Projet réseau: synchronisation de fichier

Matthieu Boutier*, Fabien de Montgolfier†

Introduction

Le but de ce projet est d'implémenter un synchroniseur de fichiers. Le programme comporte trois parties :

— établir la liste des fichiers à synchroniser de part et d'autre ;
— la synchronisation locale ;
— la synchronisation à distance.

1 Description du programme minimal

Votre projet consistera de deux programmes : un serveur, qui ne servira qu'à monitorer, et un client qui sera chargé d'interfacer avec l'utilisateur.

Le serveur Votre serveur prendra en argument le répertoire à synchroniser et le numéro de port sur lequel il doit se lancer. Il ne fera qu'attendre les requêtes du client : son rôle est essentiellement passif dans la synchronisation.

Le client Le client aura plusieurs modes de fonctionnement. Il prendra en arguments les deux répertoires à synchroniser. Lorsque les deux répertoires sont locaux, il prendra exactement ces deux répertoires, mais lorsqu'un des deux répertoires est distant, le programme prendra comme premier argument le répertoire local, puis l'adresse IP du serveur, suivi du numéro de port du serveur.

Un exemple Pour synchroniser deux répertoires locaux, le lancement d'aucun serveur n'est requis, et il suffira d'exécuter une commande :

```
> sync-client dir1 dir2
```

Pour synchroniser deux répertoires distant, il est nécessaire de lancer le serveur sur la machine distante, puis d'exécuter le client en local. On peut bient sûr tester tout ceci en local, en lançant le serveur et le client sur la même machine :

```
> sync-server dir2 4210 &
> sync-client dir1 127.0.0.1 4210
```

*boutier@pps.univ-paris-diderot.fr
†fm@liafa.univ-paris-diderot.fr

1.1 Dry run

Il vous est demandé d'implémenter, côté client, l'option `-d` (ou `-dry-run`), qui affichera l'ensemble des modifications que le synchroniseur va faire, sans toutefois les faire. Cela vous sera utile pour déboguer. Il y a cinq opérations à prévoir :

— import d'un fichier,
— export d'un fichier,
— (conflit,)
— (suppression d'un fichier local,)
— (suppression d'un fichier distant.)

1.2 Suppression des fichiers et conflits

Lorsque votre programme détectera qu'un fichier existe d'un côté, et non de l'autre, il n'aura pas de moyen de savoir s'il s'agit d'un fichier supprimé d'un côté, ou d'un fichier créé de l'autre. Il n'est pas demandé de gérer la suppression dans le projet minimal : votre programme considèrera que le fichier est manquant. Des détails sont donnés dans la partie extension pour savoir comment gérer la suppression des fichiers.

De même, un conflit apparaît lorsque le même fichier est modifié des deux côtés. Dans le projet de base, il n'est pas demandé de gérer les conflits : vous ne garderez que le plus récent des deux fichiers. Référez-vous à la partie sur les extensions pour plus de détails.

2 Communication, et format de paquets

La communication entre le client et le serveur se fait en TCP. Vous êtes libres d'établir une ou plusieurs connexions entre le client et le serveur.

Les messages echangés sont formatés par les TLVs (Type-Length-Value) définis ci-après. Remarquez que les racines des répertoires à synchroniser sont donnés aux programmes à leur lancement. Les noms de fichiers échangés par le protocole sont donc des chemins relatifs à chaque arborescence.

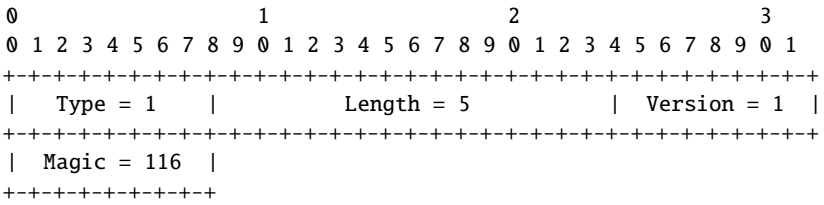
2.1 Format général des TLVs

Les TLVs de notre protocole sont constitués d'un premier champ Type de 8 bits, suivi du champ Length sur 16 bits, représentant la taille totale du TLV (y inclus les 3 octets des deux champs Type et Length).

Certains TLV contiennent des champs `Reserved` : ils seront ignorés à la lecture, et mis à 0 en écriture. Ils pourront être utilisés pour des utilisations ultérieures, mais sont là pour que la représentation soit plus jolie (Historiquement, les protocoles avaient une réelle raison d'avoir des champs `Reserved` : ils servaient à ce que les champs soient alignés sur un multiple de 4 octets.)

2.2 En-tête de connexion

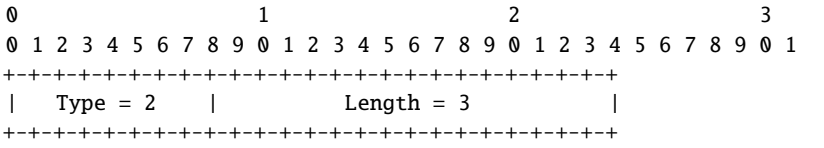
Toute nouvelle connexion au serveur commencera par ce TLV. Une connexion ne commençant pas par ce TLV sera fermée. À la fois le client et le serveur écriront ce TLV à l'établissement de la connexion (et l'attendront comme premier TLV).



— Version : représente la version du protocole. Ici, il s'agira de la version 1.
— Magic : vaut 116.

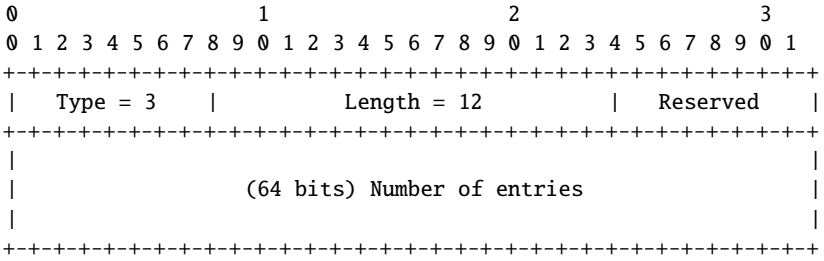
2.3 Demande la liste des fichiers

Ce TLV est utilisé par le client pour demander au serveur la liste de ses fichiers. Le serveur, à la réception de ce TLV, envoie la liste de ses fichiers, à l'aide des TLV décrits en sections 2.4 et 2.5.



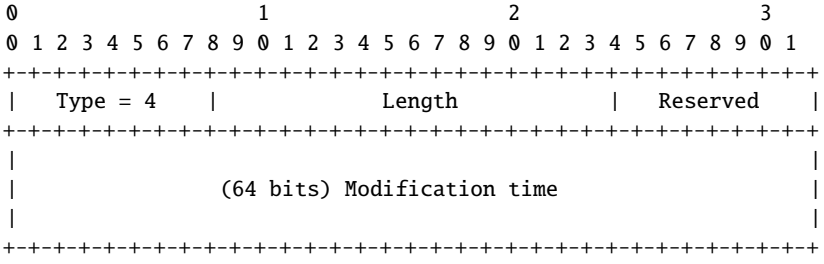
2.4 En-tête d'envoi de liste des fichiers

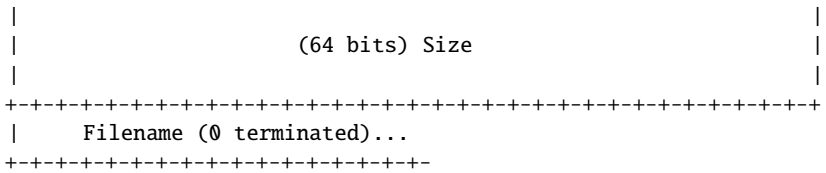
Avant d'envoyer la liste de ses fichiers, avec le TLV défini en section 2.5, le serveur envoie le nombre d'entrées qu'il va envoyer. Le client saura donc qu'il a la liste complète des fichiers du serveur après avoir lu ce nombre d'entrées.



2.5 Entrée de liste de fichier

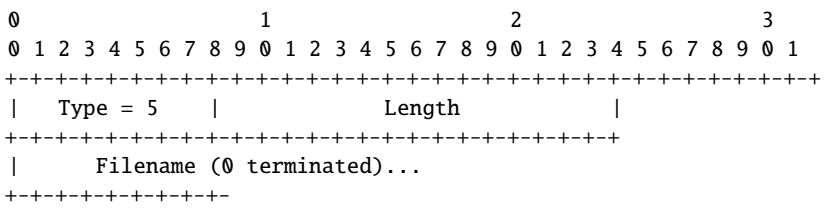
Il est utilisé par le serveur après le TLV défini en section 2.4 pour donner la liste de ses fichiers.





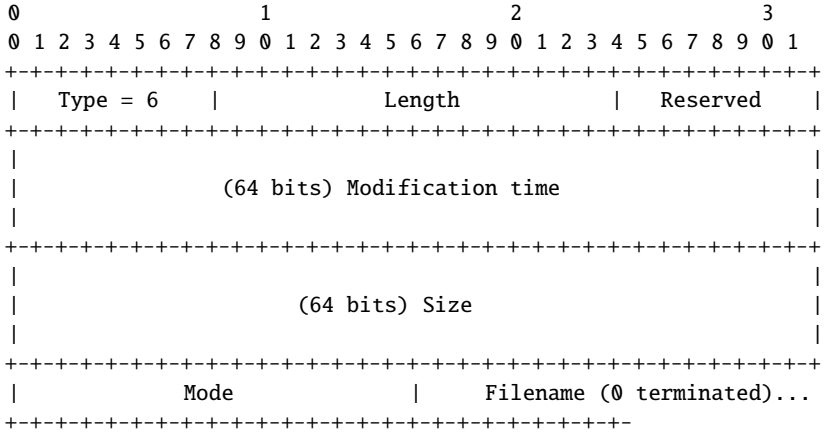
2.6 Demande d'un fichier

Ce TLV est utilisé par le client pour lui demander le contenu d'un fichier. À la réception de ce TLV, le serveur doit envoyer au client le fichier correspondant, à l'aide du TLV décrit en section 2.7.



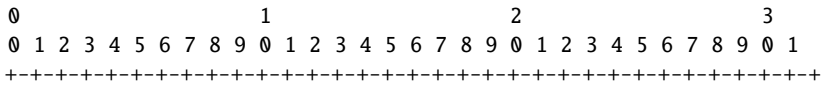
2.7 En-tête d'envoi de fichier

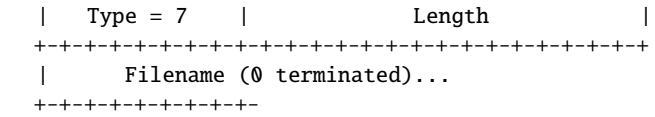
Ce TLV précède les données binaires d'un fichier. Il contient ses méta-informations, et est immédiatement suivi des `Size` octets du fichier. Le client ou le serveur qui reçoit ce TLV doit enregistrer le fichier, et mettre-à-jour le contenu du fichier, ses droits, etc. Remarquez que les données du fichier ne sont pas incluses dans le TLV (le champ Length serait trop petit).



2.8 Demande de suppression d'un fichier

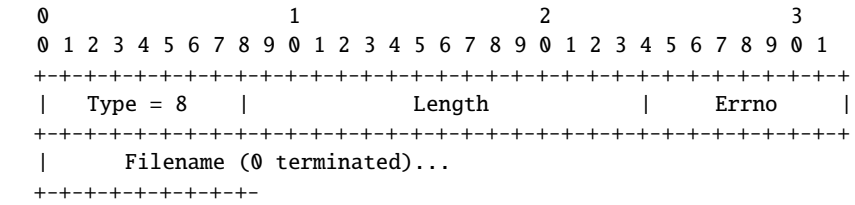
Ce TLV est utilisé par le client pour demander au serveur de supprimer un fichier. À la réception de ce TLV, le serveur doit supprimer le fichier en question.





2.9 Notification d'erreur

Ce TLV est utilisé par le serveur pour avertir le client d'une erreur. Le serveur envoie un tel TLV dès qu'une erreur se produit. Un tel TLV est facultatif : le client peut détecter les erreurs en redemandant la liste des fichiers du serveur. Cependant, il est pratique de pouvoir notifier le client d'une erreur dès qu'elle se produit, et surtout sans avoir à lister tous les fichiers.



Les erreurs possibles sont :
— 0 : aucune erreur, l'action a été réalisée avec succès ;
— 1 : erreur générique (erreur quand au traitement de ce fichier) ;
— 2 : Le fichier côté serveur a changé.

3 Extensions

Vous êtes libres d'étendre à votre guise le projet, y compris en modifiant le protocole, ajoutant des TLV ou sous-TLV, etc. Les extensions les plus basiques, car les plus utiles, constituent la détection des fichiers supprimés, et la propagation de cette suppression, ainsi que la gestion des conflits. Toute autre extension sera la bienvenue. Dans la mesure du possible, les modifications au protocole devront s'efforcer de garder la compatibilité.

3.1 Suppression de fichiers et conflits

Afin de déterminer quels sont les changements effectifs de part et d'autres depuis la dernière sauvegarde, le client pourra sauvegarder dans un fichier l'état de l'arbre local et distant à la dernière modification. Le client pourra alors, pour chaque fichier et de chaque côté, déterminé s'il est inchangé, modifié ou supprimé depuis la dernière synchronisation.

Lorsque les deux fichiers ont été modifiés de part et d'autre, il s'agit d'un conflit. Il n'y a pas de bonne solution à un conflit : il convient de notifier l'utilisateur qu'une synchronisation automatique risque de corrompre les données, et au choix de lui demander de faire la synchronisation manuellement, ou de lui demander quel fichier garder.

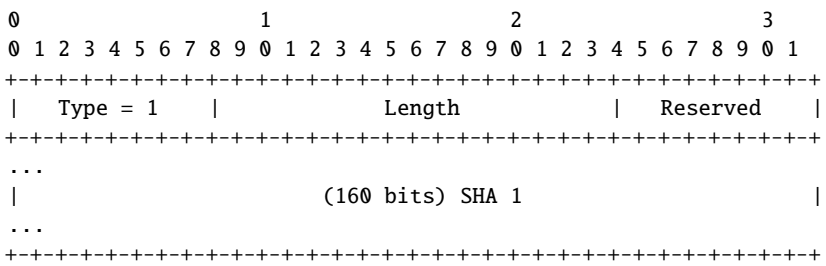
3.2 Intégrité des données

Détecter un changement de contenu de fichier par sa seule date de dernière modification n'est pas une approche satisfaisante : si généralement on peut considérer qu'un

fichier dont la date n'a pas été modifiée est resté le même, en revanche il arrive qu'un fichier se soit vu changé de date sans avoir modifié son contenu. Une solution consiste à *hasher* les fichiers, et de comparer leur hash. On pourra par exemple utiliser SHA1 comme fonction de hashage.

Afin de ne pas hasher systématiquement tous les fichiers, on pourra se restreindre à ceux qui ont vu leur date de dernière modification changer.

Le hash d'un fichier pourra être stocké dans un sous-TLV au TLV défini en section 2.4. Un sous-TLV est un TLV inclus dans un autre TLV : la taille du TLV englobant inclus la taille de ses sous-TLV.



3.3 Fichier de configuration

On pourra décider d'un fichier de configuration qui permet de configurer la synchronisation. Avoir un, ou plusieurs fichiers de configuration possible pourrait aussi permettre de synchroniser plusieurs répertoires, de filtrer les entrées que l'on veut synchroniser, etc.

3.4 Notifications de changements

Afin d'automatiser la synchronisation, lorsque cela est possible, on pourra utiliser les API de détection de changement de l'état des fichiers sur disque. Il s'agit de `inotify` sur Linux [1].

3.5 Sécurité

Notre protocole, en l'état, est totalement non-sécurisé. N'importe qui peut récupérer ou supprimer nos données, saturer le disque dur avec de faux fichiers ou s'en servir comme espace de stockage... L'ensemble de la connexion pourrait-être cryptée et signée.

4 Modalités de soumissions

Le projet sera individuel. Votre solution devra consister d'un programme écrit en C et utilisable sous Linux [2]. Elle consistera de :

— un fichier texte nommé `README` contenant vos noms et indiquant brièvement comment compiler et utiliser votre programme ;

1. Sur MacOS, il faut regarder du côté des `FSEvents`. Mais n'oubliez pas que votre programme est censé être testé sur Linux...
2. Ou deux programmes : un pour le serveur et un pour le client.

— un rapport sous format PDF de 2 à 12 pages environ décrivant ce que vous avez fait, quelles extensions vous avez traitées, et expliquant (ou justifiant) les choix de conception ou d'implémentation que nous pourrions ne pas comprendre du premier coup ;
— tout autre fichier nécessaire à la compilation et à l'exécution, par exemple un fichier `Makefile`, un script permettant de compiler vos sources, des fichiers d'exemples, un fichier man, etc. Votre soumission devra consister d'une seule archive compressée `tar.gz`. L'archive devra obligatoirement s'appeler `nom.tar.gz`, et s'extraire dans un répertoire `nom`, où `nom` est votre `nom`. Par exemple, si vous vous appelez Linus Torvald, votre archive devra s'appeler `torvald.tar.gz` et s'extraire dans un répertoire `torvald`. La date limite et le mode de soumission seront indiqués sur Didel.