AR-010-672

An Analysis of Transmission and Storage Gains from Sliding Checksum Methods

Richard Taylor and Rittwik Jana

DSTO-TR-0743

D E P A R T M E N T ◆ O F D E F E N C E

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

An Analysis of Transmission and Storage Gains from Sliding Checksum Methods

Richard Taylor and Rittwik Jana#*

***Communications Division**
#Information Technology Division
Electronics and Surveillance Research Laboratory

DSTO-TR-0743

ABSTRACT

In a previous report we described, modelled and analysed a protocol "rsync" for synchronising related files at different ends of a communications channel with a minimum of transmitted data. This report identifies the extent of the gains that this method may provide by performing experiments on large repositories of data. The outcome is a new method of compressing large directories of data that together with conventional methods provides major gains in compression factors. As is explained in the report, these gains in compression are also indicative of the data transmission gains available when mirroring a collection of files at a remote site using rsync.

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

AQF99-06-1191

Published by

DSTO Electronics and Surveillance Research Laboratory
PO Box 1500
Salisbury South Australia 5108

Telephone: (08) 8259 5555
Fax: (08) 8259 6567
© Commonwealth of Australia 1998
AR-010-672
November 1998

An Analysis of Transmission and Storage Gains from Sliding Checksum Methods

Executive Summary

In a previous report we described, modelled and analysed a protocol "rsync" for synchronising related files at different ends of a communications channel with a minimum of transmitted data. This protocol is suitable for supporting such activities as collaborative writing of documentation and synchronisation of distributed databases in the situation where no one location is aware of the differences and similarities between their files and remote related files.

Rsync may be particularly useful in synchronising databases that have significant disconnections or outages, resulting in an inefficiency or inability to synchronise data based on large numbers of missed updates. In this situation Rsync may be used to efficiently synchronise databases without any version control or common reference point. Another major application of Rsync is in maintaining web pages which are regularly being changed at the server and have to be synchronised with the clients. Thus the changes to client files are identified and only updates to files are sent from the server. This is achieved without the need for the server to maintain any records of client files or to store old versions. The characteristics of Rsync make it a good match to the paradigm of high computing - low bandwidth, as well mobile information systems with dynamic operational status. This makes Rsync an attractive tool for tactical C3 environments.

The power of this method as a tool to minimise the use of bandwidth depends on the likelihood and extent to which transmitted data is similar to data already held by the receiver. This report investigates the gains that this method may provide by performing experiments on large repositories of data.

By building on the sliding checksum method of rsync, a novel method of compressing large data repositories is developed. The report shows how this method can be used together with conventional compression methods to produce major gains in compression factors. The experimental data presented indicates that the compression factor may be doubled with these methods in comparison to conventional methods, depending on the application types.

These gains in compression are also indicative of the data transmission gains available when mirroring a file structure at a remote site using the rsync protocol.

Authors

Richard Taylor
Communications Division

Richard Taylor is Head of the Network Integration Group of the Defence Science and Technology Organisation's (DSTO) Communications Division. A PhD in Mathematics from the University of Melbourne, Richard has worked at the Telecom Research Laboratories in Victoria, and has over 9 years experience in the fields of communication reliability and security.

Rittwik Jana
Information Technology Division

Rittwik Jana is a Professional Officer with the Intelligence Systems Group of the Defence Science and Technology Organisation's (DSTO) Information Technology Division. His main research interests include transmission of imagery over low bandwidth communication channels. He is currently pursuing a PhD in telecommunications at the Australian National University.

Contents

1. Introduction

In a previous report we described, modelled and analysed a protocol called rsync [1] for "synchronising" related files at different ends of a communications channel with a minimum of transmitted data. This protocol is suitable for supporting activities such as collaborative writing of documentation and synchronisation of distributed databases in the situation where no one location is aware of the differences and similarities between their files and remote related files. The key concept in the rsync protocol is the use of sliding checksums to efficiently identify similarities between data sets. Note that the extent of the utility of this method as a tool to minimise use of bandwidth depends on the likelihood and extent to which transmitted data is similar to data already held by the receiver.

In this report we use the sliding checksum method of rsync to identify similarities in large repositories of data, which has two major outcomes,

- providing a novel method of compressing large data repositories, "Rzip", which can be used together with conventional compression methods to produce major gains in compression factors,

- as a measure of the possible utility of rsync in terms of transmission savings.

To understand how the second outcome follows, consider a situation where a large directory of data is developed over time at one location α and is required to be duplicated at another location β. Thus new files when completed are added to the directory at α and need to be sent to β. In this case the compression achieved by Rzip of the directory at time t provides an indication (neglecting the checksum and other overheads of Rsync) of the total data that would need to be transmitted up to time t for location β to be up to date.

This report analyses the behaviour of the algorithm under varying conditions and in particular a comparative study is made with a standard compression algorithm GnuZip (or Gzip), which uses Lempel-Ziv (LZ77) coding.

2. Background - The Rsync protocol

The aim is to update File B (the old version) with File A (the current version). There are three important transactions that occur during the execution of the algorithm, as shown by the arrows in Figure 1.

- Step 1: α notifies β that an rsync operation is to be initiated from File *A* to File *B*.
- Step 2: β partitions File *B* into non-overlapping fixed size blocks each of size *b* bytes. For each of these blocks a simple 32 bit checksum and a much stronger 128 bit checksum (MD5 see [2], [3]) is calculated. These checksums are consolidated into a table and sent back to α.
- Step 3: α scans through File *A* and calculates checksums for all blocks of length *b* bytes at all offset positions. These checksums are used to determine blocks of data in File *B*(in any position) that match blocks in File *A*. 32 bit checksums are calculated and checked first, if a match is found within the received table then the 128 bit checksum is calculated and checked to be surer of the match.
- Step 4: α sends β a sequence of instructions for constructing a copy of *A*. Each instruction is either a reference to a block of data or literal data. Literal data is sent only for those blocks of *A* which are different to any of the blocks in *B*.

The algorithm may be better understood with reference to Figure 1 (see [1], [4] for more details):

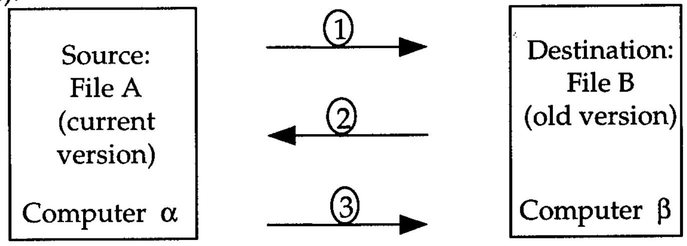

Figure 1 - Information flow during rsync operation.

3. How Rzip works

In this section we outline the algorithm and highlight some of the potential areas that can be optimised for speed. In simple terms the algorithm works by identifying repeated blocks in the data and noting that pointers to repeated blocks may be stored rather than the data itself. The challenge is to identify repeated blocks efficiently at whatever byte offset they may occur.

There are two principal phases of execution. In the first pass, the data is partitioned into non overlapping fixed size blocks, each of size *b* bytes (typically in the range 100 to 1000 bytes). For each of these blocks a 64 bit checksum is calculated. These checksums are consolidated into a table and sorted. In the second pass, 64 bit checksums are calculated for all blocks at all byte offset positions. For each byte offset "sliding" checksums are efficiently calculated by an incremental method, and searched for in the sorted table. If a match is found within the table, its offset location is registered in another table, the rows of which correspond to the block numbers of the sorted table. In general, redundancy in the file occurs if there is a match with the sorted table such that the corresponding blocks do not overlapp. By noting all the redundancies it is possible to determine the overall compression factor. The offset locations of repeated blocks may be used as the basis of a compression method by using pointers to repeated data blocks. The details of the algorithm to estimate the compression factor in a file are outlined below:

1. Set redundancy R to *0*. R is a measure of the amount of repeated data identified by the algorithm in the form of repeated blocks.
2. Partition file into non overlapping fixed size blocks of size b bytes.
3. Calculate 64 bit checksums for each block and store them in a sorted table.
4. Start at the beginning of the file again and calculate checksum for first block.
5. Shift block one byte, and calculate checksum.
6. Search for checksum in the sorted table.
7. If the search is successful AND this is the first time this checksum has been found, note the offset position of the block. Go to 4.
8. If search is successful AND this is at least the second time this checksum has been found AND the current block does not overlap the previous matching block, increment R by b (at this point a block that occurs previously in the file has been identified). Shift the block by b bytes and calculate checksum. Go to 5.

4. Sliding Checksums

The computationally intensive component of the algorithm is Step 4, since checksums are calculated and matched within a table for every byte offset in the file. In order to make these computations feasible, we use checksums that are simple to compute, and in particular can be updated quickly as the block offset is incremented (or "slides"). In a previous report [4] we have designed checksums that are both strong as block fingerprints, and can be updated for each new block offset with just one multiplication together with elementary operations (add, subtract, assign, and shift).

It is important to note that the length of the checksum required to provide a given level of confidence that the checksum provides a unique fingerprint of the data blocks varies significantly with the file size. We have estimated (see [4]) that in a file of size Y bytes, with block size b bytes, and checksum size n bits then under certain statistical assumptions the probability p of at least one "collision" (different blocks with the same checksum) may be bounded by

$$p \leq \frac{\left(\frac{Y}{b}\right) \times \left(Y - \frac{Y}{b}\right)}{2^n}.$$

If for example we have a file of 1 Gigabyte (Y=10^9 bytes), a block size of 400 (b=400), and checksums of 64 bits (n=64) then p is upper bounded by 1.35×10^{-4}.

In this report we have utilised the following four 16 bit functions from [4],

$$D1(k,l) = X_l + 3X_{l-1} + 3^2 X_{l-2} + \ldots + 3^{b-1} X_k \bmod[2^{16} - 1],$$
$$D2(k,l) = X_l + 5X_{l-1} + 5^2 X_{l-2} + \ldots + 5^{b-1} X_k \bmod[2^{16} - 3],$$
$$D3(k,l) = X_l + 7X_{l-1} + 7^2 X_{l-2} + \ldots + 7^{b-1} X_k \bmod[2^{16} - 5],$$
$$D4(k,l) = X_l + 17X_{l-1} + 17^2 X_{l-2} + \ldots + 17^{b-1} X_k \bmod[2^{16} - 7].$$

The corresponding sliding updates may be evaluated as,

$$D1(k+1,l+1) = 3D1(k,l) + X_{l+1} - 3^b X_{k+1} \bmod[2^{16} - 1],$$
$$D2(k+1,l+1) = 5D1(k,l) + X_{l+1} - 5^b X_{k+1} \bmod[2^{16} - 3],$$
$$D3(k+1,l+1) = 7D1(k,l) + X_{l+1} - 7^b X_{k+1} \bmod[2^{16} - 5],$$
$$D4(k+1,l+1) = 17D1(k,l) + X_{l+1} - 17^b X_{k+1} \bmod[2^{16} - 7].$$

The 64 bit checksum is created by concatenating D1, D2, D3 and D4.

5. Experimental Testing

The algorithm was tested with three different data sets. The first consists of a large tar file. This was generated from a common working directory visible to all the groups at the DSTO C3 Research Centre in Fernhill Park, Canberra. For each of the groups within the Centre a tar file was generated, consisting of all file types present (.docs, .ppt, .c, .exe, .exel, .gif, etc). The second consists of all Microsoft Power Point files from all of the groups concatenated into a large file with no compression. Similarly, the third consists of all Microsoft Word files from all of the groups in one large file. All of the data presented in Figures 2 and 3 corresponds to a block size *b* of 400 bytes.

Shown below in Figure 2 is an experiment where the original data is first processed using Rzip, the output of which is then further compressed using a standard compression algorithm GnuZip. This is compared with the result of compression with GnuZip only. By preprocessing the data with Rzip one eliminates large blocks of data which are identical, however they may be scattered throughout the file. This is the key difference between Rzip and run-length coding schemes. In general run length compression operates on a small window over which it allocates codewords to frequently occurring symbols. We note that Rzip and GnuZip appear to have somewhat orthogonal properties, so that they can work in series to maximise the compression of the original data. This effect is dependent on the order of compression however, as we have found that Rzip does not significantly compress a file that has previously been compressed with GnuZip. It is important to note that the block size can also be adaptively resized to take into account redundancies present at different levels. Thus for small window sizes Rzip will mimic standard compression algorithms such as GnuZip.

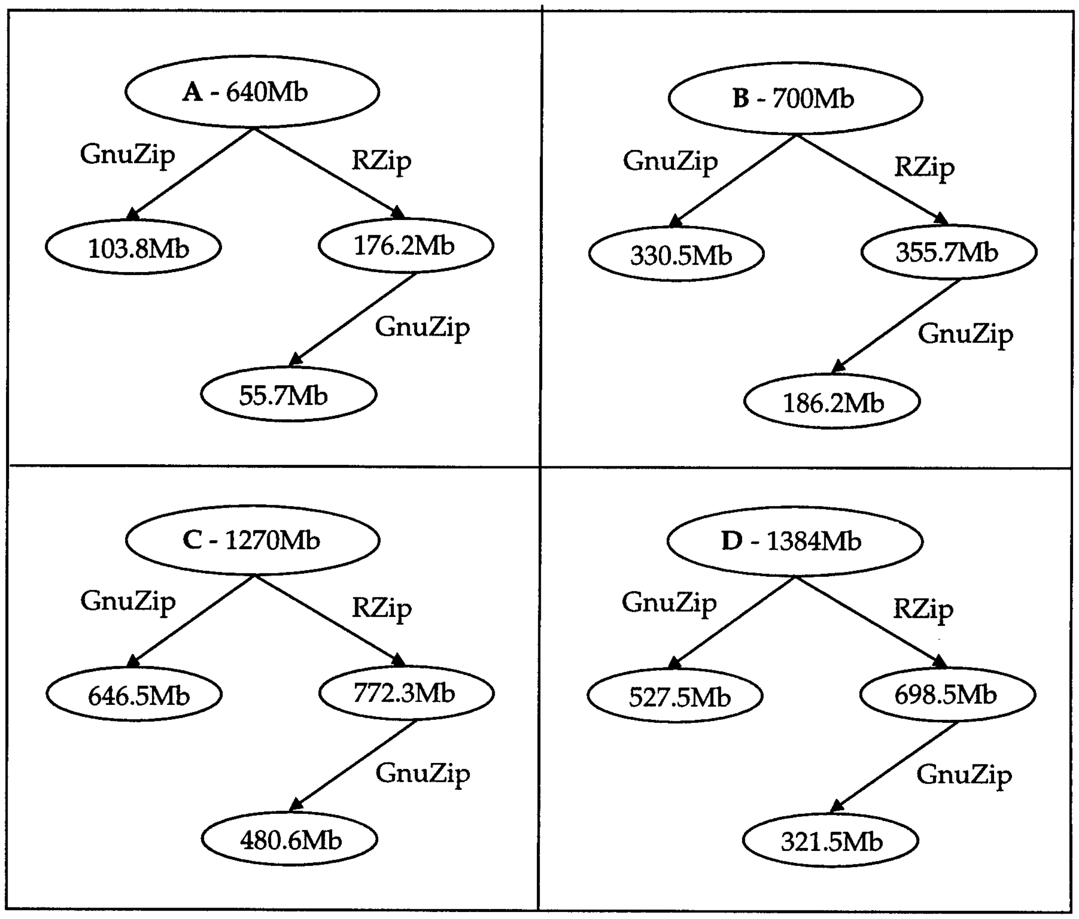

Figure 2

In Figure 3 similar results are shown for data sets based on application type. The compression figures shown are for all Microsoft PowerPoint files and all Microsoft Word files across the C3 Research Centre.

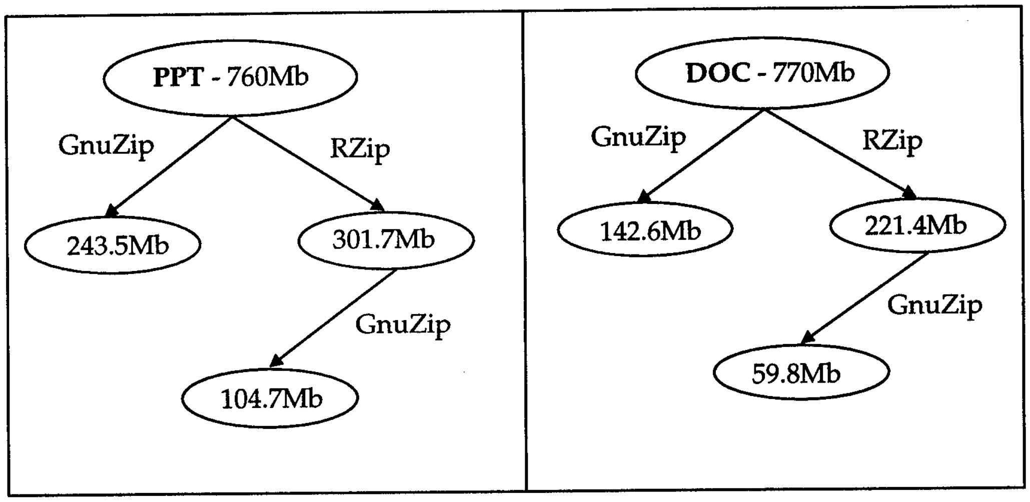

Figure 3

The block size used in rsync and Rzip plays an important part in the efficiency of both methods. Larger blocks mean that the hash tables constructed and used in both methods are smaller which leads to efficiencies in both storage, processing and for rsync transmission time. Of course this comes at the price that rsync and Rzip will identify less repeated data, between and within data sets, as the block size increases. Thus there is a compromise between block size and the overheads associated with generating and transmitting checksums and location pointers.

Figure 4/Table 1 shows that as the block sizes increase, the redundancy (as measured by block repetitions) identified by Rzip decreases surprisingly slowly. The files tested were a 135 MByte file formed from the concatenation of a general directory of Microsoft Powerpoint files, and a 150 MByte file formed from the concatenation of a general directory of Microsoft Word files.

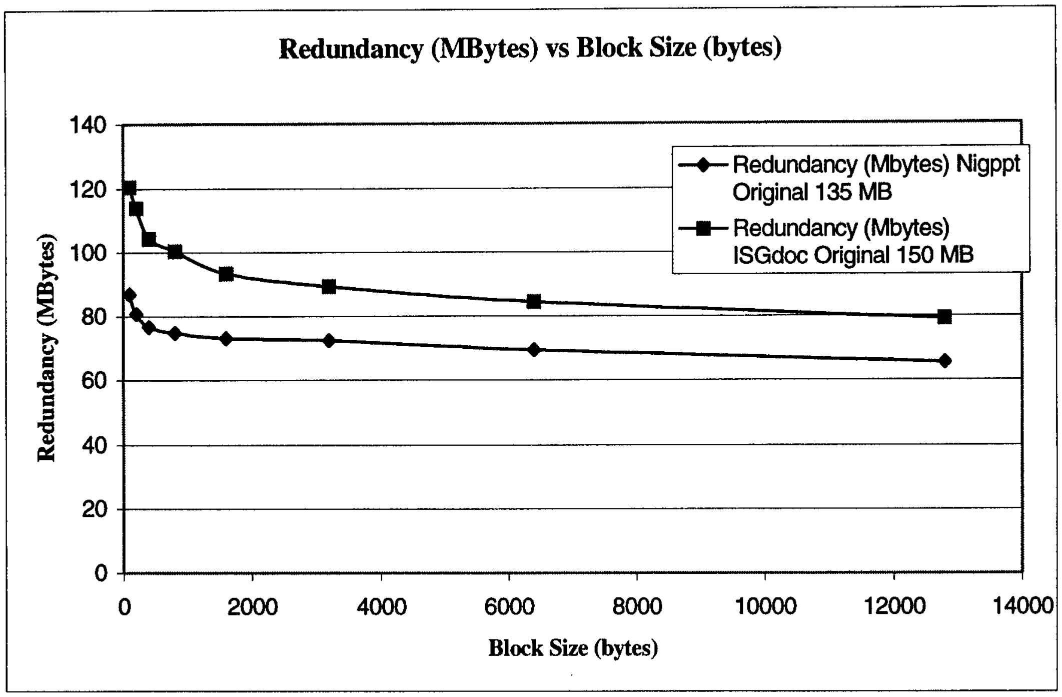

Figure 4 – Redundancy asymptotes as block sizes increase

Block Size (bytes)	Redundancy (Mbytes) 135 MByte ppt directory	Redundancy (Mbytes) 150 Mbyte doc directory
100	86.87	120.57
200	80.68	113.99
400	76.63	104.32
800	74.82	100.46
1600	73.02	93.35
3200	72.34	89.22
6400	69.40	84.33
12800	65.53	79.08

Table 1 – Redundancy asymptotes as block sizes increase

Computational Performance

The table below indicates the computational performance of Rzip in comparison to Gzip in processing the data sets from Figure 4/Table 1. This indicates that the computational requirement of Rzip is of the same order of magnitude as Gzip (note that the Rzip code we have produced is not optimised for speed).

Execution Time (minutes)	Gzip	Rzip
PPT file (135 Mbyte)	13	27
DOC file (150 Mbyte)	17	33

Table 2 – Computational performance of Gzip and Rzip

Adding to Rzip

Figure 4 suggests that it makes sense to modify Rzip in the following way so that new data may be added to the compressed data generated by Rzip incrementally. Thus if in Step 1 and 2 of Rzip the sorted checksum list is generated and stored in a file, then when adding a new file to the stored compressed file it is only necessary to calculate rolling checksums on the new file and compare with the sorted checksum list. The sorted checksum list may then be updated and the new file added to the compressed data as a combination of both pointers to repeated blocks and new data. Thus the large amount of stored compressed data need not be read or processed in any way, but rather the stored checksum list is used instead. As the size of the stored checksum list is inversely proportional to the block size used, the bigger the blocksize while combined with a good level of redundancy detection provides the best results in terms of storage and speed. As an example a block size of 3200 bytes (see Figure 4), with checksums of 64 bits and block location pointers of 32 bits generates a sorted checksum list of 12/3200 = 0.00375 or (0.375%) of the original data.

6. Conclusions

The results of the tests presented in this report indicate that repositories of stored data contain significant repetitions of large blocks. This fact may be taken advantage of to make significant improvements to existing methods for both compressing and transmitting data.

7. References

[1]Tridgell, A. and Mackerras P, "The rsync algorithm", Joint Computer Science Technical Report Series, TR-CS-96-05, June 1996.

[2]R. L. Rivest, "The MD5 Message Digest Algorithm", RFC 1321, April 1992.

[3]B. Schneier, "One Way Hash Functions", Dr. Dobbs Journal, v. 16, no. 9, pp. 148 151, September 1991.

[4]Taylor R., Jana R and Grigg M, "Checksum Testing of Remote Synchronisation tool", DSTO Technical Report 0627, Novemeber 1997.

DISTRIBUTION LIST

An Analysis of Transmission and Storage Gains from Sliding Checksum Methods

Richard Taylor and Rittwik Jana

AUSTRALIA

1. DEFENCE ORGANISATION

a. Task Sponsor
Director General C3I Development

b. S&T Program
Chief Defence Scientist
FAS Science Policy } shared copy
AS Science Corporate Management
Director General Science Policy Development
Counsellor Defence Science, London (Doc Data Sheet)
Counsellor Defence Science, Washington (Doc Data Sheet)
Scientific Adviser to MRDC Thailand (Doc Data Sheet)
Director General Scientific Advisers and Trials/Scientific Adviser Policy and Command (shared copy)
Navy Scientific Adviser (Doc Data Sheet and distribution list only)
Scientific Adviser - Army (Doc Data Sheet and distribution list only)
Air Force Scientific Adviser
Director Trials

Aeronautical and Maritime Research Laboratory
Director

Electronics and Surveillance Research Laboratory
Director

Chief, Information Technology Division
Chief, Communications Division
Research Leader Military Information Networks
Head Network Integration
Co-Author(s): R. Jana

DSTO Library
Library Fishermens Bend
Library Maribyrnong
Library Salisbury (2 copies)
Australian Archives
Library, MOD, Pyrmont (Doc Data sheet)

c. Capability Development Division
DGMD (Doc Data Sheet)

DGLD (Doc Data Sheet)

d. **Navy**
SO (Science) - MHQ

e. **Army**
ABCA Office, G-1-34, Russell Offices, Canberra (4 copies)
SO (Science) - LHQ, 3 Bde, 1 Bde, HQ Training Command

f. **Air Force**
SO (Science) - AHQ

g. **Intelligence Program**
Defence Intelligence Organisation
DDI, Defence Signals Directorate (Doc Data Sheet only)

h. **Acquisition and Logistics Program**
PD JCSE
PD JISE
PD BCSS

i. **Corporate Support Program (libraries)**
OIC TRS, Defence Regional Library, Canberra
Officer in Charge, Document Exchange Centre (DEC) (Doc Data Sheet only)
DEC requires the following copies of public release reports to meet exchange agreements under their management:
*US Defence Technical Information Centre, 2 copies
*UK Defence Research Information Center, 2 copies
*Canada Defence Scientific Information Service, 1 copy
*NZ Defence Information Centre, 1 copy
National Library of Australia, 1 copy

2. **UNIVERSITIES AND COLLEGES**

Australian National University Library
Australian Defence Force Academy
Library
Head of Aerospace and Mechanical Engineering
Deakin University, Serials Section (M list), Deakin University Library
Senior Librarian, Hargrave Library, Monash University
Librarian, Flinders University

3. **OTHER ORGANISATIONS**

NASA (Canberra)
AGPS
State Library of South Australia
Parliamentary Library, South Australia

OUTSIDE AUSTRALIA

4. **ABSTRACTING AND INFORMATION ORGANISATIONS**
   INSPEC: Acquisitions Section Institution of Electrical Engineers
   Library, Chemical Abstracts Reference Service
   Engineering Societies Library, US
   Materials Information, Cambridge Scientific Abstracts, US
   Documents Librarian, The Center for Research Libraries, US

5. **INFORMATION EXCHANGE AGREEMENT PARTNERS**
   Acquisitions Unit, Science Reference and Information Service, UK
   Library - Exchange Desk, National Institute of Standards and Technology, US

SPARES (6 copies)

Total number of copies: 60

Page classification: UNCLASSIFIED

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION DOCUMENT CONTROL DATA	1. PRIVACY MARKING/CAVEAT (OF DOCUMENT)
2. TITLE An Analysis of Transmission and Storage Gains from Sliding Checksum Methods	3. SECURITY CLASSIFICATION (FOR UNCLASSIFIED REPORTS THAT ARE LIMITED RELEASE USE (L) NEXT TO DOCUMENT CLASSIFICATION) Document (U) Title (U) Abstract (U)
4. AUTHOR(S) Richard Taylor and Rittwik Jana	5. CORPORATE AUTHOR Electronics and Surveillance Research Laboratory PO Box 1500 Salisbury SA 5108

6a. DSTO NUMBER DSTO-TR-0743	6b. AR NUMBER AR-010-672	6c. TYPE OF REPORT Technical Report	7. DOCUMENT DATE November 1998

8. FILE NUMBER	9. TASK NUMBER ADF96/295	10. TASK SPONSOR DGC3ID	11. NO. OF PAGES 8	12. NO. OF REFERENCES 4

13. DOWNGRADING/DELIMITING INSTRUCTIONS	14. RELEASE AUTHORITY Chief, Communications Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

Approved for public release

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE CENTRE, DIS NETWORK OFFICE, DEPT OF DEFENCE, CAMPBELL PARK OFFICES, CANBERRA ACT 2600

16. DELIBERATE ANNOUNCEMENT

No Limitations

17. CASUAL ANNOUNCEMENT Yes

18. DEFTEST DESCRIPTORS

Data synchronisation, Computing tools, Communications

19. ABSTRACT

In a previous report we described, modelled and analysed a protocol "rsync" for synchronising related files at different ends of a communications channel with a minimum of transmitted data. This report identifies the extent of the gains that this method may provide by performing experiments on large repositories of data. The outcome is a new method of compressing large directories of data that together with conventional methods provides major gains in compression factors. As is explained in the report these gains in compression are also indicative of the data transmission gains available when mirroring a collection of files at a remote site using rsync.

Page classification: UNCLASSIFIED