Nicolas BOUGET
✉ nicolas.bouget@esial.net
2A TRS1

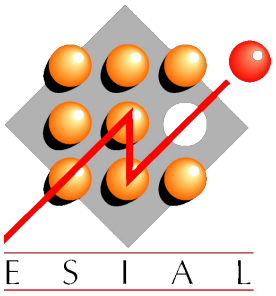
Julien VAUBOURG
✉ julien@vaubourg.com
2A TRS2

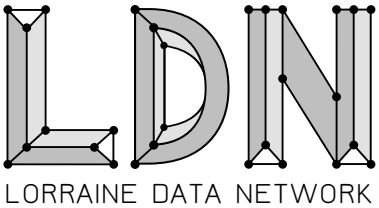
Sauvegardes de données pair-à-pair sur les boîtiers d'un FAI associatif

Rapport

Projet interdisciplinaire
encadré par Martin QUINSON

Le 16 juin 2012





Remerciements

– **Martin QUINSON**, pour avoir proposé ce sujet et nous avoir suivi

– **Jean-Marie MOUREAUX**, pour avoir accepté ce projet

– **Ludovic COURTÈS**, pour son excellent travail sur libchop et sa sympathie

– **Julien AUBÉ**, pour sa disponiblité et son généreux don de Neufbox

– **Gérald OSTER**, pour ses conseils

– **Isabelle COURBOT**, pour nous avoir laissé le temps de finir ce rapport

– **Lucas NUSSBAUM**, pour nous avoir fait découvrir libchop

– **Les membres de FFDN**, pour leur soutien

Table des matières

1 Introduction

Lorraine Data Network [1] (LDN) est une association de type loi 1901, qui défend *un Internet libre, neutre et décentralisé.* Outre ses activités de sensibilisation au respect de la vie privée [2], et à l'importance d'avoir un réseau mondial neutre et acentré, elle est enregistrée à l'ARCEP [3] et dispose donc de la possibilité de délivrer des accès à Internet publics. Elle assume son rôle de fournisseur d'accès à Internet (FAI) en permettant à ses abonnés d'être des adhérents actifs plutôt que des clients numérotés, et tente de proposer des solutions pour s'auto-héberger facilement derrière son accès participatif.

C'est sur ce dernier point que notre travail s'est consacré durant ce projet. L'auto-hébergement consiste à revenir aux fondamentaux du réseau Internet : chaque personne qui est reliée au réseau devient un nœud de celui-ci, et participe à son expansion en l'élargissant. Depuis quelques années, et notamment avec le développement des services des deux mastodontes de la Toile que sont Google et Facebook, l'utilisation d'Internet s'est recentrée en accentuant la dichotomie entre le client (l'abonné qui ne se connecte que pour utiliser les services) et le serveur (celui qui fournit le service, qui devient de plus en plus gros et incontournable). Les nombreux problèmes éthiques et techniques sous-jacents - que nous ne détaillerons pas dans ce rapport - posés par le passage obligatoire et systématique de nos données sur les serveurs de quelques entreprises, encouragent des associations comme LDN à aider le plus grand nombre de personnes à prendre de l'indépendance en hébergeant chez elles leur vie électronique [4].

L'auto-hébergement consiste donc à disposer d'une machine allumée chez soit, capable de répondre en permanence aux éventuelles requêtes externes.

Trois problèmes se font identifier rapidement :

1. Les compétences techniques de l'internaute moyen trop faibles
2. Le coût écologique de la consommation d'une machine en permanence sous tension
3. La perte de ses données en cas d'incendie, vol, panne matériel, etc.

En couplant ses activités de FAI à ses missions de sensibilisation, LDN dispose d'une solution qui répond à deux des trois problèmes posés : fournir un boîtier de connexion ADSL (*box*) configuré, qui permette l'hébergement à domicile de son courrier électronique ainsi qu'une page web personnelle. La solution d'auto-hébergement minimal est livrée clé en main, et il n'y a pas de consommation électrique supplémentaire [5] en utilisant une machine que tout le monde laisse déjà sous tension, sans interruption.

Le dernier des trois problèmes évoqués (perte des données) est encore à résoudre, et c'est précisément l'objectif de notre projet interdisciplinaire.

Nous avons eu plusieurs fois l'occasion d'assister à la réunion mensuelle de la fédération des FAI participatifs French Data Network [6] (FFDN) de laquelle LDN est membre fondateur, et qui rassemble une dizaine de FAI partout en France. Nous avons reçu un accueil chaleureux de la part des autres FAI, qui nous ont dors et déjà indiqués être intéressés pour exploiter notre solution une fois qu'elle sera fonctionnelle.

1. http://ldn-fai.net
2. Cf. par exemple http://ldn-fai.net/med/images/3/39/Sexealcoolvieprivee_avec_notes.pdf
3. *L'Autorité de Régulation des Com. Électroniques et des Postes*, qui définie entre autres les règles du marché Internet en France.
4. Cf. par exemple http://ldn-fai.net/med/images/7/78/Garder-la-main-sur-son-courrier_annote.pdf
5. La conso. électrique d'une Neufbox (mesurée au wattmètre) est aux alentours de 6-8W, contre environ 200W pour un ordinateur fixe.
6. http://ffdn.org

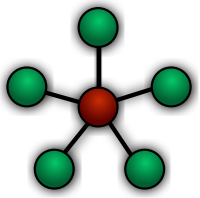
2 Analyse du problème

2.1 Justification de la topologie

2.1.1 Sauvegardes centralisées

La solution la plus simple serait de configurer les boîtiers pour les contraindre à envoyer régulièrement l'ensemble des données dont ils ont la charge à un serveur qui appartienne à l'association. En cas de perte de ses contenus, l'abonné n'aurait donc qu'à contacter ce serveur pour récupérer ce qui lui appartient, moyennant une authentification.

Il s'agit de l'approche centralisée, assimilable à une topologie en étoile (serveur de sauvegarde au centre en rouge) :



Si on peut considérer que l'aspect éthique des inconvénients posés par la centralisation sont moindre dans le cas d'une association respectueuse de la vie privée, l'aspect technique n'en reste pas moins problématique. Une solution centralisée ne passe pas l'échelle : plus la communauté de l'association grandira, plus le serveur de récupération des données sera surchargé et coûteux. À terme, c'est donc le succès du service qui tendra à le détruire.

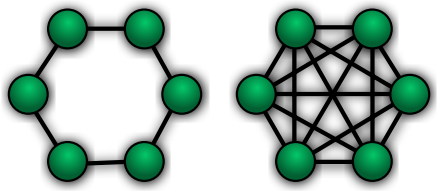
L'objectif étant de permettre au plus grand nombre de pouvoir devenir indépendant, il n'est pas concevable pour LDN d'être contraint de se limiter en restreignant sa communauté : il faut donc envisager une autre solution.

2.1.2 Sauvegardes pair-à-pair

Les réseaux pair-à-pair (*P2P*) possèdent des propriétés inverses des réseaux centralisés : plus ils grossissent, plus ils deviennent efficaces. Chaque nœud est à la fois un client et un serveur, ils ont tous une relation d'égal à égal.

Les réseaux pair-à-pair utilisent des topologies qui s'apparentent à l'anneau (chaque nœud est relié à son voisin) ou au maillage (chaque nœud est relié à tous les autres nœuds). Cette façon de faire permet aux membres de la communauté de parler directement entre eux, sans intermédiaire. Ainsi, puisqu'aucune machine n'est prépondérante, personne n'est indispensable et le réseau continue toujours de fonctionner, tant qu'il reste un minimum de nœuds.

Topologies pair-à-pair (anneau à gauche, maillage à droite) :



Les sauvegardes devront donc se faire de cette façon, entre abonnés. Chaque boîtier qui aura besoin de sauvegarder ses données sur le réseau devra être capable d'accueillir une partie des données des autres membres de la communauté. Une sauvegarde consistera donc à diviser ses données, et à en envoyer un bout à chacun des membres.

Le choix plus précis de la topologie se fera selon la solution technique retenue pour l'étape de la découverte des nœuds. Nous reviendrons sur cet aspect dans la section résolution.

Cette solution alléchante pose de nouveaux problèmes, notamment en terme de préservation de sa vie privée et de disponibilité des données.

2.2 Assurer la confidentialité et la redondance

Malgré les valeurs qui rassemblent les membres d'une telle communauté, considérer que chacun sera suffisamment honnête pour ne pas aller consulter les données qu'il héberge, semble un axiome beaucoup trop ambitieux pour pouvoir être sérieusement considéré. Ce serait également oublier qu'un boîtier peut éventuellement se faire pirater par une personne mal intentionnée.

Il faut donc que notre solution assure la confidentialité en ne permettant pas aux membres de pouvoir consulter ce qu'ils hébergent pour les autres.

Puisque nous avons considéré qu'un boîtier pouvait mettre en péril les données de son propriétaire, il en est de même pour les données tierces. Ainsi, notre solution devra pouvoir reconstituer l'ensemble des données du membre, même si certains des membres qui hébergeaient des fragments ne répondent plus.

Toutefois, chiffrer et morceler l'information pour assurer la confidentialité et la redondance peut demander des ressources importantes, pour une machine qui n'est pas destinée à des traitements de ce type.

2.3 S'accommoder d'un environnement limité

Un boîtier de FAI est à la fois un routeur, un commutateur réseau, un modem ADSL et un point d'émission Wifi. Pour assurer ces fonctions en étant cohérent en terme de coûts et de consommation électrique, ces machines ont à leur disposition le minimum des ressources physiques qui leurs sont nécessaires.

Les traitements liés aux fonctionnalités d'auto-hébergement et de sauvegardes réparties, sont de haut niveau pour une machine qui n'a pas été calibrée pour cet usage. Les contraintes sont donc fortes en terme de réalisation du projet : obtenir des programmes peu gourmands en ressources, et peu dépendants d'autres logiciels ou bibliothèques.

La contrainte supplémentaire réside dans la particularité de son système et de son architecture. Dans notre cas, il s'agit d'un processeur MIPS à basse consommation, populaire dans l'embarqué et notamment utilisé pour les matériels réseaux de Cisco, qui tend à être remplacé par la plus récente technologie ARM. Notre programme, mais aussi chacune de ses dépendances, devra donc être capable de fonctionner sous ce type d'architecture exotique, tout en s'intégrant à l'atypique système GNU/Linux embarqué (il s'agit d'une distribution OpenWRT modifiée).

Nous avons également eu pour charge de valider le choix qui avait été fait en terme de matériel.

Côté réseau, le débit montant d'une ligne de particulier est très limité. Ainsi, le programme sera contraint de trouver une solution pour compresser les données qu'il envoie d'un boîtier à un autre.

3 Organisation

3.1 Première confrontation avec la complexité du problème

Contrairement à ce qu'on pourrait croire au premier abord, les difficultés ne sont pas seulement dans l'aspect embarqué. La partie pair-à-pair n'est pas sans poser de problème de son coté. Parmi les nombreux problèmes posés, citons par exemple :

- Quelle est la meilleure solution pour réussir à contacter l'ensemble des participants sans être contraint de maintenir un serveur qui serve de point fixe ?
- Comment s'assurer que les nœuds qui hébergent nos données sont toujours vivants ?
- Comment réagir lorsqu'un événement asynchrone perturbe un échange (par exemple, un nœud nous confirme qu'il a de la place pour nous héberger, mais quelqu'un d'autre l'occupe le temps que le demandeur ne s'assure qu'il a bien trois machines différentes à sa disposition) ?
- Comment réagir si un nœud est détecté comme mort, et qu'il revient sur le réseau (données oubliées) ?
- Comment s'assurer qu'il n'y a jamais de boucle lorsqu'on contacte un nœud, qui transmet à son voisin, qui transmet lui-même à ses voisins (topologie en anneau) ?
- Comment un nœud peut-il s'assurer que les données qu'il héberge n'appartiennent pas à quelqu'un qui a définitivement quitté la communauté ?

Après avoir passé notre première journée de réunion de conception à découvrir ces problèmes en tentant de leur trouver des solutions, nous avons finalement pris conscience de la complexité du projet. Sous les conseils de notre tuteur, nous sommes partis en quête de solutions existantes. Cette étape est détaillée dans la section *Recherches*.

3.2 Implémentation de la distribution pair-à-pair

Après plusieurs semaines à chercher des solutions en consultant les rapports ou les thèses associés aux nombreux projets dénichés, c'est dans un état d'esprit tout à fait différent que nous avons abordé notre seconde réunion de conception. Nous avions compris exactement où se situaient les points sensibles, et nous étions au fait des solutions classiques aux problèmes que nous avions décelés la première fois.

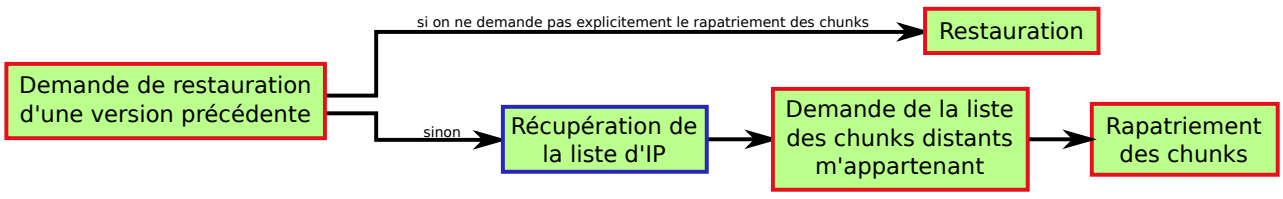
Nous avons travaillé grâce à des schémas de conception. Un premier qui représente l'enchainement des différentes étapes des processus de sauvegarde/restauration, en traitant un maximum de cas, et en représentant chacune d'entre elle par un bloc nommé. Et un second qui représente chacun de ces blocs en identifiant clairement les entrées/sorties, ainsi que quelques actions à effectuer pour préciser sa fonction. Nous avons différenciés un côté client et un côté serveur, bien que ce soit le même logiciel des deux côté et que le serveur serve parfois à répondre pour le client.

Une fois ces blocs clairement identifiés et la conception pouvant être estimée comme traitant tous les cas envisageables, nous nous sommes répartis ces blocs.

Pour être cohérent avec notre façon de travailler, nous avons choisi une approche modulaire pour l'implémentation. Le projet étant codé en C (pour des raisons de légèreté), nous avons conçu chacun de ces blocs comme une bibliothèque indépendante, associable en fonction des besoins lors de l'édition des liens, aux principaux programmes qui constitueront notre projet. Nous avons conçu un système de *makefiles* organisé en arborescence et en héritage, pour obtenir un environnement de travail collaboratif, efficace et évolutif.

Ces schémas ont été rassemblés dans un document vectoriel. Grâce à des codes couleur, il nous a permis de nous tenir informés de l'évolution de part et d'autre du projet. Chaque contour de bloc portait la couleur de celui qui s'était engagé à l'implémenter, et sa couleur de fond indiquait l'état d'avancement du bloc.

Extrait des schémas de conception, avec les codes couleur :



En distribuant cette image (et néanmoins document texte) sur le gestionnaire de versions, il est devenu un document de travail qui a évolué au cours du temps et qui nous a permis de maîtriser notre temps. L'ensemble du projet a évidemment bénéficié de ce gestionnaire de version (forge de l'école).

La dernière étape avant de pouvoir travailler aisément chacun de notre côté a été de lister l'ensemble des structures de données nécessaires au fonctionnement du programme et à la communication entre deux boîtiers. Ces structures ont été nommées et programmées dans un fichier d'entêtes que nous nous sommes partagés en même temps que le schéma de conception.

Les premières bibliothèques implémentées ont permises de donner le ton en terme de convention de nommage. Rapidement, nous nous sommes mis d'accord sur des règles claires, pour obtenir un tout cohérent.

Nous avons utilisé le wiki de l'association pour écrire notre propre documentation [7] et garder une trace du suivi des courriers envoyés.

3.3 Intégration du projet au boîtier

Puisque les boîtiers utilisent une architecture exotique, nous ne pouvions pas nous contenter de compiler notre projet et toutes ses dépendances sur nos ordinateurs personnels, en les envoyant ensuite sur le boîtier.

Étant donné que les capacités du disque dur et du processeur étaient extrêmement limités, nous ne pouvions pas non plus compiler nos sources directement sur un boîtier. Ainsi, nous avons été contraint de construire un environnement émulé à base d'une Debian et de l'émulateur Qemu, pour ensuite y injecter l'arborescence des fichiers du boîtier. En fournissant le noyau du boîtier à la machine émulée, nous avions un environnement puissant qui disposait de la bonne architecture et du bon noyau, pour compiler l'ensemble de nos sources.

4 Recherches

4.1 Chercher pour ne pas réinventer

4.1.1 Documentation

Avec l'explosion du succès d'Internet, de nombreux chercheurs et industriels ont pris conscience que la méthode la plus fiable et la moins coûteuse était d'utiliser la puissance du réseau. Ainsi, ces dernières années, de nombreux documents ont été publiés, et de nombreux projets ont été mis en place. C'est dans cette banque de données que nous sommes partis à la recherche d'un programme, d'une bibliothèque ou d'un algorithme, qui puisse répondre à l'une ou l'autre de nos problématiques.

Notamment grâce à l'aide de Martin QUINSON et Gérald OSTER, nous avons finalement pu en dénicher quelques-uns, plus intéressants que les autres.

Probablement parce qu'on a particulièrement l'habitude de documenter dans ce milieu, la plupart des documents étudiés étaient des rapports de thèse. Nombre d'entre elles ont été passionnantes à lire, et promettaient monts et merveilles. Mais quasiment aucune n'aboutissait à un projet fonctionnel et éprouvé. Nous avions donc des idées, souvent détaillées outre mesure sur des points qui dépassaient notre objectif, mais sans jamais aucune solution concrète.

Cinq solutions ont particulièrement retenu notre attention.

4.1.2 Distributed Internet Backups System (DIBS)

Son nom laissait à penser que notre projet était déjà implémenté. Selon la thèse de son auteur, DIBS [8] permet de chiffrer, versionner, et découper un dossier de données, pour ensuite les envoyer en pair-à-pair à d'autres nœuds sur Internet. Comble du bonheur, son code-source était disponible.

7. http://ldn-fai.net/wiki/PIDR
8. MAR. E. (2006)

En cherchant les derniers signes de vie du projet, nous avons constaté qu'il n'avait plus eu d'activité depuis 2006. Nous avons essayé de l'installer, et nous nous sommes retrouvés à devoir corriger le premier script python pour pouvoir l'exécuter correctement, sans arriver à un résultat totalement fonctionnel. Des commentaires postés sur des forums de discussion évoquaient des problèmes majeurs qui n'ont jamais été corrigés.

Nous avons essayé de contacter l'auteur pour savoir si le projet était toujours maintenu, en vain. Après avoir récolté plusieurs bonnes idées de conception, nous avons continué nos recherches en espérant trouver un projet abouti.

4.1.3 Cooperative File System (CFS)

Il s'agit de notre second grand espoir. L'idée était osée et allait au delà de ce que nous avions imaginés.

CFS [9] est un système de fichiers capable de s'étendre sur plusieurs machines à la fois. En se contentant de monter un simple système de fichiers localement, nous aurions pu, d'après la thèse, écrire des données comme sur un simple disque dur, et les laisser se répartir et s'envoyer automatiquement. L'auteur évoquait à plusieurs reprise des tests concluants avec un projet de 7000 lignes de code, ce qui nous assurait de ne pas être de nouveau sur un document qui ne concrétise pas la réflexion.

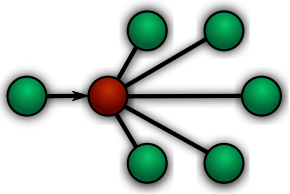
Toutefois, un bémol de taille s'est présenté : impossible de mettre la main sur ce fameux code. Nous avons contacté l'auteur pour en savoir plus. Frank DABEK nous a finalement répondu : « *CFS was never more than a research prototype, you don't want to use it for anything in production.* ». Nous n'avons même pas eu le code des tests, nous avons été contraint de totalement abandonner cette solution, trop compliquée à implémenter nous-mêmes.

4.1.4 La multidiffusion

En étudiant la possibilité de se passer totalement d'un serveur de listes, nous avons eu l'idée d'utiliser notre spécificité : travailler sur un réseau.

La multidiffusion est une forme de diffusion d'un émetteur vers un groupe de récepteurs. Ainsi, ce sont les équipements réseau qui s'occupent de gérer les abonnements et désabonnements des nœuds à une adresse IP dédiée à notre communauté. Lorsque des messages sont envoyés à tous les membres en même temps (recherche d'espace disponible, par exemple), la multidiffusion permet de ne transporter qu'un seul message et de ne le dupliquer qu'au dernier moment, pour ne pas encombrer les tuyaux.

Fonctionnement de la multidiffusion (l'équipement réseau final est en rouge) :

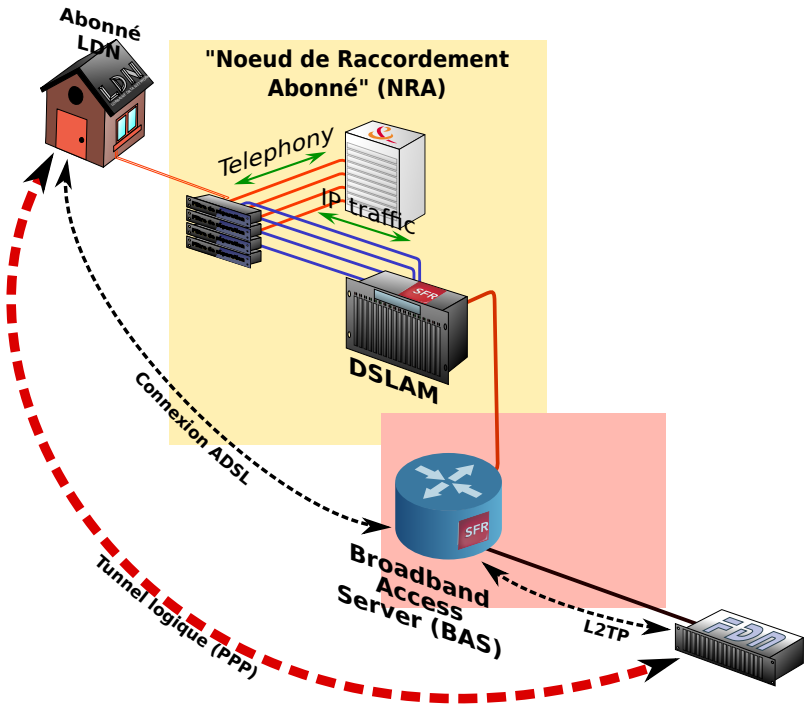


Cette méthode de diffusion implique une topologie en maillage, l'une des deux solutions qui a été présentée ci-dessus. Un abonnement à une adresse IP sur le réseau LDN correspondant au service de sauvegardes aurait suffit pour insérer un membre dans la communauté, sans avoir à maintenir un point fixe centralisé.

Nous avons contacté les administrateurs des infrastructures utilisées par LDN, pour savoir si une telle utilisation du réseau était possible : la réponse fût négative. Non seulement la collecte de l'abonné final jusqu'aux équipements utilise un tunnel logique (impossible donc de bénéficier de l'économie du transport des données), mais en plus le logiciel utilisé pour le routage ne supporte pas ce mode de diffusion.

9. DAB. F. (2001)

Raccordement d'un abonné jusqu'aux infrastructures de LDN, avec un tunnel logique (en pointillés rouges épais) :



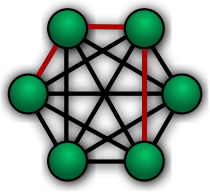
Les administrateurs nous ont proposés un accès aux LNS (les équipements qui assurent le routage) pour nous permettre d'apporter nous-mêmes cette fonctionnalité au logiciel afin de bénéficier au moins des possibilités d'abonnements/désabonnement. Le temps s'étant considérablement réduit et le travail risquant d'être conséquent, nous avons reporté cette possibilité.

Cette solution avait de plus l'inconvénient non négligeable de ne pas pouvoir ouvrir les communautés en dehors du FAI.

4.1.5 Chord

Évoqué en réunion par notre tuteur, Chord [10] est un protocole pair-à-pair utilisé dans de nombreuses applications distribuées connues du grand public.

Sa spécificité est de ne nécessiter aucun serveur de listes. Cette solution correspond plutôt à la topologie en anneau : chaque nœud possède un prédécesseur et un successeur, et se transmet l'information. L'algorithme inclus toutefois des optimisations qui visent à ne jamais parcourir l'anneau en entier, mais à le couper régulièrement en deux pour le parcourir :



CFS s'appuie fortement sur le projet Chord, et nous a poussé à nous y intéresser de plus près. Nous avons contacté la liste de diffusion du projet pour avoir quelques renseignements supplémentaires, mais nous n'avons encore jamais reçu de réponse.

10. KAA. F., KAR. D., MOR. R., SIT E., STR. J. (2009)

L'implémentation semble plutôt aisée et permet une découverte des nœuds en anneau, avec la capacité de refaire ses chemins et d'introduire de nouveaux participants, le tout sans aucune centralisation et avec un algorithme de complexité moindre. Nous avons retenu cette solution pour la découverte des nœuds uniquement, la suite du projet n'étant pas en adéquation avec nos objectifs (Chord permet de ditribuer des fichiers publics, proportionnellement disponibles au nombre de personnes qui les téléchargent).

4.1.6 libchop

Ce logiciel nous a été conseillé la première fois par Lucas NUSSBAUM, au tout début de notre projet. Il s'agit d'un projet de thèse développé par le français Ludovic COURTÈS à l'INRIA.

Comme son nom l'indique (*chop*, le hachoir), les outils associés permettent non seulement de découper l'ensemble des données contenues dans un répertoire, mais aussi de les chiffrer et de les versionner. Le projet est récent, le code est disponible, et nous avons constaté qu'il fonctionnait à la perfection.

Le chiffrement est géré par des clés de version (*tuples*), délivrées à chaque nouvelle sauvegarde du répertoire de données. Il s'agit d'une chaîne de caractère complexe, qui servira à la fois à désigner une version, mais aussi et surtout à la déchiffrer.

Une bonne nouvelle n'arrivant jamais seule, à notre grande surprise, nous n'avons pas eu à contacter l'auteur puisque c'est lui qui nous a contactés. Nous avions évoqué notre projet lors d'une réunion mensuelle de la fédération des FAI en mentionnant l'utilisation probable de libchop [11], et nous avons eu alors le plaisir de découvrir que Ludovic COURTÈS était lui-même actif dans un FAI participatif dans le Sud de la France. Il suivait donc les comptes-rendus de ces réunions.

Puisque, hormis l'aspect pair-à-pair, le projet ressemblait à DIBS, nous lui avons demandé directement ce qu'il en pensait. Il nous a confirmé que le projet semblait mort et bancal sur certains aspects, mais que lui-même n'avait pas concrétisé l'aspect pair-à-pair. En nous confirmant toutefois que libchop était parfaitement adapté à cette utilisation.

Devant une première recommandation, un projet d'une telle qualité, et la disponibilité de l'auteur, nous n'avons pas eu d'autre choix que d'intégrer ce travail à notre projet.

4.1.7 Projets retenus

Nos heures de lecture de rapports, de thèses, et de recherche sur la Toile, nous auront permises de dégager deux solutions exploitables : Chord et libchop, en laissant la multidiffusion de côté pour l'instant (qui a l'avantage de la simplicité). Ironiquement, ce sont finalement les deux premières solutions qui nous ont été soufflées qui sont retenues. Vis-à-vis de la contrainte de l'environnement limité, libchop étant écrit en C, il ne nécessite donc pas de machine virtuelle.

Des solutions ont été trouvées pour la découverte des nœuds, pour le chiffrement, le découpage et même la gestion des versions des sauvegardes, il ne reste plus qu'à trouver une solution pour distribuer les données ainsi formatées en pair-à-pair. Nos recherches ayant déjà pris énormément de temps, nous avons choisi de les abandonner pour écrire nous-même cette partie, à la lumière de ce que nous avions appris.

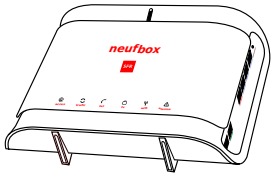
5 Étude du boîtier

5.1 Choix d'un modèle

Concernant les boîtiers, LDN n'en distribuant pas actuellement, nous avons eu pour charge d'estimer si le choix qu'ils avaient fait était judicieux. Leurs recherches avaient abouties sur l'utilisation de la Neufbox 4, de SFR. Outre l'avantage d'avoir toutes les fonctionnalités attendues pour un boîtier, elle dispose d'un système d'exploitation quasiment libre, et est disponible légalement et en nombre pour un prix raisonnable sur les sites d'occasion.

11. COU. L. (2012)

La Neufbox 4 de SFR (conçue par l'entreprise Efixo, à la demande de feu Neuf-Cégétel) :



12

Ayant déjà eu vent de travaux pratiques sur des Neufbox en troisième année pour les options Logiciels Embarqués, nous avons pris l'initiative de contacter Alexandre PARODI pour lui demander plus d'informations. Celui-ci nous a finalement indiqué que ça n'était pas lui qui s'occupait de ces travaux, mais un ancien élève de l'ESIAL qui répond au nom de Julien AUBÉ. Malgré son engagement à nous envoyer son courriel, nous ne l'avons pas reçu.

Lors d'une autre de ces réunions mensuelles de la fédération des FAI, nous avons évoqué le choix des Neufbox. Une des personnes présentes nous a alors immédiatement indiqué avoir quelqu'un dans son FAI, à Toulouse, qui connaissait particulièrement bien ce matériel : un certains obinou, de son vrai nom Julien AUBÉ. Si l'expression « *le monde est décidément petit* »devait figurer à un endroit de ce rapport, ce serait certainement ici.

Nous sommes donc entrés en contact avec lui, d'abord par IRC [13], puis par téléphone après avoir convenu d'une réunion tous ensemble. Après avoir discuté une petite heure, au sujet du développement du système des Neufbox, de la difficulté d'avoir des morceaux qui restent propriétaires, et de l'intérêt de notre projet, il nous a finalement proposé de nous envoyer gratuitement quatre Neufbox pour nous aider à avancer. À noter que Martin QUINSON nous en avait déjà prêté une, dès le début du projet.

5.2 Un système quasiment libre

La Neufbox 4 utilise un système OpenWRT modifié. Celui-ci a le bon goût d'être entièrement libre, puisqu'il s'agit en réalité d'une distribution GNU/Linux orientée routeurs.

Cependant, le matériel Broadcom dont est constitué le boîtier n'est pas libre lui-même. Ainsi, les pilotes du modem et de l'émetteur Wifi n'ont pas été libérés et ne sont fournis que pour la dernière version du système utilisé par la Neufbox. Celui-ci correspondant à un OpenWRT d'ancienne génération, certaines fonctionnalités comme l'IPv6 ne sont pas disponibles. Le mettre à jour permettrait aussi de nettoyer le système de tous les ajouts de SFR, pour des fonctionnalités inutiles dans notre cas (télévision, par exemple).

Cette opération ne s'annonce pas aisée : pour cela, il faut réussir à changer l'intégralité du système, tout en gardant une compatibilité avec les pilotes fermés qu'on ne peut pas faire évoluer. Julien AUBÉ a évoqué, lors de notre entretien, une volonté de Efixo de distribuer une mise à jour prochainement, mais il n'y a eu aucune nouvelle de ce côté depuis.

Une solution alternative serait de ne pas bénéficier de la mise à jour, mais de réussir à exploiter et modifier le système actuel pour intégrer les fonctionnalités d'auto-hébergement ainsi que notre programme de sauvegardes.

6 Résolution

6.1 Processus de traitement des sauvegardes automatisées

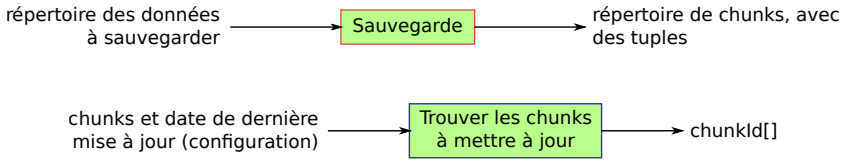
Grâce à libchop, certains de nos problèmes initiaux sont dors et déjà résolus :
- Les informations sont chiffrées
- Les données sont compressées

12. Dessin de Vincent MOLLIMARD.
13. *Internet Relay Chat*, réseaux de discussions instantanées particulièrement appréciés des informaticiens.

– Les sauvegardes sont découpées

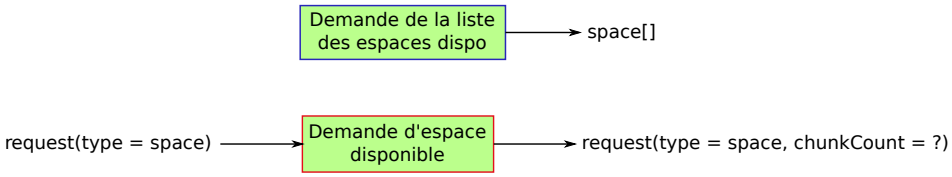
Puisque libchop nous permet de morceler la sauvegarde, nous avons considéré que l'unité de mesure de l'espace disponible serait le fragment (*chunk*).

Après avoir effectué une sauvegarde de ses données, notre programme détecte, grâce aux dates du système, les fragments qui ont été ajoutés.



Avant de tenter de décider où partira chacun des fragments, il demande à l'ensemble des nœuds la place qu'ils ont de disponible sur l'espace qu'ils réservent pour les fragments tiers. Les réponses sont donc exprimées en quantité de fragments qui peuvent encore être accueillis.

Côté client / serveur :



À partir de cette liste, le client commence la répartition. Il dispose pour cela de trois algorithmes différents :

1. Le premier trie la liste des serveurs en fonction du nombre de fragments qu'ils peuvent accueillir. Il affecte ensuite chacun des fragments à envoyer à un serveur, en commençant par le serveur ayant le moins de place. Il en profite pour affecter une copie des fragments aux deux serveurs suivants. Puisque la liste est triée de manière croissante, l'algorithme est en permanence assuré qu'il y a toujours au moins autant d'espace disponible sur les serveurs suivants.

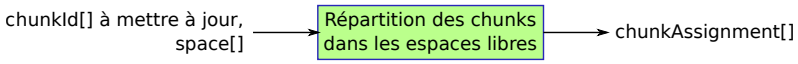
   Nous constatons cependant que cet algorithme introduit des déséquilibres au niveau de la répartition. Certains serveurs sont surchargés alors que d'autres ne sont quasiment pas exploités. De plus, un placement systèmatique tel que celui-ci, est contraint de s'assurer en permanence que la copie d'un fragment n'est pas envoyé sur le même serveur qu'une copie précédente. Cet algorithme converge parfois vers des mauvaises solutions.

2. Le second algorithme trie la liste des serveurs selon le même critère, mais de façon décroissante. Puis, il tente de remplir les serveurs qui ont le plus d'espace disponible, afin de répartir au maximum les données. Nous avons essayé de réduire les déséquilibres, mais cet algorithme à le même défaut que le précédent : il n'aboutit pas toujours à une solution viable, même si elle existe.

3. Le dernier effectue un placement initial des fragments de manière cyclique (*round-robin*). Il place le premier fragment sur le premier serveur disponible, puis le copie sur les deux suivants. Il copie ensuite le second fragment sur le prochain serveur disponible, etc. De cette façon, il place tous les fragments sans se préoccuper des erreurs de répartition (on essayant toutefois de les minimiser).

   L'algorithme tente ensuite de détecter les erreurs de placement en vérifiant sur chacun des serveurs s'il existe des doublons (cette étape est en réalité réalisée en simple décalage de la précédente). Enfin, pour une liste d'erreurs donnée, des permutations aléatoires des fragments sont effectuées, produisant une nouvelle liste d'erreurs. Si le nombre d'erreurs a été réduit, cette nouvelle répartition est conservée. Ces étapes se répétent jusqu'à trouver une répartition viable. N'ayant trouvé aucun critère de preuve d'existence d'une possibilité de convergence vers une solution viable, nous avons limité le nombre d'exécutions de la boucle.

Pour l'instant, ces algorithmes sont encore en test pour déterminer celui qui donne le plus souvent les meilleurs résultats. Dans tous les cas, il s'arrange pour envoyer tous les fragments en trois exemplaires, en s'assurant que deux exemplaires ne sont jamais envoyés sur la même machine (la communauté a donc besoin d'être à un minimum de quatre nœuds pour fonctionner). Il obtient ainsi une liste de fragments associés à des nœuds précis.

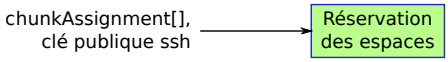


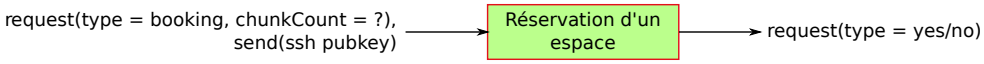
Durant cette étape, le programme peut conclure que l'espace disponible dans la communauté n'est pas suffisant pour accueillir sa mise à jour. Si c'est le cas, il se contente d'envoyer un courriel au propriétaire du boîtier, qui sera averti que la dernière sauvegarde n'a pas pu être effectuée. Celui-ci aura pour charge de contacter la communauté au travers de la liste de diffusion pour avertir du problème, et encourager les autres membres à agrandir leurs espaces (ou faire entrer de nouveaux nœuds).

Pour éviter la saturation des espaces disponibles, chaque nœud doit être en mesure de fournir au minimum le triple de la taille actuelle de ses sauvegardes (c'est à dire l'équivalent de ce qu'il consomme sur l'ensemble des autres nœuds). Les sauvegardes se faisant sur des clés USB reliées aux boîtiers, et les données (souvent des fichiers textes) compressées n'étant pas excessivement encombrantes, nous avons estimé que ce cas devrait être relativement rare. Il y a une grande probabilité pour que les utilisateurs proposent systématiquement bien plus d'espace que la limite inférieure imposée.

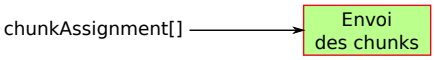
Puisque le client est sûr de pouvoir placer tous ses fragments et donc d'assurer la sauvegarde complète, il peut lancer ses réservations à partir de la liste obtenue. Il contacte donc chacun des nœuds qui sont censés accueillir les fragments, pour leur demander de réserver de la place pour le nombre de fragments qu'il souhaite leur envoyer. Cette étape devrait bien se passer, puisque les nœuds ont confirmé leur espace disponible avant de faire la répartition. Mais il existe des cas où l'espace qui était libre avant l'étape de répartition locale, n'est plus le même après. Soit parce qu'un autre nœud a pris la place en faisant la réservation, soit parce qu'une machine qui répondait avant ne répond plus à ce moment-là. Si un tel cas se présente, le programme annule ses prévisions, redemande la liste des espaces disponibles, réparti à nouveau ses fragments en fonction, et retente les réservations. Jusqu'à ce que cela fonctionne ou que les espaces ne soient pas suffisants. À chaque fois qu'une réservation est effectuée, le serveur a la possibilité de demander au client sa clé publique SSH, s'il n'héberge pas déjà des fragments à lui.

Côté client / serveur :





Une fois que tous les espaces sont correctement réservés, le client n'a plus qu'à contacter chacun des nœuds, en SSH [14], pour leur transmettre ses fragments. Si par malheur un nœud devient indisponible à cette étape, les fragments qui n'ont pas pu être transmis sont à nouveau considérés comme étant nouveaux, et le programme relance tout le processus, depuis la demande des espaces disponibles.

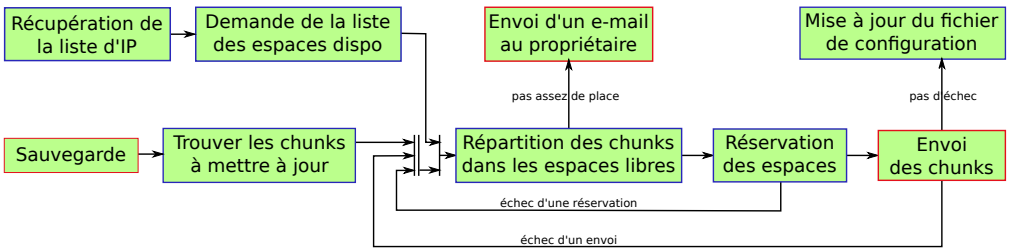


L'ultime étape consiste à mettre à jour le fichier de configuration. Celui-ci est écrit en XML [15] et contient toutes les informations relative à ce boîtier en particulier. Le programme écrit la liste des associations nœud/fragment, la date courante comme date de dernière mise à jour, et la clé délivrée par libchop qui permettra de restaurer cette version en

14. *Secure SHell*, un protocole qui permet entre autres de transmettre des données de façon authentifiée et sécurisée.
15. *Extensible Markup Language*, un format de données normé et facilement exploitable.

particulier si besoin est. Chaque fois qu'une nouvelle clé est enregistré, le fichier XML de la configuration est envoyé à une adresse externe spécialement dédiée à ça.

Schéma récapitulatif du processus complet de sauvegarde :



Il est question sur ce schéma de récupération de liste d'IP. En effet, dans un premier temps, nous avons préféré utiliser la méthode simple pour trouver les nœuds à contacter. Cependant, notre méthode de programmation modulaire permettra à terme de changer très rapidement ce comportement (en lui préférant Chord, par exemple), en ne modifiant que le module concerné.

6.2 Processus de restauration à la demande de l'utilisateur

À n'importe quel instant, l'utilisateur dispose de l'interface web intégrée à son boîtier, pour consulter les différentes versions des sauvegardes.

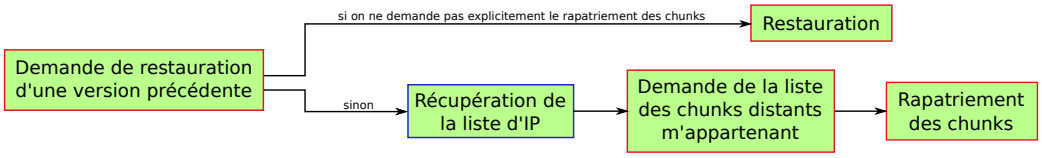
En consultant une version antérieure, il peut demander à revenir à ce point de restauration :

Si le boîtier n'est plus accessible et qu'il a perdu l'ensemble de ses données, l'utilisateur peut demander au programme de rapatrier l'ensemble des fragments depuis le réseau en cochant la case associée.

Si cette case est active, le système demande à l'ensemble des nœuds la liste des fragments hébergés qui lui appartiennent. Il se contente ensuite de parcourir ces listes, pour récupérer en SSH ce qui lui appartient.

Dans les deux cas, selon la version qui a été demandée, le système utilise libchop pour restaurer le répertoire de données à partir des fragments reconstitués et de la version (correspondant à la clé) demandée.

Schéma récapitulatif du processus complet de restauration :



6.3 Auto-régulation du système

Les deux principales fonctions de notre programme étant assurées, il reste à gérer tous les cas d'exception.

Par exemple, que se passe-t-il lorsque qu'un nœud commence à faire une série de réservations, et que l'une d'entre elles échoue ? Dans ce cas, nous avons vu que le programme abandonnait pour recommencer tout le processus, mais qu'advient-il des réservations déjà effectuées ? Dans le même ordre d'idée, que se passe-t-il si un nœud déserte la communauté, en laissant ses fragments encombrer les autres membres ? Ou bien si par malheur tous les nœuds qui hébergeaient des répliques de quelques-uns de mes fragments ne sont plus accessibles ?

Le mécanisme que nous avons mis en place pour assurer la stabilité du système s'appelle le *heartbeat* (donner un signe de vie). Régulièrement, les nœuds qui hébergent des fragments étrangers, contactent leur propriétaire, en lui envoyant la liste des fragments lui appartenant qu'il possède.

Côté client, deux cas de figure se présentent alors :

1. Il manque des fragments dans sa liste, alors qu'il est censé les avoir (suppression pour une raison quelconque)
2. Il y a des fragments qu'il ne devrait pas avoir (la nœud avait été laissé pour mort, et les fragments qu'il hébergeait réassignés, alors qu'il est réapparu)

Dans le premier cas, le système supprime les fragments en question de sa configuration, et tente de replacer dans la communauté les fragments disparus (en suivant la même procédure que pour la routine de sauvegarde).

Dans le second cas, le nœud recontacte son hébergeur, pour lui demander explicitement de supprimer ces fragments qui l'encombrent pour rien.

Ces visites régulières ont une autre conséquence : les clients s'assurent que leurs hôtes sont toujours vivants. Ainsi, à chaque passage de *heartbeat*, le client met à jour la date de dernière visite du nœud. En vérifiant régulièrement ces dates, si un nœud ne lui a pas rendu visite depuis un mois, il décide de replacer l'ensemble des fragments qu'il lui avait confié.

Du côté serveur, la conséquence est à peu près la même : la date de dernière réponse d'un client est stockée, et mise à jour à chaque fois qu'il arrive à le contacter pour lui envoyer sa liste. Régulièrement, il vérifie la liste des dates, et supprime les données d'un utilisateur si celui-ci n'a pas répondu depuis six mois. Si un boîtier prend feu ou n'est plus disponible pour une raison quelconque, l'utilisateur a donc six mois pour récupérer ses biens ou donner un signe de vie.

Chaque fois qu'une réservation est passée, le serveur ajuste le quota alloué au client en l'augmentant de la taille exacte de ce qu'il a accepté de recevoir. Pour s'assurer que des réservations n'ont pas été abandonnées, le serveur vérifie régulièrement que tous les espaces alloués sont pleins. Sinon, il réajuste les quotas en conséquent.

Ces procédures fonctionnent grâce à des CRON. À noter que la différenciation client/serveur est particulièrement fausse dans ce contexte. En effet, lorsque le serveur contacte ses clients pour leur envoyer la liste, il contacte la partie serveur qui écoute les requêtes. Cette schizophrénie est typique des programmes en pair-à-pair.

6.4 Interface web

Les boîtiers sont totalement autonomes. Mais pour configurer les données qu'il traitent, vérifier que tout se passe bien, consulter les versions enregistrées, ou encore en restaurer une, l'utilisateur a besoin de communiquer avec lui.

Le boîtier intégrant un serveur web pour permettre à l'utilisateur de configurer ses fonctionnalités de routeur, de point d'accès Wifi ou de serveur DNS [16], nous avons décidé d'ajouter notre propre section à cette interface, en l'intégrant le mieux possible.

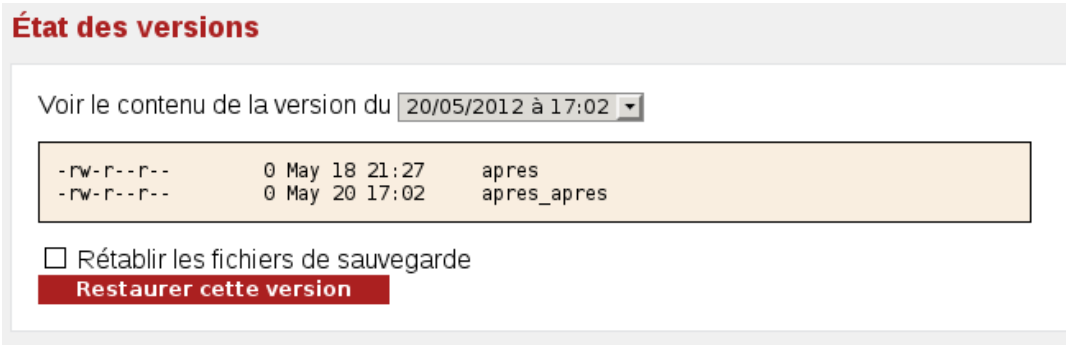
La contrainte la plus forte dans ce domaine a été l'absence de possibilité d'utiliser un langage dynamique pour gérer nos pages. La solution que nous avons trouvé a été de l'écrire entièrement en Bash et d'utiliser l'exécution CGI [17] du

16. *Domain Name System*, système mondial de gestion des noms de domaine sur Internet.
17. Le mode CGI permet de lancer n'importe quel exécutable et de récupérer le contenu de sa sortie standard en tant que page web.

serveur. Pour faciliter les opérations et apporter plus de fluidité entre les demandes de l'utilisateur et les appels à libchop, nous avons utilisé la bibliothèque javascript JQuery pour communiquer en Ajax [18] avec le serveur.

Puisque LDN avait déjà conçu un thème graphique pour l'interface web de la Neufbox (cf. annexes), nous l'avons repris pour permettre à notre panel de parfaitement s'intégrer.

La partie consultation des versions sauvegardées a été déjà présentée dans la section précédente. Outre permettre la restauration, elle permet simplement de récupérer la liste des fichiers, avec leurs attributs, à l'instant de ce point de restauration. C'est un élément décisif pour trouver jusqu'à quand on souhaite remonter, et trouver où sont les données disparues qui nous intéressent :



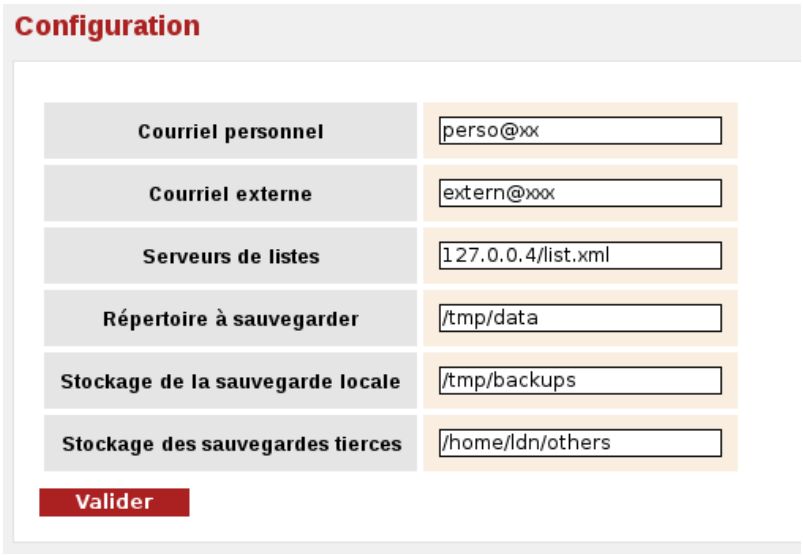
Afin d'avoir tous les outils en main pour comprendre l'évolution de ses propres données, l'interface permet aussi de visualiser précisément ce qui s'est passé d'une version à l'autre :

L'affichage utilise le code suivant :

- **A** - le fichier a été ajouté
- **M** - le fichier a été modifié
- **T** - le fichier n'a pas été modifié, mais ses propriétés l'ont été
- **D** - le fichier a été supprimé
- Les fichiers renommés sont matérialisés par une flèche qui indique le changement de nom.

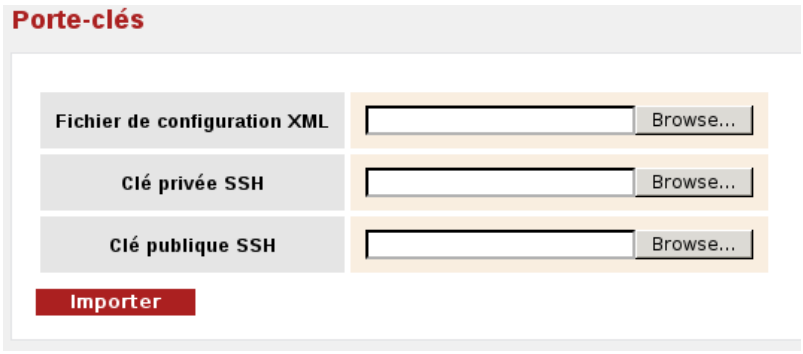
Pour configurer ses sauvegardes, l'utilisateur dispose d'une section dédiée à ses paramètres :

18. *Asynchronous JAvascript and Xml*, méthode de conception de sites qui permet, grâce à javascript, de communiquer avec le serveur sans recharger la page principale.



Le courriel personnel servira à le contacter en cas de sauvegardes avortées. Le courriel externe servira pour lui envoyer régulièrement tout ce qui sera nécessaire pour rétablir son boîtier en cas de perte de l'ensemble des données. Nous différencions les deux en considérant que beaucoup d'utilisateurs hébergeront leurs courriels sur leur boîtier, nous leur demandons donc d'ouvrir une boîte de courriel éventuellement dédiée à ça et jamais consultée, sur un serveur externe. À terme, LDN pourra fournir des adresses dans ce but.

Enfin, en cas de remplacement du boîtier, l'utilisateur peut injecter les fichiers qu'il aura reçu son courriel externe pour rétablir l'ensemble de sa configuration :



S'il a conservé le contenu de sa clé USB, la communauté ne se rendra même pas compte du changement de boîtier.

Afin de s'assurer du bon fonctionnement de son système de sauvegardes, une zone est prévue pour informer de l'état actuel des choses :

6.5 Intégration du projet au boîtier

Le passage par la virtualisation du système grâce à Qemu a déjà été évoqué dans une section précédente. Pour recréer l'environnement de la Neufbox, il a fallu réussir à lancer un système en MIPS, en utilisant le même noyau. Puis, utiliser

chroot [19] pour reproduire l'arborescence du boîtier.

L'objectif était de pouvoir compiler tranquillement l'ensemble de notre projet, ainsi que ses dépendances (libchop, ses propres dépendances, libxml2) sur une machine puissante. Mais certains outils fondamentaux nous manquent aussi : par exemple, le boîtier intègre déjà un client SSH mais pas de serveur. Puisque le système utilise une OpenWRT, en reproduisant son arborescence, nous pourrions utiliser son gestionnaire de paquets, pour installer aisément ce qui est manquant.

Pour ce qui est de la place disponible sur la Neufbox, un projet de gestion des fichiers par clé USB est maintenu depuis un certain temps par l'utilisateur fxmx86 [20]. Celui-ci permet de modifier le système de la Neufbox pour qu'elle aille systématiquement chercher ses fichiers sur la clé en priorité, et qu'elle se rabatte sur son propre disque en cas d'indisponibilité. Cette astuce permet de rajouter des programmes sans contrainte d'espace. Nous avons essayé d'ajouter une simple commande au système de base, mais quelques kilos suffisent pour dépasser la taille maximale. Cette solution est donc à privilégier.

Concernant l'accès en mode super-utilisateur et la modification des fichiers, nous avons réussi en plus de l'émulation, à changer des fichiers du système embarqué. Concernant l'ajout des programmes, nous n'avons pas encore abouti à une victoire à l'heure où ce rapport est écrit.

6.6 Sécurité

Afin de sécuriser les accès SSH sur les boîtiers, nous avons étudié les possibilités offertes par le protocole. Nous avons finalement réussi à aboutir à une solution qui ne permet aucune marge de manœuvre aux éventuels attaquants. Les seules façons d'utiliser nos accès SSH sont les suivantes :

- Créer un dossier à distance : `ssh ldn@host-ip nom_du_dossier`, avec un nom qui ne contient que des lettres ou des chiffres.
- Envoyer un fragment avec SCP [21] : `scp nom_du_fragment ldn@host-ip:dossier/nom_du_fragment`, avec une destination qui ne contient que des lettre ou des chiffres, et un seul slash.

Les requêtes sont ensuite réécrites pour être interprétées, et redirigée dans le bon répertoire, selon la clé SSH utilisée. Cette méthode nous semble plus efficace et moins consommatrice de ressources que l'utilisation de chroot, qui ne nous a pas satisfait pour cet usage.

Concernant les quotas, nous sommes dépendants de ce que nous arriverons à faire en terme d'installation sur les boîtiers. L'idéal serait d'utiliser un système de fichiers comme XFS qui permet de gérer des quotas par dossier. Dans tous les cas, des quotas disques strictement limités à la taille de ce qui a été réservés seront systématiquement appliqués.

L'authentification se fait actuellement en fonction de l'IP source de la personne qui contacte le serveur. Cette méthode impose d'avoir des IP fixes (comme les fournissent tous les FAI sérieux), et considère qu'il n'est pas possible de d'usurper une IP publique. Factuellement, cette condition est vérifiée puisque les FAI ne laissent jamais sortir des paquets d'une ligne sans vérifier l'IP source correspond bien au tuyau dans lequel ils transitent. Notre système a été programmé dans le but de supporter pleinement l'IPv6.

L'inconvénient majeur de ce mode d'authentification concerne le cas de l'incendie de la résidence. On peut alors estimer que l'habitant souhaitera rapidement récupérer ses données pour les héberger ailleurs, alors qu'il n'aura plus accès à la ligne qui lui fournissait la bonne IP. Nous nous sommes donc arrangés pour que l'authentification en SSH (qui permet de récupérer ses données) ne se fasse qu'en fonction de la clé publique, que l'utilisateur pourra récupérer sur son courriel externe. Ainsi, il ne pourra plus bénéficier de son réseau - qui disparaîtra de lui-même - mais il pourra réinjecter ses données depuis une nouvelle ligne.

19. Permet de changer la racine des fichiers d'un système, pour lui *faire croire* qu'il évolue dans un autre.
20. http://fxmx86.mine.nu
21. *Secure CoPy*, l'utilitaire de transmission de données de SSH.

7 Conclusion

Outre l'utilisation du projet Chord pour la découverte des nœuds qui reste à implémenter, quelques modules indépendants liés à la préservation de la stabilité du système ne sont pas encore totalement terminés à l'heure de l'achèvement de ce rapport. Cependant, tous les principaux modules fonctionnent parfaitement, et le projet permet dors et déjà de sauvegarder ses données en les distribuant sur un réseau de nœuds, en pair-à-pair et en toute autonomie. Grâce à l'utilisation de libchop, les sauvegardes sont compressées, versionnées, chiffrées et découpées. La restauration à une version précise, depuis une interface web parfaitement intégrée, est également pleinement fonctionnelle. Le peu de choses qui reste à faire a été étudié et documenté, et nous estimons que notre conception suffira pour les implémenter facilement, de la même façon que nous avons réussi à écrire efficacement le cœur du projet grâce à son aide.

La seule étape qui demande encore de la recherche et de la réflexion concerne l'intégration du projet aux boîtiers. Nous travaillons activement dessus actuellement, mais nos recherches n'ont pas encore abouties. Toutefois, nous avons pris soin de mettre toutes les chances de notre côté, en créant une liste de diffusion [22] du côté de la fédération des FAI, dédiée à l'entraide autour du reconditionnement du système de la Neufbox. Déjà 34 personnes (qui recevront ce rapport) se sont inscrites à cette liste, ce qui confirme l'intérêt pour le projet, qui avait déjà été exprimé lors des réunions mensuelles de la fédération.

Le projet a été entièrement conçu pour être le plus simplement possible intégrable à une Neufbox. Une fois que cette dernière étape que nous avons enclenchés sera achevée, la solution que nous proposons aujourd'hui répondra parfaitement au dernier problème de LDN pour diffuser ses boîtiers : avoir l'assurance que les données ne peuvent pas être perdues, sans être contraint de limiter la taille de sa communauté.

Une mise à jour de libchop très récemment confirme que nous avons fait le bon choix en décidant de travailler avec cet outil, qui semble encore plein d'avenir.

Les attentes de plusieurs FAI concernant ce projet laissent à penser que ses perspectives sont grandes. Nous continuerons à développer cette solution, jusqu'à ce que LDN ait la possibilité de distribuer des boîtiers à ses abonnés. Ce projet n'étant pas contraint par l'environnement de la Neufbox, il pourra à terme être utilisé pour faire des sauvegardes entre serveurs, et permettra de constituer des réseaux hybrides.

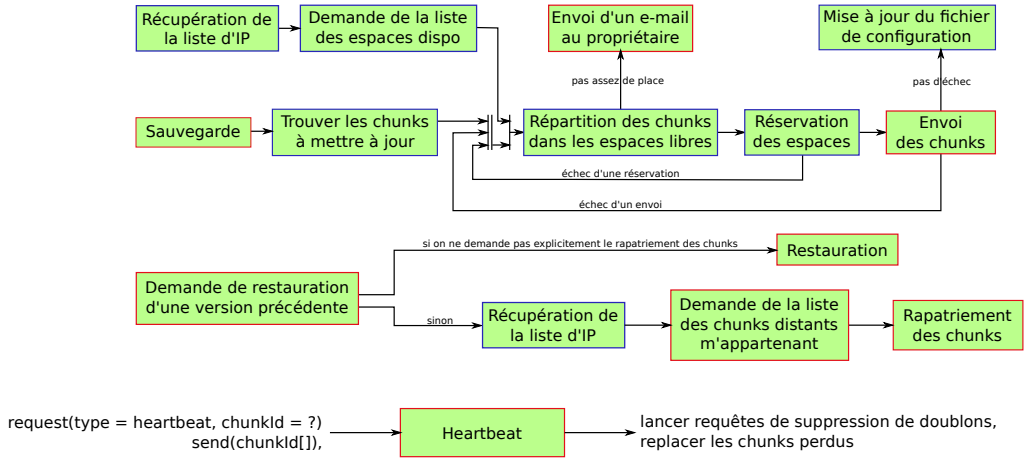
D'un point de vue plus personnel, nous avons apprécié pouvoir prendre le temps de chercher les bonnes solutions, et ne pas être stressés par des obligations précises de résultat. Les méthodes de travail que nous avons décidés de mettre en place pour ce projet étaient inédites pour nous, et nous ont permises de travailler en collaboration, plus efficacement que nous ne l'avions jamais fait dans aucun autre projet.

22. https://lists.ffdn.org/wws/info/neufbox

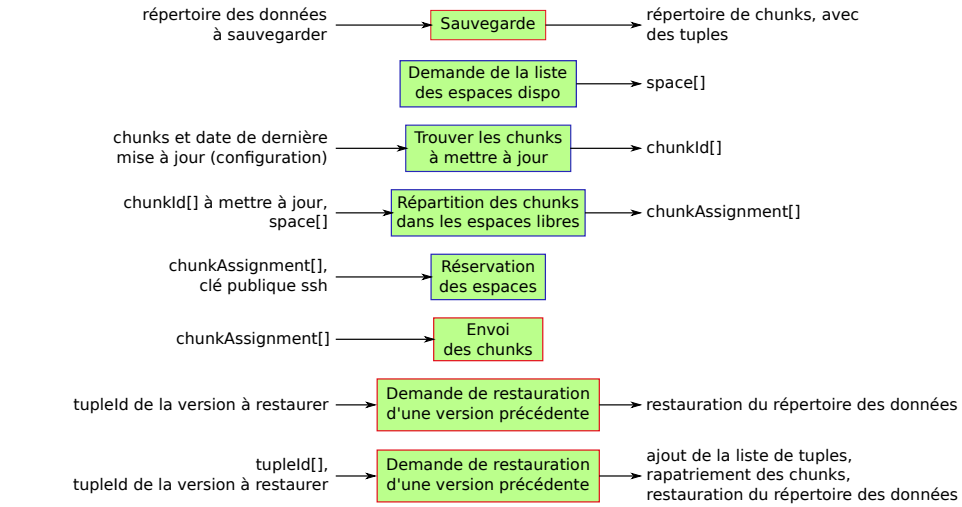
8 Annexes

8.1 Document de conception

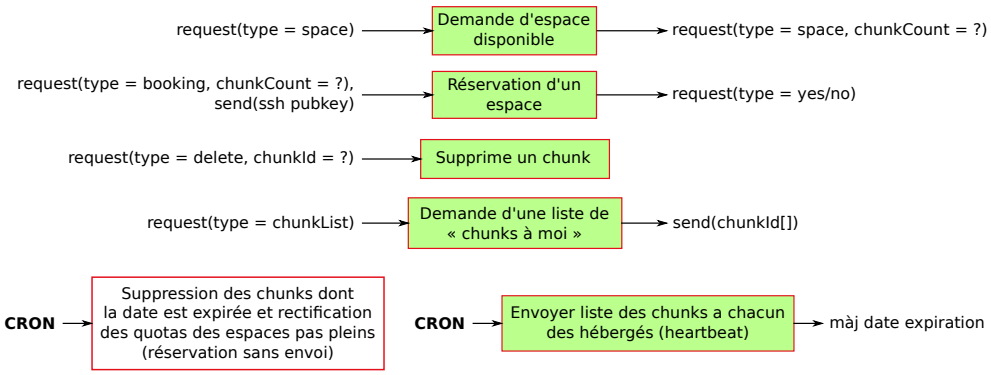
Côté client : algorithme général



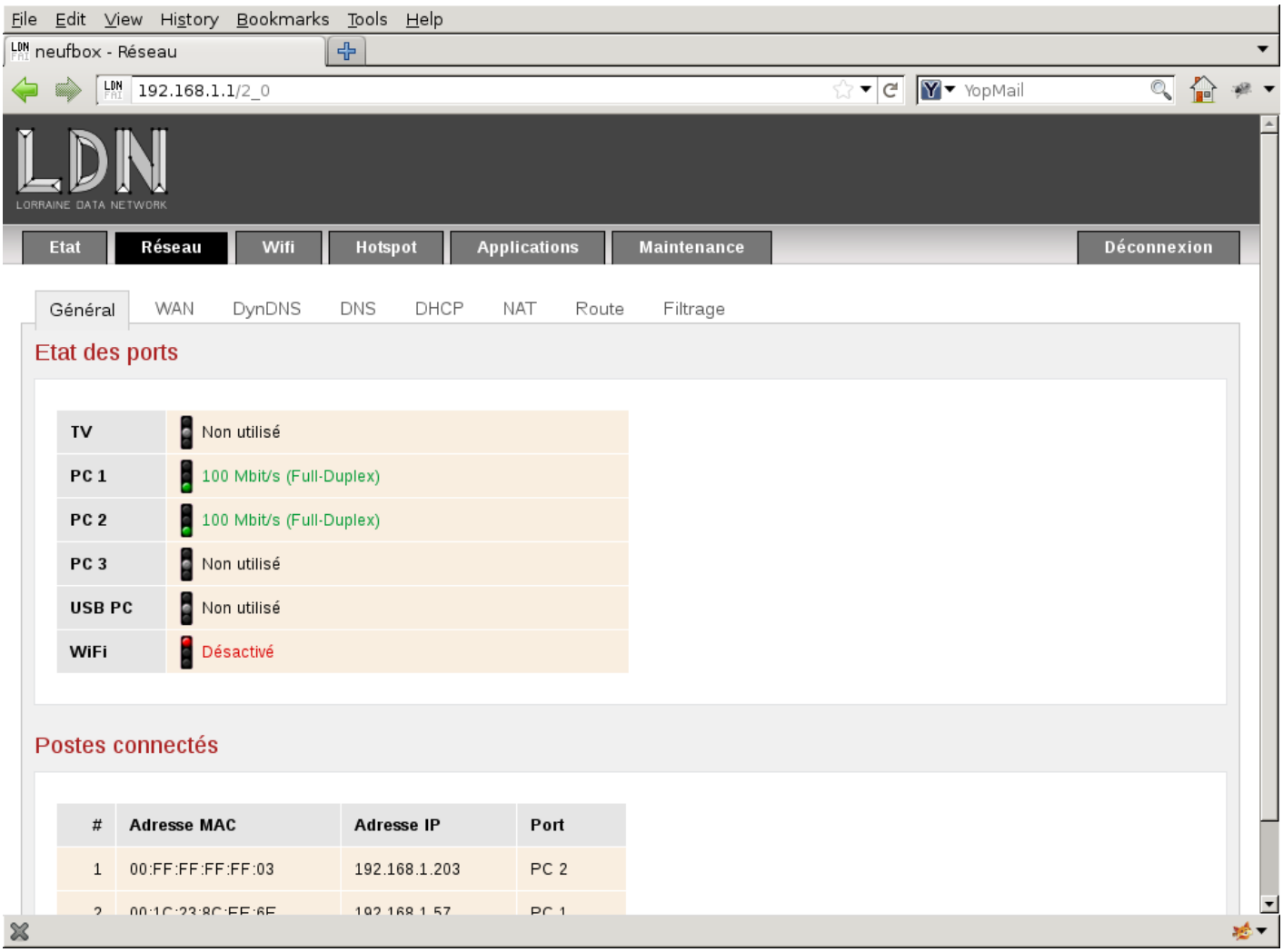
Côté client : entrées/sorties



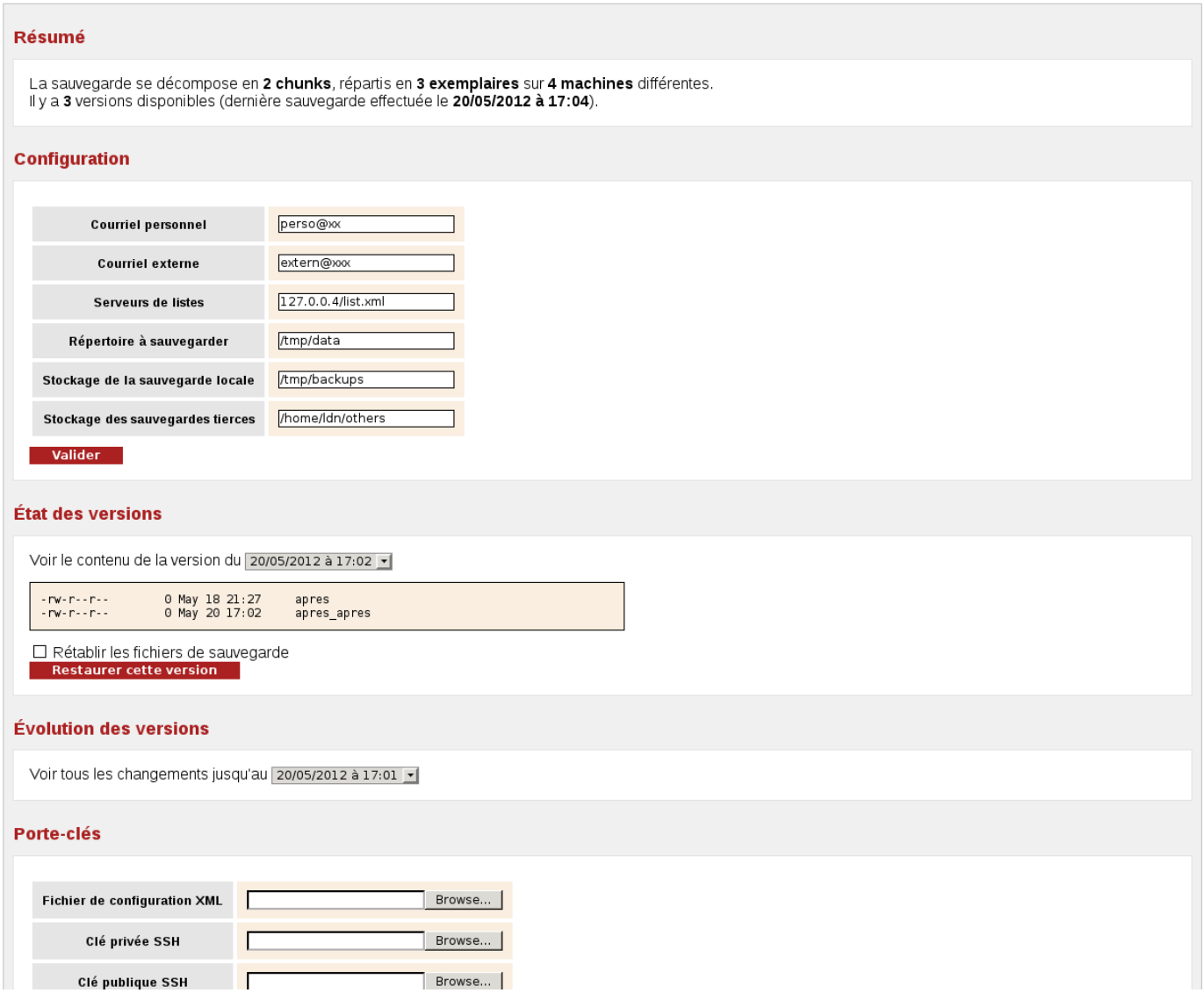
Côté serveur : échange des requêtes



8.2 Interface de la future LDNbox



8.3 Vue globale de l'interface web intégrable



9 Références

Sites utiles

- Société Efixo (disponibilité des sources depuis 2007, libération en 2010)
  Neuf Box 4 Open Source Software
  http://www.efixo.com/neufbox4/freesoftware

- Projet OpenBox (2007)
  Site communautaire dédié à la Neufbox
  http://www.neufbox4.org/blog

- Fxmx86 (2007)
  Projets associés à la modification de Neufbox
  http://fxmx86.mine.nu

- Lorraine Data Network (2010)
  FAI associatif pour la défense d'un Internet libre, neutre et décentralisé
  http://ldn-fai.net

- Fédération French Data Network (2011)
  Fédération de FAI associatifs
  http://ffdn.org

Références retenues

- Ludovic COURTÈS (2010)
  Libchop, a set of utilities and library for data backup and distributed storage
  http://nongnu.org/libchop

- Frans KAASHOEK, David KARGER, Robert MORRIS, Emil SIT, Jeremy STRIBLING (2009)
  Chord : A scalable peer-to-peer look-up protocol for internet applications
  http://pdos.csail.mit.edu/chord

- Emin MARTINIAN (2006)
  DIBS - Since disk drives are cheap, backup should be cheap too
  http://www.mit.edu/~emin/source_code/dibs

- Frank DABEK (2001)
  Wide-area cooperative storage with CFS
  http://research.google.com/pubs/author2377.html