When File Synchronization Meets Number Theory

Antoine Amarilli, Florian Bourse, Robin Morisset,
David Naccache, and Pablo Rauzy

École normale supérieure, Département d'informatique
45, rue d'Ulm, F-75230, Paris Cedex 05, France.
surname.name@ens.fr

Abstract. In this work we [to be completed by David]

1 Introduction

In this work we [to be completed by David]

2 A Few Notations

We model the directory synchronization problem as follows: Oscar possesses an old version of a directory $\mathfrak{D}$ that he wishes to update. Neil has the up-to-date version $\mathfrak{D}'$. The challenge faced by Oscar and Neil[1] is that of *exchanging as little data as possible* during the synchronization process. Note that in reality $\mathfrak{D}$ and $\mathfrak{D}'$ may differ both in their files and in their tree structure.

In tackling this problem we separate the "*what*" from the "*where*": namely, we disregard the relative position of files in subdirectories. Let $\mathfrak{F}$ and $\mathfrak{F}'$ denote the multisets of files contained in $\mathfrak{D}$ and $\mathfrak{D}'$. We denote $\mathfrak{F} = \{F_0, \ldots, F_n\}$ and $\mathfrak{F}' = \{F'_0, \ldots, F'_{n'}\}$.

Let `Hash` denote a collision-resistant hash function[2] and let F be a file. Let $\texttt{NextPrime}(F)$ be the prime immediately larger than $\texttt{Hash}(F)$ and let u denote the size of `NextPrime`'s output in bits. Define the shorthand notations: $h_i = \texttt{NextPrime}(F_i)$ and $h'_i = \texttt{NextPrime}(F'_i)$.

3 The Contents Synchronization Protocol

To efficiently synchronize directories, we propose a new protocol based on modular arithmetic. In terms of asymptotic complexity, the proposed procedure is comparable to prior publications [] (that anyhow reached optimality) but its interest lies in its simplicity and novelty.

3.1 Description of the Basic Exchanges

Let t be the number of discrepancies between $\mathfrak{F}$ and $\mathfrak{F}'$ that we wish to spot.*i.e.*:

$$t = \#\mathfrak{F} + \#\mathfrak{F}' - 2\#(\mathfrak{F} \bigcap \mathfrak{F}')$$

We generate a prime p such that:

[1] Oscar and Neil will respectively stand for *o̲ld* and *n̲ew*.
[2] *e.g.* SHA-1

$$2^{2tu+1} \leq p < 2^{2tu+2} \tag{1}$$

Given $\mathfrak{F}$, Neil generates and sends to Oscar the redundancy :

$$c = \prod_{i=1}^{n} h_i \bmod p \tag{2}$$

Oscar computes:

$$c' = \prod_{i=1}^{n} h'_i \bmod p \quad \text{and} \quad s = \frac{c'}{c} \bmod p$$

Using [?] the integer s can be written as :

$$s = \frac{a}{b} \bmod p \quad \text{where the } G_i \text{ denote files and } \begin{cases} a = \displaystyle\prod_{G_i \in \mathfrak{F}' \wedge G_i \notin \mathfrak{F}} \texttt{NextPrime}(G_i) \\ b = \displaystyle\prod_{G_i \notin \mathfrak{F}' \wedge G_i \in \mathfrak{F}} \texttt{NextPrime}(G_i) \end{cases}$$

Note that since $\mathfrak{F}$ and $\mathfrak{F}'$ differ by at most t elements, we have that a and b are strictly lesser than 2^{ut}. Theorem **??** (see [?]) shows that given s one can recover a and b efficiently. The algorithm is based on Gauss' algorithm for finding the shortest vector in a two-dimensional lattice [?].

Theorem 1. *Let $a, b \in \mathbb{Z}$ such that $-A \leq a \leq A$ and $0 < b \leq B$. Let p be some prime integer such that $2AB < p$. Let $s = ab^{-1} \mod p$. Then given A, B, s and p, one can recover a and b in polynomial time.*

Taking $A = B = 2^{ut} - 1$, we have from (**??**) that $2AB < p$. Moreover, $0 \leq a \leq A$ and $0 < b \leq B$. Therefore, we can recover a and b from s in polynomial time. By testing the divisibility of a and b by the h_i and the h'_i, Neil and Oscar can easily identify the discrepancies between $\mathfrak{F}$ and $\mathfrak{F}'$.

Formally, this is done as follows:

Oscar		**Neil**
	$\xrightarrow{c}$	
		computes a, b
		if a does not factor properly
		then output $\perp$ and halt
		else let $\mathfrak{S} = \{F'_i \text{ s.t. } a \bmod h'_i = 0\}$
	$\xleftarrow{\mathfrak{S}, b}$	
deletes files s.t. $b \bmod h_i = 0$		
adds $\mathfrak{S}$ to the disk		

As we have just seen the "output $\perp$ and halt" should actually never occur if bounds on parameter sizes are respected. However, a file synchronization procedure that works *only* for a limited number of differences is not of major practical usefulness. In the next subsection we will show how to extend the protocol even in the case where the differences exceed the informational capacity of the modulus p used.

3.2 The Case of Insufficient Information

To extend the protocol to an unlimited number of differences, Oscar and Neil will use more than one p by agreeing on an infinite set of primes $p_1, p_2, \ldots$ As long as the protocol fails, Neil will keep amassing information about the difference as shown in the appendix. Note that no information is lost and that information adds-up until it reaches a threshold that suffices to identify the difference.

4 Variants

In this section we explore two strategies allowing to reduce the size of p and hence improve transmission by *constant factors* (from a complexity standpoint, nothing can be done as the protocol transmits information proportional in size to the difference).

4.1 Probabilistic Decoding: Reducing p

Generate a prime p smaller than previously, namely:

$$2^{ut+w-1} < p \leq 2^{ut+w} \tag{3}$$

for some small integer $w \geq 1$ (we suggest to take $w = 50$). For large $\eta = \max(n, n')$ and t the size of the new prime p will be approximately half the size of the prime p generated in the previous section. The resulting redundancy c is calculated as previously but approximately twice smaller. As previously, we have :

$$s = \frac{a}{b} \bmod p \quad \text{and} \quad \begin{cases} a = \displaystyle\prod_{G_i \in \mathfrak{F}' \wedge G_i \notin \mathfrak{F}} \texttt{NextPrime}(G_i) \\ b = \displaystyle\prod_{G_i \notin \mathfrak{F}' \wedge G_i \in \mathfrak{F}} \texttt{NextPrime}(G_i) \end{cases}$$

and since there are at most t differences, we must have :

$$ab \leq 2^{ut} \tag{4}$$

The difference with respect to the basic protocol is that we don't have a fixed bound for a and b anymore; equation (**??**) only provides a bound for the product ab. Therefore, we define a finite sequence of integers:

$$(A_i = 2^{w \cdot i}, B_i = \lfloor (p-1)/(2 \cdot A_i) \rfloor) \quad \text{where} \quad B_i > 1$$

.

$\forall i > 0$ we have $2A_iB_i < p$. Moreover, From equations (**??**) and (**??**) there must be at least one index i such that $0 \leq a \leq A_i$ and $0 < b \leq B_i$. Then using Theorem **??**, given (A_i, B_i, p, s) one can recover a and b, and eventually the difference.

The problem is that (by opposition to the basic protocol) we have no guarantee that such an (a, b) is unique. Namely we could in theory stumble upon another (a', b') satisfying (**??**) for some index $i' \neq i$. We expect this to happen with negligible probability for large enough w, but this makes the modified protocol heuristic only.

5 File Laundry: Reducing u

What happens if we shorten u in the basic protocol?

BEGIN OLD

As foreseen by the birthday paradox, we should start seeing collisions. As a direct function of our two parameters (η, u), we would roughly expect:

$$E(\eta, u) = \frac{\binom{\eta}{2}}{2^u} = \frac{\eta(\eta-1)}{2^{u+1}} \text{ collisions}$$

For instance, for $\eta = 10^6$ files and 32-bit hash values, one should expect about 116 collisions.

That being said, a collision can only yield a *false positive* and never a *false negative*. In other words, whilst a collision may make the parties blind to a difference[3] a collision will never create an nonexistent difference *ex nihilo*.

Hence, it suffices to replace the function $\texttt{Hash}(F)$ by a chopped $\texttt{MAC}_k(F) \bmod 2^u$ to quickly filter-out file differences by repeating the protocol for $k = 1, 2, \ldots$ At each round the parties will detect new files and deletions, fix these and "launder" again the remaining files.

Indeed, the probability that a stubborn file persists colliding decreases exponentially with the number of iterations. Assuming the η remains invariant between rounds, the probability that a given file will successfully keep playing hide-and-seek after at the second iteration is:

$$\frac{E(\eta, u)}{\eta} = \frac{\eta - 1}{2^{u+1}}$$

Hence the probability that after the second round at least one false positive will survive is:

$$z = 1 - \left(1 - \frac{E(\eta, u)}{\eta}\right)^{E(\eta, u)}$$

For the $(\eta = 10^6, u = 32)$ instance considered previously, gives $z \simeq 1.34\%$.

END OLD

As foreseen by the birthday paradox, we should start seeing collisions. Let us analyse them.

We see the hash function $\texttt{Hash}$ as a random function from $\{0,1\}^*$ to $\{0, \ldots, 2^u - 1\}$. Let X_i^1 be the random variable equal to 1 when the file number i has a collision with another file, and equal to 0 otherwise. Clearly, we have $\Pr[X_i = 1] \leq \frac{\eta - 1}{2^u}$. And the number of files which collides are, in average:

$$\mathbb{E}\left[\sum_{i=1}^{\eta} X_i\right] \leq \sum_{i=1}^{\eta} \frac{\eta - 1}{2^u} = \frac{\eta(\eta-1)}{2^u}.$$

[3] *e.g.* result in confusing `index.htm` and `iexplore.exe`.

For instance, for $\eta = 10^6$ files and 32-bit hash values, one should expect about at most 233 files which collide.

That being said, a collision can only yield a *false positive* and never a *false negative*. In other words, whilst a collision may make the parties blind to a difference[4] a collision will never create an nonexistent difference *ex nihilo*.

Hence, it suffices to replace the function $\texttt{Hash}(F)$ by a chopped $\texttt{MAC}_k(F) \bmod 2^u$ to quickly filter-out file differences by repeating the protocol for $k = 1, 2, \ldots$ At each round the parties will detect new files and deletions, fix these and "launder" again the remaining files.

Indeed, the probability that a stubborn file persists colliding decreases exponentially with the number k of iterations, if the hash functions are random and independent for each iteration. Assume the η remains invariant between iterations. Let Y_i be the random variable equal to 1 when the file number i has a collision with another file for all the k iterations, and equal to 0 otherwise. Let X_i^l be the random variable equal to 1 when the file number i has a collision with another file during iteration l, and equal to 0 otherwise. By independence, we have

$$\Pr\left[\, Y_i = 1 \,\right] = \Pr\left[\, X_i^1 = 1 \,\right] \ldots \Pr\left[\, X_i^k = 1 \,\right] \leq \left(\frac{\eta - 1}{2^u}\right)^k .$$

Therefore the number of files which collides is, in average

$$\mathbb{E}\left[\sum_{i=1}^{\eta} Y_i\right] \leq \sum_{i=1}^{\eta} \left(\frac{\eta-1}{2^u}\right)^k = \eta \left(\frac{\eta-1}{2^u}\right)^k .$$

Hence the probability that after k rounds at least one false positive will survive is

$$\epsilon_k \leq \eta \left(\frac{\eta-1}{2^u}\right)^k$$

For the $(\eta = 10^6, u = 32)$ instance considered previously, this gives $\epsilon_2 \leq 5.43\%$ and $\epsilon_3 \leq 2 \cdot 10^{-3}\%$.

THIS section does not work but maybe only a small thing is missing.

However, I HOPE WE CAN (one can) improve a lot the algorithm, using the following trick: we can remove the files which are different in the first possible iteration, and only work with common files and files which collided (in a bad way, blinding a difference). Now, the only collision which can be bad for round k, are the collisions of a file i with a file j such that i and j both have collided at all the previous iterations. And let write Z_i^l the random variable equal to 1 when the file i has a bad collisions for all the l first iterations.

We have, $Z_i^1 = X_i^1$, and for $l \geq 2$, we have

$$\begin{aligned} \Pr\left[\, Z_i^l = 1 \,\right] &= \Pr\left[\, \exists j \neq i,\ X_{i,j}^l = 1,\ Z_i^{l-1} = 1 \text{ and } Z_j^{l-1} = 1 \,\right] \\ &\leq \sum_{j=1, j\neq i}^{\eta} \Pr\left[\, X_{i,j}^1 = 1 \,\right] \Pr\left[\, Z_i^{l-1} = 1 \text{ and } Z_j^{l-1} = 1 \,\right] \end{aligned}$$

[4] *e.g.* result in confusing `index.htm` and `iexplore.exe`.

I HAVE NOT FOUND A WAY TO FINISH THIS, even if I think, we can win a bit. The problem is that the Z... are not independent at all (imagine $\eta = 2$ to understand why). If we use $\Pr\left[Z_i^{l-1} = 1 \text{ and } Z_j^{l-1} = 1\right] \leq \Pr\left[Z_i^{l-1} = 1\right]$, we get the previous bound unfortunately...

6 Implementation

6.1 Program Structure

6.2 Time Measurements

7 Conclusion and Further Improvements

In this work we [to be completed by David]

8 Acknowledgment

The authors acknowledge Guillain Potron for his early involvement in this project.

References

1. A. Tridgell, Efficient Algorithms for Sorting and Synchronization, Ph.D. Thesis, The Australian National University, February 1999.
2. D. Wagner, Edit Lenses, POPL'12, [to be completed].
3. J. Stern, P.-A. Fouque and G.-J. Wackers, *CryptoComputing with Rationals*. Proceedings of Financial Cryptography 2002, Lecture Notes in Computer Science (2002), Springer-Verlag.
4. B. Vallée, *Gauss' algorithm revisited*. J. Algorithms, 12:556-572, 1991.

A Extended Protocol

First phase during which Neil amasses modular information on the difference		
Oscar		**Neil**
		start protocol with p_1
	$\xrightarrow{c_1}$	
		computes a, b using p_1
		if a factors properly then Terminate Phase
		else switch to p_2
	$\xrightarrow{c_2}$	
		computes $c \bmod p_1 p_2 = \mathrm{CRT}_{p_1,p_2}(c_1, c_2)$
		computes a, b using $p_1 p_2$
		if a factors properly then Terminate Phase
		else switch to p_3
	$\xrightarrow{c_3}$	
		computes $c \bmod p_1 p_2 p_3 = \mathrm{CRT}_{p_1,p_2,p_3}(c_1, c_2, c_3)$
		computes a, b using $p_1 p_2 p_3$
		if a factors properly then Terminate Phase
		else switch to p_4 ...
	$\vdots$	

Terminate Phase		
		Let $\mathfrak{S} = \{F'_i \text{ s.t. } a \bmod h'_i = 0\}$
	$\xleftarrow{\mathfrak{S},b}$	
deletes files s.t. $b \bmod h_i = 0$		
adds $\mathfrak{S}$ to the disk		

Note that the parties do not need to store the p_i's in full. Indeed, the bits of each p_i could be generated using a pseudo-random number generator and a small corrected additive constant of an average value of $\ln(p_i) \cong \ln(2^{2tu+2}) \cong 1.39(tu+1)$ which storage requires essentially $\log_2(tu)$ bits.