SANDIA REPORT

SAND2013-10141

Unlimited Release

Printed November 2013

Reed-Solomon Error-Correction as a Software Patch Mechanism

Kevin D. Pendley

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: <http://www.osti.gov/bridge>

Available to the public from

U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd.
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: <http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online>

SAND2013-10141
Unlimited Release
Printed November 2013

Reed-Solomon Error-Correction as a Software Patch Mechanism

Kevin D Pendley
Decision Support Systems
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-MS0576

Abstract

This report explores how error-correction data generated by a Reed-Solomon code may be used as a mechanism to apply changes to an existing installed codebase. Using the Reed-Solomon code to generate error-correction data for a changed or updated codebase will allow the error-correction data to be applied to an existing codebase to both validate and introduce changes or updates from some upstream source to the existing installed codebase.

ACKNOWLEDGMENTS

The author would like to thank Ernie Helmer of the Decision Support Systems for several conversations and discussions that refined my explanation of how the Reed-Solomon code can be used as a software patch mechanism.

The author would also like to thank Tom Loughry of the Decision Support Systems for several conversations and discussions about the block-encoding nature of Reed-Solomon codes that led to more research about how to overcome early problems caused by adaptive compression in the source archive.

CONTENTS

1. Introduction.....	7
2. Reed-Solomon Patching Approach.....	9
2.1 Some Complications to Consider.....	9
3. Experimental Results	11
3.1 File Sizes.....	11
4. Conclusions.....	13
5. Future Work	15
6. References.....	17
Appendix A: First Reed-Solomom Code Run	20
Appendix B: Adaptive Compression Problems	22
Appendix C: Patching without Compression.....	24
Appendix D: Patching a Codebase	28

NOMENCLATURE

codebase	For the discussion of this publication, “codebase” will refer to the entire collection of delivered software, including binary files, configuration files, property files, or scripts (Unlike the conventional meaning encompassing only source code, here it refers to any electronic software, documentation, or source code that is ‘installed’ when it is placed in an operational configuration).
redundant data	The data generated by a Reed-Solomon code using a Galois Field that can be used to recover lost or damaged data in the original input. This is also referred to as ‘error-correction data’.
software patch	A file or package representing changes to an installed codebase that is intended to modify an originally installed codebase with the changes.
SNL	Sandia National Laboratories
checksum	A number that is the total of the digits in a piece of data that has been stored or sent in digital form. It is used to check that nothing has gone wrong with the transmitted data (this definition was taken from the Online Macmillan English Dictionary).

1. INTRODUCTION

This publication explores how error-correction data generated by a Reed-Solomon code may be used as a mechanism to apply changes to an existing installed codebase (This consists of both the source code used to generate end-user software and the software itself).

Reed-Solomon codes are generally used to provide error correction for transmitted signals or streaming data (e.g., DVD movies).

This publication looks at using Reed-Solomon-generated data to ‘correct’ an existing software installation, which in effect applies a software patch.

Using the Reed-Solomon code as a patch mechanism provides two main benefits. First, it allows the packaging of the updates and changes in software on the originating side and then once delivered to the destination, allows the application of these new packaged updates and changes to the already-installed software on the destination side. Second, it allows the validation that the code that is at the destination side is exactly what was packaged on the source side. One additional benefit is that it will allow repair and/or regeneration of changed or lost pieces of the software at the destination side (if necessary) without further interaction with the originating side. For example, consider the case when a key configuration file was modified and the archive of the original file was either damaged or modified. The original file and its backup archive file would now be useless. In the process described here the Reed-Solomon-code-generated error-correction data is simply used as a ‘patch’ to recover the archive containing the original file.

The original intent of the work in this paper was to address a fairly inefficient but reliable software patch mechanism. In this mechanism, changed files were identified using file checksums and for each file containing a change, the entire file was archived and transmitted to be installed at the destination. In this situation, even a minor modification in a file resulted in the entire file being sent for installation along with the checksum files to verify the codebase. The purpose in exploring this concept was the belief that using data from a Reed-Solomon code could create smaller files to transmit and install at the destination without sacrificing the reliability. There were seven experiments that were conducted to explore this concept with the four key experiments listed in the Appendices of this publication, and the other unlisted experiments just exploring input vs. output size and adaptive compression options in the input archive files.

2. REED-SOLOMON PATCHING APPROACH

In a very abstract way, the recovery-block data resulting from running a Reed-Solomon code on some original set of data provides a function from which changed or missing points in the original data could be interpolated, provided that enough of the original data remains available and correct for an accurate interpolation.

Consider the changes to some baseline codebase (the original ‘signal’) that may be introduced by a software patch or upgrade to be a burst error or deletion (the noise or attenuation in the transmitted signal), then the application of the Reed-Solomon recovery data could be used to ‘correct’ the ‘signal’, i.e. the codebase, in effect applying the patch.

From this viewpoint, the uncorrupted data aspired to with the Reed-Solomon ‘corrections’ becomes the final, end version of the software with the ‘communication channel’ being not just the normal medium of transmission but with an added *time aspect* where bugs and missing enhancements in the original installed codebase are the *noise* and the original baseline delivery is the *corrupt product*.

One of the inspirations for this patch mechanism is from a tool in wide use by the USENET community at the turn of the century called “Parchive.” This tool generates “PAR files” that can be used with the proper portions of the input files to repair other portions of the input files that had been damaged by transit through USENET. Here is a short description of how the Reed-Solomon codes use PAR files from the *Parity Volume Set Specification 2.0*:

“The redundant data in the PAR files is computed using Reed-Solomon codes. These codes can take a set of equal-sized blocks of data and produce a number of same-sized recovery blocks. Then, given a subset of original data blocks and some recovery block, it is possible to reproduce the original data blocks. Reed-Solomon codes can do this recovery as long as the number of missing data blocks does not out number the recovery blocks.” (Michael Nahas 2003)

The Reed-Solomon code simply uses the codebase as input (rather than the traditional signal that is to be transmitted, or video signal streamed off of a DVD for example.). The error-correction data it generates can be used to ‘correct’ a damaged codebase. In the context of software, this type of ‘correction’ is traditionally referred to as a software patch, and the ‘damaged’ codebase is simply the installed codebase without the intended updates.

2.1 Some Complications to Consider

One complexity to consider is that Reed-Solomon is a block-encoding algorithm and the codebase that will be delivered and patched is generally imagined to be structured as a tree (stored in directories on disk) at the source and destination and that it only becomes transmittable as blocks of data with some manipulation. Specifically, the directories and files of the codebase become transmittable by adding them to an archive, for example. Archiving the codebase eliminates the need to plan if the tree traversal should proceed depth- or breadth-first while transmitting from the starting point to the endpoint by allowing the archiving program to manage

that aspect. The archiving mechanism by its nature contains the structure of the codebase within a single entity that can be streamed from the source to the destination. It is this data stream that the Reed-Solomon code will be applied to. When this publication refers to running the Reed-Solomon code on the codebase, with the abstraction provided by the archiving, the intent is that the Reed-Solomon code will operate on an *archive* containing the codebase.

In archiving it, however, in the case of added or removed branches (directories), this change will only rarely occur at the ‘end’ of the archive. Reed-Solomon encoding is for block-based error-correction. The recovery data can certainly be used to restore or remove data in specific blocks in the archive, but there is some question as to whether this addition/deletion will cause changes in later data blocks in the archive.

Depending on the compression algorithm, the changed data may allow for a more or less optimized compression of the data. What amounts to changing two bytes of data in the codebase could conceivably cause nearly all of the compressed data to change, even if it’s only in the order of blocks of compressed data.

One way to overcome the potential inefficiency of having these minor changes cascading changes through the entire compressed archive might be to ensure that all of the changed data appears in a single place, at the end of the archive. To this end, appending ‘diff’ patches or compressed changed files at the end of the transmitted/delivered package would make it seem that these were just lost blocks of data that the Reed-Solomon error-correction code can recover.

This would require a method to clearly differentiate between the already-sent archived data and the appended archived data on the delivery end in order to be able to extract and use the ‘diff’ patches. This also implies that the ‘diff’ patches will be used to update the installed files themselves, which might seem to contradict the thesis of this paper. With the archiving of the installed software, however, as a simple method to abstract the software directory tree, the archived data itself becomes the delivered software, and the installation and updating of the software-that-is-actually-executed is merely the proper application (extracting the archive contents) of the delivered software. The ‘software patch’ of the thesis, then, is the proper modification of the archived data that when installed will produce the desired changes in the software-that-is-to-be-executed.

A much simpler method to overcome this would be to not compress the archive at all before using the Reed-Solomon code to generate the redundant data. In this way, the changes could conceivably be confined to just parts of the archive, leveraging the block nature of the Reed-Solomon code to modify only those parts of the archive that are changed because of directory or file changes.

3. EXPERIMENTAL RESULTS

As expected, the adaptive compression algorithms commonly used for file compression produced archive data that was altogether different across multiple invocations for even minor data modifications. The option to the compression program that was supposed to ‘reset’ the compression table and allow data modification to be localized to only small parts of the compressed file did not work properly on any of the test machines.

The initial successful test was with a single file. The file was copied to the destination. The original file at the source location was modified. The Reed-Solomon code was used to generate error-correction data. The error-correction data was then copied to the destination. The Reed-Solomon code applied the error-correction data to ‘correct’ the destination file. The modified source file and ‘corrected’ destination file were compared. The hypothesis that Reed-Solomon codes could be used as a patch mechanism was verified when the compared files were shown to be identical. The error-correction data was roughly 25% of the size of the file, cutting almost 75% off of the size of the copied data.

The next experiment closely matched the intended patch mechanism in that the Reed-Solomon code was run on an archive containing multiple files. Two files were chosen to represent the source codebase. These files were archived using the ‘tar’ command without compression. The archive was copied to the destination. A modification was made in one of the files and a new archive was created using the ‘tar’ command also without compression. The Reed-Solomon code was used to generate the error-correction data. The error-correction data was copied to the destination. The Reed-Solomon code at the destination was once again provided with the error-correction code to ‘correct’ the archive. The files were extracted and compared to the files at the source. These files were also identical and verified that a Reed-Solomon code can be used to generate data which can be used as a software patch mechanism.

3.1 File Sizes

Reed-Solomon is a block encoding algorithm and the input file is divided into blocks that the PAR2 program (Parchive 2.0) refers to as ‘slices’. The default for the PAR2 program used for this experiment divides the input file into 100 slices with 5% redundancy. The PAR2 program has options to specify block size, with the level of redundancy or the recovery block count. The default setting of 100 recovery blocks and 5% redundancy was used for all of these experiments.

A Reed-Solomon (RS) code is specified as the following:

$$RS(n,k)$$

where the code that takes **k** data symbols and adds redundant data symbols to produce an **n** symbol codeword.

A Reed-Solomon code can correct up to **t** erroneous symbols in a codeword where:

$$t = (n-k)/2$$

In the case of the PAR2 files, the data **k** is sent unmodified, and the full codeword as commonly understood above only appears internally as the PAR2 program processes the files. The redundant data symbols are within packets in the generated *.par2 files. The redundant data packet symbols:

$$\mathbf{d} = \mathbf{n} - \mathbf{k}$$

This is simply two times the number of desired correctable symbols:

$$\mathbf{d} = 2\mathbf{t}$$

The Reed-Solomon code used for these experiments stores the recovery data in 4-byte arrays.

The Reed-Solomon code used is “Parchive v2.0” which defines a number of structures in the files used to verify and correct the original data. These files and structures are defined in the “Parity Volume Set Specification 2.0.” (Michael Nahas 2003) For small input files (i.e., less than one megabyte (MB)) the overhead of this infrastructure is large. For example, with a really small input file of 406 bytes, the PAR2 files generated a total of 14,824 bytes in size (i.e., over thirty-six times the input file size). While in another example, a large input file of 8.6MB, the PAR2 files generated a total of 1.5 MB in size (i.e., about one-fifth less than of the input file size). Note that not all of the output files are required to patch the delivered codebase (i.e., if the changes are less than 5% of the total codebase).

4. CONCLUSIONS

These simple experiments have demonstrated that Reed-Solomon-generated redundant data can be used as a software patch mechanism. This patch mechanism provides the added advantage that the delivered code can be verified to be exactly the same at the destination as it was at the source. Additionally, when there is some code that is damaged or missing (depending on its size) it can be easily repaired without additional interaction required with the original code provider.

First install procedure:

- 1) At the source, tar the codebase that will be delivered but don't compress it.
- 2) Use the Reed-Solomon code with the tar archive as input to generate the files containing redundant data. (This isn't really necessary for the first installation, but will allow verification and archive repair at the destination if necessary.)
- 3) Transport the tar archive and Reed-Solomon-generated files to the destination. (The tar archive and files could all be compressed for this, but must be uncompressed at the destination. The Reed-Solomon files are optional for the initial installation.)
- 4) (Optional) Run the Reed-Solomon code with the delivered files as input to verify that what's at the destination matches what was sent from the source.
- 5) Untar the codebase, preserving the tar archive for use with later deliveries.
- 6) (If necessary) Perform installation or setup procedures on the un-tarred delivered code.

Patch install procedure:

- 1) At the source, tar the modified codebase that is intended for delivery but don't compress it.
- 2) Use the Reed-Solomon code with the tar archive as input to generate the files containing redundant data.
- 3) Transport only the Reed-Solomon-generated files to the destination. (The files could be compressed for delivery, but must be uncompressed at the destination for further installation steps.)
- 4) Run the Reed-Solomon code with the just-delivered files and the previously-delivered tar archive to modify the tar archive to be the same as the one at the source.
- 5) Untar the codebase, preserving the tar archive for use with later deliveries.
- 6) (If necessary) Perform installation or setup procedures on the un-tarred delivered code.

This Reed-Solomon-generated patch mechanism is more efficient than the reference patch mechanism because the reference patch mechanism sends the files in their entirety even when there is just a small change. This Reed-Solomon patch mechanism is not as efficient as a patch mechanism that transmits only simple 'diff' files. The Reed-Solomon patch mechanism also provides verification of the delivered code, however, and has the ability to repair the delivered code (if necessary) that neither other patch mechanism can provide.

5. FUTURE WORK

The Reed-Solomon patch mechanism can be used as a basis for future work. It has been demonstrated that the Reed-Solomon-generated recovery data can be used to update software or fix damaging changes to a codebase. There are further possibilities that could be explored and examined as demonstrated with the examples below.

A parametric study looking at block number and redundancy level vs. codebase and patch size would be useful for determining values providing the most efficient data transfers from the source to the destination.

The next version of Parchive is going to be different enough that it would probably be useful to repeat the experiments with the newer Parchive program, when it's released.

A 'roll-back' capability could be created that might allow previous versions to be recreated by properly managing the Reed-Solomon 'patch' files in version control.

Some systems use a 'sandbox' area that can be used to override or supplement existing installed software. The Reed-Solomon data generated from those areas could be version controlled and then used as a way to install or 'roll-back' different configurations.

6. REFERENCES

- Blahut, Richard E. *Algebraic Codes for Data Transmission*. New York, NY: Cambridge University Press, 2003.
- George C. Clark, Jr., and J. Bibb Cain. *Error-Correction Coding for Digital Communications*. New York, NY: Plenum Press, 1982.
- jwarnier. *Rsyncable gzip*. February 3, 2005.
<http://beeznest.wordpress.com/2005/02/03/rsyncable-gzip/> (accessed October 7, 2013).
- Loughry, Thomas A. "General Purpose Graphics Processor Unit (GPGPU) Based High Rate Rice Decompression and Reed Solomon Decoding." 2013.
- Martyn Riley, Ian Richardson. "Reed-Solomon Codes." *www.cs.cmu.edu*. 1998.
http://www.cs.cmu.edu/afs/cs.cmu.edu/project/pscico-guyb/realworld/www/reedsolomon/reed_solomon_codes.html (accessed October 7, 2013).
- Michael Nahas, Paul Nettle, Ryan Gallagher. "Parity Volume Set Specification 2.0." *archive.sourceforge.net*. May 11, 2003.
<http://archive.sourceforge.net/docs/specifications/parity-volume-spec/article-spec.html> (accessed August 16, 2013).
- NotProvided. "RSCodeHDTV.pdf." *pubs.media.mit.edu*. n.d.
<http://pubs.media.mit.edu/pubs/papers/RSCodeHDTV.pdf> (accessed October 7, 2013).
- Numerous. *MultiPar*. October 7, 2013.
<https://www.livebusinesschat.com/smf/index.php?board=396.0> (accessed October 7, 2013).
- Parchive*. September 21, 2013. <http://en.wikipedia.org/wiki/Parchive> (accessed October 7, 2013).
- Reed-Solomon error Correction*. September 30, 2013.
http://en.wikipedia.org/w/index.php?title=Reed%E2%80%93Solomon_error_correction&action=history (accessed October 7, 2013).
- Plank, James S. *A tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-like Systems*. July 1996. <http://web.eecs.utk.edu/~plank/plank/papers/CS-96-332.html> (accessed October 7, 2013).
- Pool, Martin. *About Integration of rsync and Debian*. May 14, 2002.
<http://rsync.samba.org/rsync-and-debian/rsync-and-debian.html> (accessed October 7, 2013).

- Roberto Pereyra, Lasse Collin, Ole Tange, Robert de Bath. *LZMA Utils/Discussion/Open Discussion: xz rsync-friendly*. May 13, 2009.
<http://sourceforge.net/p/lzmautils/discussion/708858/thread/2ba1b02f> (accessed October 7, 2013).
- Rue, Dan. *Compression Can Play Nice With Deduplication (and rsync)*. November 8, 2010.
<http://therub.org/2010/11/08/compression-can-play-nice-with-deduplication-and-rsync/> (accessed October 7, 2013).
- Syed Shahzad Shah, Saqub yaqub, and Fiasal Suleman, Chameleo Logics. "Self-correcting codes Conquer Noise Part 2: Reed-Solomon codecs." *ftp.radionetworkprocessor.com*. March 15, 2001. <ftp://ftp.radionetworkprocessor.com/pub/reed-solomon/reedsolomon-chameleon.pdf> (accessed October 7, 2013).
- W. Cary Huffman, and Vera Pless. *Fundamentals of Error-Correcting Codes*. New York: Cambridge University Press, 2003.

APPENDIX A: FIRST REED-SOLOMOM CODE RUN

This simply illustrates how the Reed-Solomon files were generated. It also shows how large the generated files were relative to the very small input archive files in the initial test.

Experimental setup

Using par2cmdLine 0.4 downloaded from the following:

<http://sourceforge.net/projects/parchive/files/par2cmdline/0.4/par2cmdline-0.4.tar.gz/download>

Modified to declare templates as gcc version 4.1.2 expects.

Original baseline – two text files with three lines of text and a single subdirectory with a text file also with three lines of text. From within the ‘original’ subdirectory, archived with: tar -cvzf original.tgz . Par files created with: par2 create original.tgz

```
drwxrwxr-x 3 johndoe johndoe 4096 Jun 10 11:37 original
-rw-rw-r-- 1 johndoe johndoe 406 Jun 10 11:38 original.tgz
-rw-rw-r-- 1 johndoe johndoe 2444 Jun 10 11:56 original.tgz.par2
-rw-rw-r-- 1 johndoe johndoe 2516 Jun 10 11:56 original.tgz.vol0+1.par2
-rw-rw-r-- 1 johndoe johndoe 4932 Jun 10 11:56 original.tgz.vol1+2.par2
```


APPENDIX B: ADAPTIVE COMPRESSION PROBLEMS

This demonstrates the problem that was predicted in the beginning of this report with trying to use a block-encoding algorithm with a set of archives using adaptive compression.

With compression, the RS patch solution suffers from the same shortcomings as ‘rsync’ (a ‘smart’ copy program that tries to copy only the changed pieces of files) when it comes to compressed files. The compression algorithm uses adaptive compression, meaning it examines the data it just compressed to determine how to best compress the data it will compress next. So a change to even a byte or two of data could change the entire output of the compressed data.

Experimentally, this is exactly what happened.

Experimental setup:

Start with the dataset for baseline two (the contents of the WorldWind version 1.4.0 zip archive, take two). From within the ‘orig.wwnd’ subdirectory, archived with: `tar -cvf ../wwnd.tar .; cd ..; gzip --rsyncable wwnd.tar` Par files created with: `par2 create wwnd.tar.gz`

```
-rw-rw-r-- 1 johndoe johndoe 82663395 Jun 17 12:23 wwnd.tar.gz
-rw-rw-r-- 1 johndoe johndoe 40404 Jun 17 12:24 wwnd.tar.gz.par2
-rw-rw-r-- 1 johndoe johndoe 81812 Jun 17 12:24 wwnd.tar.gz.vol000+01.par2
-rw-rw-r-- 1 johndoe johndoe 163524 Jun 17 12:24 wwnd.tar.gz.vol001+02.par2
-rw-rw-r-- 1 johndoe johndoe 286644 Jun 17 12:24 wwnd.tar.gz.vol003+04.par2
-rw-rw-r-- 1 johndoe johndoe 492580 Jun 17 12:24 wwnd.tar.gz.vol007+08.par2
-rw-rw-r-- 1 johndoe johndoe 864148 Jun 17 12:24 wwnd.tar.gz.vol015+16.par2
-rw-rw-r-- 1 johndoe johndoe 1566980 Jun 17 12:24 wwnd.tar.gz.vol031+32.par2
-rw-rw-r-- 1 johndoe johndoe 1774020 Jun 17 12:24 wwnd.tar.gz.vol063+37.par2
```

Create another directory, wmod1. Unarchive the original Whirlwind tgz in it. Add another file with three lines of text. Create another archive: `tar -cvf ../wwnd.tar .; cd ..; gzip --rsyncable wwnd.tar` Create par files for it with: `par2 create wwnd.tgz`

```
-rw-rw-r-- 1 johndoe johndoe 82667454 Jun 17 12:31 wwnd.tar.gz
-rw-rw-r-- 1 johndoe johndoe 40404 Jun 17 12:32 wwnd.tar.gz.par2
-rw-rw-r-- 1 johndoe johndoe 81816 Jun 17 12:32 wwnd.tar.gz.vol000+01.par2
-rw-rw-r-- 1 johndoe johndoe 163532 Jun 17 12:32 wwnd.tar.gz.vol001+02.par2
-rw-rw-r-- 1 johndoe johndoe 286660 Jun 17 12:32 wwnd.tar.gz.vol003+04.par2
-rw-rw-r-- 1 johndoe johndoe 492612 Jun 17 12:32 wwnd.tar.gz.vol007+08.par2
-rw-rw-r-- 1 johndoe johndoe 864212 Jun 17 12:32 wwnd.tar.gz.vol015+16.par2
-rw-rw-r-- 1 johndoe johndoe 1567108 Jun 17 12:32 wwnd.tar.gz.vol031+32.par2
-rw-rw-r-- 1 johndoe johndoe 1774168 Jun 17 12:32 wwnd.tar.gz.vol063+37.par2
```

After the first comparison there is a complete replacement of the contents of the archive, and as a consequence, the resulting par files.

APPENDIX C: PATCHING WITHOUT COMPRESSION

This simply illustrates the experimental setup of the patching of a file using Reed-Solomon generated data.

Experimental setup

Starting with the 'worldwind.jar' file, create RS recovery files. Modify a single byte in the jar file and regenerate the recovery files. Compare the recovery files.

```
-rw-r--r-- 1 johndoe johndoe 5560375 Jun 19 10:31 worldwind.jar
-rw-rw-r-- 1 johndoe johndoe 40428 Jun 19 10:33 worldwind.jar.par2
-rw-rw-r-- 1 johndoe johndoe 43276 Jun 19 10:33 worldwind.jar.vol000+01.par2
-rw-rw-r-- 1 johndoe johndoe 86452 Jun 19 10:33 worldwind.jar.vol001+02.par2
-rw-rw-r-- 1 johndoe johndoe 132476 Jun 19 10:33 worldwind.jar.vol003+04.par2
-rw-rw-r-- 1 johndoe johndoe 184196 Jun 19 10:33 worldwind.jar.vol007+08.par2
-rw-rw-r-- 1 johndoe johndoe 247308 Jun 19 10:33 worldwind.jar.vol015+16.par2
-rw-rw-r-- 1 johndoe johndoe 333204 Jun 19 10:33 worldwind.jar.vol031+32.par2
-rw-rw-r-- 1 johndoe johndoe 347444 Jun 19 10:33 worldwind.jar.vol063+37.par2
```

Modified:

```
-rw-r--r-- 1 johndoe johndoe 5560375 Jun 19 10:36 worldwind.jar
-rw-rw-r-- 1 johndoe johndoe 40428 Jun 19 10:36 worldwind.jar.par2
-rw-rw-r-- 1 johndoe johndoe 43276 Jun 19 10:36 worldwind.jar.vol000+01.par2
-rw-rw-r-- 1 johndoe johndoe 86452 Jun 19 10:36 worldwind.jar.vol001+02.par2
-rw-rw-r-- 1 johndoe johndoe 132476 Jun 19 10:36 worldwind.jar.vol003+04.par2
-rw-rw-r-- 1 johndoe johndoe 184196 Jun 19 10:36 worldwind.jar.vol007+08.par2
-rw-rw-r-- 1 johndoe johndoe 247308 Jun 19 10:36 worldwind.jar.vol015+16.par2
-rw-rw-r-- 1 johndoe johndoe 333204 Jun 19 10:36 worldwind.jar.vol031+32.par2
-rw-rw-r-- 1 johndoe johndoe 347444 Jun 19 10:36 worldwind.jar.vol063+37.par2
```

Comparison: With a single-byte change in the original file, the resulting par files still display multiple multi-byte changes. A diff-patch mechanism would be faster and more efficient. The summed size of the resulting par files *are* smaller than the original file. So transmitting these would be more efficient than transmitting the full file, but not as nearly as efficient as transmitting just patch files.

The next step is to use the RS recovery files from the changed-file case to 'patch' the original file to the new changed-file.

Experimental setup

Create a new directory and copy the original unmodified worldwind.jar into it. This will represent the original install of the codebase. Use the par files generated for the worldwind.jar that was modified to 'patch' the worldwind.jar to match the modified version (i.e., this will be the 'next version'). Verify that the patched file and the modified version of worldwind.jar match.


```
[johndoe@mymachine rstesting]$ mkdir sfile.patch1
[johndoe@mymachine rstesting]$ cd sfile.patch1
[johndoe@mymachine sfile.patch1]$ cp ../orig.singlefile/worldwind.jar .
[johndoe@mymachine sfile.patch1]$ cp ../fmod1/*.par2 .
par2 verify worldwind.jar
par2cmdline version 0.4, Copyright (C) 2003 Peter Brian Clements.
```

par2cmdline comes with ABSOLUTELY NO WARRANTY.

This is free software, and you are welcome to redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version. See COPYING for details.

```
Loading "worldwind.jar.par2".
Loaded 4 new packets
Loading "worldwind.jar.vol001+02.par2".
Loaded 2 new packets including 2 recovery blocks
Loading "worldwind.jar.vol003+04.par2".
Loaded 4 new packets including 4 recovery blocks
Loading "worldwind.jar.vol015+16.par2".
Loaded 16 new packets including 16 recovery blocks
Loading "worldwind.jar.vol007+08.par2".
Loaded 8 new packets including 8 recovery blocks
Loading "worldwind.jar.vol031+32.par2".
Loaded 32 new packets including 32 recovery blocks
Loading "worldwind.jar.vol063+37.par2".
Loaded 37 new packets including 37 recovery blocks
Loading "worldwind.jar.vol000+01.par2".
Loaded 1 new packets including 1 recovery blocks
```

```
There are 1 recoverable files and 0 other files.
The block size used was 2,780 bytes.
There are a total of 2,001 data blocks.
The total size of the data files is 5,560,375 bytes.
```

Verifying source files:

Target: "worldwind.jar" - damaged. Found 2,000 of 2,001 data blocks.

Scanning extra files:

```
Repair is required.
1 file(s) exist but are damaged.
You have 2,000 out of 2,001 data blocks available.
You have 100 recovery blocks available.
```

Repair is possible.
You have an excess of 99 recovery blocks.
1 recovery blocks will be used to repair.
par2 repair worldwind.jar
par2cmdline version 0.4, Copyright (C) 2003 Peter Brian Clements.

par2cmdline comes with ABSOLUTELY NO WARRANTY.

This is free software, and you are welcome to redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version. See COPYING for details.

Loading "worldwind.jar.par2".
Loaded 4 new packets
Loading "worldwind.jar.vol001+02.par2".
Loaded 2 new packets including 2 recovery blocks
Loading "worldwind.jar.vol003+04.par2".
Loaded 4 new packets including 4 recovery blocks
Loading "worldwind.jar.vol015+16.par2".
Loaded 16 new packets including 16 recovery blocks
Loading "worldwind.jar.vol007+08.par2".
Loaded 8 new packets including 8 recovery blocks
Loading "worldwind.jar.vol031+32.par2".
Loaded 32 new packets including 32 recovery blocks
Loading "worldwind.jar.vol063+37.par2".
Loaded 37 new packets including 37 recovery blocks
Loading "worldwind.jar.vol000+01.par2".
Loaded 1 new packets including 1 recovery blocks

There are 1 recoverable files and 0 other files.
The block size used was 2,780 bytes.
There are a total of 2,001 data blocks.
The total size of the data files is 5,560,375 bytes.

Verifying source files:

Target: "worldwind.jar" - damaged. Found 2,000 of 2,001 data blocks.

Scanning extra files:

Repair is required.
1 file(s) exist but are damaged.
You have 2,000 out of 2,001 data blocks available.
You have 100 recovery blocks available.
Repair is possible.
You have an excess of 99 recovery blocks.

1 recovery blocks will be used to repair.

Computing Reed Solomon matrix.

Constructing: done.

Solving: done.

Wrote 5,560,375 bytes to disk

Verifying repaired files:

Target: "worldwind.jar" - found.

Repair complete.

Comparison of the 'patched' worldwind.jar and modified version indicates that applying the RS recovery data did correctly patch the file. The patched file and the modified worldwind.jar match.

This verifies the hypothesis that the RS recovery data can be used to patch existing software. The existing 'original install' was modified using the RS recovery data to produce the 'next version'.

APPENDIX D: PATCHING A CODEBASE

This demonstrates how to patch a file using the Reed-Solomon data to patch an archive of files.

Experimental Setup

Create a new directory and copy the original unmodified worldwind.jar into it. Copy another jar file into the directory and tar the two files together without compression. This will represent the original install of the codebase.

Tar worldwind.jar and worldwindx.jar without compression, “tar cvf twofiles.tar worldwind*”. Modify a byte in worldwindx.jar and tar the files together again without compression, “tar cvf nexttwofiles.tar worldwind*”.

```
-rw-r--r-- 1 johndoe johndoe 5560375 Aug 21 12:34 worldwind.jar
-rw-rw-r-- 1 johndoe johndoe 9021440 Aug 21 12:37 twofiles.tar
-rw-r--r-- 1 johndoe johndoe 3452092 Aug 21 12:38 worldwindx.jar
-rw-rw-r-- 1 johndoe johndoe 9021440 Aug 21 12:38 nexttwofiles.tar
```

Here, ‘twofiles.tar’ represents the original install of the codebase and the ‘nexttwofiles.tar’ represents the next version of the codebase. Now if the Reed-Solomon redundant data files are generated for the ‘next-version’ codebase, these files can be applied to the ‘original-install’ codebase (the name of that tar file will need to be changed in this case so that the par files that were generated will work on that file.) After ‘par2 create nexttwofiles.tar’:

```
-rw-rw-r-- 1 johndoe johndoe 40408 Aug 21 12:54 nexttwofiles.tar.par2
-rw-rw-r-- 1 johndoe johndoe 44988 Aug 21 12:54 nexttwofiles.tar.vol000+01.par2
-rw-rw-r-- 1 johndoe johndoe 89876 Aug 21 12:54 nexttwofiles.tar.vol001+02.par2
-rw-rw-r-- 1 johndoe johndoe 139344 Aug 21 12:54 nexttwofiles.tar.vol003+04.par2
-rw-rw-r-- 1 johndoe johndoe 197972 Aug 21 12:54 nexttwofiles.tar.vol007+08.par2
-rw-rw-r-- 1 johndoe johndoe 274920 Aug 21 12:54 nexttwofiles.tar.vol015+16.par2
-rw-rw-r-- 1 johndoe johndoe 388508 Aug 21 12:54 nexttwofiles.tar.vol031+32.par2
-rw-rw-r-- 1 johndoe johndoe 411408 Aug 21 12:54 nexttwofiles.tar.vol063+37.par2
```

In a new directory with the ‘original-install’ tarball renamed as the ‘next-version’ tarball, using ‘par2 verify nexttwofiles.tar’ reports that:

```
Repair is required.
1 file(s) exist but are damaged.
You have 1,999 out of 2,000 data blocks available.
You have 100 recovery blocks available.
Repair is possible.
You have an excess of 99 recovery blocks.
1 recovery blocks will be used to repair.
```

Using 'par2 repair nexttwofiles.tar' patches the tarball to match the actual 'next-version' tarball. 'diff' verifies that they are the same file. After untarring the patched tarball, the 'worldwindx.jar' file matches with the expected modified 'worldwindx.jar'.

This further verifies the hypothesis that the RS redundant files can be used as a software patch mechanism. *The important distinction in this case is that the modified file is not the actual file that is modified directly by the Reed-Solomon code, there is other software packaged with the modified file and it is the 'package' that the Reed-Solomon code operates on.*

Distribution

2	MS0576	Kevin Pendley	5519
1	MS0568	Susan Harty	5516
1	MS0986	Jim Martinez	5519
1	MS0899	Technical Library	9536 (electronic copy)