Checksum Testing of Remote Synchronisation Tool

Richard Taylor, Rittwik Jana and Mark Grigg

DSTO
Electronics and Surveillance Research Laboratory

DSTO-TR-0627

ABSTRACT

This report presents testing and improvements to a protocol, rsync, for the synchronisation of similar data files in different locations. When copying a file A at location α to a remote location β it is often the case that A has much in common with some data file B already stored at β. In this situation rsync may be used to effectively send File A from α to β in such a way that much less data than that contained in A is transmitted. Moreover this is achieved without requiring both files to be located at either α or β. This paper provides new checksum functions that offer significant improvements over the existing functions, and proposes that checksum sizes be adaptive based on a verified model of the required checksum size.

RELEASE LIMITATION

Approved for public release

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

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-010-457
March 1998

APPROVED FOR PUBLIC RELEASE

Checksum Testing of Remote Synchronisation Tool

Executive Summary

This report presents testing and improvements to a protocol, rsync, for the synchronisation of similar data files in different locations. Implementation of the recommendations provided here can result in bandwidth savings of at least 50% for large files, as well as giving significant reductions in computer effort, resulting in smaller delays.

When copying a file A at location α to a remote location β it is often the case that A has much in common with some data file B already stored at β. In this situation rsync may be used to effectively send File A from α to β in such a way that much less data than that contained in A is transmitted. Moreover, this is achieved without requiring both files to be located at either α or β.

Rsync may be particularly useful in synchronising databases that have had significant disconnections or outages, resulting in failure of the existing synchronisation scheme. 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 the 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 data efficiencies that rsync offers make it an attractive choice for communications channels of limited bandwidth, which occur in many forms of military communications.

The specific contribution of this report is to provide new checksum functions that offer significant improvements over the existing functions, some of which are shown to have close to ideal properties. We also propose that checksum sizes be adaptive, depending on the file size, and verify a model of the required checksum size.

Authors

Richard Taylor
Communications Division

Richard Taylor is the 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 telcommunications at the Australian National University.

Mark Grigg
Information Technology Division

Mark Grigg has a B.Eng (Electronics) and a B.App.Sci. (Physics) from RMIT, and a PhD in Physics from the University of Melbourne. He joined DSTO in 1994 as a Research Scientist with the Intelligence Systems Group in ITD. His current research interests lie in the areas of digital image coding, signal processing, and the application of Web based technologies to the access and dissemination of information.

Contents

1. INTRODUCTION 1

2. HOW RSYNC WORKS 1

3. ROLLING CHECKSUMS 2

4. CHECKSUM ANALYSIS 5

5. SIMULATIONS 6

6. ADAPTIVE CHECKSUM SIZE 9

7. CONCLUSIONS 10

8. REFERENCES 10

1. Introduction

When copying a File A at location α to a remote location β it is often the case that A has much in common with some data File B already stored at β. In this situation the rsync protocol [5] may be used to effectively send File A from α to β in such a way that much less data than that contained in A is transmitted. Moreover this is achieved without requiring both files to be located at either α or β. This is particularly useful if the communication channel is of limited bandwidth, which occurs in many forms of military communications.

At the heart of the rsync protocol are checksums which are used to uniquely identify blocks of data, and so identify the similarities and differences between A and B. This report analyses the strength of the checksums provided with rsync, and investigates new checksum designs with improved strength. The analysis suggests that the bandwidth requirement of rsync may be considerably reduced without a significant risk of failure due to checksum collisions. The potential use of the rsync algorithm is in synchronising distributed information systems. In particular it may be used to synchronise databases that have had significant disconnections, resulting in failure of the existing synchronising scheme, or Web pages which are regularly being changed at the server and have to be in synchronisation with the clients.

2. How Rsync works

The algorithm may be better understood with reference to Figure1 (see [5] for more details):

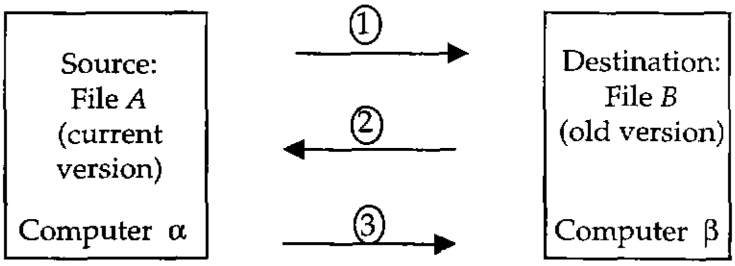

Figure 1 - Information flow during rsync operation.

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 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 that the blocks actually 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 computationally intensive component of the algorithm is Step 3, since checksums are calculated and matches within a table sought for every byte offset in File *A*. In order to make the computations feasible, checksums that are simple to compute, and in particular can be updated quickly at each new offset, are used.

3. Rolling Checksums

Consider a block of bytes X_k ,.., X_l. If it is possible to efficiently calculate the checksum for the block of bytes X_{k+1} ,..,X_{l+1} given the checksum for the buffer X_k ,.., X_l and the values of the bytes X_k and X_{l+1} then we say that the checksum has the *rolling* property. Let *P mod [Q]* denote the member of the residue class *P modulo* Q that lies in the range *0* to *Q–1*. In the original paper [5], a simple rolling checksum *S(k,l)* was used, namely:

$$S(k,l) = T(k,l) + 2^{16}U(k,l) \text{ where}$$

$$T(k,l) = \left(\sum_{i=k}^{l} X_i\right)\text{mod}[2^{16}],\ U(k,l) = \left(\sum_{i=k}^{l}(l-i+1)X_i\right)\text{mod}[2^{16}].$$

The rolling property of *S(k,l)* follows since

$$T(k+1,l+1) = (T(k,l) - X_k + X_{l+1})\text{mod }[2^{16}],$$
$$U(k+1,l+1) = (U(k,l) - (l-k+1)X_k + T(k+1,l+1))\text{mod }[2^{16}].$$

We present two families of 16 bit rolling checksums, each with four functions, *C1, C2, C3, C4*, and *D1, D2, D3, D4*. We analyse their strength, and compare with that of *T* and *U*. A similar analysis of all 6 concatenated pairs of *C1,…,C4* and *D1, D2, D3, D4,* with *S* is performed to see how well the 16 bit checksums combine. The two families of functions are examples of the trade-off between simplicity and speed as against checksum strength. The C family of functions are simpler and quicker to compute, but as we shall show the *D* functions are stronger and combine better.

Define *C1, C2, C3, C4* on the data block X_k, ... X_l with $b=l-k+1$ elements by

$$C1(k,l) = X_l + 2X_{l-1} + 2^2 X_{l-2} + \ldots + 2^{b-1} X_k \bmod[2^{16} - 1].$$

Then

$$\begin{aligned} C1(k+1,l+1) &= X_{l+1} + 2X_l + 2^2 X_{l-1} + \ldots + 2^{b-1} X_{k+1} \bmod[2^{16} - 1] \\ &= 2C1(k,l) + X_{l+1} - 2^b X_{k+1} \bmod[2^{16} - 1]. \end{aligned}$$

Similarly define

$$\begin{aligned} C2(k,l) &= X_l + 8X_{l-1} + 8^2 X_{l-2} + \ldots + 8^{b-1} X_k \bmod[2^{16} - 1], \\ C3(k,l) &= X_l + 32X_{l-1} + 32^2 X_{l-2} + \ldots + 32^{b-1} X_k \bmod[2^{16} - 1], \\ C4(k,l) &= X_l + 128X_{l-1} + 128^2 X_{l-2} + \ldots + 128^{b-1} X_k \bmod[2^{16} - 1]. \end{aligned}$$

Then,

$$\begin{aligned} C2(k+1,l+1) &= 8C2(k,l) + X_{l+1} - 8^b X_{k+1} \bmod[2^{16} - 1], \\ C3(k+1,l+1) &= 32C2(k,l) + X_{l+1} - 32^b X_{k+1} \bmod[2^{16} - 1], \\ C4(k+1,l+1) &= 128C2(k,l) + X_{l+1} - 128^b X_{k+1} \bmod[2^{16} - 1]. \end{aligned}$$

By choosing *b* to be a multiple of 16 we have $8^b \equiv 32^b \equiv 128^b \equiv 1$ mod $[2^{16}-1]$. Multiplication by a power of 2 may be evaluated efficiently using the left shift operation <<. Thus for *b* a multiple of 16,

$$\begin{aligned} C1(k+1,l+1) &= (C1(k,l) \ll 1) + X_{l+1} - X_{k+1} \bmod[2^{16} - 1], \\ C2(k+1,l+1) &= (C2(k,l) \ll 3) + X_{l+1} - X_{k+1} \bmod[2^{16} - 1], \\ C3(k+1,l+1) &= (C3(k,l) \ll 5) + X_{l+1} - X_{k+1} \bmod[2^{16} - 1], \\ C4(k+1,l+1) &= (C4(k,l) \ll 7) + X_{l+1} - X_{k+1} \bmod[2^{16} - 1]. \end{aligned}$$

To evaluate the modulus function efficiently we show how a 32 bit non-negative integer *x* may be evaluated mod $2^{16}-1$ (see [1] and [4] for previous uses of these methods). This can then be used as the basis of evaluating any expression mod $2^{16}-1$. Let *y* and *z* be the top and bottom 16 bits of x, respectively. Thus

$$\begin{aligned} x &= (y * 2^{16} + z), \text{where } 0 \le y, z < 2^{16} - 1 \\ &= (y(2^{16} - 1) + y + z) \\ &\equiv (y + z) \bmod (2^{16} - 1). \end{aligned}$$

To update the *C1* function set,

$$x = (C1(k,l) << 1) + X_{l+1} + 65535 - X_{k+1}.$$

If $x=y*2^{16}+z$ then since $C1 \le 2^{16}-2$ and $X_i \le 2^8-1$ it follows that $0 \le y \le 3$. Thus $0 \le y+z \le 2^{16}$ and so $x \ mod[2^{16}-1]$ is very likely to be simply y+z. Similarly for *C2, C3* and *C4* we have $0 \le y+z \le 2^{16}+6$, $2^{16}+30$, and $2^{16}+126$ respectively.

Define the second family of functions as

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

The corresponding rolling updates may then be evaluated as

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

Multiplication by 3, 5, 7, and 17 can be done with a shift and an add or subtract operation (eg 17x=(x<<4)+x). The powers are fixed for a given *b* and so can be pre-calculated. To evaluate the modulus functions, let *y* and *z* be the top and bottom 16 bits of x, respectively. Thus

$$\begin{aligned} x &= (y * 2^{16} + z), \text{where } 0 \le y, z < 2^{16} - 1. \\ &= (y(2^{16} - i) + iy + z) \\ &\equiv (iy + z) \bmod (2^{16} - i). \end{aligned}$$

Here again, multiplication by *i* may be performed with a shift and add as above. To update *D1,* set

$$\begin{aligned} x &= 3D1(k,l) + X_{l+1} + (2^{16}-1)(2^{8}-1) - 3^{b} X_{k+1} \bmod[2^{16}-1], \\ &= 3D1(k,l) + X_{l+1} + 16711425 - 3^{b} X_{k+1} \bmod[2^{16}-1]. \end{aligned}$$

It follows that $0 \le 3y+z \le 2^{16}+769$. Similarly for *D2, D3*, and *D4* we have $iy+z \le 2^{16}+1293$, $2^{16}+1825$ and $2^{16}+4605$, respectively. Thus $x \; mod[2^{16}-i]$ is likely to be simply $iy+z$, and if not then $x \; mod[2^{16}-i]=iy+z-(2^{16}-i)$.

Thus a single multiplication, together with elementary operations (add, subtract, shift, assign) are used in updating each of the rolling functions.

4. Checksum Analysis

To test the strength of checksum functions we construct a theoretical model for evaluating the probability of checksum collisions (matching checksums corresponding to different data blocks).

In the operation of rsync, once α has received the list of checksums of the blocks of File *B*, it must search File *A* for any blocks at any offset that match the checksum of some block of *B*. The 32 bit rolling checksum for a block of length *b* is computed for each byte offset in File *A*. This is then compared against the table to find any matches. Once a match has been found, α then sends β the corresponding reference to the data in *A*.

Since File *A* and *B* are assumed to be largely similar, and it is in this case that the checksums are more likely to fail we shall assume that *A* and *B* are the same file. Thus we shall examine the ability of the checksums to differentiate between boundary blocks and offset blocks within a given file. Let *Y* be the total number of data bytes in the file, so the number of blocks in the file is *Y/b*. The total number of shifts less those that lie on the block boundaries is *Y–Y/b*. Let the expected number of checksum matches in which the blocks are different(or False Alarms) be *FA*. Let *n* be the number of bits in the checksum. Assuming that the incidence of boundary blocks that match some non-boundary block is small in comparison to False Alarms, and that the checksum has ideal statistical properties in differentiating between different blocks we have,

$$FA = \frac{\left(\frac{Y}{b}\right) \times \left(Y - \frac{Y}{b}\right)}{2^{n}}. \qquad (1)$$

Conversely, if we are comparing the strength of different checksum functions, the number of False Alarms *FA* can be computed for particular files and the effective bit strength *n* of the checksum calculated from the above.

5. Simulations

Checksums were tested with three different files. The first consists of pseudorandom data, the second consists of the data corresponding to a large map, the third is a large tar file generated from a directory of Powerpoint files.

Over the pseudorandom data, the effective bit strengths of all *16* bit checksums are close to ideal (*16* bits), with the exception of function *T*. This may be explained by noting that *T* is a sum of *8* bit integers modulo 2^{16} and so each added integer is likely to change only the lower *8* bits of the checksum. Indeed one would expect that at least *512* bytes need to added together to reach 2^{16}, and so bring the modulus into effect.

There is some variation of checksum strengths over the structured data tests. In general however all the functions perform well with the exception of *T*.

In pairwise combinations there are significant differences in the measured strengths of the 32-bit checksums formed, ranging from 26.8 to 32.1. In general the function *S* is the weakest function, the *C* combinations somewhat stronger and the *D* combinations the strongest of all. In fact the *D* combination checksums have a strength that is consistently close to ideal (32) over the test data (from 31.8 to 32.1).

16 Bit Checksums, Random Data File, Block Size R=400 bytes, 1000 Blocks in File		
Checksums	**False Alarms, FA**	**Effective bit strength**
T	84343	12.2
U	6102	16.0
C1	5935	16.0
C2	6225	16.0
C3	6160	16.0
C4	6106	16.0
D1	6068	16.0
D2	6018	16.0
D3	5955	16.0
D4	6036	16.0

Table 1. Analysis of 16 bit checksum functions for random data

16 Bit Checksums, Map Data File, Block Size R=400 bytes, 1000 Blocks in File		
Checksums	**False Alarms, FA**	**Effective bit strength**
T	32071	13.6
U	6121	16.0
C1	6187	16.0
C2	5925	16.0
C3	6073	16.0
C4	6184	16.0
D1	6223	16.0
D2	6209	16.0
D3	6061	16.0
D4	6108	16.0

Table 2. Analysis of 16 bit checksum functions for structured data (map file)

32 Bit Checksums, Map Data File, Block Size R=400 bytes, 50000 Blocks in File		
Checksums	**False Alarms, FA**	**Effective bit strength**
S=T | U	7007	27.1
C1 | C2	705	30.4
C1 | C3	3608	28.0
C1 | C4	688	30.4
C2 | C3	676	30.4
C2 | C4	3537	28.1
C3 | C4	723	30.4
D1 | D2	236	32.0
D1 | D3	213	32.1
D1 | D4	247	31.9
D2 | D3	238	32.0
D2 | D4	270	31.8
D3 | D4	224	32.1

Table 3. Analysis of pairwise combinations of 16 bit checksum functions for structured data (map file)

<table>
<tr><th colspan="3">16 Bit Checksums, PowerPoint Tar File,
Block Size R=400 bytes, 1000 Blocks in File</th></tr>
<tr><th>Checksums</th><th>False Alarms, FA</th><th>Effective bit strength</th></tr>
<tr><td>T</td><td>27052</td><td>13.9</td></tr>
<tr><td>U</td><td>8072</td><td>15.6</td></tr>
<tr><td>C1</td><td>10098</td><td>15.2</td></tr>
<tr><td>C2</td><td>7133</td><td>15.7</td></tr>
<tr><td>C3</td><td>5388</td><td>16.2</td></tr>
<tr><td>C4</td><td>12069</td><td>15.0</td></tr>
<tr><td>D1</td><td>6547</td><td>15.9</td></tr>
<tr><td>D2</td><td>6712</td><td>15.9</td></tr>
<tr><td>D3</td><td>5817</td><td>16.1</td></tr>
<tr><td>D4</td><td>8999</td><td>15.4</td></tr>
</table>

Table 4. Analysis of pairwise combinations of 16 bit checksum functions for structured data (PowerPoint tar file)

<table>
<tr><th colspan="3">32 Bit Checksums, PowerPoint Tar File,
Block Size R=400 bytes, 50000 Blocks in File</th></tr>
<tr><th>Checksums</th><th>False Alarms, FA</th><th>Effective bit strength</th></tr>
<tr><td>S=T | U</td><td>8752</td><td>26.8</td></tr>
<tr><td>C1 | C2</td><td>2193</td><td>28.8</td></tr>
<tr><td>C1 | C3</td><td>2291</td><td>28.7</td></tr>
<tr><td>C1 | C4</td><td>2152</td><td>28.8</td></tr>
<tr><td>C2 | C3</td><td>2167</td><td>28.8</td></tr>
<tr><td>C2 | C4</td><td>2271</td><td>28.7</td></tr>
<tr><td>C3 | C4</td><td>2167</td><td>28.8</td></tr>
<tr><td>D1 | D2</td><td>209</td><td>32.0</td></tr>
<tr><td>D1 | D3</td><td>211</td><td>32.1</td></tr>
<tr><td>D1 | D4</td><td>248</td><td>31.9</td></tr>
<tr><td>D2 | D3</td><td>210</td><td>32.0</td></tr>
<tr><td>D2 | D4</td><td>249</td><td>31.8</td></tr>
<tr><td>D3 | D4</td><td>225</td><td>32.0</td></tr>
</table>

Table 5. Analysis of pairwise combinations of 16 bit checksum functions for structured data (PowerPoint tar file)

Adapting the design of the checksums given here to take advantage of computers with larger word sizes, it would seem plausible that the functions modified by increasing the moduli should provide correspondingly strong checksums. Thus for 64 bit word size for example, simply replace the moduli $2^{16}, 2^{16}-1, 2^{16}-3, \ldots$ by $2^{32}, 2^{32}-1, 2^{32}-3, \ldots$.

6. Adaptive Checksum Size

From equation (1) we may make predictions about adequate checksum sizes with the use of checksums with consistent properties. This may be used to minimise the size of the table generated in Step 2 of rsync and the corresponding bandwidth requirement, while keeping the probability of a False Alarm relatively low. To support this approach, Step 3 of rsync should also involve the sending of a single strong checksum (say 128 bits) over the entire File *A* (noted in [2]). This will flag β that rsync has failed, and the protocol needs to be repeated with some small file changes.

As the number of False Alarms indicated by (1) increases with the square of the file size, clearly the size of adequate checksums will vary accordingly. Let *p* denote the probability of at least one False Alarm occurring in rsync. Then using (1) and assuming that Files *A* and *B* are approximately the same size *b* we may bound *p* by

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

For small values of the right hand side of (2) both *p* and *FA* approximate the probability of exactly one False Alarm.

Using $Y - Y/b \cong Y$, we may estimate the number of checkbits *CB* (in a checksum with ideal properties) required to meet such *P* as

$$CB \approx 2\log_2(Y) + \log_2(1/bp). \qquad (3)$$

For example if $p=10^{-6}$, $b=1000$, then $CB(Y)$ may be approximated from (3) as *CB*(10^4 *or 10 Kbyte)=36.5, CB*(10^6 *or 1 Mbyte)=49.8, CB*(10^8 *or 100 Mbytes)=63.1, CB*(10^{10} *or 10 Gbytes)=76.4.*

Thus the length of checksums required to provide a given level of confidence varies significantly with the file size. We therefore suggest that the checksum sizes be dynamically chosen for each rsync transaction based on equation (3). This will significantly reduce the data transmitted in Step 2 of the rsync protocol.

Another important factor in the computational efficiency of rsync is the use of non-rolling checksums (such as MD5) as a backup check in Step 3, since each such backup check requires MD5 to be calculated from scratch over the entire block. This is important for large files since the incidence of 32 bit mismatches rises with the size of the file. For this reason stronger rolling checksums will also reduce the computational effort in Step 3 of rsync, especially for large files.

7. Conclusions

A methodology is given with which the checksums provided with the rsync protocol have been tested and compared to new checksum functions given in this report. The new checksums are shown to be stronger than those used in rsync, moreover a particular family of new checksums has a strength that is consistently close to ideal over the test data. These checksums may be updated for each byte offset with just one multiplication and elementary operations (add, subtract, assign, and shift). Based on these new checksums we suggest that rsync should dynamically determine checksum sizes depending on the file size. This will significantly reduce the bandwidth requirement of the rsync protocol while controlling the chance that the protocol will fail because of checksum collisions (and need to be repeated).

8. References

[1] H. J. Knobloch, "A Smartcard Implementation of the Fiat-Shamir Identification Scheme. Advances in Cryptology- EUROCRYPT'88, Proceedings, Springer-Verlag (1989) pp. 87-96.

[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] R. Taylor, "An Integrity Check Value Algorithm for Stream Ciphers", Advances in Cryptology - CRYPTO' 93, Proceedings, *Lecture Notes in Computer Science* 773, Springer-Verlag 1994, pp. 40-48.

[5] Tridgell, A. and Mackerras P, "The rsync algorithm", Joint Computer Science Technical Report Series, TR-CS-96-05, Department of Computer Science, The Australian National University, June 1996.

[6] Tridgell, A. and Mackerras P, Private communication.

DISTRIBUTION LIST

Checksum Testing of Remote Synchronisation Tool

Richard Taylor, Rittwik Jana and Mark Grigg

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, M. Grigg

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 AUSTACSS

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 Checksum Testing of Remote Synchronisation Tool	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, Rittwik Jana and Mark Grigg	5. CORPORATE AUTHOR Electronics and Surveillance Research Laboratory PO Box 1500 Salisbury SA 5108

6a. DSTO NUMBER DSTO-TR-0627	6b. AR NUMBER AR-010-457	6c. TYPE OF REPORT Technical Report	7. DOCUMENT DATE March 1998

8. FILE NUMBER	9. TASK NUMBER ADF96/295	10. TASK SPONSOR DGC3ID	11. NO. OF PAGES 12	12. NO. OF REFERENCES 6

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

This report presents testing and improvements to a protocol, rsync, for the synchronisation of similar data files in different locations. When copying a file A at location α to a remote location β it is often the case that A has much in common with some data file B already stored at β. In this situation rsync may be used to effectively send File A from α to β in such a way that much less data than that contained in A is transmitted. Moreover this is achieved without requiring both files to be located at either α or β. This paper provides new checksum functions that offer significant improvements over the existing functions, and proposes that checksum sizes be adaptive based on a verified model of the required checksum size.

Page classification: UNCLASSIFIED