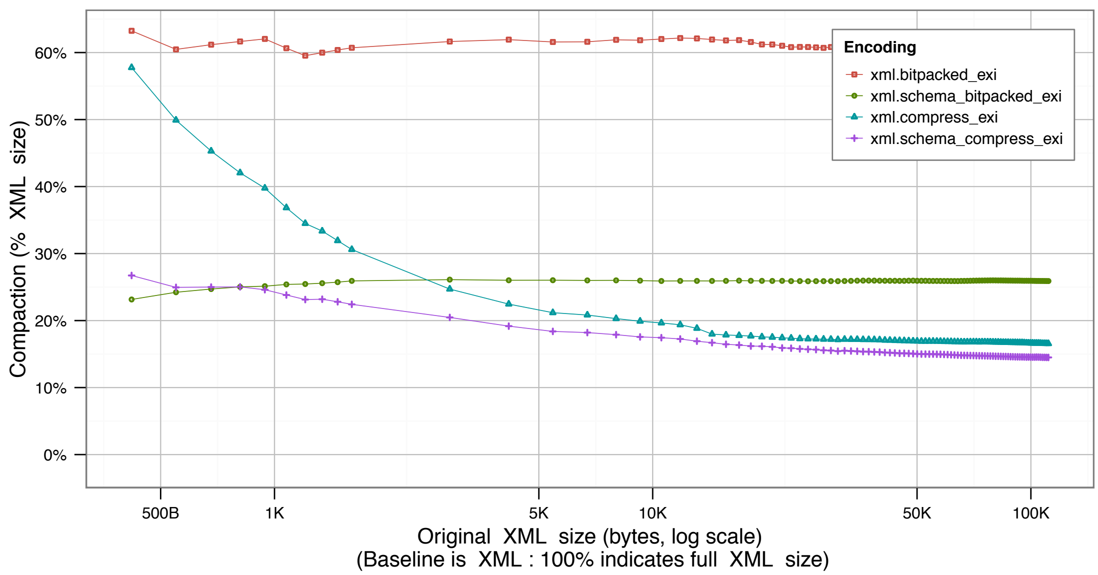

Figure 29. Plot for GPX use case, focus question C.

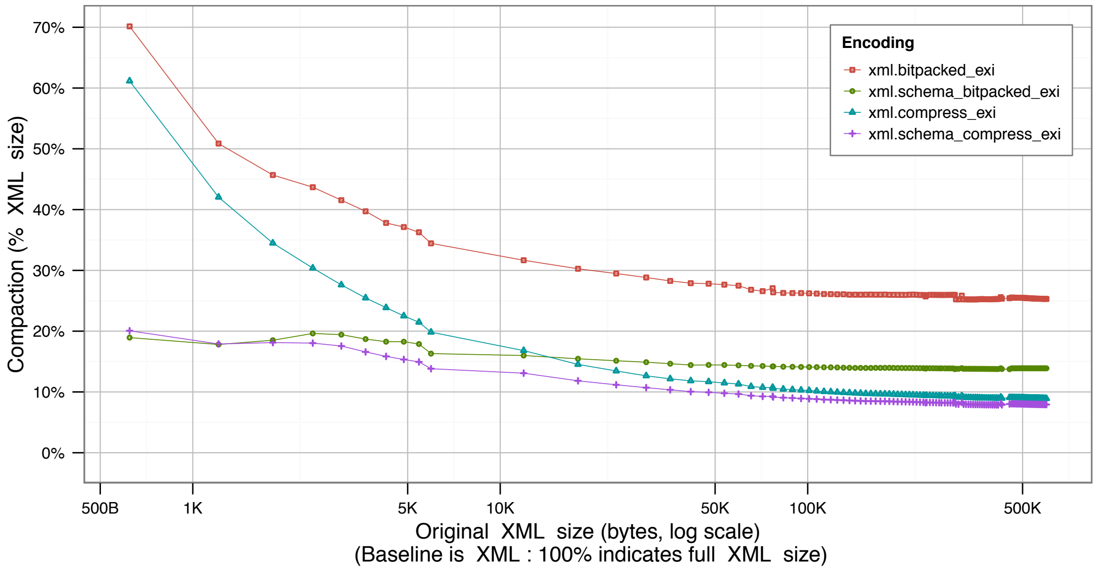

Figure 30. Plot for OpenWeatherMap use case, focus question C.

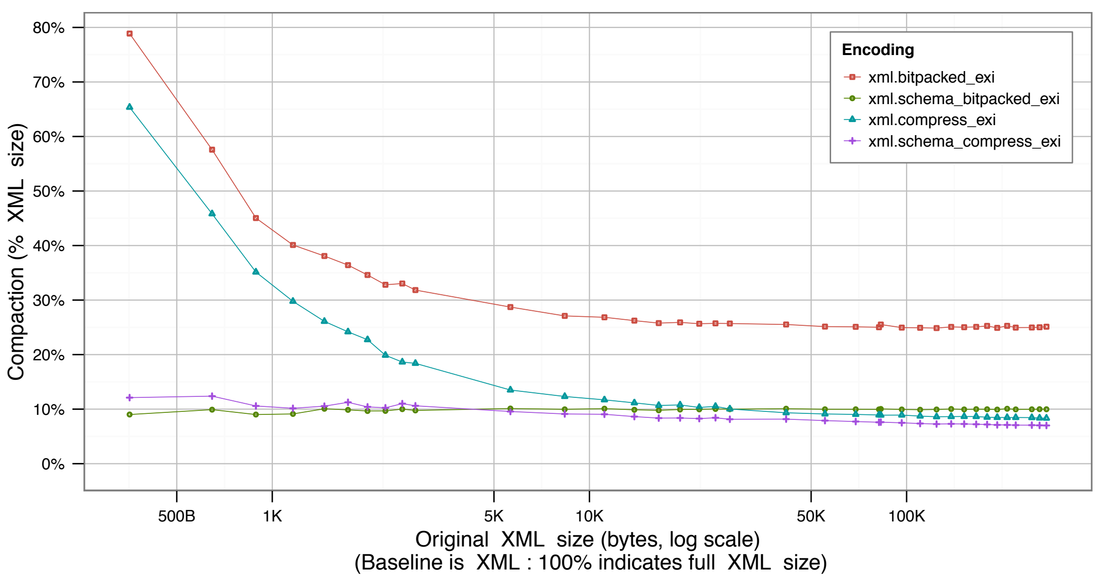

Figure 31. Plot for AIS use case, focus question C.

For schemaless EXI encodings, compress mode was consistently more compact than bitpacked mode across all three use cases. The difference between bitpacked and compress mode encodings increased as subset size increased; compress mode performed better on large subsets than did bitpacked mode.

Bitpacked encodings demonstrated significant compaction benefits from schema information because they use data type information in the schema to reduce the encoding size for those values. For the smallest subset in each use case, the schema-informed, bitpacked encoding compacted the file to ~11–36% of the schemaless bitpacked encoding, though this margin decreased as the subset size increased.

In the AIS and GPX use cases, compaction for schema-informed bitpacked encodings decreased as subset size increased. It improved slightly for the OpenWeatherMap use case. For all three use cases, compaction for schema-informed compress encodings increased with subset size. For the 10 smallest subsets, schema-informed, bitpacked encodings were smaller than schema-informed, compress encodings. Past that point, which was different for each use case, schema-informed, compress encodings were consistently smaller.

4. Focus Questions D and E: Comparison of EXI, BSON and CBOR

To simplify comparison and discussion, focus questions D and E are addressed in tandem here. They are, respectively: Is bitpacked EXI more compact than BSON or CBOR? Is compress-mode EXI more compact than BSON or CBOR post-compressed with conventional compression algorithms?

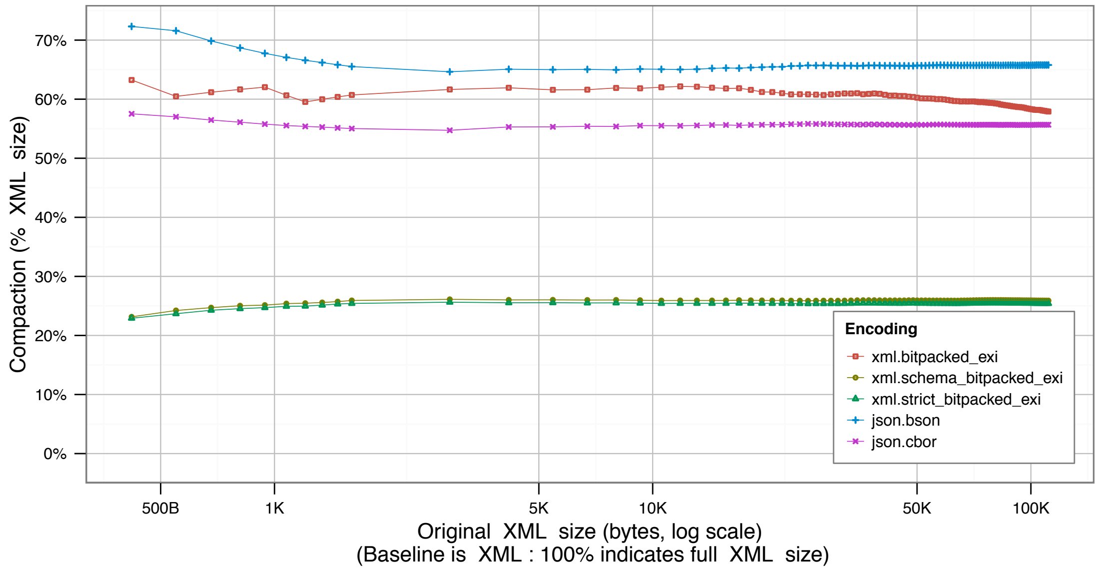

Figure 32. Plot for GPX use case, focus question D.

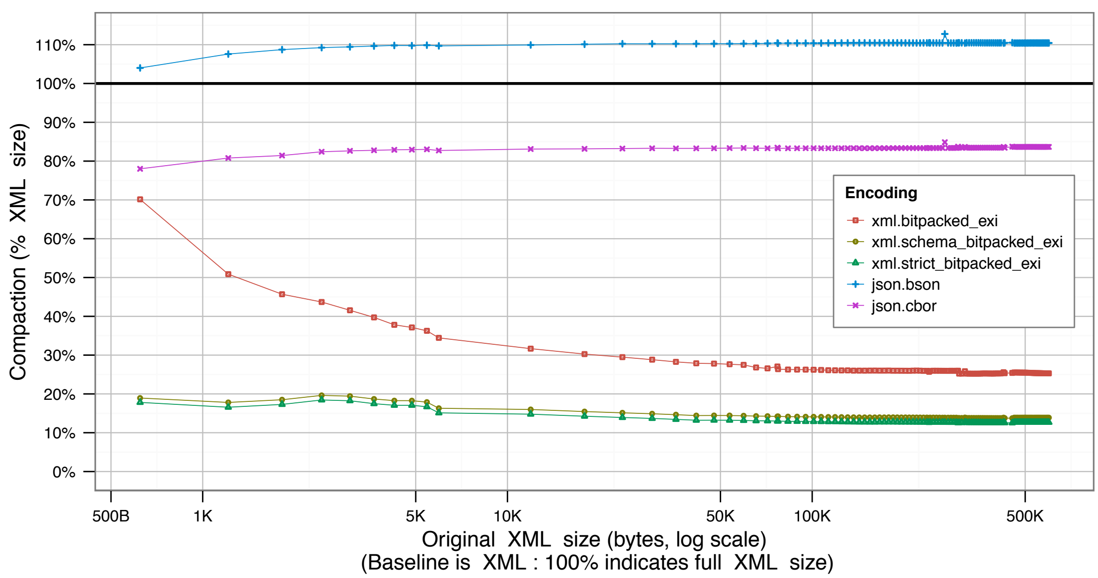

Figure 33. Plot for OpenWeatherMap use case, focus question D.

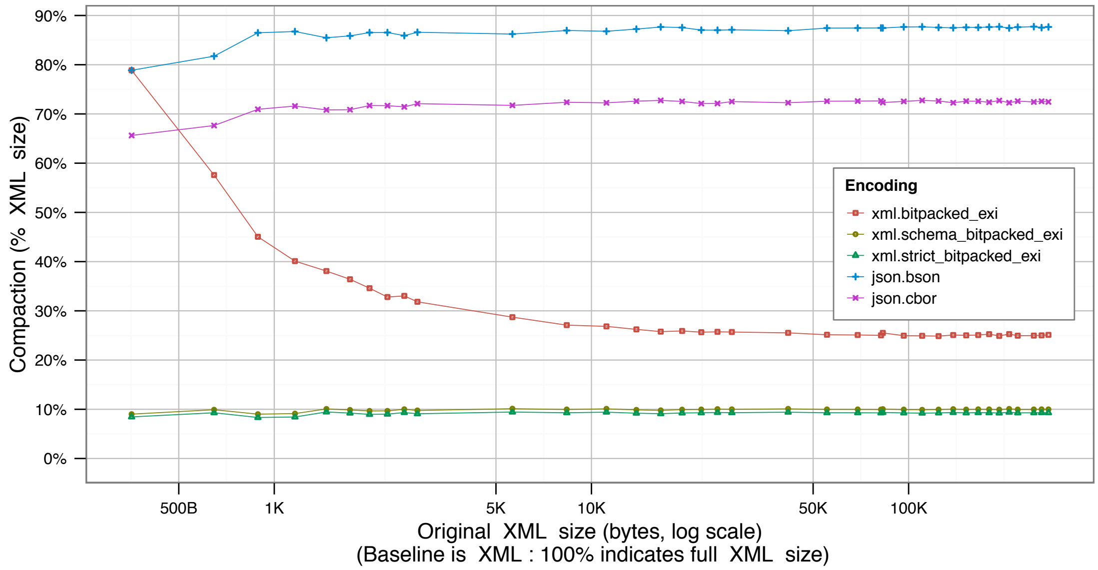

Figure 34. Plot for AIS use case, focus question D.

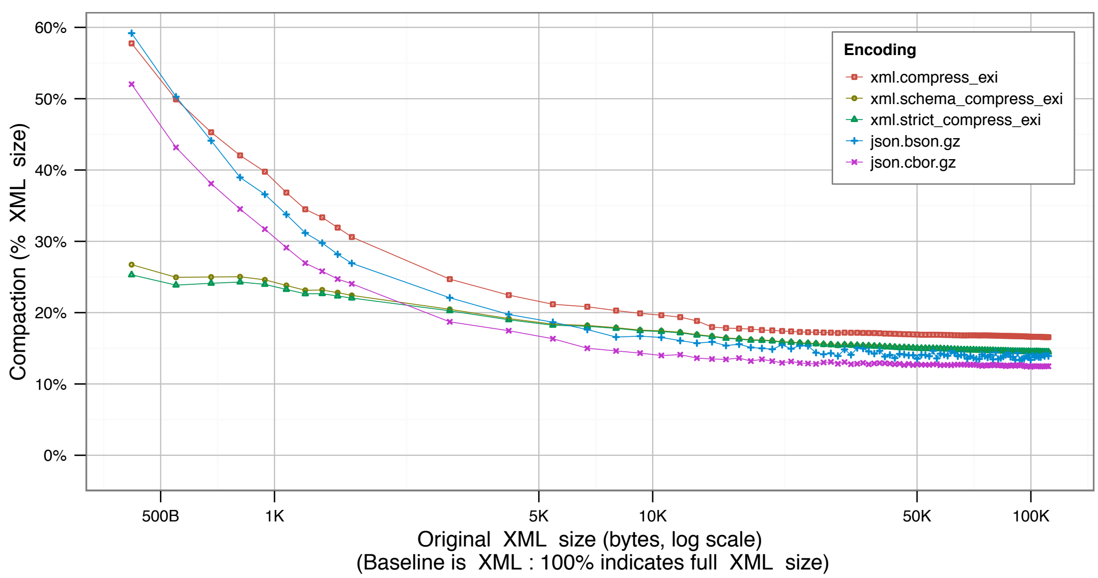

Figure 35. Plot for GPX use case, focus question E.

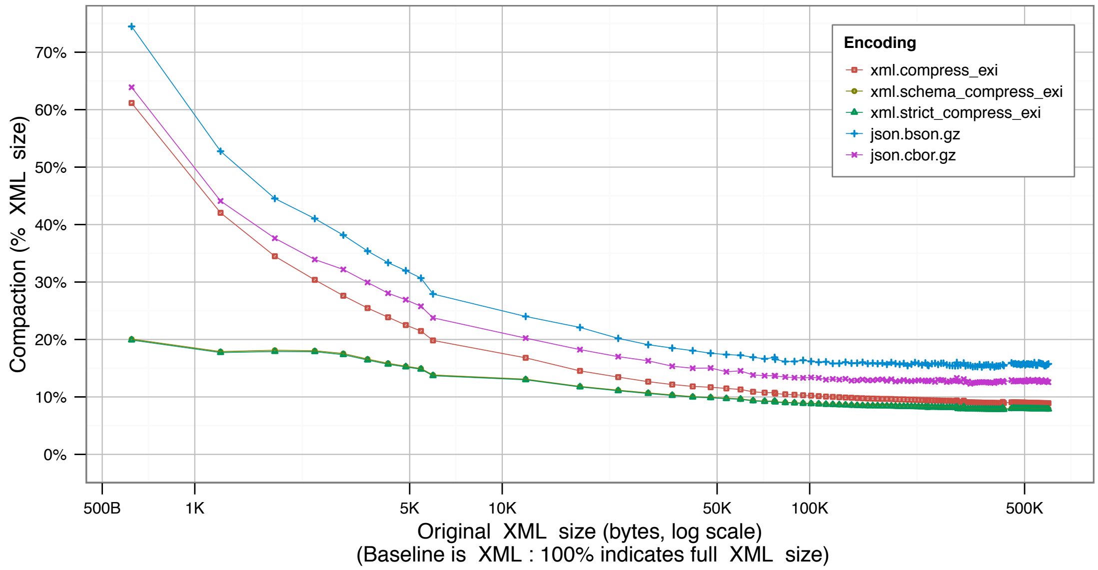

Figure 36. Plot for OpenWeatherMap use case, focus question E.

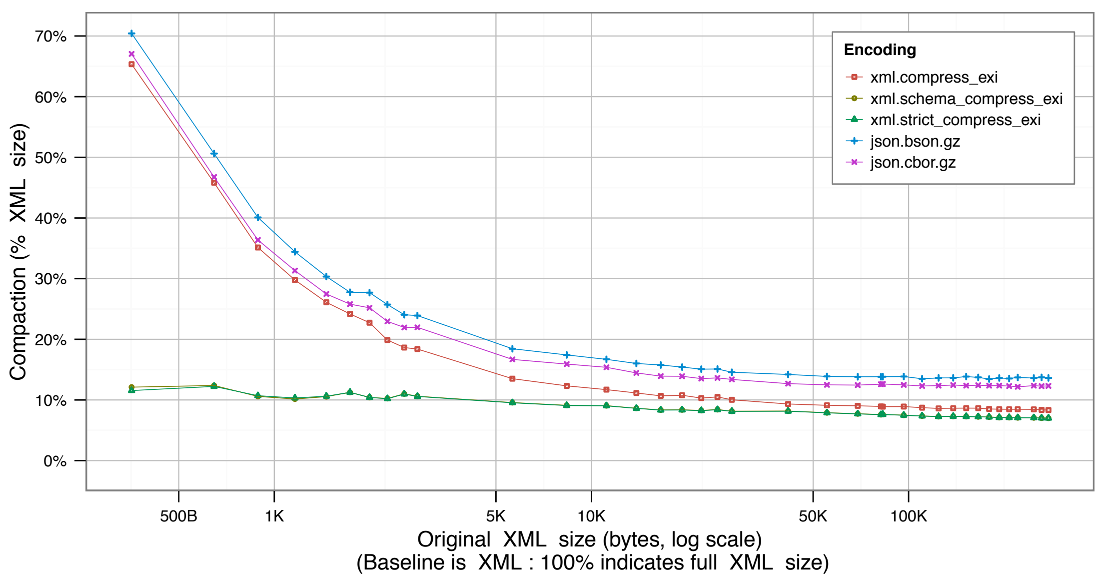

Figure 37. Plot for AIS use case, focus question E.

A relevant comparison of EXI to binary JSON encodings must be divided into two categories. The first, addressed in focus question D, compares encodings with no post-compression, i.e., bitpacked EXI compared to plain CBOR and BSON, with no Gzip or other conventional compression applied. Here, schemaless, bitpacked EXI was more compact than both CBOR and BSON in the AIS and OpenWeatherMap use cases. For the GPX use case, it was less compact than CBOR, but more compact than BSON. However, schema-informed, Bitpacked EXI was between ~15-50% of CBOR size for all three use cases, with an even larger margin over BSON. The trend-lines for BSON and CBOR are generally flatter than for EXI, because these formats do not reuse variable-name tokens as EXI does through grammar-based encoding. BSON is the least compact in all three use-cases, presumably due to additional overhead incurred with string-length prefixing, a feature intended to improve parsing speed at the expense of compactness (BSONSpec.org, 2014).

The second comparison category, addressed by focus question E, compares encodings with post-compression, i.e., compress-mode EXI compared to CBOR and BSON further compressed by Gzip or other conventional compression. Since applying conventional compression as a second step requires additional processing, this must be considered in a separate category. Here, schemaless, compress-mode EXI was more compact than both CBOR and BSON in the AIS and OpenWeatherMap use cases. For the GPX use case, it was less compact than both Gzip-compressed CBOR and BSON. Configured as schema-informed, compress-mode EXI was more compact than CBOR and BSON in the AIS and OpenWeatherMap use cases, particularly for smaller files where it was between ~15-50% of Gzip-compressed CBOR size, with an even larger margin over Gzip-compressed BSON. For the GPX use case, schema-informed, compress-mode EXI was more compact than both CBOR and BSON for small files, but for files larger than ~5KB, both Gzip-compressed CBOR and BSON were more compact at ~80-90% of schema-informed, compress-mode EXI size.

The GPX use case represents an outlier: CBOR and BSON achieve superior compaction due to different encodings of high-precision decimal values. The GPX files used 13–15 decimal places of precision for elevation, latitude and longitude values, which comprise ~70% of the plain-text XML file. The corresponding XSD defined these with the decimal data type. In EXI, a decimal data type value between -180 and 180 with 14 places of precision encodes as 66–82 bits, whereas in CBOR and BSON, it encodes as a 72 bits for the identifier tag and a 64-bit double. Given that 14 decimal places of precision exceed the actual sensitivity of the GPS recording device or available GIS systems (for example, the GPX sample files recorded elevations at sub-millimeter precision), the truly useful information in the message could likely be represented with less precision, and thus fewer bits using EXI (J. Schneider, personal communication, October 20, 2014). The floating-point format used in CBOR and BSON is a potentially lossy representation of decimal values, but the author's visual inspection of the decoded files showed no loss of fidelity. Further investigation using an XML float data type is warranted.

5. Focus Question F: Improvement over Gzip

For a network already using Gzip compression, do any of the tested binary encodings offer better compactness?

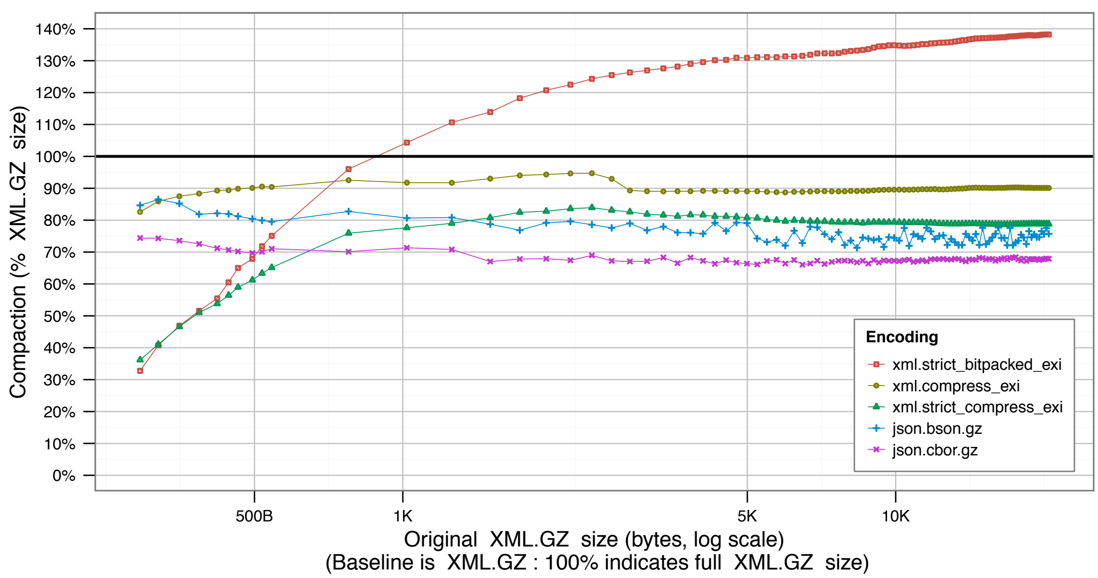

Figure 38. Plot for GPX use case, focus question F.

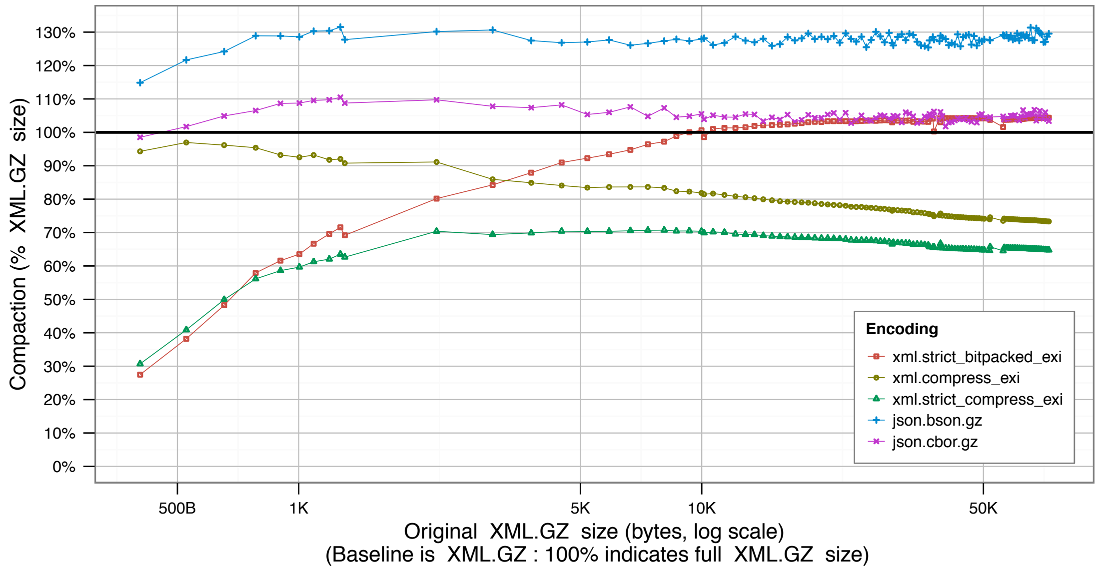

Figure 39. Plot for OpenWeatherMap use case, focus question F.

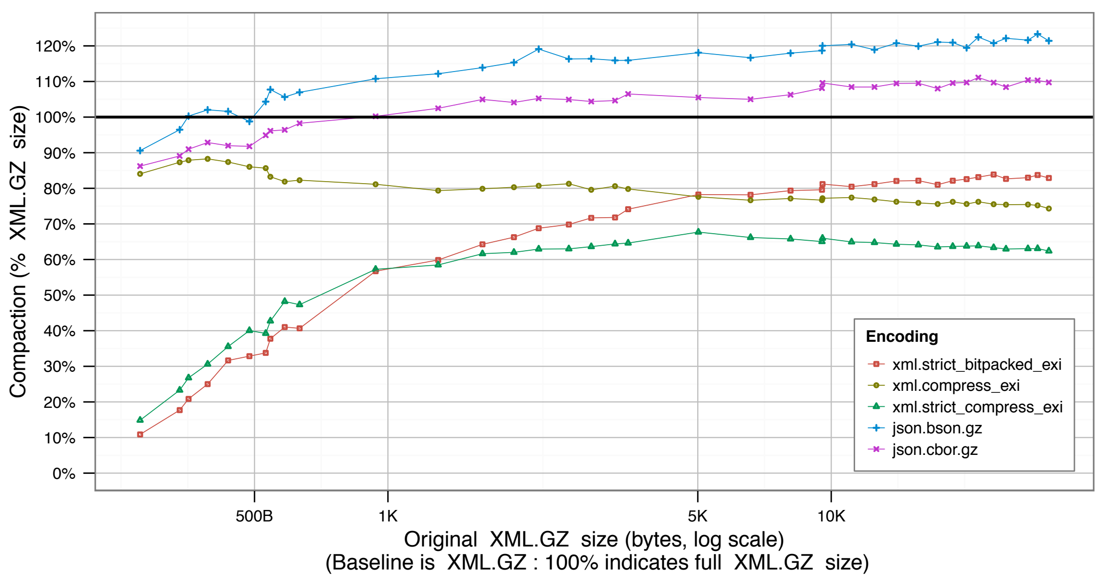

Figure 40. Plot for AIS use case, focus question F.

The performance of binary formats varies with both the subset size and with various characteristics of a use-case. For the 10 smallest subsets, the schema-informed EXI encodings, in both bitpacked and compress modes, produced files ~10–50% the size of Gzip-compressed plain-text XML. For larger subsets, the schema-informed, compress-mode EXI encodings produced files ~60–75% the size of Gzip-compressed plain-text XML. By comparison, the JSON-based binary encodings for small subsets produced files ~75–95% the size of Gzip-compressed plain-text XML, and for larger subsets they ranged from ~75–135%.

6. Focus Question G: Schema Impact

Do restrictions on data types and range restrictions in an XSD significantly impact the compaction of schema-informed encodings? This question is addressed only for the AIS use case.

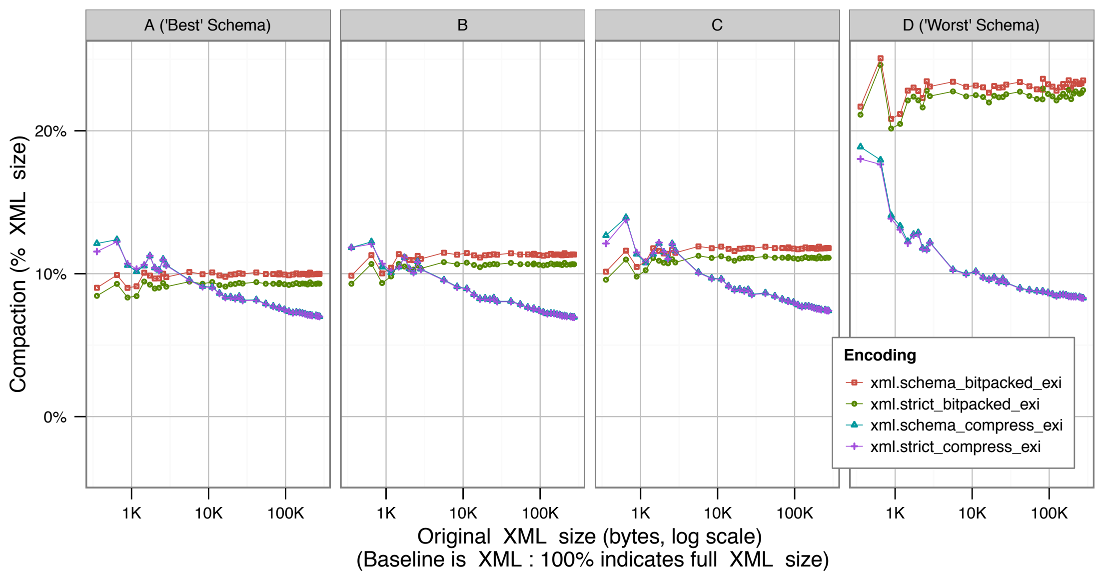

Figure 41. Plot for AIS use case, focus question G.

The sample files for the AIS use case were relatively simple, with few nested elements or complex structures. Modifications to the schema focused on proper selection of data types and ranges and the resulting variation in compaction hinged upon those changes. For each successive improvement in the schema, compaction for both bitpacked and compress-mode EXI encodings improved. The encoding with the most restrictive schema resulted in files ~45% the size of those encoded with the least restrictive schema.

C. RESULTS BY FOCUS QUESTION—LARGE-FILE CATEGORY

This section presents empirical data for each focus question in the large-file category. The focus question is reiterated, followed by corresponding plots from each of the three large-file use cases: DFXML, PDML and OpenStreetMap, in that order. A brief discussion follows, addressing results across the use cases in response to the focus question.

1. Focus Question A: Primary EXI Modes

How do the primary EXI modes compare for schemaless and schema-informed encodings?

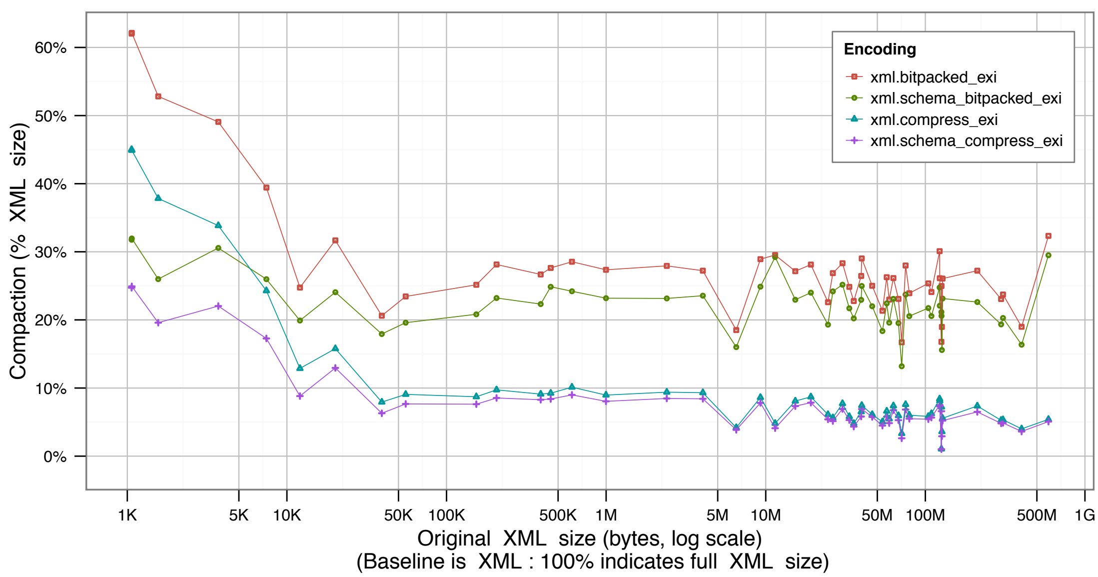

Figure 42. Plot for DFXML use case, focus question A.

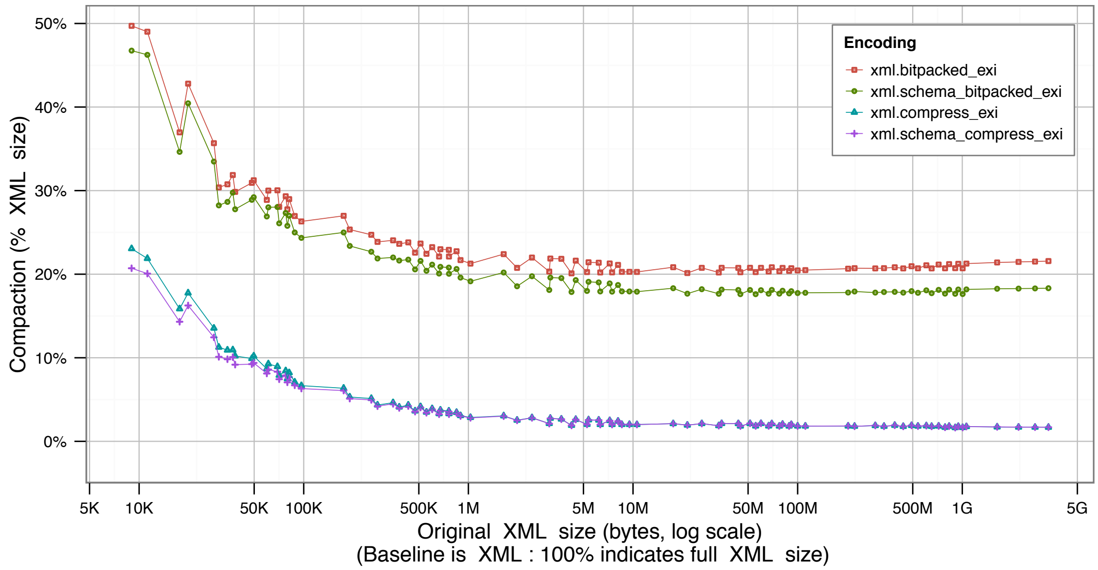

Figure 43. Plot for PDML use case, focus question A.

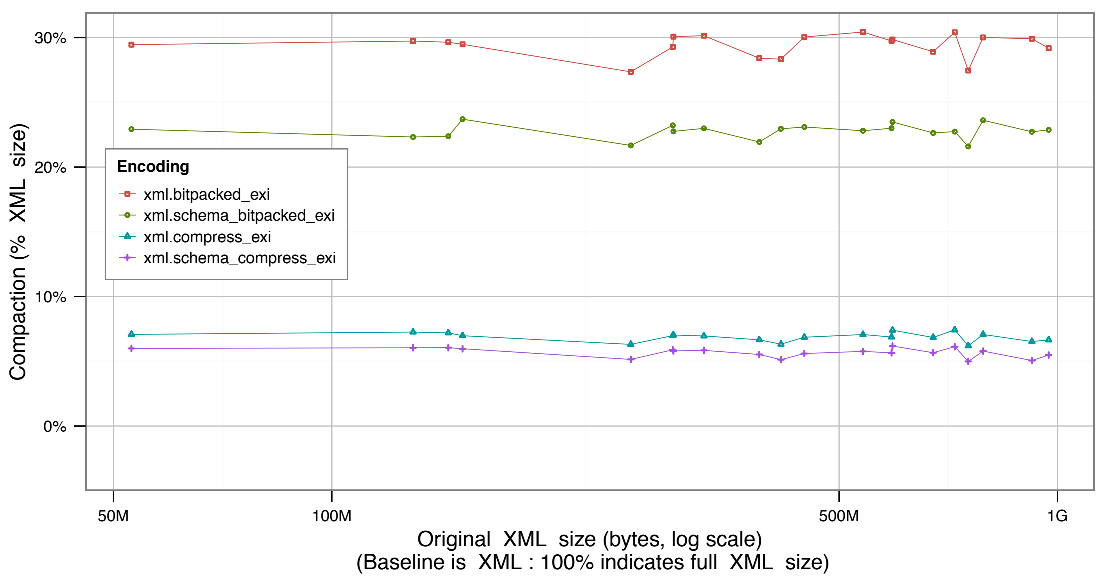

Figure 44. Plot for OpenStreetMap use case, focus question A.

Across all three use cases, compress mode consistently produced the smallest files, ~1–10% of plain-text XML size for original file sizes larger than 1 MB, and ~16–35% the size of the corresponding bitpacked encoding. With schema-informed encodings, all results were equal to or more compact than their schemaless counterparts. In the

DFXML and PDML use cases, the schema information impacted files larger than 1MB less than it did for files below 1MB. For plain-text file sizes greater than 1GB, the difference between schemaless and schema-informed compress-mode encodings was insignificant.

2. Focus Question B: Strict Schema Conformance

Does the strict option significantly improve compaction for schema-informed encodings?

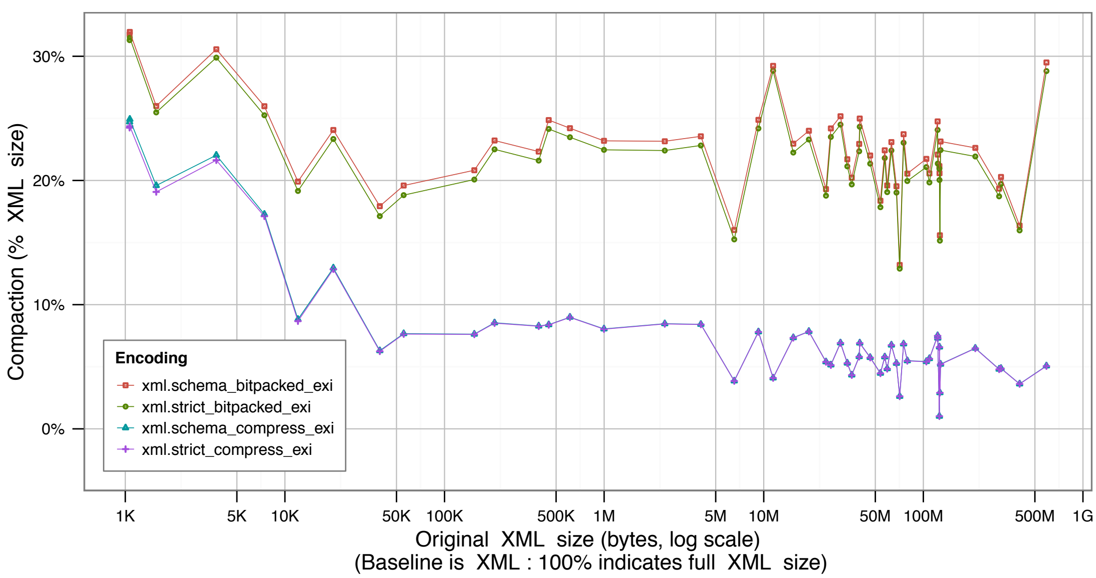

Figure 45. Plot for DFXML case, focus question B.

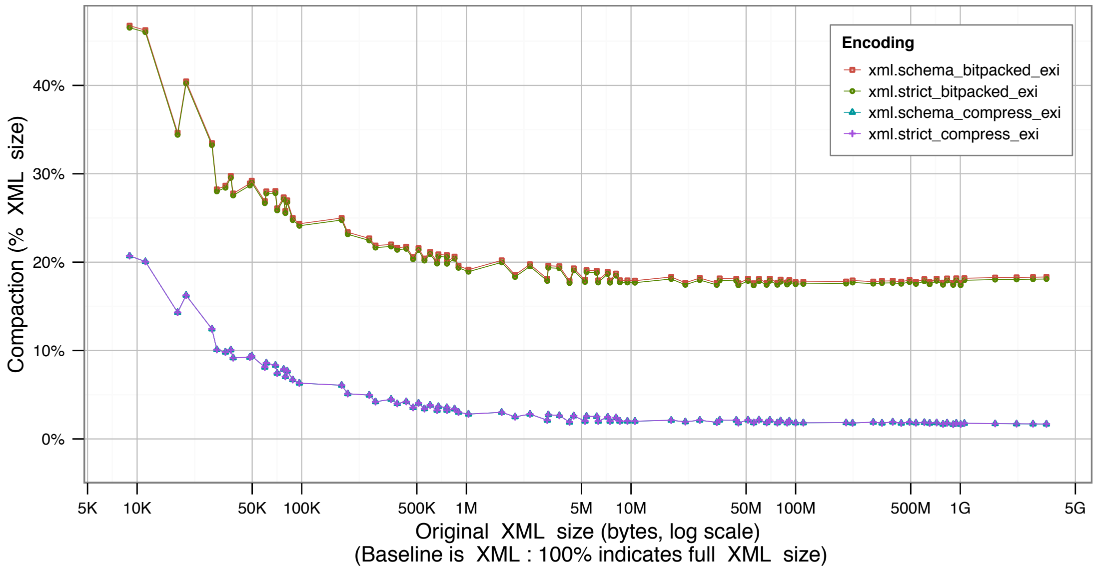

Figure 46. Plot for PDML use case, focus question B.

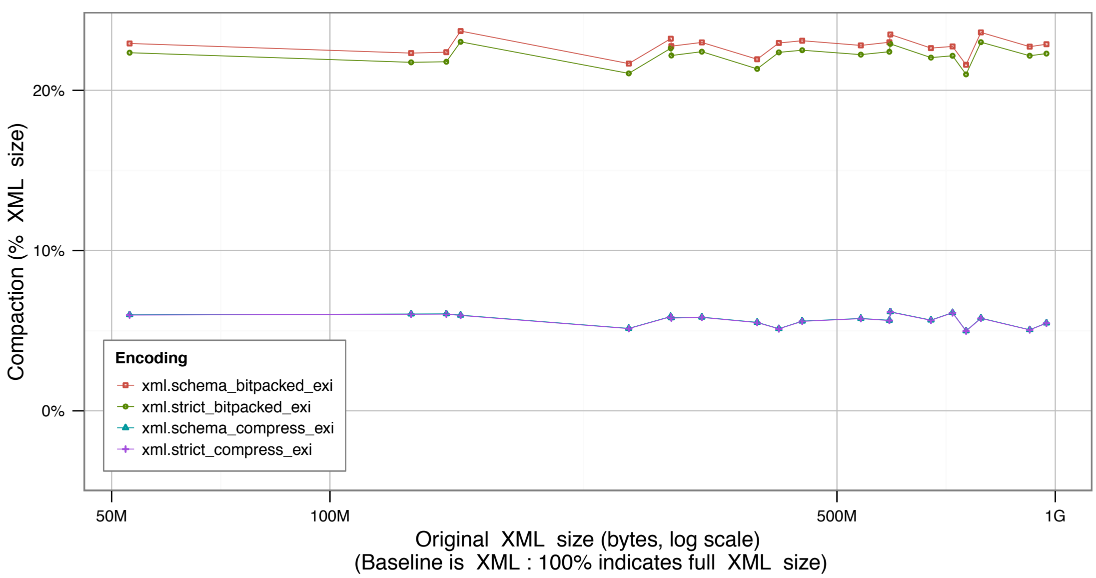

Figure 47. Plot for OpenStreetMap use case, focus question B.

In all three use cases, regardless of whether bitpacked or compress modes were used, the strict option made only minor impacts on compaction. It resulted in files ~97–100% the size of their non-strict counterparts. EXI encodings created without the strict option set will accept XML that does not conform to the given schema. The difference in compaction in these plots shows the benefits associated with configuring systems to generate schema-valid XML.

3. Focus Questions C and D: Post-Compression other than DEFLATE

As these two questions are closely related with similar outcomes, they are presented together here for simplicity. They are, respectively: Do any of the tested conventional compression algorithms better compact a schemaless, precompress EXI document than EXI's default DEFLATE? Do any of the tested conventional compression algorithms better compact a schema-informed, precompress EXI document than EXI's default DEFLATE?

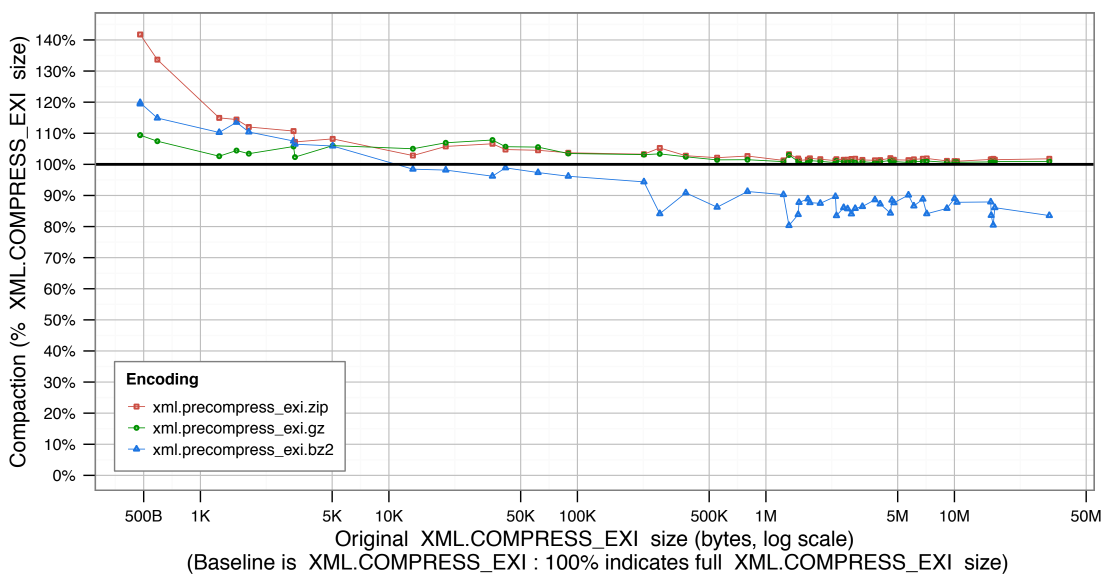

Figure 48. Plot for DFXML case, focus question C.

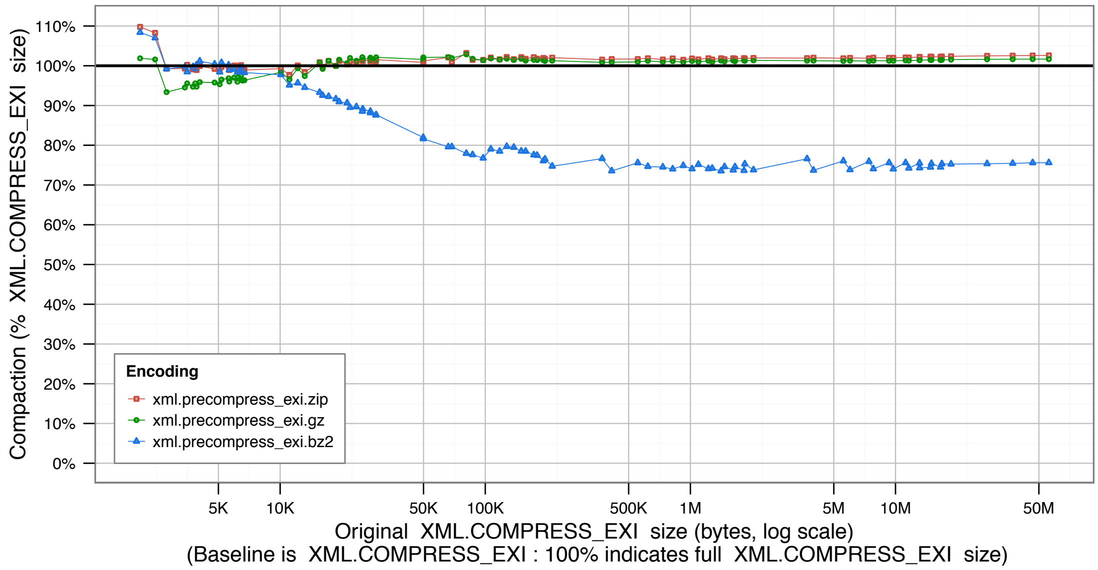

Figure 49. Plot for PDML use case, focus question C.

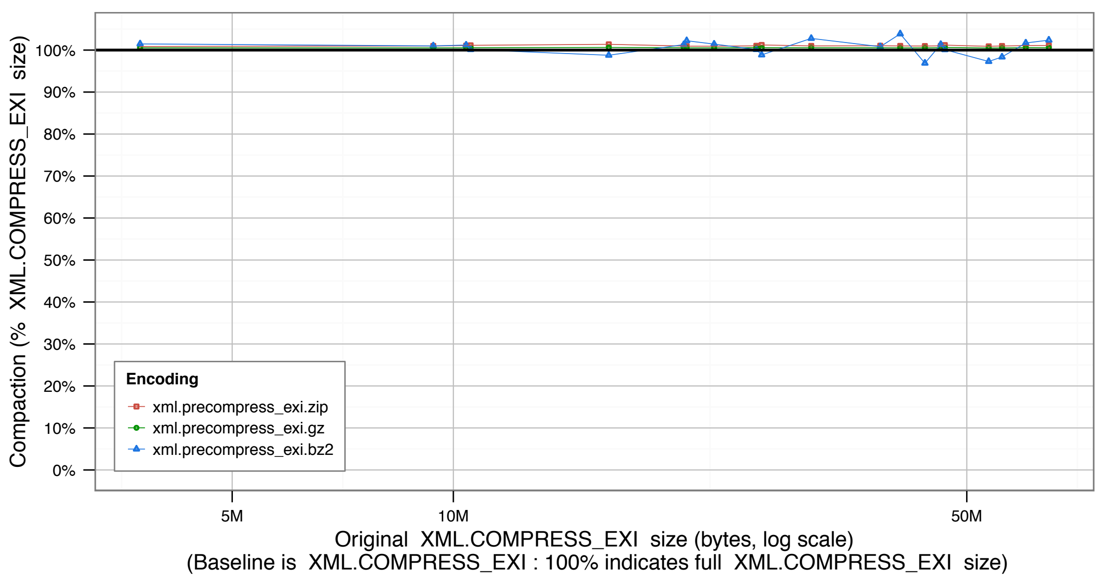

Figure 50. Plot for OpenStreetMap use case, focus question C.

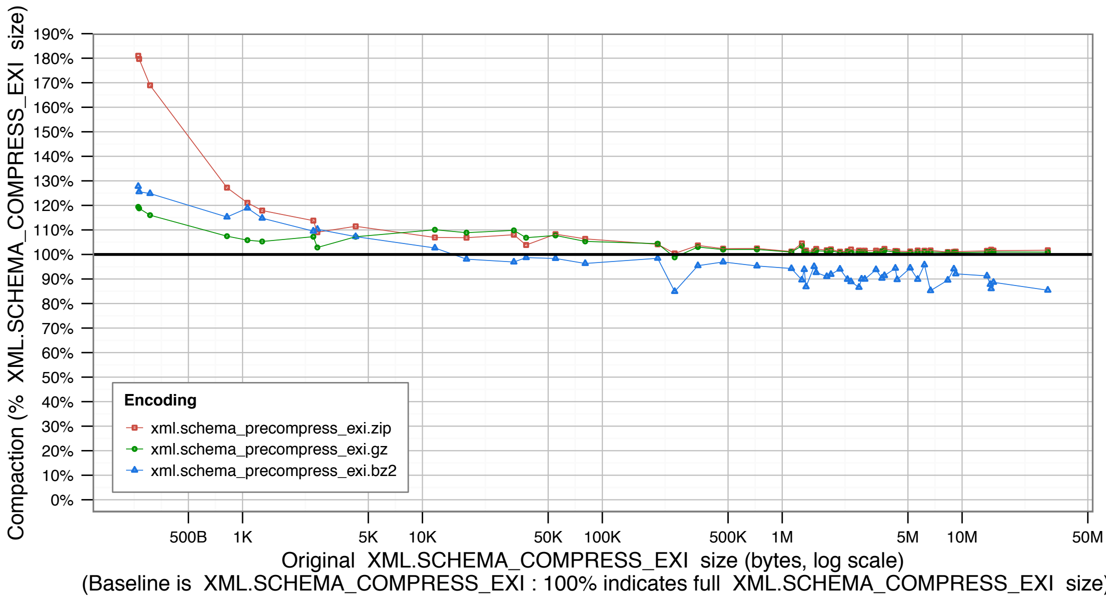

Figure 51. Plot for DFXML case, focus question D.

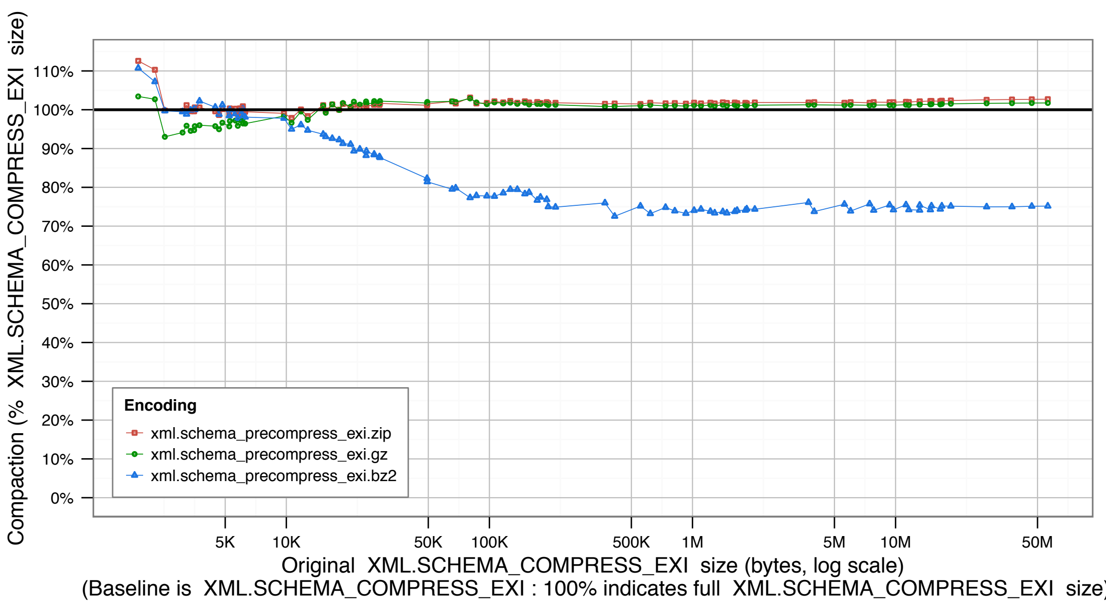

Figure 52. Plot for PDML use case, focus question D.

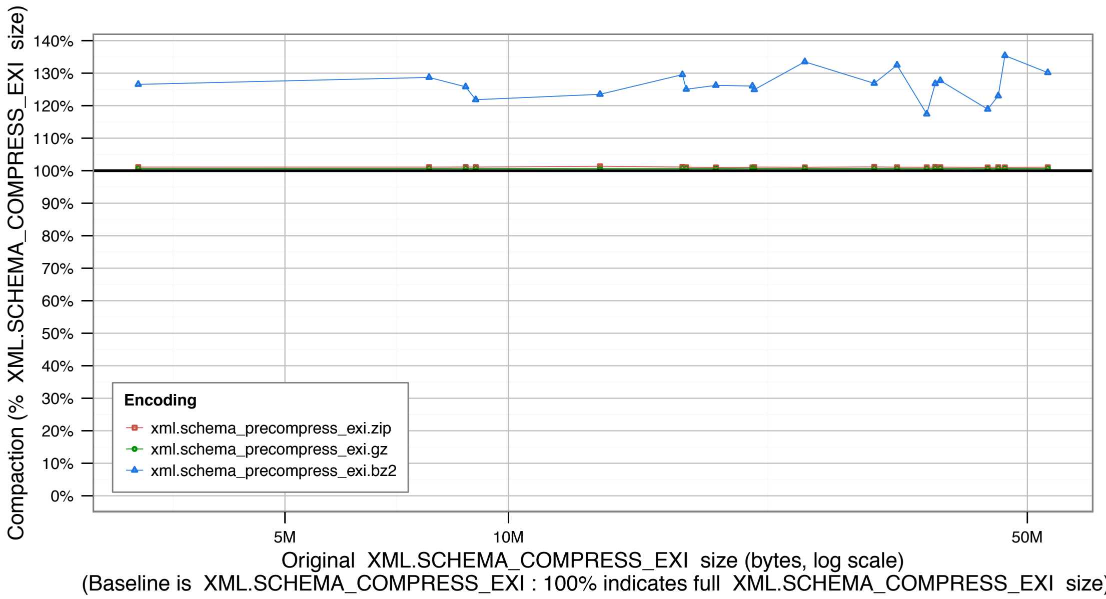

Figure 53. Plot for OpenStreetMap use case, focus question D.

In general, Gzip and Zip did not perform better on either schemaless or schema-informed documents than the EXI's DEFLATE algorithm. The one exception was in the PDML use case, where Gzip compacted files with original XML sizes of ~2.5–25KB to ~92–99% of precompress EXI using DEFLATE. For files larger than 1MB, Bzip2 improved compaction in the DFXML and PDML use cases, but decreased compaction in the OpenStreetMap use case.

4. Focus Question E: Most Compact EXI Configuration

Which EXI encoding is the most compact?

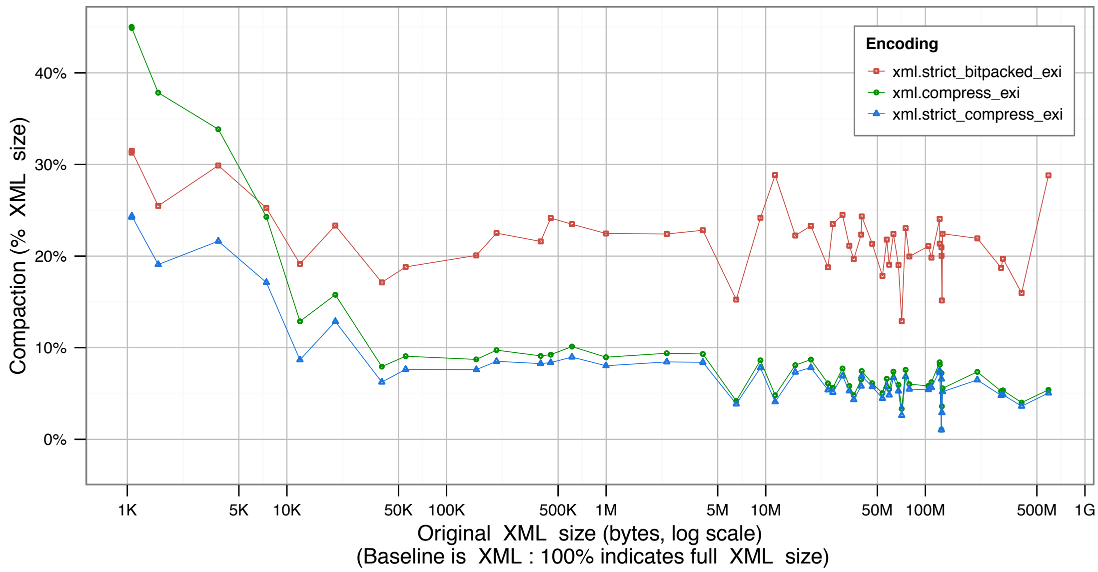

Figure 54. Plot for DFXML case, focus question E.

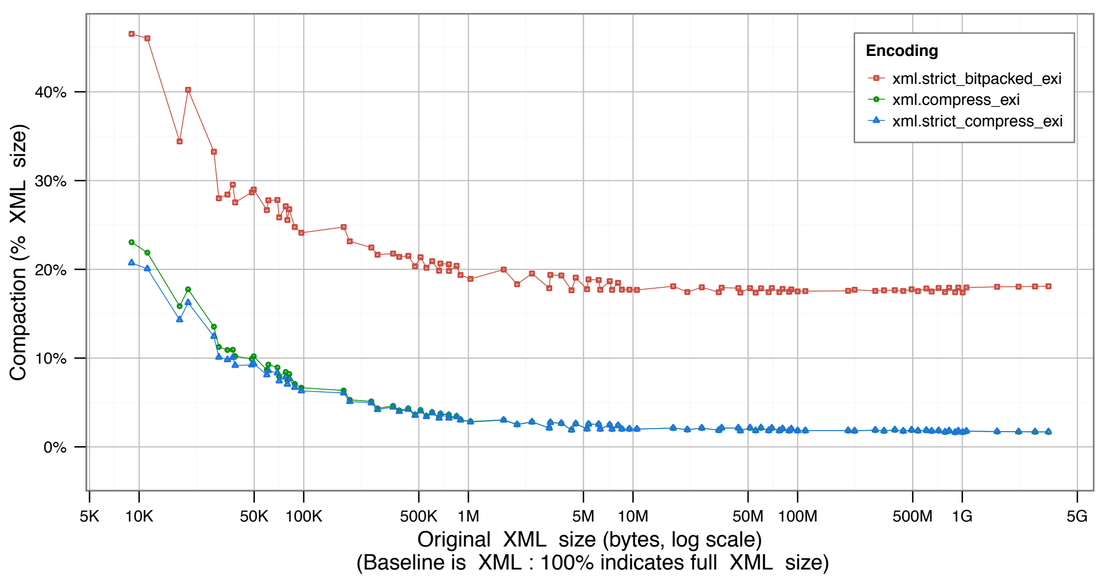

Figure 55. Plot for PDML use case, focus question E.

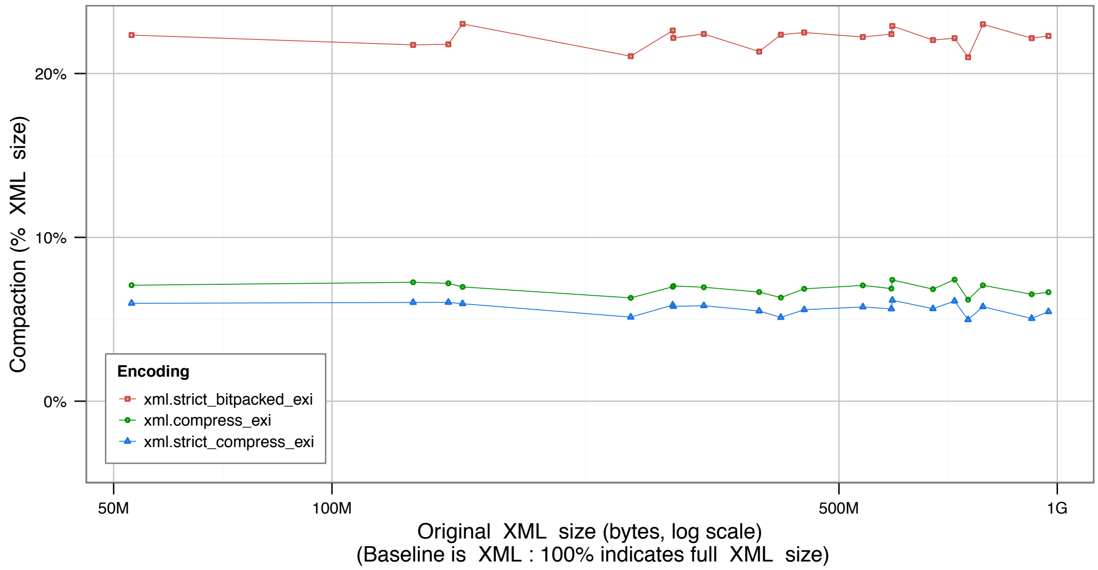

Figure 56. Plot for OpenStreetMap use case, focus question E.

For sample files across all three use cases with original XML size greater than 10KB, bitpacked EXI with the strict option set was the least compact encoding, and produced files two or more times the size of the closest compress-mode EXI encoding.

Overall, compress-mode and strict compress-mode encodings were the smallest for large files. For sample files in the DFXML and PDML use cases with original XML size greater than ~1–5MB, compaction for compress-mode EXI with and without the strict option set was nearly identical, but for smaller sizes diverged, with the strict option adding significant compaction to smaller sample files. In the DFXML use case, the strict option reduced a 1KB sample file to 54% the size of the encoding without the strict option. For all sample files in the OpenStreetMap use case, whose original XML size ranged from ~50MB–1GB, the strict option increased compaction, reducing the size of compress-mode EXI encodings by ~17–20%.

5. Focus Question F: Improvement over Gzip

For a network already using Gzip compression, do any of the EXI encodings offer better compactness?

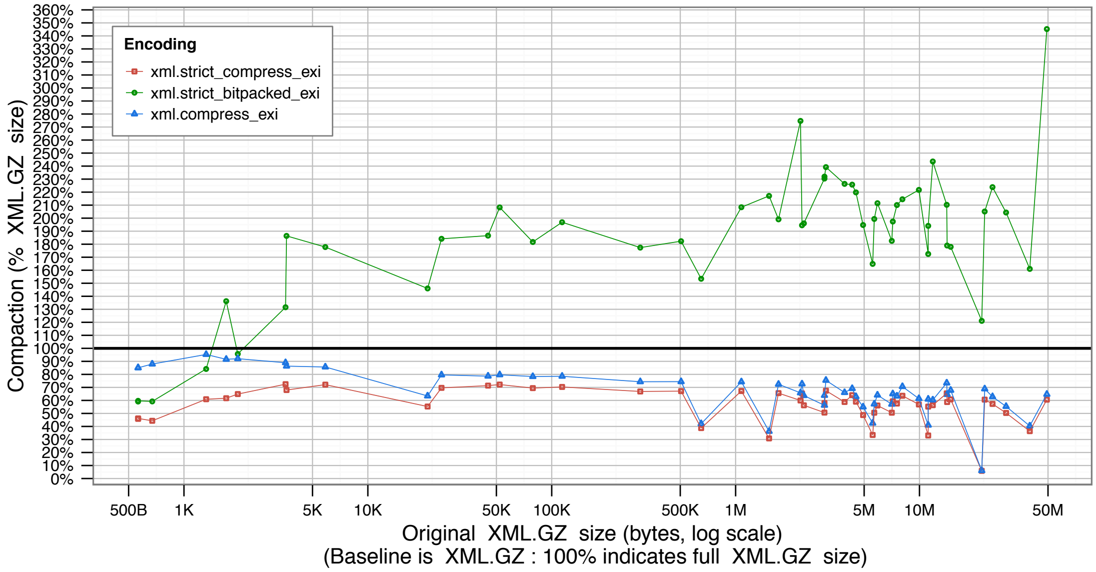

Figure 57. Plot for DFXML case, focus question F.

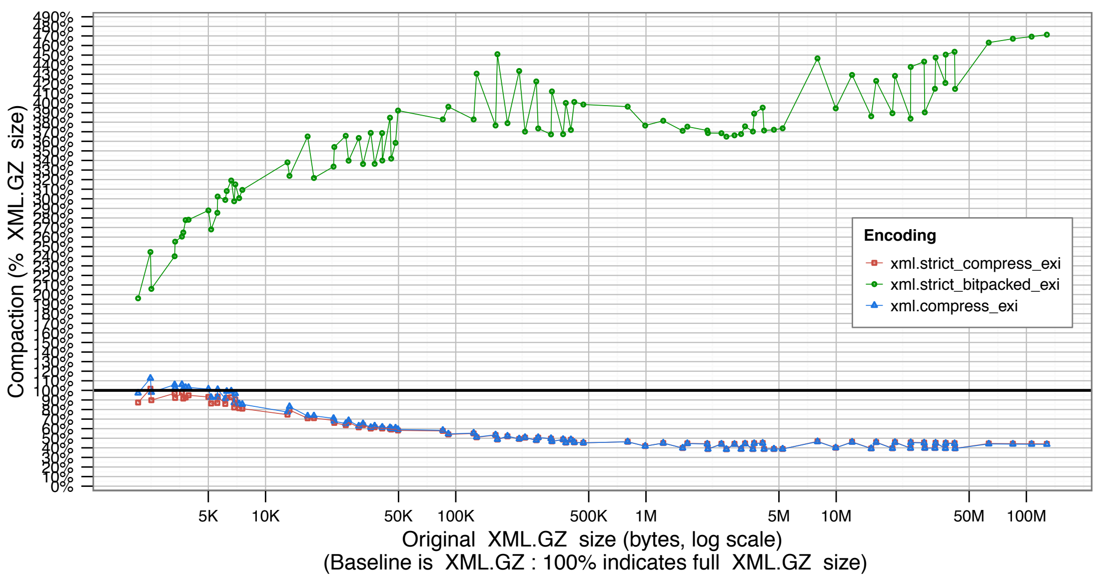

Figure 58. Plot for PDML use case, focus question F.

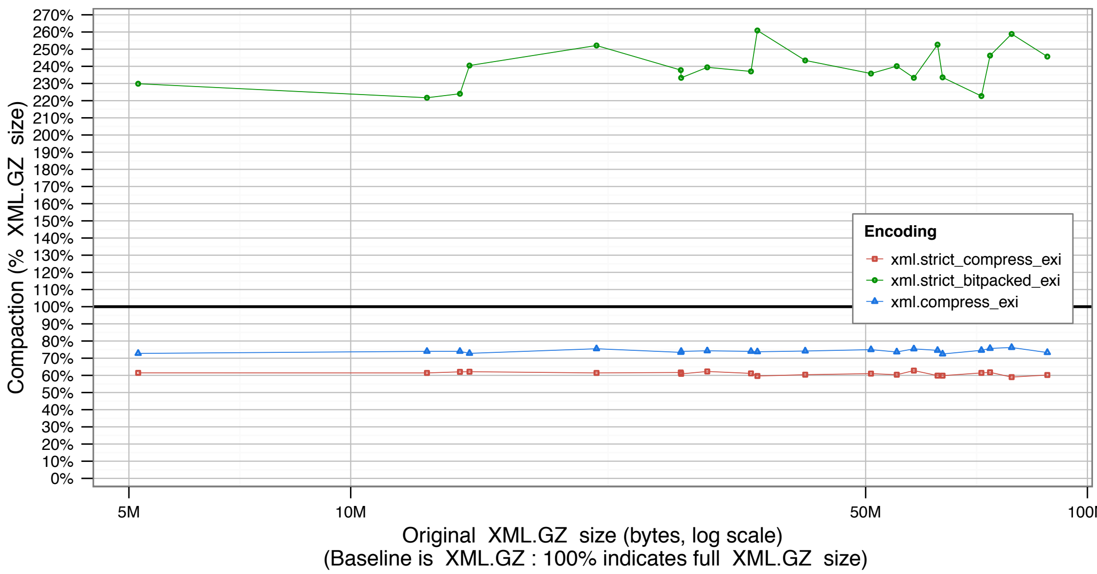

Figure 59. Plot for OpenStreetMap use case, focus question F.

For all three use cases, bitpacked EXI using the strict option generally resulted in files two or more times the size of plain-text XML compressed with Gzip.

Compress-mode EXI, both with and without the strict option, resulted in significant improvements. In the DFXML use case, the compress-mode EXI encodings were ~30-85% the size of plain-text XML compressed with Gzip for files larger than 1MB. The corresponding results for the PDML and OpenStreetMap use cases were ~40–95% and ~60–75%, respectively.

D. CHAPTER SUMMARY

Experimental results for small-file use cases showed significant variance in EXI encoding size depending on the structure of the XML data itself, the data type information in the associated XSD, and the configuration of the EXI codec itself. In general, for any sample file in the small-file category, there was at least one EXI configuration that produced a more compact encoding than BSON or CBOR while maintaining semantic equivalence. Results for large-file category suggest that performs no worse on large XML files than suggested by previous work, but that beyond 1MB file sizes, compaction rates remain static within a use case as file size increases.

V. CONCLUSIONS AND RECOMMENDATIONS

This chapter presents broad conclusions from comparing EXI to binary JSON formats and from testing EXI compaction of large XML files, and presents the author's recommendations for future work identified in the course of the research.

A. SMALL-FILE CATEGORY

The following section outlines the major findings of this research stemming from the small-file use cases: diminishing gains from compressing larger files suggest an optimal point to begin transmission to maximize network throughput; the configuration of EXI options as well as an XSD make significant differences in the final size of an EXI encoding; and in most cases there is at least one EXI configuration that produces smaller files than BSON and CBOR.

1. When To Send?

A common metric for a web-application's performance is responsiveness, or how quickly a user receives a response after making a request. The size of data transmitted in an AJAX-style web application is one of myriad factors affecting response time. For applications using AJAX for frequent data exchanges between the client and server, intentionally waiting to aggregate multiple responses prior to transmission can decrease overall data size. Tsai, Chen, Huang and Hu (2011) investigated this in the context of image aggregation through a proxy server for mobile mashup applications, and found that it increases performance and reduces HTTP overhead. Liefke and Suciu (2000) also explored the relationship between file size and XMill compression ratio with results suggesting that more messages, if aggregated, compact more. Only one of their datasets, Weblog, had a temporal component relevant to this concept, i.e., the increasing file size reflected more data generated over time. For the AIS and GPX use cases analyzed in this research, there is a similar temporal relationship between each "message" that extends their results for the EXI algorithm.

By compressing sequential subsets of a larger file, this experiment investigates this aggregation for textual data in conjunction with conventional compression. The results suggest that similar responses, when combined, tend to create additional redundancy in the data and improve compaction when compressed with generic lossless compression algorithms such as DEFLATE, Gzip, or Zip. Beyond web applications, the notion of aggregation has potential for remote sensor networks collecting real-time information and transmitting back to a central node. In such a scenario, the decision to transmit data every 5ms, 5s or 5min presents a tradeoff in overall timeliness of information versus bandwidth consumption. In tests of a single-server streaming architecture over HTTP, a buffering delay of 0.5 seconds, both improved compression and decreased processor utilization for a high-capacity web server (Fisteus, García, Fernández, & Fuentes-Lorenzo, 2014).

However, the intentional delay during aggregation may decrease responsiveness in a situation where available bandwidth is not the limiting factor. Also, the results indicate that for a specific use case, compaction increases quickly for small aggregations, and then flattens as subset sizes increase. Thus, delaying transmission too long is not likely to increase performance. As the compaction curve flattened at different points for each of the small-file use cases, the optimal ‘when to send’ point is likely a function of the application’s data, the needs of the application, and the characteristics of the larger network. The variation across use cases in this research suggests empirical investigation is the best method for identifying the balancing point between compaction benefits and operational requirements.

2. Significance of EXI Configurations

EXI encodings are highly configurable, and no single configuration is the best for all use cases. The EXI standard defines several options that affect the final size of an EXI encoding, as well as memory footprint, implementation simplicity, and processing speed (Schneider et al., 2014). It lists default settings for each option that decrease the size of XML documents in all cases. However, as this experiment’s results indicate, in many cases those defaults do not produce the smallest possible file size, and may not produce files smaller than conventional Gzip compression. A default configuration meeting both of those criteria for all possible XML documents is simply not possible. Configuration of the EXI codec, informed by empirical testing on a case-by-case basis, is necessary to achieve the best compaction. Each adjustment thus made will also affect other factors such as memory consumption or encode and decode speed, which are not addressed in this research.

Even within a single use-case or application, there may not be a single best EXI configuration. As the three small-file use case results indicate, there can be a transition point in file sizes where Compress-mode begins to produce more compact files than Bitpacked. If an application transmits messages on both sides of this divide, a static EXI configuration for all transmissions is not optimal. Another scenario could be where a server transmits EXI documents to some clients with memory limitations and others without, where DEFLATE decompression for large block sizes may not be feasible. In such cases, a dynamic, mixed-mode encoder could adjust the EXI parameters on a per-message basis and achieve better results overall.

3. Significance of XML Schema

The XML Schema specification identifies the purposes of XSDs as primarily definition, documentation and robust validation of XML documents (W3C, W3C). However, the role of the XSD in EXI compression adds a new function to this list. The empirical results in this work, both for the small and large-file use cases, reinforce the notion that XSDs improve EXI compaction regardless of which mode EXI is configured for, particularly true for small XML files and Bitpacked encodings.

The simple presence of a schema, however is not sufficient to maximize EXI compression. Rather, as suggested in the EXI Specification and validated through the empirical results in this work, selection of data types and range restrictions in an XSD can significantly improve compaction for Bitpacked encodings (Schneider et al., 2014). For the AIS use case, a schema with well defined, restricted data types resulted in Bitpacked file-sizes less than half that produced with no data types at all. An XSD schema, formerly a tool strictly for validation of data, serves also a tool for compression with EXI. This also presents a tradeoff space between maintaining generality of a schema, or tightly restricting it to achieve maximum compaction. Data type representation maps, described in the EXI specification but not tested during this research, provide another method for efficiently representing custom data types that do not conform to the basic data types included in the EXI specification (Schneider et al., 2014).

In general, the author believes that for all of the sample files tested, there likely exists an alternate combination of XSD and EXI codec configurations that could produce semantically equivalent but more compact encodings than found in this research. The implication for developers working with EXI is that though the default options are indeed more efficient than plain-text transmission, the true potential of EXI lies in an in-depth understanding of one's own data, the EXI specification, and a customized marriage of the two.

4. EXI and Binary JSON Encodings

The process of collecting sample files and testing compression figurations to highlighted a major challenge, compounded with the introduction of multiple binary formats and configurations, inherent to any comparison of XML to JSON: conversions from XML to JSON and vice-versa are not one to one. Many configurations exist which this work did not examine. Also, the tested binary encodings of JSON differ from EXI in that specific data type encodings for CBOR and BSON are limited, relative to the impact XSD information makes on EXI encoding sizes. Technologies such as JSON Schema, if incorporated into the BSON/CBOR representations, could provide similar benefits.

As tested, however, a properly parameterized EXI encoding is more compact that JSON in the majority of cases. However, identifying such parameters may require examination and adjusting of data on a per-use case basis. This shows a need for defining best-practice EXI properties for common XML documents and schemas, both in the fleet and in industry.

B. LARGE-FILE CATEGORY

The findings from the large-file use cases, in addition to reinforcing the previously statements about significance of EXI configurations and schema, provided two additional findings. First, EXI performs as well on files between 100MB and 4GB as it does on files less than 100MB, but it does not necessarily perform better. Second, large XML files can be computationally expensive to process and applications that must transfer them over networks may be better served to break them into multiple, smaller files.

1. EXI Performs Well on Large Files

The results from the three large-file use cases both extend and validate work by the W3C working group and previous NPS research indicating that EXI consistently produces more compact encodings than Gzip for plain-text XML files up to 100MB (Snyder, 2010; White et al., 2007). In brief, this work found that across the three use cases, EXI consistently produced more compact encodings than conventional compression algorithms for plain-text XML files between 100MB and 4GB. For the three use cases tested, schemaless EXI in Compress mode produced files between ~35–75% the size of Gzip. Schema-informed EXI in Compress mode offered equivalent or better results, generally ~30–60% the size of their schemaless counterparts. As noted in the EXI standard, Bitpacked mode is not ideal for large files, and in all three use cases, produced encodings significantly larger than Gzip (Schneider et al., 2014).

2. Compaction Plateaus as Size Increases

Though EXI does perform well on large documents, the test results did not indicate that, within a given use case, EXI performs better on plain-text files larger than 100MB documents than on files between 1MB and 100MB. Trends from all three use cases suggest that EXI compaction for a given use case plateaus as plain-text file size grows past ~1MB. The author's experience with testing EXI compression for large files during this research was that large XML files can be memory intensive and slow to process, not solely for EXI compressors but other software tools as well. Thus, since large XML files can be difficult to process and EXI compaction doesn't improve with additional file size beyond 1MB, it may be better to identify the point of diminishing compaction gains and use that to break an XML file into chunks prior to EXI encoding and transmission. For applications performing just-in-time compression immediately prior to transmission this approach could provide a balance between file-manageability and compression—it would begin transmission sooner and thereby reduce unutilized bandwidth. However, for applications that maintain large files in persistent storage prior to transmission, EXI compression overhead incurred once could pay dividends over the course of many downloads.

C. RECOMMENDATIONS FOR FUTURE WORK

Since the EXI standard is well suited for optimizing machine-readable, interoperable data moving over constrained networks such as those in afloat Navy units, the technology warrants future research. The below recommendations summarize issues identified in this research that could help to speed fleet adoption of EXI and once adopted, ensure its potential is maximized.

1. Conventions for JSON/XML Interoperability

Given that JSON and XML are commonly used for identical functions in web services and applications, and that the tradeoff space between the two is multivariate, selecting one over the other should be a matter of picking the best tool for the application under consideration. Further work to clarify and compare their respective capabilities and limitations in terms of security, expressive power, performance and other dimensions relevant to Navy information systems would provide insight and best practices for future systems development. Such work might explore consistent conventions for expressing XML as JSON and vice versa in the context of Navy web applications. With such conventions in place, maintaining parallel development tracks to implement both technologies may become unnecessary.

2. Holistic Profiling

In comparing EXI to BSON and CBOR, this work only measured file sizes, but file size is only one of many factors to consider in selecting a data serialization format. Extending this research to include profiling capabilities across other dimensions would allow for a more holistic comparison of EXI to other serialization formats, as well as between various EXI configurations. Additional variables identified in this research include: serialization, deserialization, and round-trip memory-to-memory transmission times; memory consumption; power consumption; fidelity; and transmission size with layer 4 and below overhead included.

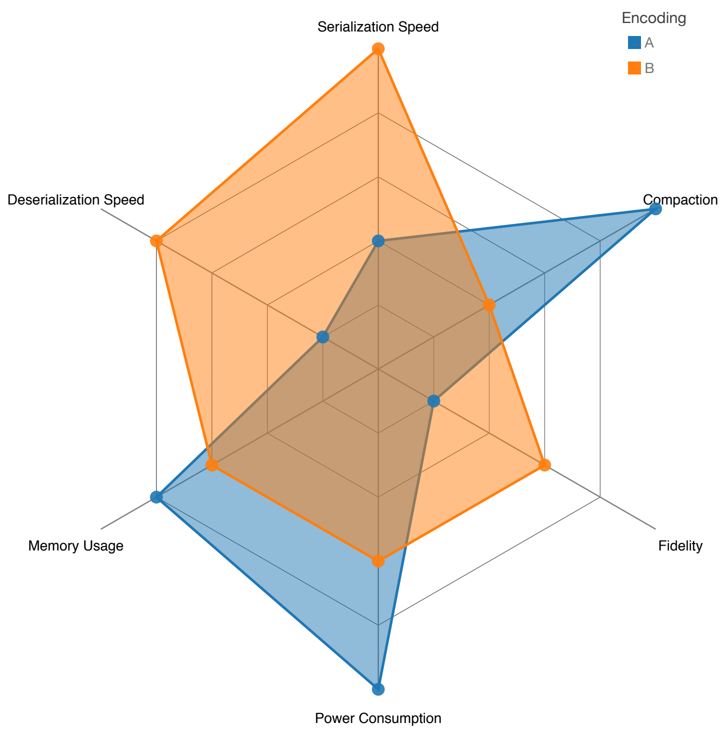

Figure 60. Radar chart comparing hypothetical, multivariate profiles for two data exchange encodings. Presents potential opportunities for further work. Visualization adopted from Bremer (2013) and Brutzman (personal communication, December 2, 2014).

3. Need for Best Practices

As the EXI algorithm is highly configurable, and each configuration parameter affects various attributes of the encoding, a semi-formalized best practices methodology is needed to effectively incorporate it into an existing or developing system. The process could begin with requirements collection defining which parameters the system should be optimized for, perhaps leveraging an instrument similar to Figure 60. Next, it would address the optimizing from two angles: adjusting the system's XML format and XSD to best leverage EXI, and then adjusting EXI's configuration parameters to find a sensible default that fits the system requirements and constraints.

This research approached part of this problem through a brute-force approach, trying all possible configuration permutations and comparing the results. More sophisticated approaches are possible, such as automating tools to examine the data types present in an XML document based on information from its XSD as well as regular expressions, and locate areas that could benefit from additional schema tuning, custom data type representation maps, or other configurations. Also, such an approach applied across many Navy or DOD systems could result in a library of EXI patterns available for reuse across the organization.

4. Expanding EXI across the Open Web Platform

At present, the EXI specification targets only XML, but the Open Web Platform includes many other formats for moving data across web applications. Prototype implementations by Kamiya (2014) and Peintner (2015) show that the grammar-based EXI algorithm can be applied directly to JSON, rather than converting between XML and JSON, with excellent results. Additionally, HTML5, CSS3, and JavaScript code can potentially be compressed with EXI-like, grammar-based algorithms (Burtscher, Livshits, Zorn, & Sinha, 2010). Finally, a JavaScript implementation of EXI could simplify deployment for web applications, as the client would not need to have an EXI codec pre-installed.

5. EXI Streaming Protocols

Given results from this and other research showing that combining multiple small transmissions prior to compression yields compaction benefits, EXI applied to streaming XML protocols could capitalize on these results (Fisteus et al., 2014). Work at the XML Messaging and Presence Protocol (XMPP) Standards Foundation is developing an EXI-based streaming protocol for XMPP (Waher & Doi, 2014). The implications of this for the fleet, both in terms of chat tools, sensor networks and other store and forward situations, warrants further research.

6. Fleet Adoption

The EXI standard is a powerful tool for compressing and optimizing XML data which could significantly benefit constrained Navy networks afloat, but is not widely adopted in fleet systems. A mix of technological and administrative efforts is required to leverage EXI. Also, a high-profile proof of concept implementation directly impacting daily Navy business could support technology adoption. One such possibility is the Microsoft Office suite's files, which are a ubiquitous format for sharing ideas and documents in the fleet and DOD. In practice, they are commonly transmitted over networks as downloads from web portals or as attachments to emails. Past research by SPAWAR has addressed compression of these files using NXPowerLite, a commercial utility that targets unnecessary data in these files, such as excessive resolution and metadata in embedded images (Jordan, 2008; Neuxpower Solutions Ltd., 2015). Though image data inside Microsoft Office files can comprise a large portion of the file size, EXI can target the remaining structured data in XML format. EXI, in conjunction with image manipulation libraries, could provide a completely open-source tool, transparent to users, that provides similar functionality to NXPowerLite and demonstrates the potential impact of EXI in the fleet. Appendix B includes preliminary compaction results for such an approach.

Also, unlike Zip and Gzip, the major operating systems do not by default include an EXI codec, so integrating EXI into Navy systems requires additional work, be it through gateway-based architecture compressing all XML data prior to transmission over

SATCOM links, or on individual clients and servers. Future work toward developing a widely integrated, open source codec pre-integrated at the operating system level could decrease the entry cost for Navy system developers considering EXI. Approaches include ongoing work by NPS faculty, students and industry partners to include the Apache-licensed OpenEXI codec in the Apache Software Foundation codebase, or to work toward inclusion into common development utilities, such as Apache Ant task or an Apache Maven repository (OpenEXI Project, 2014).

D. CLOSING THOUGHTS

EXI is a powerful enabling technology for interoperable, efficient communications over constrained or degraded networks. In the short-term, EXI can be incorporated into the Navy's tactical infrastructure via gateways or filters deployed near edge routers, prior to transmission over SATCOM links. A longer-term solution that eliminates a gateway bottleneck and maximizes performance everywhere is to deploy EXI natively into applications, both on client and server nodes. Further developments will soon change diverse IoT sensors into coherent WoT devices, with accompanying improvements in power consumption and platform endurance. EXI provides major benefits that deserve broad adoption to benefit the fleet.

APPENDIX A. EXI AND MICROSOFT OFFICE

In 2003, Microsoft introduced an XML-based file formats into its Microsoft Office suite that enabled interoperability with a wide array of external applications (Lenz, McRae, & St.Laurent, 2004). Between 2006 and 2012, the file formats became an open standard called Office Open XML[1] (OOXML) (International Organization for Standardization, 2011; Ngo, n.d.). The OOXML standard defines "strict" and "transitional" forms, and Microsoft Officer versions released during that timeframe progressed toward full compliance with the standard. As of the Office 365 release, the software gives users the option to save files in "strict" OOXML format (Thatcher & Knowlton, 2012).

The .xlsx, .pptx, and .docx files generated by Excel, PowerPoint and Word, respectively are Zip archives containing multiple subdirectories and multiple XML files, alongside a media directory containing any binary data in the document, such as images and video clips. To test the compaction performance of EXI on the XML portion of these file formats, the author collected several documents of varying size and complexity in each format, then created a script that expands the Zip archive, applies EXI to each XML document inside, and finally compresses the archive using Zip. A corresponding script reversed the sequence of operations to verify round-trip compatibility. To provide a basis for comparison, the author collected file sizes for native OOXML files further compressed with Zip. To focus on the impact of EXI compaction, the author removed all images and binary data from the files. Figure 61 presents preliminary compaction results.

[1] Note that OOXML here should not be confused with the similar Open Document Format (ODF) specification used in many office productivity applications.

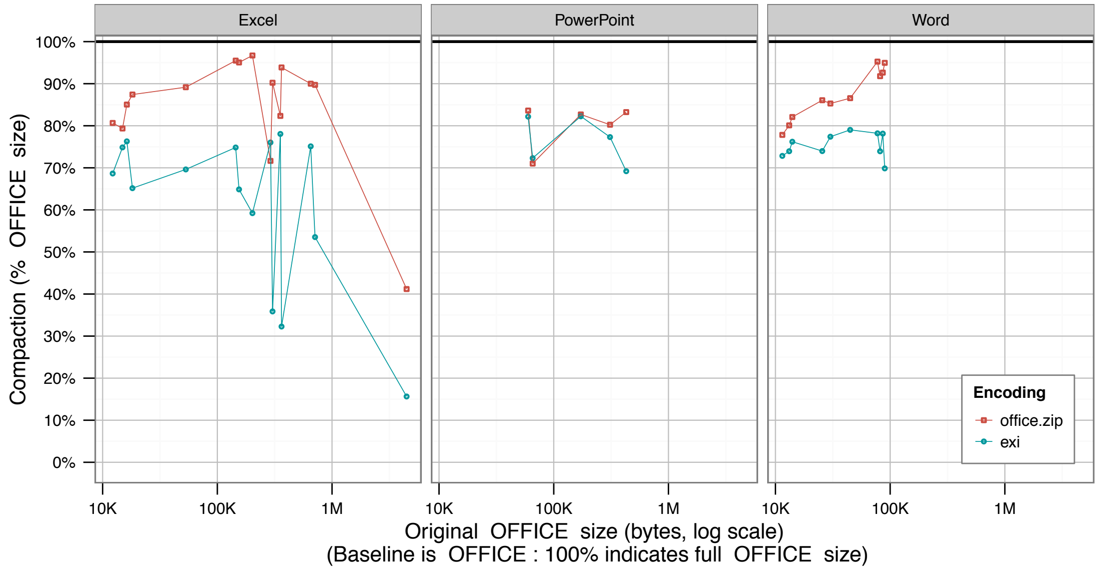

Figure 61. Comparison of EXI and Zip compaction of Microsoft Office documents.

Overall, the proposed method of applying EXI to Microsoft Office documents reduced them to 80% of original size or better—for files containing only XML data. For files with embedded images, compaction was significantly less, and in many cases negligible. The proposed method performed best on Excel documents, compacting them to as little as ~33% of original OOXML size. One result for Excel compaction was ~15% of original OOXML size, though this is likely an outlier. The document contained borders around thousands of empty worksheet cells, resulting in long stretches of highly repetitive XML tags and thus high compaction.

Several avenues for improving on this methodology are to: (1) include image processing functionality to reduce the resolution of media files to that of the anticipated briefing screen; (2) to adjust or extend the OOXML XSDs; (3) to aggregate the XML files inside an OOXML archive prior to EXI encoding to achieve better compaction during the DEFLATE phase of EXI compression. This final approach is promising because, for example in a PowerPoint presentation, each slide is as a separate and relatively small XML file.

APPENDIX B. SOURCE CODE

The source code for all format conversion, data processing and plotting conducted in this research may be retrieved from GitHub at https://github.com/hillbw/exi-test and https://github.com/hillbw/exify-office. Each repository's README.md file provides documentation on the general organization and usage of the test suite, and comments in each source file describe its function. All sample XML and JSON files for the GPX and OpenWeatherMap use cases are included in the repository. A few sample files for the Microsoft Office tests are included, but most are not as they contained sensitive information.

Files from the AIS use-case are not licensed for open distribution, and are available from https://savagedefense.nps.navy.mil/exi/BruceHillThesis. Sample files from the OpenStreetMap, DFXML and PDML use cases, which are too large for GitHub hosting, are also available at that address

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. BEING EFFICIENT WITH BANDWIDTH

As part of this research, the author of this thesis, along with colleagues at the Naval Postgraduate School, published an article entitled "Being Efficient with Bandwidth" in the July 2014 issue of *Proceedings*, published by the United States Naval Institute (Debich, Hill, Miller, & Brutzman, 2014). The article, which explores foundational concepts from this research, is available online at http://www.usni.org/magazines/proceedings/2014-07/professional-notes, and the enclosed version in this appendix is reprinted from *Proceedings* with permission; Copyright © 2014 U.S. Naval Institute (http://www.usni.org).

Being Efficient with Bandwidth

By Lieutenant Commander Steve Debich, Lieutenant Bruce Hill, Captain Scot Miller (Retired), U.S. Navy, and Dr. Don Brutzman

Naval information dominance hinges on three fundamental capabilities: assured command and control (C2), battlespace awareness, and integrated fires. None of these are possible without effective communications links. Networks—and more specifically, the information flowing through them—are now a center of gravity for the Fleet. [1] Maritime tactics and operational plans rely on levels of synchronization only possible through high-bandwidth communications. Satellite communication (SATCOM) is the Fleet's primary path for high-bandwidth C2. However, afloat units may be denied access due to equipment failure, technical problems, weather phenomenon, or enemy actions, forcing reliance on lower-bandwidth alternatives.

For afloat units, bandwidth has become a critical but painfully finite resource that must be conserved. SATCOM carries data from a large number of disparate systems often referred to as "stovepipes." These systems vary in function from tactical to administrative, and the data formats for each application vary greatly. The result is communications only occurring vertically within a system, but not across the breadth of different systems. When many such stovepipes contend for access to the same ship-to-shore transport path, even the largest SATCOM channels can become congested. Future assured C2 requires interoperability between stovepipes and better prioritization of network traffic.

Before identifying the solution, we must understand the factors that impose constraints on the transmission path: bandwidth, latency, and throughput.

Bandwidth: Not The Same As "Throughput"

Bandwidth is literally the "width" of the frequency band used to carry a data signal. It is more often described as the transmission capacity of the communications medium, measured in terms of bits per second. [2] To increase the capacity of an electromagnetic communications channel, modulation technologies and methods would need improvement, or an additional antenna could be installed. Both approaches illustrate significant engineering and financial constraints associated with increasing bandwidth, particularly in the shipboard environment.

SATCOM connections are often depicted as lightning bolts connecting deployed units with relay systems. These lightning bolts convey the impression that data are instantaneously transmitted from unit A to unit B through an optimally placed satellite node. Unfortunately SATCOM transmissions are far from instantaneous: They incur significant delays in comparison to terrestrial communications paths. The combined delay is known as latency.

Latency is an accumulated series of delays that can occur in each step of the communications path between the sender and receiver. Such delays occur as part of propagation delay during signal transmission, network processing and interface delays, varying methods for buffering and queuing, and cumulative router and switch delays. [3] Latency from the perspective of network traffic is the delay from the time of the start of packet transmission at the sender host to the time of the end of packet reception at the receiver host.

Unfortunately latency has significant effects on throughput. This is due in part to the degradation experienced by the primary networking protocol TCP when operating over a high latency network. [4] SATCOM channels routinely operate with latency between 500 to 800 milliseconds. Response "waiting time" is a particular problem for communications protocols like TCP that includes frequent acknowledgement among participants. Increased latency ultimately results in decreased throughput.

Throughput is the rate at which new data—actual information—is transferred through a system. Like bandwidth, it is measured in bits per second and can be considered the actual effective capacity of a channel or the "rate of successful message delivery" being achieved. A common misconception is that bandwidth and throughput are synonymous. Numerous additional constraints can limit the amount of data that can be transferred between two points, such as the overhead of communication protocols and latency delays, which may keep a channel idle. Thus bandwidth indicates the maximum possible data-transfer capacity, while throughput is what capacity actually occurs. Throughput is often significantly lower than the communications channel's bandwidth capacity. Ultimately round-trip-time dominates performance more than bandwidth does. [5]

Common Practice: SATCOM

For Navy ships at sea, the only access to high bandwidth is through SATCOM systems. In our increasingly connected world, the value placed on access to high bandwidth continues to rise. As bandwidth increases, the amount of data that can be transferred between two points also increases. As bandwidth is increased, additional capacity is quickly consumed by ever-more sophisticated sensors, unmanned vehicles, and other network-centric dependencies. [6] Most high-bandwidth paths utilize the super- and extremely-high frequency (SHF/EHF) spectrum for SATCOM communications. Though data and voice circuits exist in other portions of the spectrum, SHF and EHF carry the brunt of Navy traffic, with SHF (C/Ku/X band) ultimately providing the biggest "pipe" for data flows.

In the past, the solution to demand for increasing data transfer was to increase bandwidth, and thereby capacity. As the DoD throttles back spending, many areas must become more efficient in order to accomplish defense missions. Similar approaches for efficiency must be applied with respect to communication systems. The amount of information to be shared is not expected to decrease. Because constraints on SATCOM bandwidth make even marginal increases a costly venture, the Navy must explore new tactics. Perhaps solutions lie not in the channel itself, but in the format of data transmitted. What if we can convey the same information using just a fraction of the original zeros and ones, while at the same time connecting stovepipes through data interoperability?

XML: The Language of Interoperability

Interoperability is essential to the key information dominance capabilities. Shipboard computers must talk to each other, computers from other service branches, and computers from partner nations. To facilitate interoperability, an open-standards approach is critical. The Department of the Navy's chief information officer has designated the extensible markup language (XML) as the data-definition language of choice for information standardization, and for good reason: It is the *de facto* standard format for systems talking across the web. [7] By design, XML adds structure to data, which in turn facilitates validation of correctness and system interoperability. XML is the *lingua franca* of the world's computers.

Though XML is a path to both technical and semantic interoperability, it has an Achilles heel: It was never intended to be compact. [8] In terrestrial networks with low latency contributing to massive throughput, this is usually unimportant. For the Navy, however, large messages mean slower connections and less information to forward-deployed units relying on SATCOM. Transmitting large messages also draws more power, so XML isn't ideal for mobile or unmanned devices running on batteries. Viewed in this light, XML is less attractive, but it doesn't have to be that way. Recent advances in data compression are providing new design options.

Shrinking Data, Broadening the Web

In 2004, the World Wide Web Consortium began to address this issue, and in 2014 released the Efficient XML Interchange (EXI) Format Recommendation. [9] EXI is an alternate encoding of XML data that leverages the inherent structure of XML to tightly compress it. Since it is designed specifically for XML, the results are superior to generic compression methods. In some cases, EXI compression results in files that are less than 10 percent the size of the original XML file. [10] Perhaps even more surprising is that EXI decompresses faster, using fewer computations and therefore drawing less power than plain text-based ZIP and GZIP compression.

Given that XML enables interoperability, and that EXI shrinks it, Fleet communications architects and program managers should be interested. [11] Systems could potentially convert and transmit information in XML format, and with EXI they could send more information in less time. By incorporating EXI, web-based architectures such as CANES and C4I systems using service-oriented architectures may be viable over constrained SATCOM links. Unmanned systems and remote sensors might use EXI to conserve batteries on extended missions. A single file cut to a tenth of its original size is useful in itself, but the aggregate impact over thousands of nodes in a cloud, each sending thousands of files, could be immense.

Other impacts pertain as well. For example, encryption is usually considered independent of compression. However, by randomizing a bit stream, encryption scrambles the structure necessary for effective compression. That means encrypted streams cannot be compressed. Compression must occur before encryption when transmitting, and decompression after decryption on the receiving end. This principle is so important that the order should be checked for all Navy communications channels.

Since message size is just one of many factors in network throughput, EXI is not a silver-bullet for Navy bandwidth woes, but it certainly can't hurt. It is not mutually exclusive of other attempts to address the issue. Navy communications designers need not choose between a new SATCOM constellation and EXI, or between commercial network accelerators and EXI; they can have both. Considering that EXI is open standard, supports interoperability, and shrinks data the Navy is already sending over its networks, there is little to lose and much to gain. The Navy can be more efficient with a precious afloat resource: bandwidth.

1. Chief of Naval Operations Information Dominance, Navy Strategy for Achieving Information Dominance, 2013-2017, 1 January 2013, www.dtic.mil/docs/citations/ADA571217 [6] .

2. Anu A. Gokhale, *Introduction to Telecommunications* (Cengage Learning, 2004), http://books.google.com/books?id=QowmxWAOEtYC&pgis=1 [7] , 455.

3. Rony Kay, "Pragmatic Network Latency Engineering Fundamental Facts and Analysis" (cPacket Networks Inc, 2009), http://cpacket.com/wp-content/files_mf/introductiontonetworklatencyengin... [8] .

4. Thomas R. Henderson, Randy H. Katz, *TCP Performance over Satellite Channels* (Berkeley, CA: University of California at Berkeley, 1999), www.eecs.berkeley.edu/Pubs/TechRpts/1999/CSD-99-1083.pdf [9] .

5. Mike Belshe, "More Bandwidth Doesn't Matter (Much)," 2010, www.chromium.org/spdy [10] .

6. Isaac R. Porche, Bradley Wilson, Erin-Elizabeth Johnson, Shane Tierney, Evan Saltzman, RAND Corporation, "Data Flood: Helping the Navy Address the Rising Tide of Sensor Information," 2014, www.rand.org/pubs/research_reports/RR315.html [11] .

7. Department of the Navy Chief Information Officer, "DON policy on the use of extensible markup language (XML)," 2012, http://xml.coverpages.org/DON-XMLPolicy200212.pdf [12] .

8. Mike Cokus, Santiago Pericas-Geertsen, "XML binary characterization properties," 2005, www.w3.org/TR/xbc-properties/#xml-design-goals [13] .

9. Takuki Kamiya, Efficient XML Interchange Working Group, 2014, www.w3.org/XML/EXI [14] .

10. Sheldon Snyder, Don McGregor, Don Brutzman, "Efficient XML interchange: Compact, efficient, and standards-based XML for modeling and simulation," 2009, http://calhoun.nps.edu/public/handle/10945/5422 [15] .

11. Jeffrey Williams, "Document-based message-centric security using XML authentication and encryption for coalition and interagency operations," master's thesis, Naval Postgraduate School, 2009, http://calhoun.nps.edu/public/handle/10945/4610 [16] .

Lieutenant Commander Debich and Lieutenant Hill are information professional officers studying network operations at the Naval Postgraduate School (NPS).

Captain Miller is a retired information professional officer serving as a research associate at NPS. He is the former commanding officer of the Navy Center for Tactical Systems Interoperability.

Dr. Brutzman is a retired submarine officer working in the information sciences department, Undersea Warfare Academic Group, and MOVES Institute at NPS. **Additional insights by Dr. Dan Boger, Captain Louis Unrein (U.S. Navy) and other reviewers are gratefully acknowledged.**

LIST OF REFERENCES

Anastasi, G., Conti, M., & Di Francesco, M. (2008). Data collection in sensor networks with data mules: An integrated simulation analysis. In *2008 IEEE Symposium on Computers and Communications* (pp. 1096–1102). IEEE. doi:10.1109/ISCC.2008.4625629

The Apache Software Foundation. (2014). Apache module mod_deflate. Retrieved from http://httpd.apache.org/docs/2.4/mod/mod_deflate.html

Asanov, S., Oleg, G., Palashina, A., Krivonosova, A., & Namjittrong, T. (2014). Unicode character table. Retrieved from http://unicode-table.com/en/

Baldoni, R., Querzoni, L., & Virgillito, A. (2005). *Distributed event routing in publish/subscribe communication systems : A survey*. DIS, Universita di Roma La Sapienza, Tech. Rep. Retrieved from http://www.diag.uniroma1.it/~midlab/articoli/BV.pdf

Bartel, M., Boyer, J., Fox, B., LaMacchia, B., Simon, E., Eastlake, D., … Yiu, K. (2013). XML signature syntax and processing. Retrieved March 9, 2015, from http://www.w3.org/TR/2013/REC-xmldsig-core1-20130411/

Bennett, J. (2010). *OpenStreetMap: Be your own cartographer*. Birmingham, UK: Packt Publishing.

Bentrup, J., Otte, E., Chan, B., Vavrichek, D., & Gingras, D. (2012). *At-sea testing of a wide area network optimization device (U)*. Center for Naval Analyses Corporation. DRM-2012-U-002246-Final.

Bormann, C., & Hoffman, P. (2013). Concise binary object representation. Internet Engineering Task Force.

Bos, B. (2001). XML in 10 points. Retrieved from http://www.w3.org/XML/1999/XML-in-10-points

Bournez, C. (2009). Efficient XML interchange evaluation (Working Draft). Retrieved from http://www.w3.org/TR/2009/WD-exi-evaluation-20090407/#compactness-results

Bray, T., Paoli, J., & Sperberg-McQueen, C. M. (1998). Extensible markup language (XML) 1.0. Retrieved from http://www.w3.org/TR/1998/REC-xml-19980210

Bremer, N. (2013). Radar chart or spider chart. Retrieved February 2, 2015, from http://bl.ocks.org/nbremer/6506614

Brutzman, D., Hughes, W., Kline, J., Buettner, R., & Ekelund, J. J. (2014). *Network-optional warfare (NOW) operational concepts*. Monterey, CA: Naval Postgraduate School. Retrieved Mar 10, 2015, from https://wiki.nps.edu/display/NOW/Network+Optional+Warfare

Brutzman, D., & X3D Working Group. (2014). X3D JSON encoding. Retrieved March 9, 2015, from http://www.web3d.org/wiki/index.php/X3D_JSON_Encoding

BSON. (2014). Retrieved from http://bsonspec.org/

BSONSpec.org. (2014). Specification version 1.0, Retrieved from http://bsonspec.org/spec.html.

Burtscher, M., Livshits, B., Zorn, B. G., & Sinha, G. (2010). JSZap: Compressing JavaScript code. In *2010 USENIX Conference on Web Application Development* (pp. 39–50). Boston, MA. Retrieved from https://www.usenix.org/legacy/event/webapps10/tech/full_papers/webapps10_proceedings.pdf#page=29

Cebrowski, A. K., & Garstka, J. J. (1998). Network-centric warfare: Its origin and future. *United States Naval Institute Proceedings*, *124*(1), 28–35. Retrieved from http://search.proquest.com/docview/205987210

Cerami, E. (2002). *Web services essentials*. Sebastopol, CA: O'Reilly & Associates.

Chaplain, C. (2009). *Space acquisitions: Government and industry partners face substantial challenges in developing new DoD space systems* (GAO-09-648T). Washington, DC: Government Accountability Office. Retrieved from http://www.dtic.mil/dtic/tr/fulltext/u2/a499151.pdf

Chief of Naval Operations for Information Dominance. (2013). *Navy strategy for achieving information dominance: 2013-2017*. Washington, DC: Author. Retrieved from http://www.dtic.mil/docs/citations/ADA571217

Cokus, M., & Pericas-Geertsen, S. (2005a). XML binary characterization properties. Retrieved from http://www.w3.org/TR/xbc-properties/#xml-design-goals

Cokus, M., & Pericas-Geertsen, S. (2005b). XML binary characterization use cases. Retrieved from http://www.w3.org/TR/2005/NOTE-xbc-use-cases-20050331

Cokus, M., & Vogelheim, D. (2007). Efficient XML Interchange (EXI) Best Practices. Retrieved from http://www.w3.org/TR/exi-best-practices/

Crockford, D. (n.d.). The JSON saga [Video file]. Retrieved from https://www.youtube.com/watch?v=x92vbAN_j1k

Crockford, D. (2006). JSON: The fat-free alternative to XML. Retrieved from http://www.json.org/fatfree.html

Crockford, D. (2008). *JavaScript: The good parts*. Sebastopol, CA: O'Reilly Media.

Crockford, D., & Bray, T. (2014). RFC 7159: The JavaScript Object Notation (JSON) Data Interchange Format. Internet Engineering Task Force. Retrieved from http://www.rfc-editor.org/rfc/rfc7159.txt

Debich, S. (2015). *The role of efficient XML interchange in navy wide area network optimization.* Master's thesis, Naval Postgraduate School, Monterey, CA.

Debich, S., Hill, B., Miller, S., & Brutzman, D. (2014). Being efficient with bandwidth. *United States Naval Institute Proceedings, 140*(7), 76-77.

Department of the Navy Chief Information Officer. (2013). Update to department of the navy approach to cloud computing. Washington, DC: Author. Retrieved from http://www.doncio.navy.mil/ContentView.aspx?id=4695

Deputy Chief of Naval Operations for Information Dominance. (2014). Task force cloud charter. Department of the Navy.

Deutsch, P. (1996). DEFLATE Compressed Data Format Specification version 1.3. Internet Engineering Task Force. Retrieved from http://tools.ietf.org/pdf/rfc1951.pdf

Dvoinikov, D. (2014). Pythomnic3k (Version 1.4.1) [Computer software]. Retrieved from http://www.pythomnic3k.org/

Dzhagaryan, A., Milenkovic, A., & Burtscher, M. (2013). Energy efficiency of lossless data compression on a mobile device: An experimental evaluation. In *2013 IEEE International Symposium on Performance Analysis of Systems and Software* (pp. 126–127). Austin, Texas: IEEE. doi:10.1109/ISPASS.2013.6557156

Eck, D. J. (2011). *Introduction to programming using java* (6th ed., Vol. 2011). Retrieved from http://math.hws.edu/javanotes/

Ecma International. (2013). Ecma-404: The json data interchange format. Geneva, Switzerland: Ecma International. Retrieved from http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

Evjen, B., Sharkey, K., Thangarathinam, T., Kay, M., Vernet, A., & Ferguson, S. (2007). *Professional XML*. Indianapolis, IN: Wiley.

Fall, K. (2003). A delay-tolerant network architecture for challenged internets. In *Proceedings of the 2003 conference on Applications, technologies, architectures,* *and protocols for computer communications - SIGCOMM '03* (pp. 27–34). New York: ACM Press. doi:10.1145/863956.863960

Fawcett, J., Ayers, D., & Quin, L. (2012). *Beginning XML* (5th ed.). Somerset, NJ: John Wiley & Sons.

Fielding, R., & Reschke, J. (2014). Hypertext transfer protocol (http/1.1): Semantics and content. Internet Engineering Task Force. Retrieved from http://tools.ietf.org/html/rfc7231

Fisteus, J. A., García, N. F., Fernández, L. S., & Fuentes-Lorenzo, D. (2014). Ztreamy: A middleware for publishing semantic streams on the web. *Journal of Web Semantics*, *25*, 16–23. doi:10.1016/j.websem.2013.11.002

Foster, D. (n.d.). GPX: The GPS exchange format. Retrieved from http://www.topografix.com/gpx.asp

Galiegue, F., Zyp, K., & Court, G. (2013). *JSON schema: Core definitions and terminology* (No. draft-zyp-json-schema-04) (pp. 1–14). Retrieved from http://tools.ietf.org/pdf/draft-zyp-json-schema-04.pdf

Garfinkel, S. L. (2009). Automating disk forensic processing with Sleuthkit, XML and Python. *Fourth International IEEE Workshop on Systematic Approaches to Digital Forensic Engineering*, 73–84. doi:10.1109/SADFE.2009.12

Garfinkel, S. L. (2011). Digital forensics tool integration [PowerPoint slides]. Retrieved from http://simson.net/ref/2011/2011-12-07 DFXML.pdf

Garfinkel, S. L. (2012). Digital forensics innovation: Searching a terabyte of data in 10 minutes [Video file]. Harvard Center for Research on Computation and Society. Retrieved from https://www.youtube.com/watch?v=pI_e-4eZ2Yg

Garret, J. J. (2005). Ajax: A new approach to web applications. Retrieved from http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications/

Garrett, C. (2012). ExiProcessor (Version 2012-03-22) [Computer software]. Retrieved from https://sourceforge.net/projects/exiprocessor/

Geofabrik GmbH. (2014). OpenStreetMap Data Extracts. Retrieved September 24, 2014, from http://download.geofabrik.de/

Gil, B., & Trezentos, P. (2011). Impacts of data interchange formats on energy consumption and performance in smartphones. *Proceedings of the 2011 Workshop on Open Source and Design of Communication - OSDOC '11*, 1. doi:10.1145/2016716.2016718

Gilchrist, J. (2003). Parallel data compression with bzip2. Retrieved from http://gilchrist.ca/jeff/comp5704/Final_Paper.pdf

Goff, J. (2014). Wanted: An agile, low-cost, irregular-warfare surface combatant. *United States Naval Institute Proceedings*, *140*(10), 1340. Retrieved from http://www.usni.org/magazines/proceedings/2014-10/professional-notes

Goldman, O., & Lenkov, D. (2005). XML Binary Characterization. Retrieved from http://www.w3.org/TR/xbc-characterization/

Gonzalez, R., Woods, R., & Eddins, S. (2009). *Digital image processing using matlab* (2nd ed..). Gatesmark Publishing.

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., Nielsen, H. F., Karmarkar, A., & Lafon, Y. (2007). Soap version 1.2 part 1: Messaging framework (second edition). Retrieved from http://www.w3.org/TR/soap12-part1/

Hallam-Baker, P., & Mysore, S. H. (2005). XML key management specification (XKMS 2.0). Retrieved March 9, 2015, from http://www.w3.org/TR/2005/REC-xkms2-20050628/

Hughes, W. P. (2014). A prophet for our times. *Naval War College Review*, *67*(3), 96–97. Retrieved from https://wiki.nps.edu/download/attachments/357662779/A-Prophet-for-Our-Times.pdf?version=1&modificationDate=1399566756000&api=v2

Imamura, T., Dillaway, B., Simon, E., Yiu, K., Nystrom, M., Eastlake, D., … Roessler, T. (2013). XML encryption syntax and processing. Retrieved March 9, 2015, from http://www.w3.org/TR/2013/REC-xmlenc-core1-20130411/

International Organization for Standardization. (2007a). ISO/IEC 11404:2007: Information technology - general purpose datatypes (gpd). Geneva, Switzerland: ISO/IEC. Retrieved from http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html

International Organization for Standardization. (2007b). ISO/IEC 24824-1:2007: Information technology - generic applications of asn.1: Fast infoset. Geneva, Switzerland: ISO/IEC. Retrieved from http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html

International Organization for Standardization. (2011). ISO/IEC 29500-1INTERNATIONAL STANDARD ISO / IEC Information technology — Document description and processing languages —, *2011*.

Internet Assigned Numbers Authority. (2014). Hypertext transfer protocol (http) parameters. Retrieved from http://www.iana.org/assignments/http-parameters/http-parameters.xhtml

Jepsen, T. (2001). Soap cleans up interoperability problems on the web. *IT Professional*, (February), 52–55. doi:10.1109/6294.939937

Jordan, M. (2008). *NxPowerLite Trident Warrior 2007 experimentation and results*. San Diego, CA: SPAWAR.

Kamiya, T. (2014). Applying exi encoding technique to json [Lecture and PowerPoint presentation].

Kangasharju, J. (2008). *XML messaging for mobile devices*. Helsinki, Finland: Helsinki University Printing House. Retrieved from https://helda.helsinki.fi/bitstream/handle/10138/21347/xmlmessa.pdf?sequence=1

Kattan, A. (2010). Universal intelligent data compression systems: A review. *2010 2nd Computer Science and Electronic Engineering Conference (CEEC)*. doi:10.1109/CEEC.2010.5606482

Kay, M. H. (2009). Saxon XSLT and XQuery processor (Version 9.1.0.6) [Computer software]. Retrieved from http://saxon.sourceforge.net/

Kurose, J. F., & Ross, K. W. (2013). *Computer networking: A top-down approach* (6th ed.). Boston, MA: Pearson Education.

Kyusakov, R. (2014). *Efficient web services for end-to-end interoperability of embedded systems* (Ph.D. dissertation). Retrieved from http://pure.ltu.se/portal/files/100108844/Rumen_Kyusakov.pdf

Kyusakov, R., Makitaavola, H., Delsing, J., & Eliasson, J. (2011). Efficient XML interchange in factory automation systems. *IECON 2011 - 37th Annual Conference of the IEEE Industrial Electronics Society*, 4478–4483. doi:10.1109/IECON.2011.6120046

Le Hegaret, P. (2005). Charter of the Efficient XML Interchange Working Group. Retrieved from http://www.w3.org/2005/09/exi-charter-final.html

Lee, D. (2011). JXON: An architecture for schema and annotation driven JSON/XML bidirectional transformations. In *Proceedings of Balisage: The Markup Conference 2011*. Montreal, Canada. doi:10.424/BalisageVol7.Lee01

Lee, D. (2013). Fat markup: Trimming the fat markup myth one calorie at a time. In *Balisage: The Markup Conference 2013* (Vol. 10). doi:10.4242/BalisageVol10.Lee01

Lenz, E., McRae, M., & St.Laurent, S. (2004). *Office 2003 XML*. Sebastopol, CA: O’Reilly Media.

Liefke, H., & Suciu, D. (2000). XMill : An efficient compressor for XML data. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data* (pp. 153–164). Dallas, TX: Association for Computing Machinery. doi:10.1145/342009.335405

Ling, Y., Durbha, S. S., & King, R. L. (2006). Sensor web enablement for coastal buoy systems. American Geophysical Union. Retrieved from http://adsabs.harvard.edu/abs/2006AGUFMIN23A1215L

Maeda, K. (2012). Performance evaluation of object serialization libraries in XML, JSON and binary formats. In *2012 Second International Conference on Digital Information and Communication Technology and its Applications (DICTAP)* (177–182). Ieee. doi:10.1109/DICTAP.2012.6215346

Martin, B., & Jano, B. (1999). WAP binary XML content format. Retrieved from http://www.w3.org/1999/06/NOTE-wbxml-19990624

Microsoft. (2014). Http compression <httpCompression>. Retrieved from http://www.iis.net/configreference/system.webserver/httpcompression

MongoDB Documentation Project. (2014). Mongodb documentation. Retrieved from http://docs.mongodb.org/master/MongoDB-manual.pdf

Morse, K. G. (2005, July). Compression tools compared. *Linux Journal*, *2005*(137). Retrieved from http://www.linuxjournal.com/node/8051/print

Nelson, A., & Digital Forensics XML Working Group. (2014). Digital Forensics XML Schema Document (Version 1.1.1) [Computer software]. Retrieved from https://github.com/dfxml-working-group/dfxml_schema

NetBee.org. (n.d.). PDML XML schema document [Computer software]. Retrieved from http://nbee.org/doku.php?id=netpdl:schema

Neuxpower Solutions Ltd. (2015). File optimization technology. Retrieved from http://www.neuxpower.com/technology/

Ngo, T. (n.d.). Office open XML overview. Ecma International. Retrieved from http://www.ecma-international.org/news/TC45_current_work/OpenXML White Paper.pdf

Nogatz, F., & Frühwirth, T. (2013). *From XML schema to JSON schema: Comparison and translation with constraint handling rules* (Bachelor's thesis). Retrieved from http://www.informatik.uni-ulm.de/pm/fileadmin/pm/home/fruehwirth/drafts/Bsc-Nogatz.pdf

Nurseitov, N., Paulson, M., Reynolds, R., & Izurieta, C. (2009). *Comparison of JSON and XML data interchange formats : A case study*. Bozeman, MN: Montana State

University. Retrieved from http://www.cs.montana.edu/izurieta/pubs/caine2009.pdf

Olson, B. (2014). CBOR (Version 0.1.12) [Computer software]. Retrieved from https://code.google.com/p/cbor/

OpenEXI Project. (2014). OpenEXI. Retrieved from http://openexi.sourceforge.net

OpenStreetMap contributors. (2012). XSD for SSIS. Retrieved from http://wiki.openstreetmap.org/w/index.php?title=API_v0.6/XSD_for_SSIS&oldid=807462

OpenWeatherMap, Inc. (2015a). Current weather data. Retrieved February 9, 2015, from http://openweathermap.org/current

OpenWeatherMap, Inc. (2015b). OpenWeatherMap big data + weather technology.

Oxford University Press. (2013). About. Retrieved February 10, 2015, from http://public.oed.com/about/

Paulson, L. D. (2005). Building rich web Applications with ajax. *Computer*, *38*(10), 14–17. doi:10.1109/MC.2005.330

Pavlov, I. (2014). 7-Zip. Retrieved from http://www.7-zip.org

Peintner, D. (2015). EXI 4 JSON [PowerPoint Presentation].

Peintner, D., & Heuer, J. (2014). EXIficient (Version 0.9.3) [Computer software]. Retrieved from http://exificient.sourceforge.net/

Peintner, D., Kosch, H., & Heuer, J. (2009). Efficient XML Interchange for rich internet applications. *2009 IEEE International Conference on Multimedia and Expo*. doi:10.1109/ICME.2009.5202458

Peintner, D., & Pericas-Geertsen, S. (2007). Efficient XML interchange primer. Retrieved from http://www.w3.org/TR/2007/WD-exi-primer-20071219

Porche, I., Wilson, B., Johnson, E.-E., Tierney, S., & Saltzman, E. (2014). *Data flood: Helping the navy address the rising tide of sensor information*. Santa Monica, CA: RAND National Defense Research Institute. Retrieved from http://www.rand.org/pubs/research_reports/RR315.html

Raggett, D. (2010). The web of things : Extending the web into the real world. In J. van Leeuwen, A. Muscholl, D. Peleg, J. Pokorny, & B. Rumpe (Eds.), *SOFSEM 2010: Theory and Practice of Computer Science* (96–107). Berlin: Springer Berlin Heidelberg. doi:10.1007/978-3-642-11266-9_8

Risso, F. (2010). NetPDL Language Specification. Retrieved from http://www.nbee.org/doku.php?id=netpdl:pdml_specification

Rowden, T., Gumataotao, P., & Fanta, P. (2015). Distributed lethality. *United States Naval Institute Proceedings*, *141*(1), 18–23.

Saha, S., Jamtgaard, M., & Villasenor, J. (2001). Bringing the wireless internet to mobile devices. *Computer*, *34*(6), 54–58. doi:10.1109/2.928622

Sakr, S. (2009). XML compression techniques: A survey and comparison. *Journal of Computer and System Sciences*, *75*(5), 303–322. doi:10.1016/j.jcss.2009.01.004

Salomon, D. (2008). *A concise introduction to data compression*. London: Springer-Verlag.

Sayood, K. (2005). *Introduction to data compression* (ProQuest e.). Burlington, MA: Morgan Kaufmann.

Schneider, J., & Kamiya, T. (2011). Efficient XML Interchange (EXI) Format 1.0 (First Edition). Retrieved from http://www.w3.org/TR/2011/REC-exi-20110310/

Schneider, J., Kamiya, T., Peintner, D., & Kyusakov, R. (2014). Efficient XML Interchange (EXI) Format 1.0 (Second Edition). Retrieved from http://www.w3.org/TR/2014/REC-exi-20140211/

Seward, J. (2000). Bzip2 and libbzip2: A program and library for data compression. Retrieved from http://bzip.org/docs.html

Shannon, C. (1948). A mathematical theory of communication. *The Bell Systems Technical Journal*, *27*(3), 379–423.

Sheth, A., Henson, C., & Sahoo, S. S. (2008). Semantic sensor web. *IEEE Internet Computing*, *12*(4), 78–83. doi:10.1109/MIC.2008.87

Sheth, A. P. (1999). Changing focus on interoperability in information systems: From system, syntax, structure to semantics. In M. Goodchild, M. Egenhofer, R. Fegeas, & C. Kottman (Eds.), *Interoperating Geographic Information Systems* (pp. 5–29). Springer US. doi:10.1007/978-1-4615-5189-8

Snyder, S. (2010). *Efficient XML interchange compression and performance benefits: Development, implementation and evalution* (Master's thesis). Retrieved from Calhoun https://calhoun.nps.edu/public/handle/10945/30774

Snyder, S., McGregor, D., & Brutzman, D. (2009). *Efficient XML interchange: Compact, efficient, and standards-based XML for modeling and simulation*. Retrieved from http://hdl.handle.net/10945/30774

Sporny, M., Longley, D., Kellog, G., Lanthaler, M., & Lindstrom, N. (2014). JSON-LD 1.0: A json-based serialization for linked data. Retrieved from http://www.w3.org/TR/2014/REC-json-ld-20140116/

Stein, B. (2014). XSLTJSON (Version 1.0.93) [Computer software]. Retrieved from http://www.bramstein.com/projects/xsltjson/

Sumaray, A., & Makki, S. K. (2012). A comparison of data serialization formats for optimal efficiency on a mobile platform. In *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication - ICUIMC '12* (p. 1). New York: ACM Press. doi:10.1145/2184751.2184810

Sundin, S. (2013). *Evaluation of wireless interfaces for vehicles and applications* (Master's thesis). Retrieved from https://pure.ltu.se/portal/files/43309095/LTU-EX-2013-43243354.pdf

Szecówka, P., & Mandrysz, T. (2009). Towards hardware implementation of bzip2 data compression algorithm. In *16th International Conference on Mixed Design of Integrated Circuits and Systems* (pp. 337–340). IEEE. Retrieved from http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=5289605

Thatcher, J., & Knowlton, G. (2012). New file format options in the new office [Blog post]. Retrieved March 9, 2015, from http://blogs.office.com/2012/08/13/new-file-format-options-in-the-new-office/

Thompson, H. S., Beech, D., Maloney, M., & Mendelsohn, N. (2004). XML schema part 1: Structures second edition. Retrieved from http://www.w3.org/TR/xmlschema-1/

Tiller, M., & Harman, P. (2014). Web and network friendly simulation data formats. In *Proceedings of the 10th International Modelica Conference, 96*, 1081–1093. Lund, Sweden. doi:10.3384/ecp140961081

TopoGrafix. (2004). GPX Schema (Version 1.1) [Computer software]. Retrieved from http://www.topografix.com/gpx/1/1/gpx.xsd

Tsai, C.-L., Chen, H.-W., Huang, J.-L., & Hu, C.-L. (2011). Transmission reduction between mobile phone applications and RESTful APIs. In *Proceedings of the 2011 ACM Symposium on Applied Computing - SAC '11* (pp. 445–450). New York: ACM Press. doi:10.1145/1982185.1982280

United States Coast Guard. (2014a). Class A AIS position report. Retrieved February 9, 2015, from http://www.navcen.uscg.gov/?pageName=AISMessagesA

United States Coast Guard. (2014b). Nationwide automatic identification system. Retrieved December 3, 2014, from http://www.navcen.uscg.gov/?pageName=NAISmain

United States Coast Guard. (2015). Nationwide automatic identification system. Retrieved from http://www.uscg.mil/acquisition/nais/

United States Department of Transportation Volpe Center. (n.d.). Maritime safety and security information system. Retrieved February 6, 2015, from https://mssis.volpe.dot.gov/Main/

Waher, P., & Doi, Y. (2014). XEP-0322: Efficient XML interchange (EXI) format. Retrieved from http://xmpp.org/extensions/xep-0322.pdf

Walsh, N. (2010). Deprecating XML [Blog post]. Retrieved from http://norman.walsh.name/2010/11/17/deprecatingXML

Wang, G. (2011). Improving data transmission in web applications via the translation between XML and JSON. In *2011 Third International Conference on Communications and Mobile Computing,* 182–185. IEEE. doi:10.1109/CMC.2011.25

Wang, P., Wu, X., & Yang, H. (2011). Analysis of the efficiency of data transmission format based on Ajax applications. In *2011 International Conference of Information Technology, Computer Engineering and Management Sciences*, 265–268. doi:10.1109/ICM.2011.199

The WAP Forum. (2000). Wireless application protocol: Wireless Internet today. Retrieved from http://www.wapforum.org/what/whitepapers.htm

White, G., Kangasharju, J., Brutzman, D., & Williams, S. (2007). Efficient XML interchange measurements note. Retrieved from http://www.w3.org/TR/2007/WD-exi-measurements-20070725/#context

Winer, D. (1999). XML-RPC specification. Retrieved from http://xmlrpc.scripting.com/spec#update1

Winer, D. (2003). RSS 2.0 specification. Retrieved from http://cyber.law.harvard.edu/rss/rss.html

Wong, C., & Gonzales, D. (2014). *Authority to issue interoperability policy*. Santa Monica, CA: RAND. Retrieved from http://www.dtic.mil/docs/citations/ADA593558

Ziv, J., & Lempel, A. (1977). A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, *23*(3). Retrieved from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.118.8921

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California