Thèse

Présentée pour obtenir le grade de docteur de l'Ecole d'Ingénieur TELECOM ParisTech

Spécialité : Informatique et Réseaux

Par

Nouha Oualha

Sécurité et coopération pour le stockage de données pair-à-pair

À soutenir le 18 Juin 2009 devant le jury composé de:

Rapporteurs :	**Maryline Laurent-Maknavicius (TELECOM&Management SudParis-France)**
	Pierre Sens (Laboratoire LIP6, Université Paris 6-France)
Examinateurs :	**David Powell (LAAS-CNRS-France)**
	Pascal Urien (Telecom ParisTech-France)
	Roberto Di Pietro (Università di Roma 3-Italie)
Directeur de Thèse :	**Yves Roudier (EURECOM-France)**

Acknowledgements

First of all, I would like to dedicate my thesis to my grand-mother who passed away recently, to my dear parents and sisters.

This PhD dissertation could not be written without the help and guidance of many generous and inspiring people who encouraged me and contributed to my personal and professional development.

I thank Prof. Yves Roudier, my supervisor for his support and guidance during these three and a half years. I was visiting his office for advices daily and he has always been available.

Thanks:

To Prof. Refik Molva for his invaluable willingness to share his exceptional knowledge.

To Dr. Melek Onen and Dr. Pietro Michiardi for their persistent help and support.

To my past and present colleagues for making work a pleasure, especially Ikbal Msadaa, Lamia Romdhani, Aymen Hafsaoui, Majed Haddad, Bassem Zayen, Abdullatif Shikfa, Alessandro Dumunico, Van Hau Pham, Corrado Leita, Slim Trabelsi, Ulrich Bayer...

To all EURECOM staff for their help and assistance.

Thank you all.

Nouha.

Résumé

Chapitre I Introduction

L'intérêt pour les protocoles et algorithmes auto-organisants qui s'est manifesté notamment avec la popularité des services de partage de fichiers (*file sharing*) tels que Napster[1], Gnutella[2], KaZaA [3] et Morpheus [4] concerne maintenant un plus large domaine d'applications. En particulier, il a favorisé l'essor des services de stockage pair-à-pair (Wuala[5], AllMyData Tahoe[6] et UbiStorage[7]). Ces services permettent l'utilisation efficace de tout espace disque libre et inexploité pour construire un système de stockage fiable, disponible, passant à l'échelle et avec des coûts d'entretien réduits.

A. Cas du stockage pair-à-pair

L'avancement des technologies de l'information se traduit par l'accroissement de la quantité de données disponibles et produites dans nos systèmes informatiques. Toutefois, ceci occasionne des défis complexes par rapport à la gestion du stockage des données, stockage qui peut être réalisé en mettant en application des techniques d'auto-organisation. Les données peuvent être stockées d'une manière coopérative chez plusieurs pairs éparpillés dans le réseau pair-à-pair. Ces derniers doivent garder les données stockées jusqu'à ce que leurs propriétaires viennent les réclamer. Un tel système de stockage offre une solution fiable et robuste (*no single point of failure*), sans pour autant nécessiter une infrastructure dédiée et chère come c'est le cas avec des centres de données (*data center*s).

L'approche pair-à-pair a déjà été appliqué à des services de sauvegarde de données ([Cox et Noble 2002] et [Lillibridge et al. 2003]) et à des systèmes de fichiers ([Druschel et Rowstron 2001], [Kubiatowicz et al. 2000] et [Dingledine 2000]). Le stockage pair-à-pair est aussi intéressant pour les réseaux tolérants aux délais (DTNs), puisqu'il peut être utilisé pour livrer les messages des nœuds en dépit de leur mobilité dans le réseau (*store-carry-and-forward paradigm* de [Zhao et al. 2006]). Les services contextuels peuvent tirer un bénéfice du stockage pair-à-pair afin par exemple de déplacer des données relatives à une application pour suivre le mouvement de son utilisateur (*Desktop teleporting* [Bennett et al. 1994], [Pham et al. 2000]). Les données stockées peuvent être aussi contextuelles comme par exemple dépendantes de l'emplacement ([Marmasse et Schmandt 2000], [Huang et al. 1999], [Dey et Abowd 2000], [Beigl 2000]).

B. Les enjeux de sécurité

Une application de stockage pair-à-pair se base sur l'échange volontaire et équitable des ressources de stockage entre des pairs autonomes, pourtant il y a une tension inévitable qui régit ces pairs qui doivent trancher entre leur rationalité individuelle et le bien-être collectif. Cette

[1] http://www.napster.com/
[2] http://www.gnutella.com/
[3] http://www.kazaa.com/
[4] http://www.morpheus.com/
[5] http://wua.la/en/home.html
[6] http://allmydata.org/
[7] http://www.ubistorage.com/

tension qui menace la viabilité de l'application est le résultat d'un dilemme social qui peut mener à une tragédie des biens communs (*tragedy of commons* [Hardin 1968]).

Concevoir un système de stockage pair-à-pair fiable et sûr présente un défi important du fait de la nature ouverte, autonome, et fortement dynamique des réseaux pair-à-pair. Tout effort pour protéger ce type de système devrait assurer les objectifs suivants :

- *Confidentialité et intégrité des données :* Les données traitées dans un système de stockage pair-à-pair sont généralement personnelles (ou appartiennent à un groupe) et sont stockées chez des pairs qui ne sont à priori pas de confiance. C'est la raison pour laquelle les données devraient être protégées lors de leur transmission et de leur stockage chez un pair. Typiquement, la confidentialité et l'intégrité des données peuvent être assurées en utilisant les moyens cryptographiques habituels tels que des méthodes de chiffrement symétriques, et les fonctions de condensation (*hashing*), et de somme de contrôle (*checksum*).
- *Anonymat :* L'anonymat peut être une condition nécessaire pour un certain type d'application de stockage pair-à-pair qui a pour objectif d'empêcher la censure de l'information par exemple. L'anonymat peut se rapporter à l'identité du propriétaire des données stockées, à l'identité du pair de stockage, ou aux détails d'interaction entre les deux. En outre, l'anonymat permet d'éviter des attaques ciblées où l'attaquant vise tous les pairs qui stockent la même donnée afin de l'éliminer complètement du système. Les systèmes de stockage qui visent à fournir l'anonymat utilisent souvent des infrastructures à base de couches anonymes comme le routage d'oignon dans [Goldschlag et al. 1999].
- *Identification :* Dans un environnement distribué et ouvert, il est possible que la même entité physique apparaisse sous différentes identités. Ce problème peut mener à des attaques de type Sybil [Douceur 2002], et ainsi menacer les méthodes de réplication de la donnée qui se basent sur l'idée que les pairs de stockage sont physiquement distincts. Ce type d'attaque ne peut être éliminé qu'avec le déploiement d'une autorité centrale de certification comme démontré dans [Douceur 2002]. Cet objectif peut limiter l'anonymat. Alternativement, l'autorité peut imposer le paiement d'honoraires d'adhésion. Cependant, cette approche réduit la nature décentralisée des systèmes pair-à-pair et où un point d'étranglement. Sans tierce partie de confiance, une autre option est d'appliquer des sanctions à tous les nouveaux venus : un pair peut coopérer avec des étrangers avec une probabilité donnée (comme dans BitTorrent [Piatek et al. 2007]), ou un pair peut joindre le système seulement si un autre pair l'invite [Lesueur et al. 2008]. Autre approche plus appropriée dans un réseau pair-à-pair, les opérations acceptables peuvent être limitées si on observe des liens directs avec trop d'identités éphémères et peu fiables [Yu et al. 2006]. Cette option semble cependant freiner la mise à l'échelle du système et dégrade aussi le bien être social [Feldman et Chuang 2005].
- *Contrôle d'accès :* Le manque d'authentification peut être surmonté par la distribution des clefs nécessaires pour accéder aux données stockées. Des listes de contrôle d'accès ou des capacités peuvent être associées aux données par leurs propriétaires originaux, comme dans [Srivatsa et Liu 2005].
- *Mise à l'échelle :* Le système de stockage pair-à-pair doit pouvoir faire face à la participation d'une grande population de pairs qui y participent (mise à l'échelle horizontale). Puisque la plupart des fonctions importantes du système sont exécutées par les pairs, le système devrait alors pouvoir facilement traiter des quantités croissantes de messages de contrôle d'une complexité accrue pour la gestion des pairs et des ressources. Le système peut être géré par des groupes comme c'est le cas des réseaux sociaux, ce qui réduit la charge supportés par les pairs.

- *Fiabilité des données :* Généralement, la fiabilité d'une donnée est assurée par la redondance de la donnée stockée à plusieurs endroits dans le réseau. Les données peuvent être simplement répliquées. Le facteur de réplication devrait être maintenu pendant la durée entière du stockage. Ceci implique la réparation des données détruites ou corrompues, ce qui peut s'effectuer périodiquement ou peut être déclenché par des événements comme la détection de fautes avec des protocoles de vérification de la donnée à distance. D'autres approches de redondance peuvent être aussi employées, par exemple le codage d'effacement qui fournit le même niveau de fiabilité des données mais avec des coûts de stockage inférieurs à ceux de la réplication.
- *Survie à long terme des données :* La longévité des données stockées dans certaines applications comme la sauvegarde distribuée (*backup*) est très critique. Le système doit s'assurer que les données seront conservées de manière permanente (jusqu'à leur récupération par le propriétaire). Les techniques de redondance des données améliorent la longévité des données, mais ces techniques doivent être régulièrement ajustées pour optimiser la capacité du système. Généralement, la méthode d'adaptation utilisée est basée sur des protocoles de vérification de la présence de la donnée chez le pair de stockage. Par ailleurs, des mécanismes d'incitation à la coopération doivent être employés pour encourager les pairs de stockage à préserver les données.
- *Disponibilité des données :* Les systèmes de stockage doivent assurer que les données stockées sont accessibles et utilisables sur demande par les pairs autorisés. Les vérifications périodiques des données stockées chez les pairs de stockage permettent un contrôle régulier de cette propriété. La connexion intermittente des pairs peut être mitigée en appliquant un « délai de grâce » où les vérificateurs tolèrent l'absence de réponse du pair de stockage pour un nombre défini de défis avant de déclarer que le pair est non coopératif.

Dans cette thèse, nous nous focalisons sur les trois derniers objectifs décrits ci-dessus : comment réaliser un stockage fiable et disponible à long terme dans le contexte d'un système de stockage pair-à-pair à grande échelle. Ces trois objectifs sont souvent ignorés dans les systèmes de partage de fichiers qui suivent plutôt des approches sous obligation de moyens (*best effort*). Cette thèse suggère la nécessité d'effectuer des vérifications cryptographiques périodiquement pour permettre l'évaluation du statut de sécurité des données stockées dans le système et la conception de mécanismes d'incitation à la coopération adaptés qui préservent les propriétés de sécurité des données sur le long terme.

C. Objectifs de recherche

L'étude des systèmes auto-organisants mène vers plusieurs défis de sécurité stimulants. En premier lieu, ces systèmes sont caractérisés par une grande échelle allant de centaines à des milliers de pairs, une grande dynamicité, et un relatif anonymat des pairs participants. Ainsi, la coopération volontaire est difficilement réalisable du fait du manque de confiance. La confiance peut être réalisée d'une manière statique (basée sur l'identité par exemple) ou d'une manière dynamique (auto-organisante). La confiance statique consiste en un rapport de fidélité qui reste le même jusqu'à ce qu'il soit retiré, tandis que la confiance dynamique montre des caractéristiques d'auto-apprentissage (*self-learning*) et d'auto-élargissement (*self-amplifying*). La confiance se construit en se basant sur des évaluations de comportement dans le système et change en conséquence sans interruption.

La dimension temporelle doit être prise en compte. Les interactions coopératives entre pairs sont généralement considérées en tant qu'opérations atomiques, ce qui est une hypothèse acceptable pour une application de routage de paquets dans un réseau ad hoc ou de partage de fichiers dans un réseau pair-à-pair ; ce n'est pas le cas d'une application de stockage ou de

sauvegarde de données. Cette dernière exige un nouveau type de primitives qui permet l'évaluation « ponctuelle » de la coopération des pairs de stockage. Cette primitive vise à vérifier périodiquement la présence des données stockées sur le long terme dans le but de fournir des évaluations à court terme de la coopération des pairs de stockage. En se basant sur ces évaluations, des mécanismes d'incitation à la coopération sont construits pour stimuler la coopération des pairs et assurer l'équité de leurs contributions respectives.

Les mécanismes d'incitation à la coopération supposent des pairs stratégiques et rationnels. Par conséquent, les modèles théoriques les plus adaptés qui permettent de valider ces mécanismes utilisent la théorie des jeux. Il existe un grand nombre de modèles théoriques de jeux qui peuvent façonner le système de stockage pair-à-pair. Nous nous concentrerons en particulier sur les jeux non coopératifs répétés et évolutionnaires.

Chapitre II Architecture : éléments pour un stockage de données pair-à-pair sécurisé

Un système de stockage pair-à-pair s'appuie sur la coopération des pairs pour fonctionner correctement.

Pour permettre une architecture simple et modulable, nous proposons une organisation sous forme de couches. Les couches sont superposés les unes sur les autres et chaque couche peut utiliser des éléments fournis par les couches basses :

- Couche d'infrastructure basique
- Couche de gestion des pairs et des ressources
- Couche de gestion de la confiance et de la coopération
- Couche applicative

A. Couche d'infrastructure basique

Notre travail se concentre en particulier sur le réseau pair-à-pair. Le réseau pair-à-pair est un paradigme de communication qui permet l'échange direct des ressources entre les pairs à la place d'un échange à travers une entité centralisée dans le paradigme client/serveur. Chaque pair peut agir en tant que serveur s'il veut partager des ressources, et en tant que client s'il veut demander des ressources d'autres pairs. Tous les pairs sont égaux et ont les mêmes responsabilités et privilèges. Puisqu'il n'y a pas d'entité centralisée, les coûts administratifs et opérationnels sont réduits, permettant au réseau de contenir une grande population de pairs.

Les objectifs de sécurité nécessaires à réaliser dans un système de stockage pair-à-pair peuvent être assurés avec un environnement de confiance. Ce type d'environnement permet aux utilisateurs d'être confiants sur l'intégrité et la fiabilité de leurs propres dispositifs et d'autres dispositifs sur le réseau. Il fournit un environnement protégé d'exécution qui ne peut pas être manœuvré ni observé par un adversaire. Un environnement de confiance existe dans divers facteurs de forme allant des dispositifs de confiance dédiés dans un réseau aux plateformes de confiance intégrées à des appareils pas forcément de confiance. Une tierce partie de confiance (*trusted third party*) est une entité responsable et admise pour une fonction convenue par tous les utilisateurs. Les fonctions de la tierce partie de confiance peuvent être assurées d'une manière distribuée en utilisant des modules de type TPM (*Trusted Platform Module*) pour cartes à puce. Ces derniers sont des composants matériels passifs et programmables qui possèdent un système d'exploitation. La machine de l'utilisateur peut aussi disposer d'un système d'exploitation de confiance (*trusted operating system*) qui est un système actif conçu pour garantir la confidentialité, l'intégrité et la disponibilité des informations, des systèmes et des

ressources. Les utilisateurs ou les processus sont autorisés à effectuer seulement les actions qui leur sont permises.

B. Couche de gestion des pairs et des ressources

Les systèmes implémentés sur un réseau pair-à-pair sont en général très dynamiques : les pairs peuvent joindre le système et en partir à tout moment. Déployer un réseau de superposition offre un bon substrat pour la gestion des pairs et des ressources du système. Le réseau de superposition peut être réalisé d'une manière centralisée par laquelle la gestion dans son ensemble se fait à travers un serveur centralisé qui garde la métadonnée correspondant aux ressources du système et facilite la découverte et la recherche de ses ressources par les pairs. Le réseau de superposition peut être également décentralisé. Les services de découverte et de recherche des ressources se font par des techniques relatives à des topologies différentes. Dans le cas d'une topologie plate du système, les pairs cherchent par eux-mêmes les ressources qu'ils sollicitent en inondant par exemple le système avec leurs requêtes comme dans Gnutella[8]. La topologie du système peut être aussi hiérarchique come dans FastTrack [Liang et al. 2006] où la gestion du système se fait essentiellement à travers des super-pairs (*super-peers*) qui assistent les pairs ordinaires (*ordinary-peers*) dans leur recherche des ressources. La topologie du système peut aussi être structurée sous forme de tables de hachage distribuées (*distributed hash tables*) qui attribuent uniformément des identités aléatoires aux pairs. Des marques uniques, dites clefs, sont aussi attribuées aux ressources dont les métadonnées seront stockées par les pairs. Ces messages se trouvent dans le même espace d'adressage que les clefs.

Identification des pairs

L'identification des pairs dans le réseau est un enjeu de sécurité très important puisque le système risque des attaques de type Sybil si les pairs sont libres de choisir leur identifiants. [Douceur 2002] démontre que ce type d'attaque est complètement éliminé s'il existe une entité de certification dans le système qui fournit des identifiants fortement liés aux identités réelles des pairs. D'autres alternatives limitent l'attaque sans pour autant l'éliminer et se basent sur des tests de ressources, comme par exemple des puzzles cryptographiques dans [Vishnumurthy et al. 2003] ou la vérification d'adresse IP. Par ailleurs, SybilGuard [Yu et al. 2006] utilise des liens sociaux déjà existants entre les pairs pour détecter des attaquants de type Sybil.

Gestion de la métadonnée

La métadonnée renseigne sur les attributs d'une donnée (par exemple nom, taille, propriété et type), sa structure (par exemple, longueur et champs), son emplacement, ses droits d'accès et les clefs associées, et contient éventuellement une description courte de leur contenu. La métadonnée peut être stockée par le propriétaire de la donnée, rendue disponible par une entité centralisée, ou distribuée aux pairs du réseau qui gèrent cette information à travers un réseau de superposition structuré ou non structuré.

Sélection aléatoire de pair

Dans un réseau de superposition centralisé, la sélection aléatoire des pairs peut être simplement effectuée en choisissant un sous-ensemble aléatoire de la liste des pairs enregistrés. Dans un réseau de superposition non structuré, la sélection aléatoire peut être obtenue en se basant sur la marche aléatoire (*random walk*) [Zhong et al. 2008]. Finalement, un réseau de superposition structuré permet avec une valeur aléatoire dans l'espace d'adressage de sélectionner aléatoirement les pairs qui se trouvent au voisinage de cette valeur.

[8] http://www.gnutella.com/

C. Couche de gestion de la confiance et de la coopération

La confiance entre les pairs peut être réalisée statiquement ou dynamiquement (la Fig. 1 décrit la terminologie utilisée pour la confiance). Dans le premier cas, les pairs ont des rapports de confiance antérieurs basés par exemple sur des liens sociaux existants. Dans les réseaux ami-à-ami (*friend-to-friend*), les pairs interagissent avec les pairs qu'ils connaissent. Turtle [Popescu et al. 2004] est un système anonyme de partage d'informations qui construit un réseau de superposition pair-à-pair sur des relations d'amitié préexistantes entre les pairs. Ces relations d'amitié sont définies comme commutatives, mais non transitives. [Li et Dabek 2006] a proposé un système de stockage sur un réseau ami-à-ami. Comparée à un système ouvert de stockage pair-à-pair, l'approche proposée réduit le taux de réplication des données stockées puisque les pairs sont seulement sujets à des pannes et pas à l'égoïsme ou à la malveillance et l'intérêt est donné plus à la préservation des données qu'à leur disponibilité. Cette approche n'aide cependant pas à construire des systèmes à grande échelle avec une large réserve de ressources.

La confiance peut être assurée en utilisant une autorité de confiance comme dans le cas d'UbiStorage[9]. Ce service propose un système de fichiers qui est basé sur une infrastructure de confiance distribuée établie au-dessus d'un réseau de pairs. En effet, le service distribue des terminaux dédiés, appelés « néobox », aux pairs. Ces terminaux sont utilisés pour stocker en sécurité les données d'autres pairs.

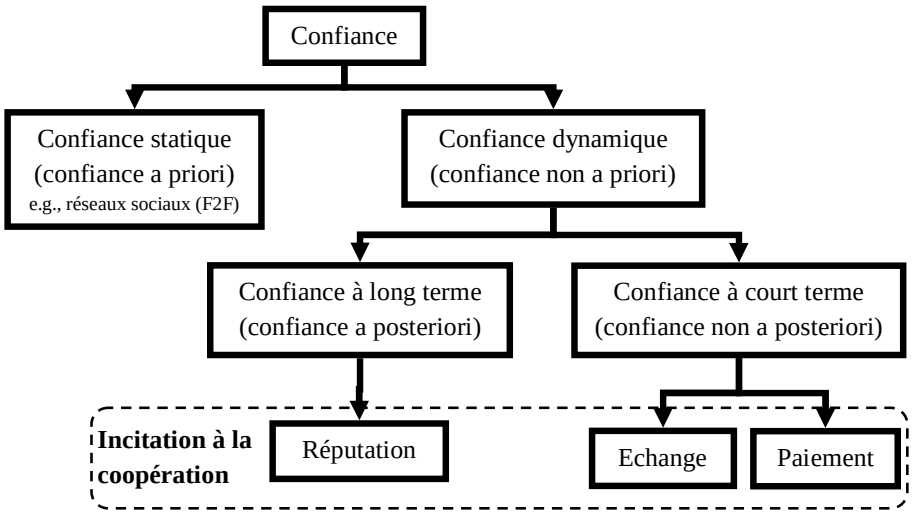


Fig. 1 Taxonomie de la confiance

La confiance dynamique évalue les interactions des pairs et se construit suite à cette évaluation. L'évaluation du comportement des pairs peut être effectuée à différentes fréquences. L'évaluation immédiate du comportement du pair est seulement possible si sa contribution se produit dans un temps très court (atomique) comme pour le routage ad hoc ([Michiardi 2004] et [Buttyán et Hubaux 2003]) ou l'échange de fichiers pair-à-pair. L'opération de stockage s'étend sur une période temporelle généralement longue ; ce qui nécessite l'utilisation de protocoles qui fournissent des preuves de possession de données. En utilisant un de ces protocoles, le pair de

[9] http://www.ubistorage.com/

stockage démontre au vérificateur (peut être le propriétaire) qu'il possède les données stockées chez lui. Ces preuves peuvent être maintenues de manière confidentielle par les vérificateurs comme elles peuvent être distribuées au reste des pairs. La distribution peut se faire d'une manière centralisée ou suivant un réseau de superposition structuré [Kamvar et al. 2003] ou non structuré [Anceaume et Ravoaja 2006]. L'information sur le comportement d'un pair en particulier est utilisée pour déduire si ce pair mérite d'avoir une bonne réputation ou à alternativement être rémunéré pour sa contribution, suivant le mécanisme d'incitation à la coopération en vigueur. Le modèle d'incitation à la coopération peut utiliser un historique des actions passées des pairs (réputation) ou une promesse de récompense ou de punition financière (paiement). Il peut se baser sur l'échange symétrique comme dans le système de partage de fichiers BitTorrent[10]. Dans BitTorrent, les pairs téléchargent des fichiers vers d'autres pairs qui leur fournissent une bande passante élevée (*tit-for-tat*).

Le déploiement d'un environnement de confiance peut permettre la gestion de l'information de la réputation des pairs et même assurer l'échange équitable entre pairs de la rémunération contre contribution dans le cas d'un mécanisme d'incitation basé sur le paiement.

D. Couche applicative

La couche du niveau applicatif est concernée par la gestion individuelle du service installé sur chaque machine. Chaque pair doit stocker les données d'autres pairs du réseau et garantir la disponibilité et la fiabilité du stockage.

Structure multiservice

Il est possible de concevoir une structure générale d'échange de ressources où les pairs peuvent échanger plusieurs types de ressources entre eux. Cette structure s'avère être intéressante dans le cas où les pairs ont des systèmes hétérogènes et des besoins différents. Chaque pair participe donc à une collection de services dont certains sont utilisés pour sa consommation personnelle et d'autres pour sa contribution à la collectivité.

La rémunération (argent réel ou virtuel) peut être considérée comme une contrepartie neutre qui peut être échangée pour n'importe quel service coopératif. Par conséquent, un système basé sur des incitations à base de paiement peut permettre aux pairs d'accéder d'une manière simultanée à des services coopératifs. L'évaluation du comportement des pairs devrait être exécutée séparément et indépendamment pour chaque service. Cependant, la rémunération pour un service rendu peut être effectuée de la même façon pour tous les services. Par exemple, la rémunération peut employer des enchères (comme dans KARMA [Vishnumurthy et al. 2003]) pour faire face à l'effet des changements de l'offre et de la demande sur les prix.

Un système d'exploitation de confiance incorporé dans la machine de chaque pair doît contrôler l'accès du pair aux ressources et aux services et peut également servir pour stimuler ou même forcer le pair à coopérer avec le système d'une manière équitable. L'incitation à la coopération peut se résumer à une différentiation du service reçu par le pair: un pair coopératif aura une bonne qualité de service contrairement à un pair non coopératif. La fonction du système d'exploitation est de permettre l'évaluation impartiale des actions du pair et de modifier ses droits d'accès sur les ressources du système en fonction de cette évaluation. En particulier, la différentiation de service peut miser sur une politique de sécurité contextuelle qui peut être renforcée avec une architecture de sécurité comme dans Flask [Spencer et al. 1999]. Ce type d'architecture permet la révocation systématique des droits d'accès précédemment accordés.

[10] http://www.bittorrent.com/

Chapitre III Vérification de la possession de la donnée à distance

Le premier objectif du stockage de données pair-à-pair est de garantir la survie à long terme des données stockées. Cet objectif exige des primitives particulières qui permettent d'assurer la vérification de cette propriété. Contrairement aux contrôles simples d'intégrité, la vérification de l'intégrité des données stockées doit prendre en considération le fait que le pair de stockage peut être défectueux mais aussi malveillant. De plus, puisque les données sont stockées à distance, la vérification ne devrait pas exiger le transfert des données dans leur intégralité. La dynamicité du système due notamment à la connexion intermittente des pairs suggère de distribuer la charge de la vérification sur plusieurs pairs dans le réseau. Il est nécessaire que ces vérificateurs ne gardent pas toute la donnée pour la vérification mais plutôt une information de petite taille (comparée à la donnée). Les vérificateurs ne sont pas forcément de confiance, donc l'information qu'ils stockent pour la vérification ne doit pas être une information secrète par rapport au pair de stockage. En tenant compte de ces dernières conditions, le protocole de vérification est donc délégable.

A. Objectifs de sécurité

Le mécanisme de vérification doit adresser les attaques potentielles suivantes auxquelles le système de stockage est exposé :

- *Détection de destruction de données :* La destruction ou la corruption des données peut être due à un pair défectueux ou malhonnête. Le protocole de vérification doit assurer cette fonction.
- *Résistance à la collusion :* Les pairs possédant les répliques de la même donnée peuvent entrer en collusion en détruisant toutes les répliques sauf une qui est utilisée pour répondre correctement aux vérificateurs. Une solution contre ce type de collusion se base sur la personnalisation des répliques de données : le propriétaire conserve des répliques qui sont personnalisées pour chaque pair de stockage.
- *Prévention contre l'attaque par le milieu (man-in-the-middle):* L'attaquant peut prétendre être à la fois le pair propriétaire de la donnée et le pair qui garde cette donnée en se plaçant entre les deux lors d'un échange de messages. La réplication peut être perturbée par cette attaque puisque le propriétaire risque de stocker sa donnée chez le même pair. Une manière typique de résoudre ce problème est d'ajouter une étape d'engagement dans les messages échangés entre les pairs de telle manière que l'attaquant ne puisse pas ouvrir ou produire ces engagements.
- *Prévention contre le déni de service (denial of service) :* Un pair de stockage peut être inondé de requêtes de vérification. Un attaquant peut aussi rejouer un message de vérification ou de réponse valide afin de perturber le processus de vérification.

Ce chapitre présente trois protocoles de vérification qui cherchent à répondre aux exigences de construction, de performance et de sécurité discutées ci-dessus en proposant différent compromis.

B. Protocole de vérification Probabiliste

Le premier protocole de vérification vise à une détection de destruction de donnée probabiliste. La donnée est stockée sous forme de fragments avec leurs signatures respectives qui sont générées par le propriétaire de la donnée. A chaque opération de vérification, le vérificateur demande un fragment dont l'index est choisi aléatoirement avec sa signature. Lorsqu'il reçoit la réponse du pair de stockage, le vérificateur teste si la signature correspond au fragment demandé. Le vérificateur réalise ce test en utilisant la clé publique utilisée lors de la signature des fragments.

Le vérificateur évalue la présence d'un fragment chez le pair de stockage. Mais, puisque le fragment est choisi aléatoirement, le pair de stockage doit garder toute la donnée stockée pour pouvoir répondre correctement à toutes les requêtes du vérificateur. Par contre, si le pair de stockage détruit une fraction d des fragments, le vérificateur doit effectuer des vérifications multiples pour réaliser une certaine probabilité de détection $p_{detection}$. Le nombre de vérifications c est dérivé comme suit:

$$c=\lceil log_{1-d}(1-p_{detection}) \rceil$$

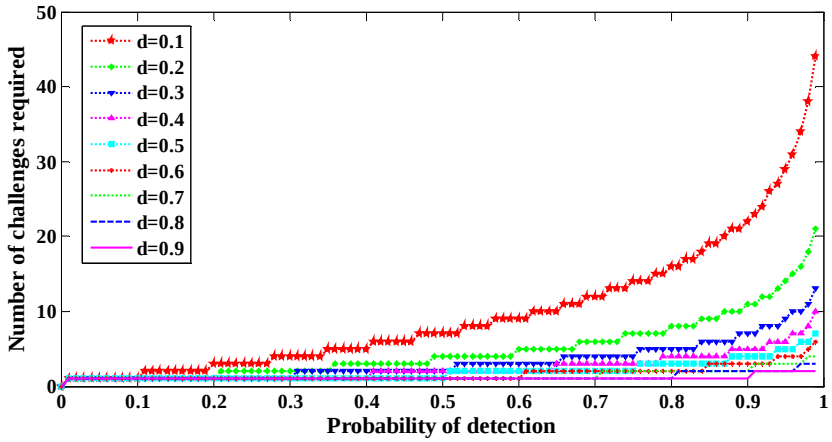


Fig. 2 Nombre de vérifications nécessaire pour assurer une certaine probabilité de détection de la destruction

La Fig. 2 démontre que même avec un nombre de vérifications c modeste, il est possible de garantir une probabilité de détection de la malveillance du pair de stockage proche de 100%.

C. Protocole de vérification déterministe restreint

Le second protocole de vérification vise à une vérification à distance sur toute la donnée stockée mais avec un nombre limité d'opérations de vérification réalisables. Le protocole proposé se base sur la notion d'unicité de la solution du problème d'interpolation de polynôme.

Le propriétaire de la donnée génère des polynômes à partir de valeurs déduites des fragments de la donnée et des valeurs aléatoires pour assurer que les défis construits soient aussi aléatoires. Il utilise pour cela la formule de Lagrange comme méthode d'interpolation de polynôme. A partir des polynômes générés, le propriétaire calcule des points particuliers. Ces points sont envoyés au vérificateur avec les valeurs aléatoires utilisées constituant ainsi une métadonnée pour la vérification. Le pair de stockage qui possède les fragments de la donnée reçoit périodiquement du vérificateur une valeur aléatoire qu'il utilise pour générer un

polynôme. Ce polynôme est construit à partir des points constitués par les fragments de la donnée et le point qui consiste en la valeur aléatoire reçue comme défi. Le pair de stockage produit une réponse unique au défi en calculant un point particulier du polynôme généré. Cette réponse est ensuite envoyée au vérificateur qui la compare à la valeur qu'il stocke comme métadonnée.

Le vérificateur peut effectuer un nombre limité de test sur la donnée stockée à distance. Ce nombre est défini par le nombre de défis pré-calculés et stockés chez le vérificateur. Augmenter la fréquence de test contraint le vérificateur à stocker une métadonnée de plus grande taille. La taille d'un défi est déterminée par la taille (maximale) d'un fragment de la donnée. Diminuer la taille des fragments (i.e., augmenter le nombre de fragments de la donnée) diminue en même temps la taille de la métadonnée stockée chez le vérificateur. Cependant, ceci affecte la sécurité du schéma puisque les points à partir desquels le polynôme est généré par interpolation deviennent aussi de petite taille et donc la solution produit beaucoup plus de faux positifs.

D. Protocole de vérification déterministe

Le protocole de vérification proposé se base sur la cryptographie des courbes elliptiques. Le propriétaire de la donnée génère une courbe elliptique sur l'ensemble $\mathbb{Z}_n$ avec n est choisi comme un modulo RSA tel que $n=pq$ où p et q sont deux nombres premiers. L'ordre de la courbe est gardé secret par le propriétaire. [Koyama et al. 1991] démontre que résoudre l'ordre sans connaître p et q revient à factoriser le modulo RSA n qui est conjecturé comme étant un problème difficile. La donnée est associée à une valeur entière d. Le propriétaire génère la métadonnée qui consiste en un point $T=d.P$ où P est un générateur de la courbe. Cette métadonnée est stockée chez le vérificateur. Pour vérifier la présence de la donnée chez un pair de stockage, le vérificateur lui envoie le point $Q=r.P$ avec r un entier choisi aléatoirement. Le pair de stockage répond à ce message en calculant $R=d.Q$. Le vérificateur teste finalement si cette égalité $R=r.T$ est vérifiée.

Le pair de stockage ne peut qu'utiliser toute la donnée pour pouvoir répondre correctement au vérificateur car il devrait sinon déduire la valeur de r de $r.P$ ou connaître l'ordre de la courbe elliptique N_n pour garder juste $d \bmod N_n$. Le premier cas correspond au problème du logarithme discret d'une courbe elliptique et le second au problème de factorisation d'un modulo RSA ; les deux problèmes sont conjecturées comme étant difficiles à résoudre.

Dans ce protocole, le pair de stockage doit faire une opération de multiplication sur toute la donnée qui peut être assez coûteuse en termes de ressources de calcul et de temps. Pour alléger cette opération, on propose de diviser la donnée en fragments et d'augmenter la taille de la métadonnée stockée chez le vérificateur. Chaque élément de la métadonnée correspond donc à un fragment de la donnée plutôt qu'à toute la donnée. Ainsi, le pair de stockage effectue une opération de multiplication sur juste un fragment. Pour permettre la vérification déterministe de la donnée, le vérificateur doit constituer en plus de r un générateur de valeurs aléatoires (*seed*) qui sont utilisées pour relier les points obtenus de la multiplication de fragments avec le point inclus dans le défi du vérificateur.

La littérature regorge de propositions pour des schémas qui permettent la vérification de l'intégrité des données à distance ([Ateniese et al. 2007], [Deswarte et al. 2004], [Sebé et al. 2007], [Filho and Barreto 2006], [Schwarz and Miller 2006], [Chang and Xu 2008], [Juels and Kaliski 2007]). Ces propositions sont assez prometteuses en termes de performance, sauf qu'aucun de ces protocoles ne suggère la délégation de la tâche de vérification à plusieurs pairs pas forcément de confiance (même si certains protocoles sont délégables). Notre protocole est le seul qui est construit autour de cette propriété qui est d'un grand intérêt pour un réseau pair-à-pair dynamique.

Chapitre IV Stockage et maintenance sécurisés de données pair-à-pair

Les protocoles de vérification de données à distance permettent au vérificateur de détecter (de manière déterministe ou probabiliste) si les données stockées sont détruites ou non. Afin de préserver la fiabilité des données dans le système, la détection de toute destruction ou corruption de ces dernières devrait déclencher leur restauration. Cette charge ne peut pas être accomplie seulement par le propriétaire des données, puisqu'il ne participe pas souvent à la vérification. Les vérificateurs et les pairs de stockage devraient plutôt coopérer pour restaurer les données en générant une nouvelle copie des données qui est stockée chez un nouveau pair. Cette nouvelle copie doit être personnalisée pour le nouveau pair de stockage ; en outre la génération de la nouvelle copie ne doit pas exiger la transmission des données plusieurs fois de suite notamment le transit à travers un vérificateur.

Dans cette section, nous présentons une nouvelle méthode de stockage et de maintenance des données qui se base sur le protocole déterministe proposé précédemment et qui permet de restaurer les données détruites sans avoir recours au propriétaire.

A. Attaques

Les différentes attaques auxquelles le protocole de stockage et de maintenance de données est exposé sont détaillées dans la section précédente relative aux attaques contre un protocole de vérification de données à distance. Notre proposition introduit cependant de nouvelles menaces en particulier liées à la phase de restauration:

- *Attaques en Déni-de-Service (DOS) :* Les vérificateurs malveillants peuvent inonder le réseau avec des messages inutiles pour la réparation. Afin d'empêcher ce type d'attaques, un seuil *t* de vérificateurs honnêtes est défini : il devrait y avoir au moins un nombre *t* de vérificateurs qui détectent un problème de destruction de données dans la phase de vérification avant de produire une nouvelle copie des données.
- *Données fausses :* Durant la phase de réparation, les pairs de stockage peuvent tricher en effectuant la régénération de données fausses. Les vérificateurs peuvent également jouer un rôle dans ce type d'attaques.

B. État de l'art des approches existantes

Le protocole de stockage et de maintenance des données devrait consister en cinq phases: les pairs de stockage potentiels sont élus par le propriétaire durant une phase de sélection, ces pairs stockent les données du propriétaire durant la phase de stockage. Le propriétaire nomme alors des vérificateurs pour vérifier l'intégrité et la présence des données stockées durant la phase de délégation et ces vérificateurs effectuent périodiquement cette tâche durant la phase de vérification. Si les vérificateurs détectent la destruction ou la corruption des données, la phase de réparation est activée durant laquelle les vérificateurs produisent une nouvelle copie des données avec l'aide des pairs de stockage encore présents dans le système.

Sélection : Le but de cette phase est de choisir un ensemble de pairs qui peuvent maintenir la fiabilité et la disponibilité des données. Il y a deux techniques possibles pour la sélection des pairs de stockage. Une sélection discriminatoire détermine les pairs d'une manière spécifique par exemple parce qu'ils satisfont une contrainte ([Dingledine 2000]) ou partagent des caractéristiques identiques à celles du propriétaire ([Toka et Michiardi 2008]). En revanche, la sélection aléatoire est généralement employée pour sa simplicité puisqu'elle consomme moins de bande passante par pair. TotalRecall [Bhagwan et al. 2004] se base sur des tables de hachage distribuées (*distributed hash tables*) pour choisir aléatoirement les pairs de stockage. [Godfrey

et al. 2006] a analysé les stratégies de sélection de pair et a prouvé l'intérêt de la sélection aléatoire. Après que les pairs de stockage sont choisis, le propriétaire peut directement les contacter. Il existe plusieurs techniques pour limiter les attaques de type Sybil [Douceur 2002], (se référer à [Levine et al. 2006]) par exemple les pairs qui joignent le système devraient en premier lieu fournir quelques ressources (crypto-puzzles dans [Vishnumurthy et al. 2003]).

Stockage : Une fois que des pairs ont été sélectionnés pour le stockage par le propriétaire, ce dernier envoie ses données à ces pairs. La disponibilité des données peut être assurée avec de la redondance. Avec la réplication, une simple copie des données est distribuée à chaque pair choisi. Par contre avec le codage d'effacement (*erasure codes*), les données sont divisées en plusieurs blocs qui vont produire des blocs supplémentaires pour permettre la reconstruction des données à partir d'un nombre de blocs seuil. La réplication, qui a été la plupart du temps employée dans les tables de hachage pour sa simplicité, offre un compromis moins intéressant entre les frais de stockage et de bande passante pour la maintenance et la tolérance aux fautes par comparaison aux codes d'effacement. C'est pourquoi, il y a plusieurs systèmes de stockage qui ont opté pour le codage d'effacement comme Wuala[11], AllMyData Tahoe[12], UbiStorage[13], et TotalRecall [Bhagwan et al. 2004]. Dans le cas de la réplication, puisque la taille des données est en général grande, les pairs de stockage peuvent entrer en collusion et tricher en stockant une seule copie des données. La personnalisation de chaque copie pour son détenteur a été présentée comme une solution à cette menace (comme présenté précédemment dans la description des protocoles de vérification, ainsi que dans [Lillibridge et al. 2003]). Ce type de collusion peut également surgir avec le codage d'effacement quoiqu'il devienne problématique seulement si le nombre de pairs en collusion excède le nombre de blocs originaux.

Délégation : Comme précédemment décrit, le protocole de stockage devrait assurer que les données sont toujours disponibles. Les réseaux pair-à-pair étant très dynamiques, le propriétaire n'est pas toujours en ligne ce qui implique que la vérification de données doit encore être assurée par des délégués du propriétaire. Le propriétaire fournit à ses délégués des métadonnées qui sont des informations sur les données stockées, et qui servent comme base à la vérification à distance.

Vérification : Des protocoles cryptographiques permettent aux pairs de stockage de prouver à distance l'intégrité des données qu'ils stockent (par exemple, les protocoles de vérification proposés précédemment, [Deswarte et al. 2004], [Sebé et al. 2007], et [Ateniese et al. 2007]). Cependant le manque de réponse de la part d'un pair de stockage est ambiguë parce qu'il ne permet pas de savoir si le pair est défaillant ou malveillant, ou bien s'il est juste déconnecté et peut revenir avec les données intactes. Ceci peut être contourné en considérant un certain délai au cours duquel le vérificateur défie le pair de stockage plusieurs fois avant de décider que ce pair est malveillant.

Réparation : Détecter qu'un des pairs de stockage a triché doit déclencher une opération de restauration afin d'assurer la disponibilité des données. Etant donné la nature dynamique des réseaux pair-à-pair, une telle opération ne peut pas se baser seulement sur l'effort du propriétaire qui peut être déconnecté lors de la détection de la destruction des données. Cette opération doit plutôt être effectuée par les vérificateurs et les pairs de stockage qui gardent encore les données stockées. Les résultats de simulation de [Bhagwan et al. 2004] démontrent que la réparation retardée des données détruites (*lazy repair*) est plus efficace en termes de compromis entre la disponibilité des données et les coûts d'une telle opération que la réparation immédiate (*eager repair*) pour des données de grande taille et un système très dynamique.

[11] http://wua.la/en/home.html
[12] http://allmydata.org/
[13] http://www.ubistorage.com/

C. Protocole de stockage et de maintenance de données basé sur le codage d'effacement

Le protocole proposé emploie le protocole de vérification déterministe basé sur les courbes elliptiques proposé à la section III.D. Il suggère aussi l'utilisation du codage d'effacement linéaire et aléatoire [Acedański et al. 2005]. Avec un tel codage, les entrées de la matrice génératrice des blocs codés sont choisies aléatoirement.

Dans le protocole proposé, les pairs de stockage sont sélectionnés aléatoirement. Ces pairs vont ensuite stocker les blocs $\{b_i\}_{1\leq i\leq k+m}$ qui sont des blocs codés par le propriétaire avec un codage d'effacement linéaire et aléatoire sur $\mathbb{Z}$ et utilisant les blocs originaux des données $\{d_i\}_{1\leq i\leq k}$. Le propriétaire choisit aussi des vérificateurs qui sont assignés chacun à un ou plusieurs pairs de stockage et vont donc recevoir une métadonnée correspondant au bloc stocké par le pair de stockage. Par exemple, le vérificateur assigné au pair qui stocke b_i reçoit la métadonnée $T_i=b_i.P$. Chaque vérificateur teste l'intégrité et la présence du bloc d'une manière périodique en se basant sur le protocole de vérification décrit au III.D de ce chapitre. Si au moins *t* vérificateurs détectent un problème chez un pair de stockage, ils décident alors de reproduire un nouveau bloc et de le stocker chez un nouveau pair. La décision de déclencher la phase de réparation revient donc à plusieurs vérificateurs pour éviter des attaques de déni de service (*flooding attack*). Les vérificateurs se mettent d'accord sur un nombre aléatoire *s* qui va être utilisé comme un générateur de coefficients aléatoires $\{c_i\}_{1\leq i\leq k}$. Ils sélectionnent aussi un nouveau pair qui va recevoir *s* avec un nombre *k* de blocs provenant des pairs de stockage restants. Le nouveau pair reproduit un bloc codé *b*' en utilisant les blocs reçus et *s* :

$$b' = \sum_{l=1}^{k} c_l \times b_{t_l}$$

Ce nouveau bloc peut s'écrire aussi en fonction des blocs originaux ($\alpha_{i,j}$ est une entrée de la matrice génératrice utilisée par le propriétaire):

$$b' = \sum_{j=1}^{k} \left(\sum_{l=1}^{k} c_l \times \alpha_{t_l,j} \right) \times d_j$$

Le nouveau bloc est donc bel et bien un bloc codé. La génération du nouveau bloc a nécessité la transmission de *k* blocs ; bien que ceci puisse encore être réduit en utilisant par exemple le codage d'effacement hiérarchique [Duminuco and Biersack 2008]. Les vérificateurs qui vont être responsables de ce nouveau pair reproduisent une nouvelle métadonnée *T*' à partir des métadonnées des autres vérificateurs et de *s* :

$$T' = \sum_{l=1}^{k} c_l \times T_{t_l}$$

Cette métadonnée peut s'écrire aussi en fonction du nouveau bloc. En effet,

$$T' = \sum_{l=1}^{k} c_l \times T_{t_l} = \sum_{l=1}^{k} (c_l \times b_{t_l}).P = b'.P$$

Donc, *T*'est une métadonnée de vérification pour le nouveau bloc *b*'. Pour éviter que les pairs n'envoient des informations fausses, sous forme de blocs ou de métadonnées, chaque

information doit être accompagnée par la signature du propriétaire. Le protocole peut même utiliser des signatures homomorphiques (par exemple, une signature algébrique [Schwarz and Miller 2006]) pour permettre au nouveau pair de reproduire une nouvelle signature pour le nouveau bloc généré pour attester de sa validité. Ce type de signature permet la vérification suivante :

$$sign_{owner}\left(b' = \sum_{l=1}^{k} c_l \times b_{t_l}\right) = \prod_{l=1}^{k} sign_{owner}(b_{t_l})^{c_l}$$

Le protocole de stockage et de maintenance de données à distance proposé est auto-organisant puisqu'il fait participer les vérificateurs et les pairs de stockage et non plus le propriétaire. La distribution de la plupart de ses fonctionnalités à ces pairs permet de limiter à la fois la connexion intermittente des pairs et leur malveillance potentielle.

Chapitre V Incitations à la coopération basées sur l'audit

Le protocole de vérification de présence de la donnée à distance constitue une primitive d'évaluation du comportement des pairs de stockage, qu'on nomme audit. A partir de cet audit, des mécanismes d'incitation à la coopération peuvent être établis pour générer de la confiance dynamique entre les pairs. On distingue des approches basées sur la réputation et d'autres basées sur la rémunération.

A. Approche de réputation

L'approche de réputation estime le degré de confiance des pairs en s'appuyant sur l'expérience et l'observation de leurs comportements passés.

Attaques

Les pairs ne sont pas nécessairement honnêtes et peuvent tromper le système de réputation pour gagner un avantage personnel non mérité.

- *Mensonge :* Un menteur est un pair qui dissémine des observations incorrectes sur d'autres pairs pour augmenter ou diminuer leur réputation. Les menteurs peuvent s'entendre et conspirer contre un ou plusieurs pairs dans le réseau en leur affectant injustement une mauvaise réputation ou au contraire en affectant une réputation excessivement élevée aux membres de leur groupe.
- *Collusion entre le propriétaire et le pair de stockage :* La collusion vise à augmenter la réputation du pair de stockage chez les vérificateurs honnêtes. Le propriétaire stocke des données factices chez le pair de stockage.
- *Collusion entre le pair de stockage et le vérificateur :* Le but d'une telle collusion est d'augmenter la réputation du pair de stockage chez le propriétaire sans pour autant garder sa donnée. L'effet de ce type de collusion est limité grâce à la distribution de la tâche de vérification à des pairs multiples ; le propriétaire peut se fier à l'ensemble (par exemple à travers un vote) de leurs résultats comme il peut finalement vérifier par lui-même le stockage.
- *Blanchissement (whitewashing):* les pairs peuvent sortir du système et le rejoindre plus tard avec une nouvelle identité afin d'effacer leurs forfaits.

- *Attaque de type Sybil :* Si les pairs peuvent produire des nouvelles identités à volonté, ils peuvent employer certaines d'entre elles pour augmenter la réputation des autres.

Description

Nous proposons que les pairs soient organisés en groupes où seules les interactions intra-groupes sont autorisées. Ainsi, les pairs établissent une estimation rapide de la réputation des autres membres de groupe. Les groupes de pairs sont créés d'une façon centralisée par une autorité (comme [Lillibridge et al. 2003]) ou décentralisée qui misent sur des protocoles de distribution de clef de groupe (par exemple, [Lee et al. 2006], [Lesueur et al. 2007]).

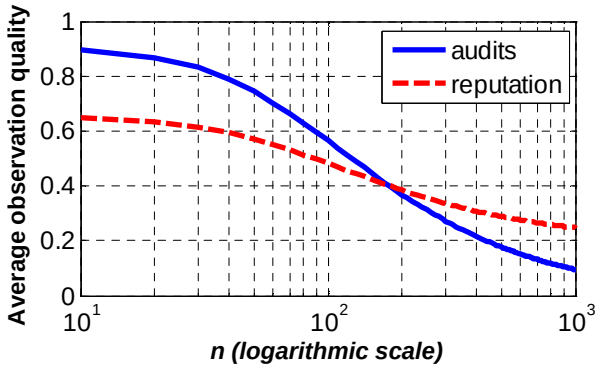


Fig. 3 La qualité d'observation en variant le nombre de pairs dans le groupe pour notre approche (*audits*) et une approche de réputation indirecte (*reputation*).

Le modèle de confiance est basé sur les listes blanches (*white-listing*) qui sont similaires à la stratégie d'œil pour œil dans BitTorrent [Piatek et al. 2007] sauf que les vérificateurs tiennent en plus en compte les résultats de vérification des données des autres pairs. Les pairs inconnus d'un pair particulier sont ajoutés à sa liste blanche d'une manière probabiliste. Chaque pair accepte de servir seulement des pairs inclus dans sa liste blanche.

Les pairs sont structurés sur une table de hachage distribuée dont on suppose qu'elle offre une recherche de clefs sécurisée ([Sit and Morris 2002] and [Castro et al. 2002]). Pour prévenir des collusions potentielles entre pairs visant à tromper le système de réputation, la sélection des vérificateurs et des pairs de stockage se fait d'une manière aléatoire dans la table de hachage.

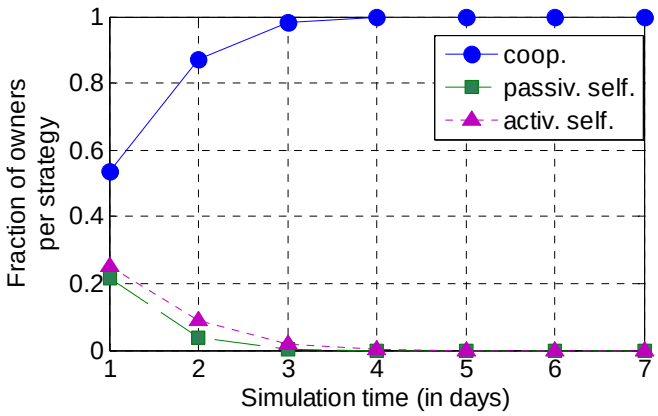


Fig. 4 Ratio des propriétaires en fonction de leur stratégie avec une composition initiale : 40% coopérateurs, 30% pairs égoïstes passives (*passively selfish*), 30% pairs égoïstes actifs (*actively selfish*).

La qualité de l'observation a été calculée analytiquement pour un système utilisant l'approche basée sur l'audit et une approche de réputation traditionnelle basée sur les recommandations. La Fig. 3 démontre que l'approche proposée est plus adaptée à un ensemble de pairs de petite taille puisqu'il affiche une bonne qualité d'observation comparé à une réputation typique qui se base sur des informations indirectes.

Le système de stockage qui utilise l'approche basée sur l'audit a été simulée dans un environnement constitué de pairs avec des stratégies comportementales persistantes. La Fig. 4 qui est un résultat de cette simulation illustre le filtrage des pairs égoïstes du système de stockage en se basant sur notre approche de réputation. Après cette phase, les pairs aptes à stocker des données dans le système sont les seuls pairs coopératifs.

B. Approche de paiement

L'approche de paiement proposée combine la surveillance périodique du stockage de données aux paiements des pairs qui les stockent et des vérificateurs.

Attaques et problèmes

Les pairs doivent participer au système conformément au protocole de paiement ; cependant les pairs peuvent se conduire d'une manière malhonnête.

- *Attaque de type Sybil :* l'attaquant peut tromper le système en s'aidant avec plusieurs identités générées par lui-même. Par exemple, il peut abuser des pairs de stockage en refusant de les payer et en prétextant de quelques vérificateurs qui les as fabriqués pour justifier son comportement.
- *Personnification :* Un pair ne doit pas être capable de personnifier un autre pair, parce que sinon il peut utiliser son argent.
- *Contrefaçon :* Des pairs sont généralement payés avec des jetons (argent virtuel, crédit, chèque, etc.). La contrefaçon se résume à reproduire frauduleusement un jeton.
- *Double dépense :* Le jeton peut être dépensé numériquement une ou plusieurs fois. Il y a deux solutions à ce problème : le bénéficiaire vérifie la validité du jeton avec la banque à chaque fois qu'il est payé, ou bien le fait de dépenser un jeton plusieurs fois expose l'identité de l'attaquant.
- *Échange équitable :* Les protocoles d'échange équitable permettent de garantir que deux parties échangent un service contre paiement sans qu'aucune partie ne gagne un avantage sur l'autre.
- *Famine :* La famine est l'incapacité d'un pair de participer au système parce qu'il n'a plus de jetons à dépenser ([Weyland et al. 2005]).

Description

Notre approche de paiement se base sur KARMA [Vishnumurthy et al. 2003]. KARMA propose de substituer à la banque (autorité de confiance) un ensemble de pairs aléatoirement attribués pour chaque pair, appelés *bank-set*. Ces banques réparties sont collectivement responsables d'augmenter et de diminuer le solde d'argent des pairs auxquels ils sont assignés. Les paiements se font sous forme de chèques électroniques certifiés par les banques.

Avant de joindre le système, les pairs doivent résoudre un puzzle cryptographique. Ceci contrecarre les attaques de type Sybil contre le système de stockage.

Les pairs sont donc organisés dans une table de hachage distribuée dont le service de recherche est supposé sécurisé. Les pairs de stockages ainsi que les vérificateurs sont choisis aléatoirement dans la table pour limiter des collusions potentielles entre eux.

Le calcul des prix du stockage de donnée et de la vérification s'effectue sous forme d'enchères afin d'atténuer les phénomènes de famine. Les prix sont calculés en fonction de la quantité d'argent que le pair possède : un pair qui a beaucoup d'argent propose des prix élevés alors qu'un pair pauvre propose des prix bas pour avoir plus de chance d'être choisi.

Une simulation du système de stockage utilisant l'approche de paiement a été réalisée. Le résultat de la simulation est décrit dans la Fig. 5 qui démontre qu'avec notre approche basée sur l'enchère, le système continue de fonctionner pendant une longue durée. Ceci est du au fait que les pairs ont un risque réduit de tomber en famine.

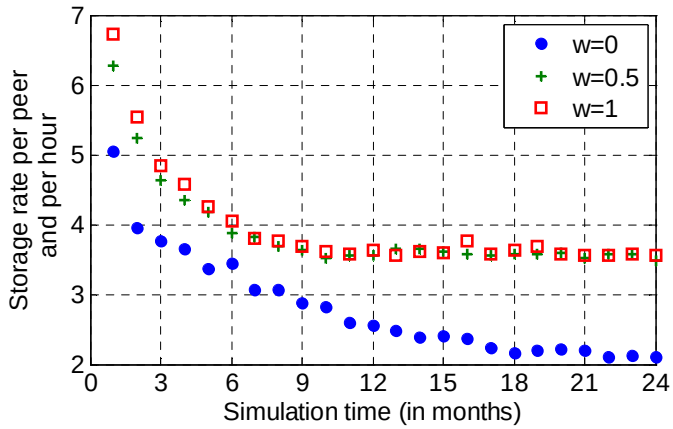


Fig. 5 Quantité de données stockées dans le système en variant le poids de l'enchère *w* (*w*=0 signifie pas d'enchère). Composition initiale : 40% coopérateurs, 30% pairs égoïstes passives, 30% pairs égoïstes actifs.

Puisque le stockage est une opération de longue haleine, on propose de mettre en séquestre les paiements dus aux pairs pour empêcher les pairs d'émettre des chèques à découvert. Le propriétaire de la donnée stockée doit dès le début bloquer la quantité nécessaire pour payer les pairs de stockage et les vérificateurs. Les pairs de stockage doivent aussi mettre en séquestre une quantité d'argent qui correspond à la rémunération que le propriétaire obtient si la donnée est détruite.

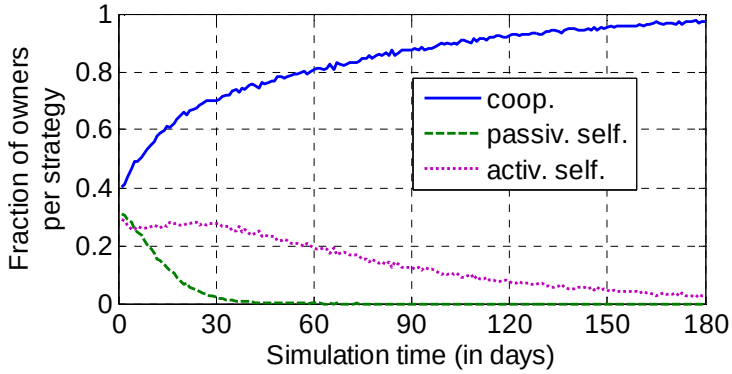


Fig. 6 Ratio des propriétaires en fonction de leur stratégie avec une composition initiale : 40% coopérateurs, 30% pairs égoïstes passives (*passively selfish*), 30% pairs égoïstes actifs (*actively selfish*).

Etant donné que le nombre de paiements reçus par les pairs est proportionnel au nombre de vérifications à effectuer, le protocole de vérification peut se baser sur un protocole qui utilise

des défis pré-calculés. Le vérificateur possède un nombre limité de défis et leurs réponses respectives qu'il emploie un par un pour tester la présence de la donnée chez le pair de stockage. Cette vérification permet au vérificateur et au pair de stockage d'être payés. C'est pourquoi la métadonnée stockée chez le vérificateur consiste en des réponses hachées. Les réponses obtenues du pair de stockage peuvent toujours être testées mais permettent en elles mêmes de déchiffrer des chèques reçus du propriétaire. Si le pair de stockage a gardé toute la donnée intacte, lui et son vérificateur sont récompensés. Par contre, si le pair de stockage a détruit la donnée, il sera détecté par un nombre suffisant de vérificateurs. Ces vérificateurs gardent des parts d'un chèque électronique au nom du propriétaire : celui-ci représente la punition du pair de stockage en cas de défaillance. Ces parts sont envoyées au propriétaire et posséder un nombre suffisant de ces parts permet de construire le chèque et de l'encaisser au près de sa banque.

La Fig. 6 est le résultat de la simulation du système de stockage basé sur l'approche de paiement et illustre la convergence du système vers un état où seuls les pairs coopératifs peuvent stocker leurs données. La convergence prend un certain temps (comparé à la réputation) pour atteindre une population de propriétaires 100% coopératifs du fait de la grande taille du système (nombre de pairs=10000).

Chapitre VI Validation par la théorie des jeux

Le rôle d'un mécanisme d'incitation à la coopération est de motiver les pairs rationnels qui accomplissent des actions stratégiques à coopérer avec les autres pairs. Il s'avère que la démonstration qu'un mécanisme satisfait cet objectif est possible avec les outils de la théorie des jeux. Les jeux non coopératifs répétés sont employés pour valider les incitations de coopération qui régissent les interactions entre pairs bien définis et donner ainsi une vue microscopique du mécanisme; par contre l'utilisation de jeux évolutionnaires qui décrivent l'évolution des stratégies chez plusieurs populations de pairs permettent de saisir une vue plus large et plus dynamique du problème.

A. Jeu non coopératif répété

Le premier modèle de jeu proposé pour le système de stockage pair-à-pair basé sur l'approche de paiement est un jeu non coopératif et symétrique qui se joue entre deux pairs : le propriétaire de la donnée et le pair qui stocke cette donnée. Le propriétaire vérifie périodiquement la présence de sa donnée chez le deuxième pair.

Le modèle de jeu de la Fig. 7 est un jeu séquentiel avec une distribution asymétrique d'information, puisqu'on considère que le propriétaire ne connaît pas le type du pair de stockage qui peut être coopératif ou égoïste ou même défaillant. Toutefois, le propriétaire a la possibilité de déduire le type en se basant sur les résultats de vérification de la donnée à distance. Après chaque vérification, le propriétaire met à jour sa croyance sur le type du pair de stockage selon la formule de Bayes. Ces vérifications sont appelées signaux et le jeu est dit jeu à signaux (*signaling game*). Un signal réussi (résultat de vérification positif) veut dire que le pair de stockage est coopératif ou égoïste car il a pu répondre correctement (avec une probabilité q) à un défi de vérification parce qu'il a gardé une portion de la donnée (qui correspond à l'ensemble d'information III). Un signal erroné (résultat de vérification négatif) signifie au contraire que le pair de stockage est défaillant ou égoïste (qui correspond à l'ensemble d'information IV). En se basant sur ces signaux, le propriétaire a le choix entre récompenser, punir ou ne rien faire contre le pair de stockage.

La solution du jeu est de trouver l'équilibre. L'équilibre de Nash se résume à la non coopération des deux parties : le pair de stockage choisit d'être égoïste et le propriétaire de le punir. L'équilibre Bayesien parfait est plus adapté à ce type de jeu avec information incomplète et réussit à produire la coopération des deux pairs mais avec des conditions qui lient les paramètres du jeu (les valeurs de récompense et de punition par exemple).

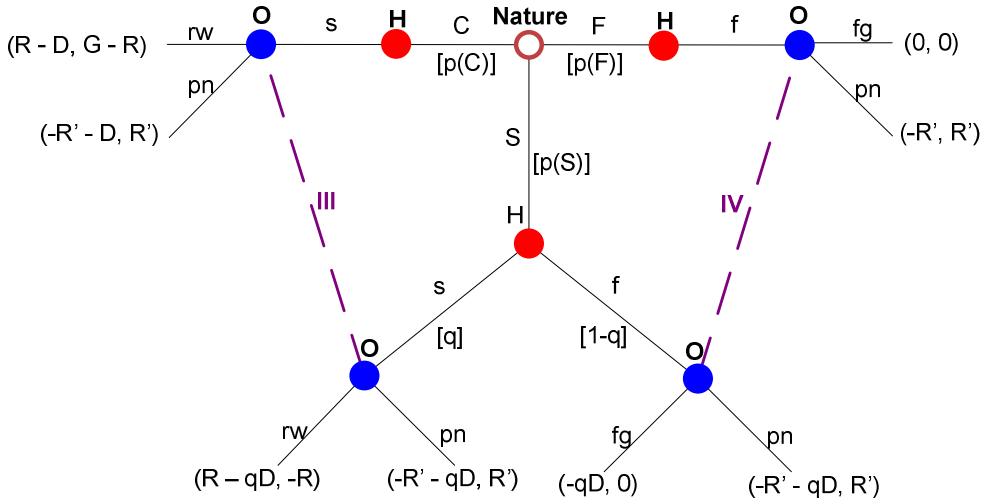


Fig. 7 Jeu à signaux

Le jeu proposé est répété plusieurs fois avec une probabilité d'arrêt de jeu *p*. Le jeu peut aussi être arrêté par le propriétaire qui choisit de punir le pair de stockage. Les profils d'action considérés sont les suivants :

a) (signal réussi, récompense), (signal réussi, récompense), (signal réussi, récompense), …
b) (signal réussi, récompense), …, (signal réussi, récompense), (signal erroné, punition)
c) (signal erroné, punition)

Le résultat de l'analyse du jeu répété prouve que l'itération du jeu (valeur basse de *p*) favorise la coopération du pair de stockage et du propriétaire. De plus, l'analyse démontre que la coopération du propriétaire est stimulée en minimisant les valeurs de la récompense et de punition et en maximisant le gain qu'il obtient du stockage à distance.

B. Jeu évolutionnaire

Le deuxième modèle de jeu du système de stockage pair-à-pair basé sur l'approche de réputation décrit l'évolution des stratégies des populations d'individus suite à des interactions locales multiples entre des individus choisis aléatoirement. Un individu joue contre un autre joueur aléatoirement choisi avec le but de maximiser son utilité (*fitness*) dans ce jeu.

Le jeu évolutionnaire proposé est similaire à celui dans [Brandt et Sigmund 2006] où les joueurs ont chacun un rôle défini : soit donateur, soit récipiendaire. Le donateur gagne un avantage *b* d'un récipiendaire à un coût *–c* chez ce dernier. Le propriétaire, le pair de stockage et le vérificateur sont des rôles qui sont jouables par n'importe quel pair. Le propriétaire est un récipiendaire dans la terminologie de [Brandt et Sigmund 2006], et les *r* pairs de stockage et les *m* vérificateurs sont des donateurs. Le propriétaire gagne *b* si au moins un pair de stockage donne à un coût *-c* ; néanmoins si aucun pairs de stockage ne donne, alors le propriétaire peut

gagner βb si au moins un vérificateur donne à un coût $-\alpha c$ ($\alpha \leq 1$) pour chaque vérificateur. Le dernier cas correspond à la situation où le vérificateur coopératif informe le propriétaire de la destruction de la donnée, le propriétaire ayant alors la possibilité de maintenir le même taux de réplication de sa donnée dans le système.

Les donateurs ont le choix entre donner (coopérer) ou pas. Le travail d'analyse se porte sur les stratégies des pairs suivants:

- *Toujours coopérer :* le pair est altruiste et donne toujours lorsqu'il est dans le rôle du donateur.
- *Ne jamais coopérer :* le pair ne donne jamais dans le rôle du donateur.
- *Discriminer :* le discriminateur donne selon des conditions : le discriminateur donne lorsqu'il ne connaît pas le joueur d'en face ou lorsque ce joueur a déjà donné dans un jeu précédent dans lequel le discriminateur était soit dans le rôle du propriétaire, soit dans le rôle du vérificateur (il était observateur). Cette stratégie s'apparente à la stratégie œil pour œil (*tit-for-tat*) mais diffère par le fait que non seulement le propriétaire tient compte des actions du pair de stockage, mais aussi que les vérificateurs considèrent ces actions dans leurs interactions futures.

La dynamique du jeu évolutionnaire se base sur la dynamique de reproduction de gènes qui définit le taux de croissance de la population de pairs avec une stratégie déterminée est proportionnelle à la valeur d'utilité acquise par la stratégie. Ainsi, la stratégie qui rapporte plus d'utilité que l'utilité moyenne du système augmente ; alors que celle qui rapporte moins d'utilité diminue en taille de population.

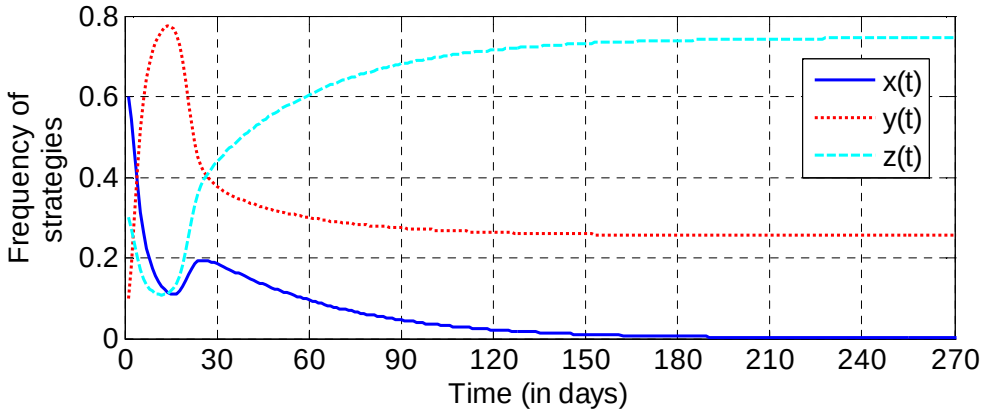


Fig. 8 Fraction des trois stratégies dans le temps : $x(t)$ pour les altruistes, $y(t)$ pour les non coopérateurs et $z(t)$ pour les discriminateurs.

La Fig. 8 montre la convergence du jeu évolutionnaire vers un équilibre où les altruistes sont éliminés du jeu et les non coopérateurs et les discriminateurs coexistent. L'analyse du jeu permet de déterminer les valeurs des paramètres du système (r, m, b, c) pour lesquelles les discriminateurs, qui emploient le modèle de réputation basé sur l'audit, peuvent gagner contre les non coopérateurs. En effet, augmenter le nombre de vérificateurs m permet d'accroître la fréquence des discriminateurs à l'équilibre. Un stockage coûteux ou un taux de réplication r élevé réduit cette fréquence.

Chapitre VII Conclusion

Les systèmes pair-à-pair ont émergé comme un nouveau paradigme intéressant pour le stockage distribué qui vise à profiter des ressources libres et inexploitées des pairs d'une manière efficace et équitable. Externaliser le stockage de données chez des pairs d'un réseau est probablement l'unique solution qui permet la disponibilité et la tolérance aux fautes des données tout en garantissant une croissance à grande échelle et en réduisant ou même supprimant les coûts d'entretien du stockage. Dans cette thèse, nous avons traité les problèmes de sécurité et de coopération auxquelles une telle application peut être confrontée une fois efficacement déployée dans le réseau.

Résumé et contributions

Tout d'abord, nous avons examiné les questions de sécurité liées au stockage de données pair-à-pair. L'opération correcte d'un système de stockage pair-à-pair se base sur la coopération équitable et efficace des pairs. Malheureusement, les pairs peuvent être malhonnêtes de diverses manières. Les pairs de stockage peuvent prétendre stocker des données qu'ils ont en fait détruites. Pour les approches basées sur la réplication, les pairs peuvent s'entendre pour stocker une seule copie des données défaisant de ce fait les mécanismes qui assurent la fiabilité des données. La collusion peut ne pas être la manière unique de faire ainsi, puisque les attaquants Sybil peuvent produire plusieurs identités et les employer d'une manière frauduleuse.

Nous décrivons des éléments d'une architecture modulaire pour un tel système fournissant les mécanismes de sécurité et de coopération nécessaires pour assurer l'opération correcte et sécurisée d'un système de stockage de données pair-à-pair. Nous détaillons comment un environnement de confiance peut empêcher des comportements malhonnêtes, en particulier concernant l'identification des pairs, la vérification de l'intégrité des données, et la gestion de la confiance.

Les actions dissimulées des pairs non coopératifs peuvent être dévoilées en utilisant un nouveau type de protocole que nous qualifions de vérification de possession de données. Ces protocoles permettent à un vérificateur de détecter si des données qui sont stockées à distance ont été corrompues ou détruites sans les transférer jusqu'au vérificateur. Nous proposons trois différentes constructions pour de tels protocoles avec différentes options pour la vérification, en particulier concernant la délégation.

Le comportement des pairs de stockage peut être évalué en se basant sur les résultats obtenus avec de tels protocoles. L'audit forme la base d'observation pour les mécanismes d'incitation à la coopération que nous proposons pour stimuler la coopération et motiver les comportements corrects. L'originalité de ces mécanismes provient de l'évaluation optimiste du comportement des pairs, suivant ainsi une approche très différente comparée aux incitations à la coopération dans les réseaux mobiles ad hoc (MANET) : tandis que le comportement d'un pair peut seulement être décidé à la fin de la période de stockage, l'audit peut être exécuté de façon régulière et nous considérons qu'un pair se comporte bien tant qu'aucune corruption de données n'est détectée. Nous proposons deux mécanismes d'incitation à la coopération, un basé sur la réputation et l'autre sur la rémunération. Les deux mécanismes sont conçus pour encourager un comportement coopératif et également pour établir la confiance, détecter et punir les pairs malhonnêtes.

L'efficacité de nos mécanismes basés sur l'audit en termes de sécurité et de coopération est démontrée par des modèles théoriques de jeu non coopératif. Nous évaluons d'abord l'efficacité des incitations avec diverses primitives d'observation probabilistes et déterministes. Des jeux évolutionnaires sont également présentés afin d'évaluer les équilibres macroscopiques réalisés.

Perspectives

Notre travail a présenté des primitives pour évaluer le comportement des pairs en ce qui concerne le stockage. La réaction qui résulte de telles évaluations sert principalement à des mécanismes d'incitation à la coopération. Cependant, les pairs, en particulier les propriétaires de données, doivent également adapter leurs stratégies de stockage basées sur de telles évaluations. Détecter un défaut de stockage devrait déclencher un processus de régénération de données pour assurer la fiabilité à long terme du stockage de données. Cependant, l'efficacité d'un tel processus dépend non seulement de la disponibilité d'un nombre suffisant de pairs de stockage, comme nous l'avons modélisé, mais également du temps nécessaire pour le transfert des blocs de données entre les pairs. Une analyse de performance d'un tel processus apporterait certainement des évaluations plus réalistes quant à la largeur de la bande passante et aux conditions de dynamicité d'une application de stockage pair-à-pair.

Les mécanismes de sécurité développés dans cette thèse, et en particulier les incitations à la coopération, sont cruciaux pour estimer le degré de confiance d'un pair et stimuler sa coopération. Bien qu'ils aient été conçus pour le stockage de données pair-à-pair, d'autres applications pair-à-pair (e.g. la téléphonie pair-à-pair sur IP) tireraient certainement bénéfice de tels mécanismes de sécurité et de coopération. Par exemple, les fournisseurs d'Internet peuvent déployer des relais Wifi pour la téléphonie sur IP avec la coopération des utilisateurs qui acceptent de configurer leurs boîtes ADSL pour mettre en œuvre ce service. En échange, ces derniers disposent d'un accès au service qu'ils contribuent à déployer. Une gestion plus fine et auto-organisante pourrait être réalisée, en particulier avec des incitations basées sur la rémunération. Wuala par exemple a commencé à déployer son infrastructure de stockage de données avec une telle approche. Les incitations à la coopération basées sur la rémunération préparent également le terrain pour des architectures qui offrent des services multiples et qui permettraient par exemple à des plateformes hétérogènes de coopérer efficacement et d'échanger de la bande passante pour du stockage.

La protection contre des attaques de type Sybil et des attaques de blanchissement (*whitewashers*) est une question centrale dans beaucoup d'applications pair-à-pair. Il convient de noter que les approches complètement auto-organisées peuvent seulement atténuer de telles attaques tout en appliquant une sanction contre les pairs honnêtes. Nous avons discuté de l'utilisation d'un environnement de confiance comme solution possible. Bien que coûteux en termes de déploiement, un environnement de confiance peut en effet fournir une solution à ce problème qui permet aussi la scalabilité. En particulier, l'architecture TCG qui est de plus en plus déployée dans les équipements d'entreprise est un candidat intéressant. Par exemple, les mécanismes d'attestation anonymes et directs (*direct anonymous attestation*) peuvent lier des données à une plateforme unique tout en préservant l'intimité de la plateforme. Il y a également une tendance de fond à établir la confiance dynamique basée sur des rapports existants de confiance et statiques, bien illustrée par l'apparition des services basés sur les réseaux sociaux (Skype, Facebook, hi5, LinkedIn, MySpace). Dans de tels systèmes, de petits groupes de pairs peuvent facilement être établis. La règle de Dunbar détermine qu'un pair donné peut maintenir des rapports sociaux stables avec au plus 150 autres pairs. Ceci peut signifier que les applications pair-à-pair pourraient à l'avenir montrer des topologies très différentes de celles utilisées dans le partage de fichiers pair-à-pair dans lequel un pair peut se relier à 3000 autres, comme dans les *swarms* de BitTorrent par exemple. La mise à l'échelle restera cependant un défi de recherche important dans de tels systèmes qui peut encourager le développement de protocoles plus efficaces pour contrôler l'interconnexion de multiples groupes reliant des pairs.

Abstract

Self-organizing algorithms and protocols have recently received a lot of interest in mobile ad-hoc networks as well as in peer-to-peer (P2P) systems. The latter in particular suggest that decisions and operation, instead of being concentrated in a relatively low number of specific devices (e.g., routers, gateways, servers, certification authority), may use the computing power, bandwidth, or disk storage space of end-user devices in the network. Such techniques have proven most successful to implement cost-effective and reliable applications to be deployed on a large scale, as illustrated by file sharing, video/audio streaming, or VoIP. P2P storage, whereby peers collectively leverage their storage resources towards ensuring the reliability and availability of user data, is an emerging field of application. P2P storage however brings up far-reaching security issues that have to be dealt with.

Providing assurances in P2P storage systems requires not only ensuring the confidentiality and privacy of the data storage process, but also the introduction of proper security and cooperation enforcement mechanisms for thwarting various peer misbehaviors. Indeed, the delegation of data storage mechanisms to autonomous peers raises new concerns, in particular with respect to peer selfishness, as illustrated by so-called free-riding attacks: the attacker may consume storage resources without contributing its fair share, or may even corrupt or destroy the data that it has promised to store while pretending it did its share of work. Systems vulnerable to free-riding either run at reduced capacity or collapse entirely because the costs of the system weigh more and more heavily on the remaining honest peers, thus encouraging them to either quit or free ride themselves. Additionally, a new form of man-in-the-middle attack may make it possible for a malicious peer to pretend to be storing data without using any local disk space. New forms of collusion also may occur whereby replica holders would collude to store a single replica of some data, thereby defeating the data redundancy requirement. Finally, Sybil attackers may create a large number of identities and use them to gain a disproportionate personal advantage.

Whilst many aspects of P2P applications have been thoroughly researched, security within these applications still remains a challenge. A trusted infrastructure that offers an interesting and powerful set of security features may be employed in order to act in response to such challenges. We provide an architectural description with a layered organization to handle the operation of the P2P storage system in a secure way. We show the different ways whereby such architecture may be enhanced by judiciously introducing a trusted infrastructure. However, with a trusted infrastructure, it is difficult to ensure a large scale P2P storage system with low administrative attention. The security assurances of such system should be provided by relying solely on peers themselves.

The continuous observation of peer behavior and monitoring of the storage process is an important requirement to secure a storage system. Observing peer misbehavior requires appropriate primitives like proofs of data possession, a form of proof of knowledge whereby the holder interactively tries to convince the verifier that it possesses the very data without actually retrieving them or copying them at verifier's memory. We present a survey of such techniques and discuss their suitability for assessing remote data storage. We also propose a new data possession verification protocol through which verification can be handed over to volunteer peers from the network. There is a potential interest in delegation method for verification, mainly because the owner or the holder may be offline such that they are not able to catch each other for the interactive verification protocol. Thus, the owner holds interest in delegating the verification task to one or multiple verifiers; though multiple verifiers' case is more desirable to avoid Byzantine failures of verifiers or even potential collusion between a verifier and the holder.

Cooperation is key to deploying P2P storage solutions, yet peers in such applications are confronted to an inherent social dilemma: should they contribute to the collective welfare or

misbehave for their individual welfare? So-called cooperation incentive schemes provide an answer to such dilemma by promoting ways of managing and organizing resources and dealing with the new security challenges that traditional security approaches cannot cope with. We review several incentive mechanisms that have been proposed to stimulate cooperation towards achieving a resilient storage. We also propose mechanisms enforcing cooperation by means of proofs of data possession periodically delivered by storage peers. This approach makes it possible to change the purpose of cooperation incentives from stimulating cooperation among peers to enforcing that cooperation and increasing its fairness.

The effectiveness of such incentive mechanisms must be validated for a large-scale system. We approach this assessment with game theoretical techniques: cooperation incentive mechanisms are proven to be effective if it is demonstrated that any rational peer will always choose to follow mechanism directives whenever it interacts with another peer. We illustrate the validation of cooperation incentives with non-cooperative one-stage and repeated Bayesian games and evolutionary games.

Table of Contents

List of Figures

List of tables

Chapter 1

Introduction

Peer-to-peer (P2P) networks have first emerged in the late 90's as specialized systems and protocols to support file sharing. They became very popular thanks to services like Napster[14], Gnutella[15], KaZaA[16] and Morpheus[17], and particularly thanks to the legal controversy regarding their copyrighted contents. Since then, the popularity of P2P systems has continued to grow such that the self-organization of a service based on the contributions of its users is now regarded as a general-purpose and practical approach that can be applied to designing applications for sharing any resource. In this context, resources may include the exchange of information, processing cycles, packet forwarding and routing, as well as cache and disk storage. In this sense, an increasing number of services ranging from telephony or audio/video streaming to ad hoc networking or nomadic computing are bound to use such architectures. P2P storage services have more recently been suggested as a new technique to make use of the vast and untapped storage resources available on personal computers. P2P data storage services like Wuala[18], AllMyData Tahoe[19], UbiStorage[20], or Cucku[21] have received some highlight. In all of these, data are outsourced from the data owner place to several heterogonous storage sites in the network, in order to increase data availability and reliability, to reduce data storage maintenance costs, and to achieve a high scalability of the system.

1.1. A case for P2P storage

Innovation and advancement in information technology has spurred a tremendous growth in the amount of data available and generated. This situation has resulted in new challenges regarding the need for scalable storage management that can only be addressed by implementing storage applications in a self-organized and cooperative form. In such storage applications, peers can store their personal data in one or multiple copies (replication) at other peers. The latter, which we call data *holders*, should store data until the *owner* retrieves them. Such P2P storage aims at maintaining a reliable storage without a single point of failure, although without the need for an expensive and energy-consuming storage infrastructure as offered by data centers (currently a lot of efforts are being undertaken to make these data

[14] http://www.napster.com/
[15] http://www.gnutella.com/
[16] http://www.kazaa.com/
[17] http://www.morpheus.com/
[18] http://wua.la/en/home.html
[19] http://allmydata.org/
[20] http://www.ubistorage.com/
[21] http://www.cucku.com/

centers efficient[22]). Peers volunteer for holding data within their own storage space on a long term basis while they expect a reciprocal behavior from other peers.

It has been some years now that P2P storage has been presented as a solution for data backup ([Cox and Noble 2002] and [Lillibridge et al. 2003]) as well as for a new generation of distributed file systems ([Druschel and Rowstron 2001], [Kubiatowicz et al. 2000], and [Dingledine 2000]). P2P storage aims at a free and more importantly more resilient alternative to centralized storage, in particular to address the fact that storage can still be considered as a single point of failure. Additionally, P2P storage may also be attractive in wireless ad-hoc networks or delay-tolerant networks (DTNs), notably since mobility introduces a store-carry-and-forward paradigm ([Zhao et al. 2006]) to deliver packets despite frequent and extended network partitions. The cooperative storage of other nodes' messages until their delivery to their destination thus might become an important feature of such networks. Context- or location-based services may also benefit from P2P storage. Desktop teleporting ([Bennett et al. 1994], [Pham et al. 2000]) for instance aims at the dynamic mapping of the desktop of a user onto a specific location. Teleporting may benefit from some caching by using the storage offered by surrounding nodes at the new user location, depending on the network topology. Location-aware information delivery ([Marmasse and Schmandt 2000], [Huang at al. 1999], [Dey and Abowd 2000], [Beigl 2000]) is another context-aware application. Each reminder message is created with a location, the message being delivered when the intended recipient arrives at that location,. In such an application storing messages at nodes situated nearby the location context rather than at the mobile node may make sense, especially if only intermittent connections of the mobile node are possible.

Though the self-organization introduced by P2P storage promises to produce large scale, reliable, and cost-effective applications, it exposes the stored data to new threats. In particular, P2P systems and even more so P2P storage systems may be subject to selfishness, a misbehavior whereby peers may discard some data they promised to store for other peers in order to optimize their resource usage. Maliciousness in the P2P context would simply consist in peers destroying the data they store in order to reduce the quality of service of the system. Because of the high churn and dynamics of peers, checking that some data have been stored somewhere is quite more complex than checking that a route has been established with another node in multi-hop MANETs for instance. In addition, such verifications cannot be instantaneous but have to be repeatedly performed. All these problems contribute to the difficulty of properly determining the actual availability of data stored onto unknown peers. Countermeasures that take into account the fact that users have full authority on their devices should be crafted to prevent them from cheating the system in order to maximize the benefit they can obtain out of peer cooperation.

1.2. Security issues related to P2P storage

A P2P storage application takes advantage of the existing and spare disk space at peers allowing the latter to leverage their collective power for the *common good*. While the fundamental premise of this is voluntary storage resource sharing among individual peers, there is an inherent tension between individual rationality and collective welfare that threatens the viability of these applications. Selfish behaviors, termed *free riding*, are the result of a social dilemma that all peers confront and may lead to system collapse in *the tragedy of the commons*

[22] The Green Grid is an association of IT professionals seeking to dramatically raise the energy efficiency of data centers: http://www.thegreengrid.org/

[Hardin 1968]: the dilemma for each peer is to either contribute to the common good, or to free ride (shirk).

Achieving secure and trusted P2P storage presents a particular challenge in that context due to the open, autonomous, and highly dynamic nature of P2P networks. We argue that any effort to protect the P2P storage system should ensure the following goals:

- *Confidentiality and integrity of data:* Most storage applications deal with personal (or group) data that are stored somewhere in the network at peers that are not especially trusted. Data must thus be protected while transmitted to and stored at some peer. Typically, the confidentiality and the integrity of stored data are ensured using usual cryptographic means such as encryption methods and checksums.
- *Anonymity:* Anonymity can be a requirement for some type of storage applications that aim at preventing information censorship for instance; however it may not be a targeted objective for all of them. Anonymity may refer to the data owner identity, the data holder identity, or the detail of their interaction. Anonymity permits to avoid attacks whereby the data of a given user are specifically targeted in order to destroy them from the system. Systems that seek to provide anonymity often employ infrastructures for providing anonymous connection layers, e.g., onion routing [Goldschlag et al. 1999].
- *Identification:* Within an open environment like P2P networks, it is possible for the same physical entity to appear under different identities, particularly in systems with highly transient populations of peers. This problem may lead to the problem of "Sybil attacks" [Douceur 2002], and may also threaten mechanisms such as data replication that rely on the existence of independent peers with different identities. Solutions to these attacks may rely on the deployment of a trusted third party acting as a central certification authority, yet this approach may limit anonymity. Alternatively, P2P storage may be operated by some authority controlling the network through the payment of membership fees to limit the introduction of fake identities. However, that approach reduces the decentralized nature of P2P systems and introduces a single point of failure or slows the bootstrap of the system if payment involves real money. Without a trusted third party, another option is to bootstrap the system through penalties imposed on all newcomers: an insider peer may only probabilistically cooperate with newcomers (like in the P2P file sharing application BitTorrent [Piatek et al. 2007]), or peers may join the system only if an insider peer with a limited number of invitation tickets introduces them [Lesueur et al. 2008]. The acceptable operations for a peer may also be limited if the connection of too many ephemeral and untrustworthy identities is observed [Yu et al. 2006]. This option however seems to be detrimental to the scalability of the system and it has even been shown that this degrades the total social welfare [Feldman and Chuang 2005]. Social networks may also partially solve the identification issue.
- *Access control:* Encryption is a basic mechanism to enforce access control with respect to read operations from one single reader. In the case of multiple readers, the distribution of the keys necessary for accessing the stored data to these readers should be enforced in order to prevent denial of service attacks against the storage peer launched by unauthorized readers. For instance, access control lists can be assigned to data by their original owners through the use of signed certificates. Capability-based access control can be also employed like in [Srivatsa and Liu 2005]. Delete operations have to be especially controlled because of their potentially devastating end result.
- *Scalability:* The system should be able to scale to a large population of peers. Since most of the important functions of the system are performed by peers, the system should then be able to handle growing amounts of control messages for peer and storage resource management and an increased complexity in a graceful manner. The system may also be clustered into small groups with homogeneous storage needs which may reduce the load

over peers. Another important issue associated with P2P applications is the fairness of resource allocation (e.g., storage, bandwidth) between peers. Generally a quota system introduced within a cooperation incentive mechanism is put in place to regulate resource sharing. The role of such system is to adjust peer consumption to their just contribution: no peer has the right to sponge off other peers.

- *Data reliability:* The common technique to achieve data reliability relies on data redundancy at several locations in the network. The data may be simply replicated at a given redundancy factor. The redundancy factor should be maintained during the entire duration of the data storage. The rejuvenation of the data may be carried out either in a periodic or event-driven fashion. For instance, in the latter approach, one or multiple new replicas should be generated whenever a certain number of replicas have been detected as destroyed or corrupted. Other redundancy schemes may be used instead of merely replicating the data into identical copies; for instance erasure coding provides the same level of data reliability with much lower storage costs.
- *Long-term data survivability:* The durability of storage in some applications like backup is very critical. The system must ensure that the data will be permanently conserved (until their retrieval by the owner). Techniques such as data replication or erasure coding improve the durability of data conservation but these techniques must be regularly adjusted to maximize the capacity of the system to tolerate failures. Generally, the employed adaptation method is based on frequent checks over the data stored to test whether the various fragments of a data are held by separate holders. Moreover, cooperation incentive techniques must be used to encourage holders to preserve the data they store as long as they can.
- *Data availability:* Any storage system must ensure that stored data are accessible and useable upon demand by an authorized peer. Data checks at holders allow the regular verification of this property. The intermittent connectivity of holders can be tolerated by applying a “grace period” through which the verifiers tolerate no response from the checked holder for a given number of challenges before declaring it non cooperative.

The rest of this thesis especially details how to achieve the last three objectives above: high reliability, availability, and long-term durability of data storage in the context of a large scale P2P storage system. These three objectives are often ignored in P2P file sharing applications which rather follow best effort approaches. Performing periodic cryptographic verifications makes it possible to evaluate the security status of data stored in the system and to design an adapted cooperation incentive framework for securing data storage in the long run.

1.3. P2P storage applications: A brief state of the art

P2P storage applications have become famous in several domains: file sharing is the flagship of such applications that it now accounts for almost 80% of total traffic [Bolton and Ockenfels 2000]; yet P2P file systems or file backup systems are also available.

PAST [Druschel and Rowstron 2001], which is based on Pastry, and OceanStore [Kubiatowicz et al. 2000], which is based on Tapestry, are well-known file systems that make use of DHT (Distributed Hash Table)-based overlay networks. Both PAST and OceanStore aim at ensuring a high data availability of files by guaranteeing the geographical separation of replicas: this is achieved by means of file replication and random distribution of the identification numbers to peers. Both PAST and OceanStore rely on remuneration means as cooperation incentives. Each OceanStore peer is supposed to pay a fee to one particular provider who buys storage space from and sells it to other providers. Legal contracts and enforcement

can be used to punish peers that do not keep their end of the bargain, based on planned billing and auditing systems. On the other hand, PAST relies on the use of smart cards to ensure that peers cannot use more remote storage than they are providing locally. Smart-cards are held by each PAST peer and issued by a third party, and they support a quota system that balances supply and demand of storage space in the system. With fixed quotas and expiration dates, peers are only allowed to use as much storage as they contribute.

The file systems described so far are not commercial infrastructures, on the contrary to the Wuala[23] start-up. Wuala is an online storage and file-sharing system that offers to "*securely store and back up files online, access them from anywhere, and share photos, videos, and music with friends and family*" [24]. In Wuala, users may choose whether to have 1GB of free storage at Wuala's servers or trade their computer's space for other Wuala members' space. User files are split into 500 encrypted fragments, each of which is stored onto other Wuala members' computers. To our knowledge, the selection of storage peers is performed randomly and centrally by Wuala. Wuala introduces an original mechanism for storage trading in that it takes into account peer availability in the network: the gained storage space is equal to the contributed data storage space times the actual availability percentage of the peer.

Pastiche [Cox and Noble 2002] is a storage system whose primary function is data backup and which is based on Pastry for locating peers. It exploits excess disk capacity to perform P2P backup with no administrative costs. Each Pastiche peer minimizes storage overhead by selecting peers that share a significant amount of data. It replicates its archival data on more than one peer. Most of these replicas are placed nearby to ease network overhead and to minimize restoration time. To address the problem of storing data on malicious peers, Pastiche uses a probabilistic mechanism to detect missing backup state by periodically querying peers for stored data. However it sacrifices a fair amount of privacy because peers can grab some information about the backup data.

The latter issue is less critical for the Cooperative Internet Backup Scheme [Lillibridge et al. 2003] where fragments of a file are stored at different geographical locations, and partners are tracked by a central server. Each peer has a set of geographically separated partner peers that collectively hold its backed up data. In return, the peer backs up a part of its partners' data. To ensure a high reliability, the scheme adds redundancy through Reed-Solomon erasure correcting code.

AllMyData Tahoe[25] also uses Reed-Solomon redundancy to provide automated online file backup. The file fragments are redundantly disseminated into the network of storing peers in such a way that only a small percentage of the fragments must be recovered in order to fully restore the file. The file is identified using a URI that includes the rights (read/write) to the data yet such capability-based access control requires URIs to always be kept secret (which is still an open issue for AllMyData).

AllMyData does not consider any cooperation incentives for thwarting free-riding, whereas in [Lillibridge et al. 2003], peers are periodically challenging each of their partners by requesting them to send a block of the backed up data. An attack can then be detected and the data blocks of the attacker that are stored in the attacked peer are consequently dropped. The scheme then uses a sort of tit-for-tat (TFT) (similarly to BitTorrent [Cohen 2003]) strategy whereby each peer takes note of its direct experience with a partner. If this partner does not voluntarily cooperate or is estimated to cooperate below some threshold, the peer may decide to dump it from its partner list.

[23] http://wua.la/en/home.html
[24] Wuala also provides typical Web 2.0 features like collaborative tagging, sharing, comments, etc.
[25] http://allmydata.org/

1.4. Research objectives

The study of P2P systems raises several stimulating security challenges owing to the intricate issues that are associated with self-organization in such systems. First of all, these systems are inherently large scale, highly churned out, and relatively anonymous. This all renders volunteer cooperation hardly achievable without some trust referential. Trust can be achieved statically (based on identity for instance) or dynamically (self-organized trust). Static trust refers to a statement of trustworthiness that remains the same until it is revoked, whereas dynamic trust exhibits self-learning and self-amplifying characteristics. The latter arises from behaviors experienced in the system and continuously changes accordingly. An entity trusts a peer more when it has direct or indirect information about that peer that prove its trustfulness. Such information may consist in the collection of its past behavior (reputation) or in a commitment of financial reward or punishment if the peer cooperates or not (payment). [Carbone et al. 2003] for instance introduces a trust model that does not only concentrate on the content of evidence but also on the amount of such evidence.

The temporal dimension should also be taken into consideration. Cooperative interactions between peers are generally understood as atomic operations which may be deemed to be acceptable in the case of packet-forwarding or file sharing; however such an assumption definitely does not hold for distributed storage, an operation that can certainly not be considered as instantaneous. The latter application requires a new primitive that allows an immediate evaluation of peer cooperation, and that we call *proofs of data possession*. The primitive aims at periodically checking the actual storage at a holder in order to provide *short-term* assessments of holder's cooperation. Based on such primitive, cooperation incentives are introduced to establish *long-term* trust between peers, to stimulate their cooperation, and to ensure the fairness of their respective contributions.

To overcome the free-riding problem and to encourage peers to cooperate, incentive mechanisms assume "strategic" peers with "rational" behavior. This generally represents a worse situation than reality as file-sharing applications have shown. Game theoretical models are the most efficient tool to evaluate whether such mechanisms will be followed by any rational peer whenever it interacts with another peer. There are a large number of game theoretical models that can fashion the P2P storage system, each one providing a different view of the problematic and challenging facets of the system. We will particularly focus on non-cooperative one-stage, repeated and evolutionary games. It should be noted that this rational behavior assumption does not take into account purely malicious behaviors that have to be addressed by some other means.

1.5. Thesis organization

The remainder of the thesis will be structured as follows. Chapter 2 describes a secure architecture that attempts to fulfill the security goals of a P2P storage system. We show the extent to which such an architecture may benefit from a trusted environment in order to realize several critical functionalities instead of letting the peers realize them themselves. In Chapter 3, we present two protocols that may be used to remotely prove data possession: the first one achieves a probabilistic proof for the sake of better performance, whereas the second one allows the prover to provide a deterministic and complete attestation of data possession. Based on such primitives, mechanisms for enforcing cooperation in a resilient and secure way are introduced in Chapter 4. Finally, Chapter 5 validates such mechanisms as cooperation incentives for rational peers using game theoretical models.

Chapter 2

Architecture: elements of a secure P2P data storage system

In a P2P storage system, data are distributed in a self-organizing manner to multiple peers instead of using a central storage outsourcing server. Such a P2P storage system is however like any P2P application more complex and chaotic than classical distributed systems. The organization and control that the system should afford is provided by the peers themselves. These peers should organize themselves in a way that provides security, scalability, and reliability of the storage. We define in this chapter the features of a secure architecture that coordinates the system while attempting to meet several security goals. These goals are presented and discussed beforehand. We suggest organizing the various functionalities offered by a P2P data storage application along several orthogonal overlays. In addition, we present two architectures, one based on self-organized peers, the other one making use of a trusted infrastructure with the aim of enforcing system security.

A P2P storage system relies on the cooperation of peers to properly operate. Such cooperation is controlled in a distributed fashion by peers themselves and whose role is critical to achieving the overall requirements of a secure storage system. In the following, we will describe each of these blocks specifying how they work and how they are connected to meet the requirements of the P2P storage system (discussed in 1.2).

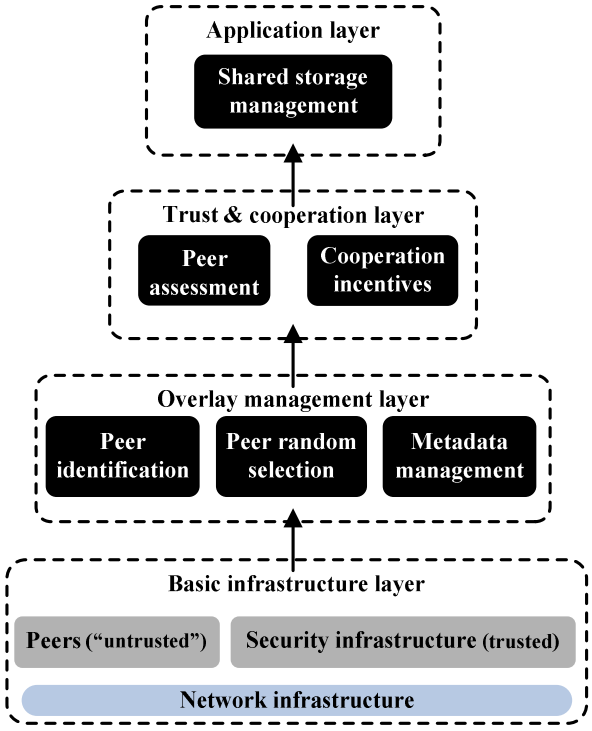


Figure 1 Architecture of the P2P storage system.

The proposed architecture follows a layered organization for the sake of modular development and separation of concerns such that each layer may use elements produced by lower layers. Wherever their functions can only be implemented in a self-organizing manner, layers are implemented by overlays on top of a P2P network. Our architecture proposal consists of four layers (depicted in Figure 1):

- *Basic infrastructure layer:* The layer defines the communication paradigm that is employed in the storage system and that particularly illustrates the direct exchange of messages and resources between peers. It also introduces a security infrastructure with various forms that can be deployed within the network in order to carry several security functions of the storage system.
- *Overlay management layer:* The layer provides the organization model of peers and resources in the system.
- *Trust and cooperation layer:* The layer comprises the tools required to guarantee the well operation of the system that relies essentially on the fair and large cooperation of peers.
- *Application layer:* The layer is concerned with managing the service offered to the user. The service is generally associated with the resource cooperatively exchanged between peers notably the distributed storage facility.

2.1. Basic infrastructure layer

This section describes the network infrastructure that is used in the storage system. The main characteristics of the infrastructure are discussed, particularly demonstrating that the peers representing the network can handle all functionalities of the storage system (explained in detail in the following sections). Yet, some functions can be handed out to a security infrastructure that can be deployed within the network. Such infrastructure is also described in this section.

2.1.1. Network infrastructure

There are three models for the computer network depending on how resources are exchanged between computers (depicted in Figure 2): client/server model and peer-to-peer (P2P) model.

- *Client/server:* In the client/server model, computers are distinguished between client computers and server computers. The client requests some information from the server that holds such requested information and transmits it to the client (e.g., HTTP, FTP). Servers make use of dedicated server operating systems that are designed to handle the load when multiple client computers access server-based resources. Moreover, servers may employ trusted computing forms thus providing a trusted environment for the system to handle several of its functionalities in a secure and protected way.
- *P2P:* In the P2P model, each computer can act as a server if it has some resources to share, and can act as a client if it wants to request some resources from other computers. Computers, termed peers, have equal roles and responsibilities. The communication model relies on the direct exchange of resources between peers (e.g., file sharing, IP telephony, publish/subscribe system).

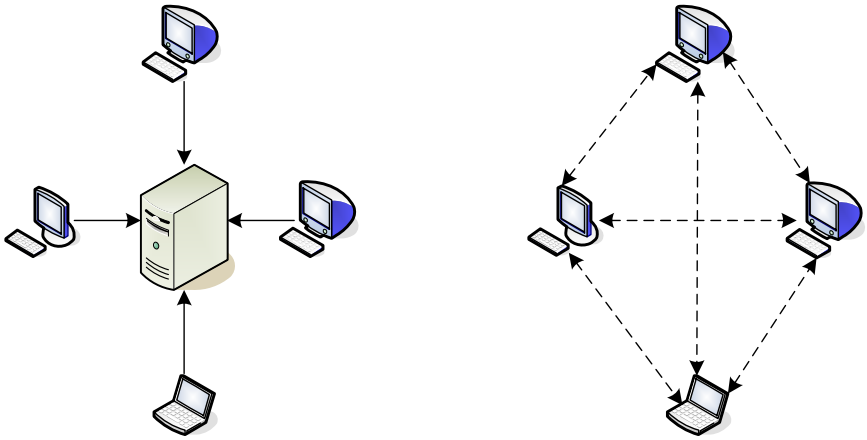


Figure 2 Network communication models: data exchange through (a) client/server and (b) P2P models.

Our work focuses on storage systems built over the latter type of networks: the P2P communication model. We may assume in some cases discussed in this chapter that the network further relies on a trusted environment based on trusted devices integrated within computers or dedicated systems disseminated into the network to secure applications.

P2P networks abide by a communication paradigm that allows the direct exchange of resources between peers rather than being exchanged through a centralized entity, the server. Each peer provides an equivalent functionality and has equivalent responsibilities and privileges. Due to the lack of dedicated servers, the control and management of such networks is handled by peers themselves. P2P applications should thus be built talking into consideration self-organization as the main feature characterizing such type of networks.

Because of the absence of a centralized server, there is no bottleneck in the network and the architecture achieves minimal administrative and operational costs which render it particularly scalable with a large population of peers.

However, the P2P network model is prone to peer failure and unpredictable peer departures from the network. Such a model is highly dynamic in nature in the sense that peers randomly join and leave the system at any time and concurrently without any central coordination (churn). This may cause the partition of the network into smaller fragments sometimes leading to an impossible communication between peers. Although P2P systems have received a lot of attention in the past few years, they have also earned a lot of criticisms for their high maintenance cost in the presence of high churn. Structured P2P networks (see next section) in which peer organization is handled by all peers through exchanged control messages constitute an example of efforts to cope with that issue. As another example, [Zhao et al. 2006] suggests the use of “throwboxes” to improve the connectivity of P2P applications deployed on top of an ad hoc network based on the “store-carry-and-forward” paradigm. Such throwboxes are trusted devices disseminated within the network.

Robustness of applications built upon P2P networks is undermined by the potential peer churn or failure. Peers are also heterogeneous in terms of the quality and quantity of resources they offer, their connectivity, and behavior. Finally, the geographical and topological distribution of peers generally prevents establishing any correlation between their disconnections, departures, or failures. With such heterogeneity and behavior independence, the robustness of applications relying on the network peers is enhanced just by distributing application functionalities to multiple peers. Security mechanisms and primitives required to

ensure the security and protection of the P2P storage system should not just fulfill the objectives discussed earlier, but they also should cope with peer churn and failure.

2.1.2. Security infrastructure

A security infrastructure can be deployed into the storage system; thus allowing to resolve several problems notably related to self-organization. The security infrastructure provides strong authentication mechanisms for peers; it may even be used to assess their behavior. Moreover, trust among peers that have no relationships may be established thanks to the security infrastructure.

The security infrastructure exists in various form factors, from dedicated trusted devices in a network to trusted platforms integrated within untrusted devices. Their purpose is to achieve confidence towards the integrity and reliability of one or several platforms in a network.

The security infrastructure may be used to correctly identify peers (see Section 2.2.3) or assess their behavior (see Section 2.3.2). Additionally, it may manage reputation ratings of cooperating peers in reputation approaches or care for the correctness of the fair-exchange and handle payments in remuneration-based approaches (see Section 2.3.3). A trusted third party may handle these functions. Techniques based on trusted operating system, trusted platform module or smart cards may provide also these same functionalities but in a distributed fashion.

Trusted third party

A trusted third party (TTP) is an established and responsible entity (e.g., dedicated server) accepted by all users as the authority for performing a given function (e.g., certification authority, fair exchange, etc.). A TTP facilitates interactions between different parties with no prior trust relationships, but which all trust the TTP and use this trust to secure their own interactions between them. A TTP is termed online when it is involved in an electronic transaction between peers, the transaction being possible only if it does not fail. Such a TTP also constitutes a single point of failure in that case. Most protocols assume that a TTP will not misbehave or collude with one of the transacting parties (e.g., [Cox et al. 1995]) whereas some assume that the TTP is only semi trusted (e.g., [Franklin and Reiter 1997]). Some protocols on the contrary rely on offline TTPs that are used in case of a dispute over the results of a protocol Finally, as explained below, tamper resistance can provide the necessary basis for implementing online authority in a distributed fashion and despite churn, thanks to the local availability of a protected execution environment that cannot be manipulated nor observed by an adversary.

Trusted operating system

A trusted operating system (trusted OS) (called also secure operating system) is designed so that agents (users or processes) can only perform actions that have been allowed. The primary objective is to preserve and protect the confidentiality, integrity, and availability of information, systems, and resources. This involves specifying and implementing a security policy: “A security policy is a statement of what is, and what is not, allowed” [Bishop 2003]. A security policy enables the proper managing and gaining access to information, systems and resources. Security policies must be applied within access control mechanisms that are based on several underlying concepts and principles.

Let us first define some aspects of access control terminology. The entity that requests access to a resource is called subject. A subject is an active entity because it initiates the access a resource. The requested resource is called object of the access, and it is the passive part of the access. So, access control is the process by which to permit or deny the use of an object (such as information, system, or resource) by a subject (such as user or process). Access control

techniques are concerned with whether subjects can access an object and how this access can occur. A trusted OS involves each object being protected by an access control mechanism.

Generally, access control techniques are categorized as either discretionary or mandatory. Alternative approaches are role-based access control, rule-based access control, or domain type enforcement.

- Discretionary access control (DAC) is a security policy that allows the owner of the information to decide who can read, write, and execute a particular object. DAC is based on the idea that the owner of data should determine who has access to them.
- Mandatory access control (MAC) is a security policy where access is determined by the system, not the owner. Administers and overseeing authorities pre-determine who can access an object. MAC assigns security label (level of sensitivity) to objects and subjects, limiting access across labels. Examples of security labels are: unclassified, sensitive but unclassified (SBU), confidential, secret, and top secret; you can see more examples of classification with their description in [Solomon and Chapple 2005]. Security labeling confers to MAC the aspect of focusing on information confidentiality, and it is often associated with Bell-LaPadula confidentiality model. Biba model developed a similar method as Bell-LaPadula model, but it aims at providing information integrity.
- Role-based access control (RBAC) or task-based access control is a security policy that requires that access rights be assigned to roles rather than to individual subjects. Subjects obtain these access rights by virtue of being assigned membership in appropriate roles.
- Domain type enforcement (DTE) is an access control technology that restricts subject accesses according to a specific security policy. DTE is an extension of Type Enforcement (TE) and is itself extended into Dynamic Typed Access Control (DTAC). Implementing TE gives priority to MAC over DAC.

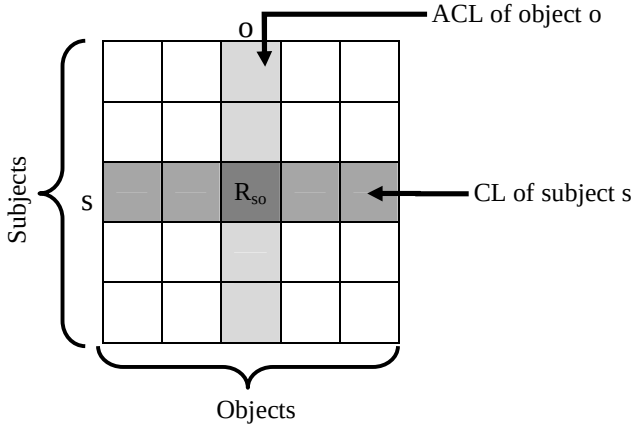


Figure 3 Access control matrix

An access control system comprises an access control mechanism and all information required to take decisions. An access control matrix (depicted in Figure 3) characterizes the rights of each subject with respect to every object in the system. The redundancy of access modes and complexity of matrix management render the model subject for theoretical analysis only. In practice, there are two implementations alternatives for access control matrix that have been developed: capabilities and access control lists.

- Capability list (CL) lists objects and rights associated with each subject. A capability is a reference to an object, which allows the subject, possessing it, to interact with an object in

certain ways. It refers to a value that references the object along with an associated set of access rights. Thus, capabilities encapsulate object identity and its location in memory.
- Access control list (ACL) lists subjects and rights associated with each object. ACL is a concept of privilege separation in which each object is associated with a set of doubles, each containing a subject and a set of rights. The list is a data structure (usually a table) which contains entries that specify subjects' rights for a specific object.

For centralized systems, the trusted OS assures the integrity of CL and subject identification in ACL. For distributed system, user management is flexible in CL if revocation operation which is difficult is not considered; however user management is complex in ACL although it allows a better resource control.

More sophisticated policies have been also suggested such as usage control [Sandhu and Park 2003] that extends traditional access control models: a usage decision is particularly made by policies of obligations and conditions. Obligations are actions that are required to be performed before or during the access process. Conditions are environment restrictions that are required to be valid before access or during access.

Access control is certainly the core function of a trusted OS. The trusted OS has the role of enforcing such control. There is generally a reference monitor that checks each subject's permission when it requests access. Permission is granted to the subject according to a security policy defined by the access control mechanism.

Trusted platform module

The TCG [26] (Trusted Computing Group) consortium, formerly named TCPA (Trusted Computing Platform Alliance), founded in October 1999 by Intel, IBM, Compaq, HP, and Microsoft, aims to "develop and promote open industry standard specifications for trusted computing hardware building blocks and software interfaces across multiple platforms, including PC's, servers, PDA's, and digital phones". Trusted environment can be established using its trusted platform modules (TPMs).

A trusted platform is "a platform that can be trusted by local users and remote entities to always behave in the expected manner for the intended purpose". To form a trusted platform, TCG's specifications define three components: a Trusted Platform Module (TPM), a Core Root of Trust for Measurement (CRTM), and a Trusted platform Support Service (TSS).

- The TPM is a hardware chip that is separate from the motherboard. TPM is a passive component, i.e. slave device that only performs action when asked (by operating system or application), and it does not have access to system resources. The TPM offers a physical true random number generator (RNG), cryptographic functions (i.e., SHA-1, HMAC, RSA encryption/decryption, signatures and key generation), and tamper resistant non-volatile memory (mainly used for persistent key storage). The TPM provides a set of platform configuration registers (PCRs) that are used to store measurements of hash values of the system/platform. The content of these registers can only be modified using the extending operation:

$$PCR_{i+1} \leftarrow SHA^{-1}(PCR_i||M)$$

with PCR_i the previous register value, PCR_{i+1} the new value, M a new measurement and || denoting the concatenation of values. The initial platform state is measured by computing

[26] https://www.trustedcomputinggroup.org/

cryptographic hashes of all software components loaded during the boot process. The Stored Measurement Log (SML) (also called the Event Log) is responsible for maintaining an ordered database for all the events for every PCR. SML may be stored outside the TPM because it can potentially grow very large. Another possible usage of the TPM is key storage: key and other data will be encrypted by a secret key only known to the TPM (binding storage), and the sensitive data can be bound to a certain platform configuration i.e., encrypted data with a reference to the configuration then only the platform in this configuration can read data (sealed storage).

- CRTM is the first code the platform executes when it is booted. The task of the CRTM is to compute a hash of the code and parameters and extend the first PCR register with this measurement. In this way a chain of trust is established from the CRTM to the operating system and potentially even to individual applications.
- The TSS offers "low level" API for applications and platforms. It is responsible for all kinds of functions that are necessary for communication with the rest of the platform or other platforms.

Smart cards

A smart card is a programmable device that has a full operating system in the dimensions of normally credit card size (ID-1 of ISO/IEC 7810 standard[27]). Therefore, applications can be managed and nowadays the preferred language to implement the application is Java Card, a subset of the Java language. A smart card has the same functionalities as a TPM (key storage, key generation, cryptographic engine). Additionally like a TPM, a smart card is a slave device, which is however portable. With this latter particular nomadic feature, a smart card typically belongs to a certain user, contrary to TPM that is physically bound to a computing platform. There are two broad categories of smartcards. Memory cards contain only non-volatile memory storage components, and perhaps some specific security logic. Microprocessor cards contain volatile memory and microprocessor components.

2.2. Overlay management layer

Given that P2P systems are highly dynamic networks of peers, organizing peers and resources in such networks requires deploying a network overlay that offers a good substrate for peer and resource management. An overlay network allows connecting peers by virtual or logical links built on top of a physical network to which the overlay is generally totally unrelated.

2.2.1. Classification

Overlays have been originally designed for file sharing applications to provide index storage as well as discovery and lookup services for peer content. Overlays can be classified based on their degree of centralization that illustrates the extent to which such overlays rely on one or more servers to facilitate the interaction between peers.

Centralized overlay

Centralized overlays, also termed "peer-through-peer" or "broker mediated", rely on a centralized server that maintains the metadata information. Content discovery and lookup are performed on the central server, and the end-to-end resource transfers are made between peers

[27] ISO/IEC 7810 in Wikipedia : http://en.wikipedia.org/wiki/ISO/IEC_7810

themselves. The Napster[28] architecture, which consists of a central index server to which peers logged in and uploaded metadata, is a perfect illustration of this architecture. The centralized approach quickly and efficiently locates data but it is vulnerable to the flash crowds, whereby popular data become less accessible because of the load of the requests on the central server, or plain denial of service. The server thus constitutes a single point of failure that is vulnerable not only to technical failure or malicious attack, but also to censorship.

Decentralized overlay

In decentralized overlays, peers combine the roles of servers and clients without the need for a central coordination of their activities. Decentralized overlays fall into two classes, depending if they have an unstructured or structured topology (see Figure 4).

An unstructured overlay is composed of peers joining the network without any prior knowledge of the topology.

- *Flat topology:* Gnutella[29] is the first system that makes use of unstructured overlays. To retrieve data over the overlay, Gnutella floods queries over the network with a limited scope. Therefore, the unstructured overlay does not scale when handling a high rate of aggregate queries and sudden increase in the system size because peers become considerably overloaded; even though the approach is effective in locating highly replicated data and is resilient to peers joining and leaving the system.
- *Hierarchical topology:* Another type of unstructured overlay is proposed by the FastTrack [Liang et al. 2006] two-tier architecture particularly used in KaZaA[30], Grokster[31], and iMesh[32]. In this architecture, some of the peers assume a more important role than the rest of the peers, acting as local central indexes as Napster for the data shared by local peers. These peers, called "super-peers", are selected for these special tasks and do not constitute single points of failure like in Napster, since they are dynamically assigned and replaced if they are subject to failure or attack. However, the architecture may create islands of sub-networks that are not connected to each other which slows discovery and lookup of shared data.

A structured overlay does not place data at random peers but at specified locations with a topology that is tightly controlled.

- *DHT-based topology:* Such structured overlays use the Distributed Hash Table (DHT) as a substrate such as CAN [Ratnasamy et al. 2001], Chord [Stoica et al. 2001], Pastry [Rowstron and Druschel 2001], or Tapestry [Zhao et al. 2000]. DHT-based overlay consistently assigns uniform random IDs to the set of peers into a large space of identifiers. Data objects are assigned unique identifiers called keys, chosen from the same identifier space. Keys are mapped by the overlay network protocol to a unique live peer in the overlay network. Although structured overlays can efficiently locate rare data items since the key-based routing is scalable, they incur significantly higher overheads than unstructured overlays for popular content.

[28] http://www.napster.com/
[29] http://www.gnutella.com/
[30] http://www.kazaa.com/
[31] http://www.grokster.com/
[32] http://imesh.com

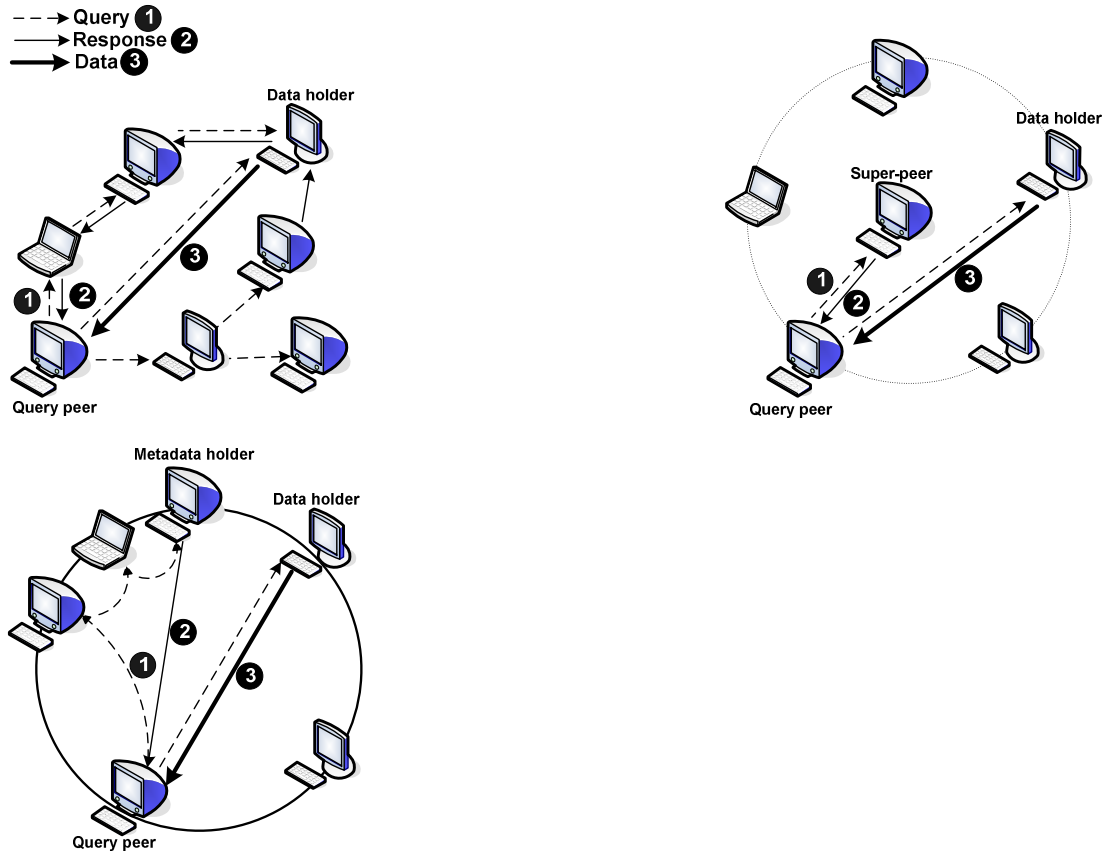


Figure 4 Decentralized overlay: (a) flat topology, (b) hierarchical topology, and (C) DHT-based topology.

Choosing structured or unstructured overlay for data storage, discovery, and lookup comes with some tradeoffs for file sharing applications. However, our considered application differs from file sharing: in the latter, peers may not actually know the location of the data and therefore need to query the network overlay to search for such information; whereas in our application, the owner may know the location of its stored data (if the). Still, an overlay is required to provide the storage of metadata in addition to other services related to peer organization like for instance the random selection of potential holders or verifiers within the P2P network.

2.2.2. Metadata usage and management

Metadata must be introduced into the overlay for describing the attributes relative to data stored (e.g., name, size, ownership, and type), their structures (e.g., length and fields), their location, or a short description of their content. Moreover, security metadata also must be introduced specifically for data storage applications. Each peer may store data and replicate them at other multiple peers. The data must initially be encrypted before being stored at the holder so as to protect them. Decryption keys are either kept by the owner (single reader) or distributed to peers allowed to access the data (multiple readers) as metadata. Signed certificates

in the form of capability-based access rights may also be granted and distributed as metadata, hence offering a low granularity of access to data. A peer wanting to retrieve some data will thus need to refer to metadata information for identifying the location of their holders, decrypting the data, or even for verifying their integrity.

Such metadata can be handled by the owner peer itself or made available at a centralized entity that cares about providing this type of information online or offline, or distributed to peers in the network that cooperatively administer this information either within a structured or unstructured network overlay. For instance, [Srivatsa and Liu 2005] proposes the LocationGuard algorithm for hiding the location of application data or of an online service on a large overlay network. The location hiding algorithm uses location keys to implement a capability-based access control mechanism whereby the token (capability) is a pseudo-filename generated from file's name and its location key.

2.2.3. Peer identification

Peer identification in a P2P network is a very important security issue. The lack of strong peer identities and the dynamicity of P2P networks with arrivals and departures of peers may lead to the problem of whitewashing whereby misbehaving peers leave the system and come back with new identities to avoid any penalty because of their misbehavior.

Additionally, without proper peer identification, the system is vulnerable to the Sybil attack (also formerly known as pseudo-spoofing) where the attacker masquerades under multiple simultaneous identities in order to gain an unfair advantage over system resources. Completely eliminating Sybil attacks can only be provided by trusted certification as proven by Douceur [Douceur 2002]. Trusted devices associated with peers can be used as an implementation to eliminate such attacks, even though a peer may buy multiple devices and then acquire multiple identities yet at a high cost. [Balfe et al. 2005] proposes a pseudonymous authentication scheme for P2P networks based on TPMs. The scheme assumes TCG-enabled peers with appropriate Direct Anonymous Attestation (DAA) credentials. DAA is an approach (supported in version 1.2 of TCG specification) that relies on cryptographic techniques to ensure the privacy of TPM users, without introducing the requirement for special trusted third parties (e.g., privacy CA). Based on DAA approach, the TPM proves that it has knowledge of a specific TPM-controlled, non-migratable secret value (the value is not revealed during the process). With the scheme of [Balfe et al. 2005], any peer is able to verify the pseudonym of another peer by challenging this latter to supply a DAA signature on some message. Such verification allows checking that the peer has a valid credential supplied by a particular issuer and also that the pseudonym is determined as a function of the P2P network name. Peers can thus be authenticated without revealing their TPM identities in the process.

Without a trusted infrastructure, Sybil attacks can only be mitigated at best. Mitigation can be achieved by relying on the topology, for instance through the test of a peer IP address range. It can also be achieved more indirectly by making the newcomer pay with computation resources, bandwidth, or storage capabilities, such as for example with crypto-puzzles [Vishnumurthy et al. 2003]. Other techniques like SybilGuard [Yu et al. 2006] rely on prior trust relationships e.g., real-world friendship between peer owners to detect Sybil attackers. [Lesueur et al. 2008] even enhances the SybilGuard approach by controlling the number of peer invitations that a group member possesses.

In the three latter methods, the costs are only one-time paid by Sybil attackers and can be then amortized during the rest of the system operation. As discussed in [Levine et al. 2006], such costs can be periodically paid by repeatedly performing resource testing on peers, thus confining the potential return on investment of Sybil attackers to a limited time slot. Even though all these proposed approaches for limiting Sybil attacks without trusted infrastructure are

scalable compared to certification-based approaches, they incur a huge cost overhead not only for Sybil attackers but also for honest newcomers, which may undermine their practicability and tolerability in actual implementations. In that respect, [Feldman et al. 2006] for instance shows that imposing a penalty on all newcomers significantly degrades system performance when the peer churn rate is high.

2.2.4. Peer random selection

In a centralized overlay, the random selection of peers can obviously be performed by parsing the list of peers registered to the network and selecting a random subset of them. Peers may also be organized in an unstructured or a structured network overlay. Both types of overlays permit the random selection of peers.

In unstructured overlays, the random selection may be based on the random walk. To solicit a number of random peers, the requesting peer starts the random walks at a subset of its neighbors chosen randomly, and runs them for TTL (time to live) steps. Each intermediate peer involved in the walk forwards the query message to a randomly chosen sub-set of its neighbors until the TTL is reached. The final peers are considered as the randomly selected peers. The random walk approach is proven to be inherently scalable [Zhong et al. 2008] because its communication overhead does not increase as the network size grows. In structured overlay, random selection of peers can be realized by randomly choosing a value from the number space, and routing to that value. However, the problem of random peer selection boils down to the problem of assigning identifiers appropriately in the network overlay. Such identification is prone to Sybil attacks that threaten also the selection based on the random walk.

2.3. Trust and cooperation layer

In P2P systems, peers often must interact with unknown or unfamiliar peers without the help of trusted third parties or authorities to mediate the interactions. As a result, peers trying to establish trust towards other peers generally rely on cooperation as evaluated on some period of time. The rationale behind such trust is that peers have confidence if the other peers cooperate by joining their efforts and actions for a common benefit.

2.3.1. Classification

Trust between peers can be achieved in two essential ways that depend on the type and extent of trust relationships among peers and that reflect the models and trends in P2P systems (the used taxonomy is depicted in Figure 5). Static trust based schemes rely on stable and preexisting relationships between peers, while dynamic trust is relying on a real-time assessment of peer behavior.

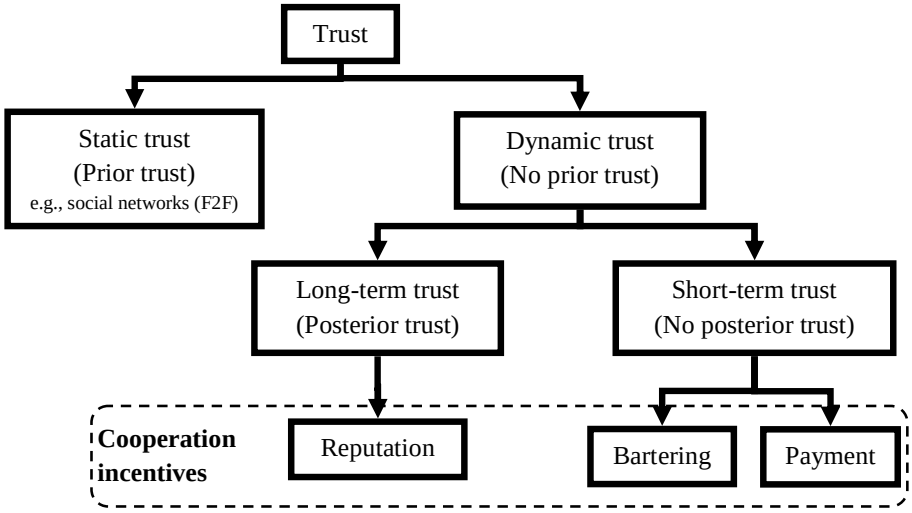


Figure 5 Trust taxonomy

Among other taxonomies have been proposed, [Obreiter and Nimis 2003] classifies cooperation enforcement mechanisms into trust-based patterns and trade-based patterns. Obreiter et al. distinguish between static trust, thereby referring to pre-established trust between peers, and dynamic trust, by which they refer to reputation-based trust. They analyze trade-based patterns as being based either on immediate or on deferred remuneration. Other authors describe cooperation in self-organized systems only in terms of reputation based and remuneration based approaches. Trust establishment, a further step in many protocols, easily maps to reputation but may rely on remuneration as well. In this work, we adhere to the existing classification of cooperation incentives in distinguishing between reputation-based and remuneration-based approaches.

Static trust

Peers may have prior trust relationships based for example on existing social relationships or a common authority. In friend-to-friend (F2F) networks, peers only interact and make direct connections with people they know. Passwords or digital signatures can be used to establish secure connections. The shared secrets needed for this are agreed-upon by out-of-band means. Turtle [Popescu et al. 2004] is an anonymous information sharing system that builds a P2P overlay on top of pre-existent friendship relations among peers. All direct interactions occur between peers who are assumed to trust and respect each other as friends. Friendship relations are defined as commutative, but not transitive.

[Li and Dabek 2006] proposes a F2F storage system where peers choose their storage sites among peers that they trust instead of randomly. Compared to an open P2P storage system, the proposed approach reduces the replication rate of the stored data since peers are only prone to failure not to departure or misbehavior. However, the approach is more applicable to certain types of storage systems like backup since it provides data durability not generally data availability: peers may not often leave the system but they me be offline. F2F-based approaches ensures the cooperation of peers which results in enhanced system stability and reduces administrative overhead; even though these approaches does not help to build large scale systems with large reserve of resources.

Trust can be established using a trusted authority like in UbiStorage[33]. Such service proposes a file system that is based on a distributed trusted infrastructure of servers built over a network of peers. Indeed, the service distributes dedicated terminals, named “néobox”, to peers. These terminals are used to securely store other peers’ data. Dedicated devices, named “throwboxes”, are also used to increase the capacity of an ad hoc network in [Zhao et al. 2006]: throwboxes store node messages before being transmitted to their destination.

Dynamic trust

A P2P storage system may rely on the cooperation of peers without any prior trust relationships. Some trust is then established as peer interactions progress, through cooperation incentive mechanisms. Peers describe their trust towards each other either directly based on reputation, or indirectly through payment incentives, money being an indirect though not always fully meaningful measure of how trusted some peer might be. The lack of prior trust between peers allows building open large scale systems that are accessible to the public. Storage systems with cooperation incentives perhaps result in more overhead than with prior trust based approaches; but however the reliability of the stored data is increased since data will be generally stored in multiple copies at different worldwide locations rather than confined at one or limited number of locations.

Inciting peers to adopt a cooperative behavior can only achieved efficiently if peer behavior can be correctly assessed. Therefore, cooperation incentive mechanisms should comprise verification methods for measuring the actual peer contributions to the P2P system. The evaluation of the behavior of each peer allows determining the right incentives to stimulate its cooperation. In turn, such incentives guide the peer in adapting its contribution level. A peer might in particular choose the best strategy that maximizes its utility gained from the system: it compensates the cost incurred due to its potential contribution with the incentives received in support for its cooperation. With such a cyclic process, the system dynamically reaches the status of “full” cooperation between peers.

Figure 6 depicts the feedback loop illustrating the correlation between peer assessment, cooperation incentives and peer strategies.

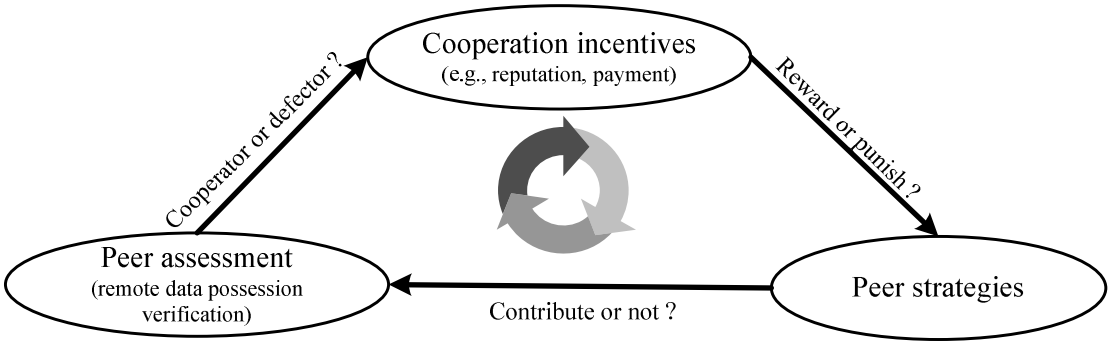


Figure 6 The feedback loop of dynamic trust

2.3.2. Peer assessment

An evaluation of the peer behavior can be performed at different timescales. An immediate evaluation of the peer behavior is only possible if the peer contribution occurs immediately like

[33] http://www.ubistorage.com/

in packet forwarding application (e.g., [Michiardi 2004] and [Buttyán and Hubaux 2003]). Otherwise, peer evaluation is deferred to the completion of the peer contribution as it is the case in data storage applications. This constitutes a problem for storage applications since misbehaving peers are left with an extensive period of time during which they can pretend to be storing some data they have in fact destroyed for free-riding or for purely malicious purposes.

Proof of data possession

Periodic peer evaluation can be achieved through proof of knowledge protocols that have been called interchangeably remote integrity checking [Deswarte et al. 2004], demonstration of data possession [Filho and Barreto 2006], proofs of data possession [Ateniese et al. 2007], or proofs of retrievability [Juels and Kaliski 2007]. Chapter 3 suggests three verification protocols of this type with various tradeoffs.

Such protocols are used as an interactive proof between the holder and the verifier or possibly the owner, in which the holder tries to convince the verifier that it possesses these very data without actually retrieving them. Interaction is based on challenge-response messages exchanged between the holder and the verifier. Verification of the holder's response is permitted through some information kept at the verifier side.

Verification protocols generally require storing security metadata for verification purposes, which are much smaller than data themselves, making it possible for multiple peers to perform periodic verifications at multiple holders. Verifiers may comprise not just the owner but also peers from the network selected and appointed by the owner. The distribution of behavior assessment tasks to multiple peers allows deploying the P2P storage application in a large scale. It also allows mitigating the selfishness of verifiers whereby they give up performing the verification task or send bogus verification results to the owner.

To ensure that a data is periodically verified, we may rely on some trusted devices rather than on the delegated verifiers. Distributed online TTPs may perform the verification operations over the data at several holders. Such TTPs may be represented as dedicated devices distributed over the network. They have storage capability and they are easy to deploy in the field without access to any infrastructure. TTPs may correspond to TPMs or smart cards that are held by each peer and that periodically perform verification of peer's storage. This approach reduces the communication overhead of the verification process since verification messages are exchanged between the TPM or the smart card and its holder.

However, the peer may detach such device avoiding its storage to be verified. Therefore, we should ensure that the incentives for cooperation must be enforced through the trusted device i.e., the peer cannot use the storage system without contacting its device, as illustrated by [Buttyán and Hubaux 2001]. For instance, verification operations whose outcome is positive increases a counter in the trusted device, such counter if it exceeds a certain threshold enables the very peer to store its own data into the system. Additionally, we may resort to a trusted OS that controls several peer functionalities such that if ever the peer selfishly disconnects from the network the trusted OS reduces the number of services or even degrades the quality of service offered to that peer.

Distribution of peer assessment information

Verification results obtained from the periodic auditing of holders can be kept private by the verifiers or instead distributed to peers other than the data owner. Private information is certainly objective but achieves a very local view about peer behavior confined to a small subset of network peers; even though delegating the verification task to multiple verifiers increases the size of such subset.

Cooperation incentive approaches have proven to be more successful (see [Lai et al. 2003]) if they rely on an objective and shared history of peer actions to compute reputation ratings.

Results obtained from remote data possession verification protocols provide such objective evaluations of effective data storage, however the knowledge of such results, which we call audits, are limited to the verifiers and the owner of the verified data. Verification results can be disseminated using a centralized or decentralized overlay network. A centralized entity may distribute verification results. It collects all information about the behavior of peers in the system, and then disseminates a history of peers' behavior to any peer either in a regular-basis or on-demand. However, the centralization creates a bottleneck problem at the central entity. The verification information can also be provided within a structured or unstructured overlay where peers may search for this kind of information, for instance through a random walk to collect the information from random peers [Anceaume and Ravoaja 2006] or through score managers that are assigned within a DHT to track the behavior of a given peer [Kamvar et al. 2003]. These approaches make collected information that may be subjective or even incorrect available to the other peers in the system. In Section 5.2.3 of Chapter 5, we introduce an analytic model that proves that this kind of indirect approach degrades the quality of collected evidences.

2.3.3. Cooperation incentives

Peer behavior assessment forms the basis of an efficient cooperation incentive mechanism. From such an evaluation, well-behaved peers will be rewarded with incentives while ill-behaved peers will be punished. Incentives may consist in exchanging identical resources (Barter), or in conferring good reputation to the well behaved peers, or in providing well behaved peers a financial counterpart for their cooperation.

Bartering

Barter based approaches do not require the interacting peers to have any preset trust relationships. They rather rely on a simultaneous and reciprocal behavior. The exchange of resources in particular takes place if both peers cooperate with each other; otherwise, there is no exchange.

Cooperation incentives may be cheaply built on a tit-for-tat (TFT) strategy ("give and ye shall receive"). The peer initially cooperates, and then responds likewise to the opponent's previous action: if the opponent previously cooperated, the peer cooperates; otherwise, the peer defects. TFT is demonstrated to be an evolutionary stable strategy (ESS) in game theory jargon: this strategy cannot be invaded (or dominated) by any alternative yet initially rare strategy.

In the Cooperative Internet Backup Scheme [Lillibridge et al. 2003], each peer has a set of geographically-separated partner peers that collectively hold its backed up data. In return, the peer backs up a part of its partners' data. To detect free-riding, each peer periodically evaluates its remote data. If it detects that one of its partners dropped the data, the peer establishes a backup contract with a different partner. Since the scheme relies on identical and immediate resource exchanges, peers must be able to choose partners that match their needs and their capabilities and that ensure similar uptimes. To this end, a central server tracks peers and their partners. Decentralized methods of finding partners in a Gnutella-like flooding approach are also suggested although not evaluated in [Lillibridge et al. 2003].

However, TFT is not perfect as illustrated by the P2P file sharing protocol BitTorrent[34]. In BitTorrent, unchoking a peer means that the peer is accepted to upload files for it. Peers follow a TFT strategy by unchoking peers that provide the highest throughput for them, and besides that they use an optimistic unchoking strategy to discover potentially better trading peers. However this strategy of (probabilistically) cooperating with newcomers blindly can be

[34] http://www.bittorrent.com/

exploited by whitewashers (peers that repeatedly join the network under new identities to avoid the penalty imposed on free-riders). [Piatek et al. 2007] describes the design of BitTyrant, a selfish client that demonstrates that BitTorrent incentives don't build robustness. The reason is that TFT is no longer an evolutionary stable strategy in the presence of whitewashers.

Reputation

Reputation relies on the evaluation of the past behavior of a peer for deciding whether to cooperate with it. Reputation then builds a long-term trust between peers based on a statistical history of their past interactions. This allows going beyond barter-based approaches (direct reciprocity) by permitting to several peers to indirectly reciprocate to the behavior of the observed peer.

A reputation mechanism consists of three phases (summarized in Figure 7):

1. *Collection of evidence:* Peer reputation is constructed based on the observation of the peer, on experiences with it, and/or on recommendations from third parties. The semantics of the information collected can be described along two dimensions:
    - **Specific vs. general information:** specific information about a given peer relates to the evaluation of its functionality such as its ability to deliver a service on time, which general information evaluates all its functionalities (e.g., measured as a weighted average).
    - **Objective vs. subjective information:** objective information (also known as direct or private information) can be obtained about a given peer through past interactions, while subjective information (also known as indirect or public information) refers to either listening to messages intended for other peers or to using the opinion of others about the peer. A message can also voluntarily piggyback evaluations collected by other peers as extra information.

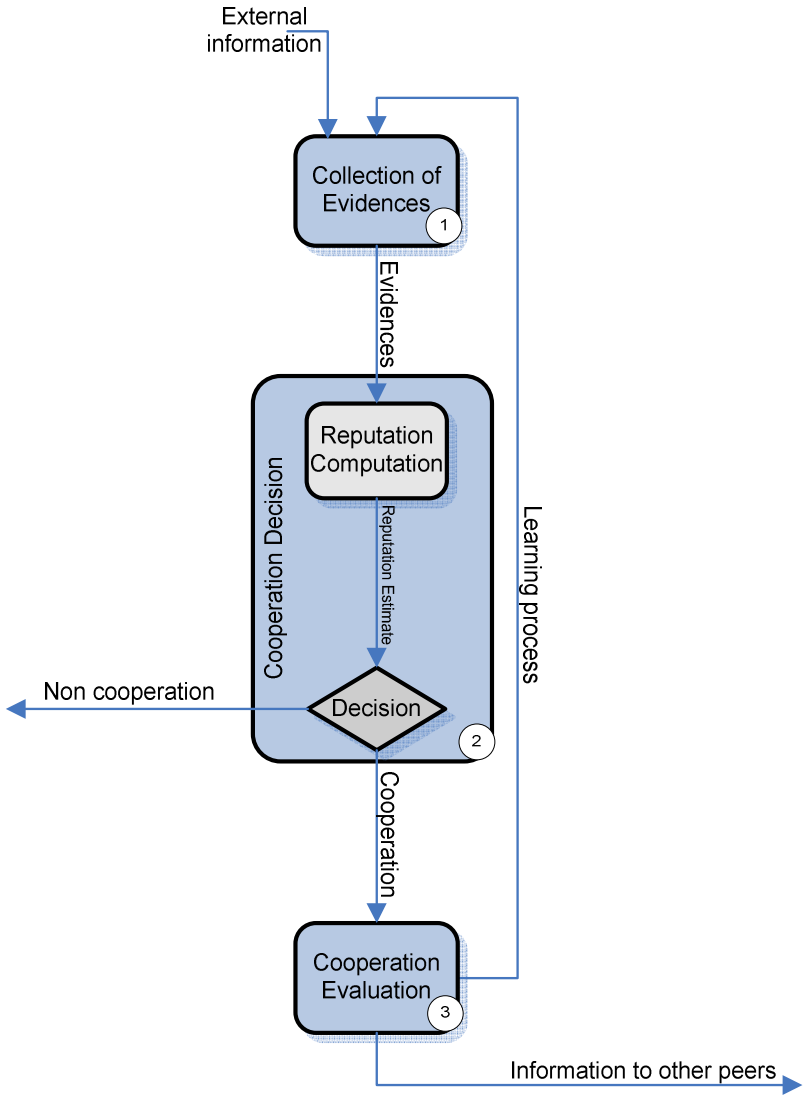


Figure 7 Reputation: diagram of operations

2. *Cooperation decision:* Based on the collected information, a peer can decide whether it should cooperate with another peer, based on the reputation of that other peer. There exists a variety of methods for computing the reputation of an entity such as voting, averaged ratings, Bayesian computation ([Jøsang and Ismail 2002] and [Mui et al. 2001]), or the flow model (e.g., PageRank [Page et al. 1998] algorithm for ranking Google[35]'s web pages[36] and EigenTrust [Kamvar et al. 2003]). More details can be found in [Jøsang et al. 2005].
3. *Cooperation evaluation:* The occurrence of an interaction with a peer is conditional on the previous phase. After interaction, the degree of cooperation of the peer involved is

[35] http://www.google.com
[36] The public PageRank measure does not fully describe Google's page ranking algorithm, which takes into account other parameters for the purpose of making it difficult or expensive to deliberately influence ranking results in what can be seen as a form of "spamming".

determined. Peers performing correct operations, that is, behaving cooperatively, are rewarded by increasing their reputation accordingly. A peer with a bad reputation will be isolated from the functionality offered by the group of peers as a whole.

The reputation mechanism may rely on an online TTP to collect auditing information, compute the reputation of the corresponding holder and disseminate such information to the rest of peers, either periodically or on-demand. However the approach results in a bottleneck problem and does not scale to large populations of peers. This task of reputation collection and dissemination can be attributed not to a single TTP but instead to several ones, even though decreasing the total shared history of peer actions while achieving a more scalable system.

Reputation values can be handled by peers through the use of TPMs or smart cards. These devices would then verify the data that the device holder has promised to store, compute the reputation of this latter accordingly, and distribute the reputation information to other peers when requested. The computed reputation would then provide an accurate and complete record of all peer actions. However, this approach may be still vulnerable to attacks whereby the peer maliciously disconnects the trusted device from the network. This would however prevent the peer from using the system and storing its data at other peers without presenting an up-to-date reputation certified by its device. For instance, a potential holder may request a data owner reputation by selecting a random number as a nonce. Then, the owner should send back its reputation along with the nonce certified by its trusted device.

Reputation may also just rely on peers themselves that compute reputation ratings for each other peer based on their personal experiences. The learning process of such ratings may be made fast by considering groups of peers rather than the whole system of peers. Group members interact with each other and accordingly compute reputation ratings for each other. The reputation approach based on this structure is describes in more detail in Section 5.2 of Chapter 5.

Payment

In contrast to reputation-based approaches, payment-based incentives constitute an explicit and discrete counterpart for cooperation and provide means to enforce a more immediate form of penalty for misconduct. Payment based approaches make it possible to secure short-term interactions between peers without relying neither on prior trust nor on some long-term history.

Payment brings up new requirements regarding the fairness of the exchange itself [Asokan et al. 1997]. This in general translates to a more complex and costly implementation than for reputation mechanisms. In particular, payment schemes require trusted third parties (TTP) such as banks; these entities do not necessarily take part in the online service, but may be contacted to resolve payment litigations. Tamper proof or tamper resistant hardware (TPH/TRH) like secure operating systems or smart cards have also been suggested as a distributed implementation of such a TTP.

A payment scheme comprises four main phases (summarized in Figure 8):

- *Negotiation:* Two peers may negotiate the terms of their interaction. Negotiating the remuneration in exchange for an enhanced service confers a substantial flexibility to the mechanism. The negotiation can be performed either between the participating peers or between peers and an authority if available.
- *Cooperation decision:* The peer is always the decision maker in a self-organizing system. During negotiation and based on its outcome, a peer can decide whether it will cooperate.
- *Cooperation evaluation:* Cooperation is evaluated by the service requesting party in terms of adequacy of the service to the request, as well as by the service providing party,

in terms of adequate remuneration. Ensuring the fairness of both evaluations may ultimately require involving a trusted third party. Depending on the service, this TTP will ensure a fair exchange for every interaction, or may only be involved if arbitration is requested by one party (see below). The TTP, which may be centralized or distributed itself, may for instance give access to information unavailable to a peer, or more generally provide a neutral execution environment.

- *Remuneration:* The remuneration can consist in virtual currency units (a number of points stored in a purse or counter) or real money (banking and micropayment), or bartering units (for instance quotas defining how a certain amount of resources provided by the service may be exchanged between entities). The latter can even be envisioned in the form of micropayments [Jakobsson et al. 2003]. Regarding real money, this solution assumes that every entity possesses a bank account, and that banks are enrolled in the cooperative system, directly or indirectly through some payment scheme. The collaborating peer is remunerated by issuing a check or making a transfer of money. In the first case, remuneration implies that a number of points are added to a counter connected in some way with the collaborating peer. The remuneration effectiveness may be immediate or delayed after a certain number of steps (e.g., reservations, then remuneration in several phases for different services).

These phases can be executed repeatedly to perform some cooperative service on a finer granularity basis, which may ease cooperation enforcement. In particular, micropayment is often envisioned rather than an actual macro-payment in remuneration based cooperation enforcement mechanisms. With this scheme, trust establishment essentially relies on the presence of peers in the system, that is, their continued ability to pay proves they cooperated.

Achieving an effective implementation of payment-based mechanism depends upon the realization of a protocol that enforces the fair exchange of the payment (credits) against some task: “A fair exchange protocol can then be defined as a protocol that ensures that no player in an electronic commerce transaction can gain an advantage over the other player by misbehaving, misrepresenting or by prematurely aborting the protocol” [Asokan et al. 1998]. The fair-exchange may be enforced through a TTP that may be used online or opportunistically. TPMs or smart cards may also be employed to carry out a fair-exchange protocol in a distributed fashion.

In a P2P network, TTPs may be represented as super-peers that play the same role as an online TTP but in a distributed fashion. One example of such architecture is FastTrack [Liang et al. 2006] which is used in P2P networks like KaZaA[37], Grokster[38], and iMesh[39]. These networks have two-tier hierarchy consisting of ordinary nodes (ONs) in the lower tier and super-nodes (SNs) in the upper tier. SNs keep tracks of ONs and other SNs and act as directory servers during the search phase of files. Additionally, one way of implementing a payment scheme would be to use super-peers distributed within the P2P network as a trusted infrastructure for payment. These super-peers would provide neutral platforms for performing an optimistic fair exchange protocol. The use of such an infrastructure of trusted peers, that would not necessarily need to be related with the payment authority, may make sense, in particular in relationship with content distribution networks (CDNs)[40]. Such networks involve the deployment of managed workstations all over the Internet, thereby providing a nice platform for payment functionalities.

[37] http://www.kazaa.com/
[38] http://www.grokster.com/
[39] http://imesh.com
[40] E.g., Akamai technologies, inc. http://www.akamai.com/

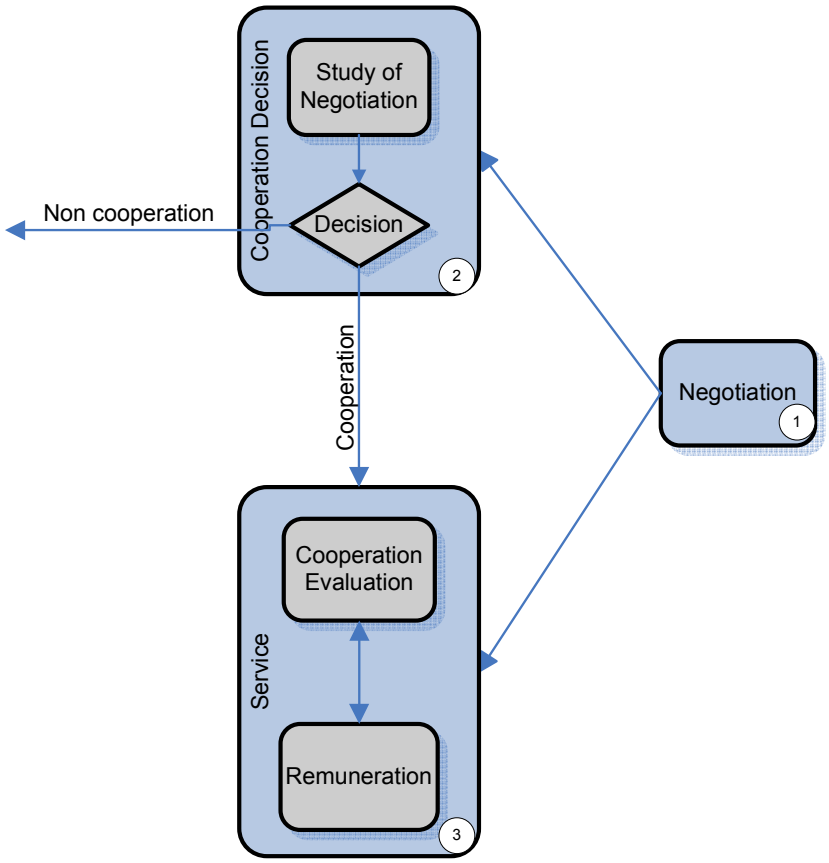


Figure 8 Payment: diagram of operations

The scale of the storage system makes it necessary to resort to a new type of protocols called optimistic protocols [Asokan et al. 1997] whereby the TTP does not necessarily take part in peer interactions, but may be contacted to arbitrate litigations between peers. In the cooperative backup system of [Lillibridge et al. 2003], a central server considered as a TTP tracks the partners of each peer participating in the backup system. Partners of a peer are peers that collectively hold its backed up data. In return, the peer backs up a part of its partners' data. Each peer takes note of its direct experience with a partner, and if this partner does not cooperate voluntarily or not beyond some threshold, the peer may decide to establish a backup contract with a different partner that is obtained through the central server.

TPMs supported approaches have been suggested within the TermiNodes [Buttyán and Hubaux 2001] and CASHnet [Weyland et al. 2005] projects. Both schemes address the security of the networking function of packet forwarding through remuneration schemes. Each device possesses a TPM that manages its account by maintaining a counter that is interpreted as a currency. However, TPM-based approaches suffer from additional attacks: if the peer device of a non cooperative or malicious user is disconnected from the other peers, their credits/tokens might not be available, which might raise starvation issues. However, the use of secure operating system as a TPM might make it possible alleviate this problem notably by more completely controlling and possibly reducing the device functionalities if the peer does not connect to the system network.

Smart cards have been used in the P2P storage system PAST [Druschel and Rowstron 2001] to ensure the fairness of peer contributions. Smart cards issued by a third party are held by each PAST peer to support a quota system that balances supply and demand of storage space in the system. Peers cannot use more remote storage than they are providing locally. With fixed quotas and expiration dates, peers are only allowed to use as much storage as they contribute.

If data storage should be achieved in a large-scale and open P2P system, designs based on a trusted environment may be unfeasible or unmanageable. In that case, implementing the optimistic fair exchange protocol would have to be done by relying solely on peers. [Asokan et al. 1998] describes design rules for such cryptographic protocols making it possible to implement appropriate fair-exchange protocols. For instance, the distribution of the banking function to multiple peers may make easier the realization of a scalable system that does not have recourse to a trusted environment. In the KARMA framework [Vishnumurthy et al. 2003], the exchange of payment against some task is supported by multiple peers that collaborate to provide a fair exchange even though mitigated by the selfishness of the latter.

2.4. Application layer

The application-level layer is concerned with the service that is installed at each peer machine. A peer should store other peers' data and keep them available for them. Additionally, it should correctly answer verifiers' challenges based on the stored data.

2.4.1. Shared storage management

In the proposed P2P data storage application, the shared resource is the extra storage space spared at each peer that is used to set up a remote data storage facility.

The common technique to provide data reliability is realized by disseminating the data into multiple copies in the network. Data redundancy can be implemented through either replication or erasure coding. With replication, the copy is a simple duplicate of the data. Whereas, erasure coded copies are coded blocks such that any threshold sized set of these blocks allows generating the original data. Redundancy entails that the size of the actual consumed storage space is larger than the data size. The overhead introduced by data redundancy can however be coped with. For instance, Wuala[41] reduces the remote storage space allocated to a peer in exchange of an equivalent local storage space based on its probability of being online: the unallocated storage space serves for trading space on other peers in order to achieve a redundant storage [Toka and Michiardi 2008].

The preservation of the remote data is handled by their holders. The management of the shared storage falls then directly in the individual sphere of the holder, thus corroborating the idea of peer cooperation as a requirement for this type of storage system.

Verifiers, which are delegates of the data owner, operate a double check on the remote storage. The data holder should correctly respond to periodic verifier challenges and also send back the data whenever their owner wants to retrieve them.

Providing data availability and survivability is not just the concern of their owner. Several holders and verifiers contributed to this task. The distribution of the work to multiple peers limits the selfishness of holders or verifiers.

Peer cooperation in providing storage resources is stimulated through the trust and cooperation layer (discussed in the previous section). The fairness of peer contributions is particularly regulated by the cooperation incentives that work as a quota system: peers consume

[41] http://wua.la/en/home.html

storage resources from the system because they contribute such resources to other peers. Other type of resource utilization (e.g., bandwidth) related for instance to the performing the verification task must also be stimulated and regulated.

2.4.2. Multi-service framework

There is a potential interest in providing a general framework of cooperative services instead of one specific to P2P storage, mainly in case where a peer desires to store data in the P2P network without sacrificing its own storage space. This situation may be rendered possible by just making the peer contribute to the community of peers with other resources that it has in abundance. The P2P storage service may be then combined with other resource sharing services that relate for example to the bandwidth (file sharing), computation, or even networking. Each peer participates to a collection of services which the peer retains some of them for consumption and others for contribution.

Payment-based approach

Remuneration (e.g., real/virtual money, token) can be regarded as a neutral counterpart that can be traded for any cooperative service. Therefore, a system based on payment-based cooperation incentives is able to allow peers simultaneously accessing multiple cooperative services (e.g., remote storage, cloud computing, distributed database).

The evaluation of the good behavior of peers should be performed separately and independently using verification mechanisms specifically designed for each service.

However, the remuneration for a service can be operated with the same manner. For instance, remuneration may rely on auctions (like in the KARMA framework [Vishnumurthy et al. 2003]) to better cope with the effect of changes in supply and demand on service prices. Each peer contributing with a service might first auction the offered service and then supply the service to the winning bidder. The service delivered by the peer would then be checked to evaluate whether it corresponds to the advertised offer. Such an evaluation permits to determine if the service provider is worthy of the remuneration earned in counterpart to the service (cf. Figure 9).

Trusted OS based approach

Additionally, peer cooperation in a multi-service framework can be enforced through the use of trusted OS. Each peer's device incorporates a trusted OS that controls the access of the peer to resources and services and may exploit such control to stimulate or even force the peer to cooperate to the P2P system in a strictly fair manner. The cooperation enforcement may be illustrated through *service differentiation*: a cooperative peer will have a good quality of service (e.g., high bandwidth) and a non cooperative peer will have a bad quality of service (e.g., intermittent connection to the P2P network).

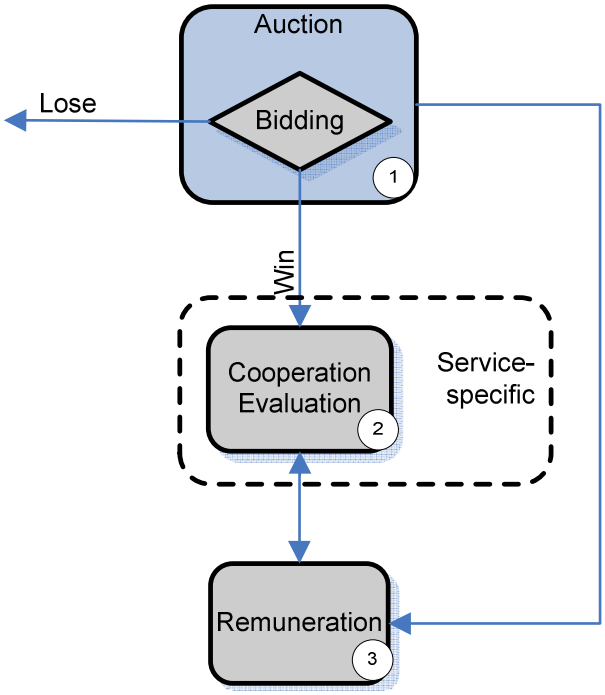


Figure 9 Multi-service framework based on payment

Data storage verification would thus serve two different functions:

- *Peer evaluation:* Based on peer assessments, the framework computes and maintains a counter that reflects the degree of peer cooperation. The counter employs different weights for resources.
- *Service differentiation:* The framework makes use of the counter as an indicator of the level of quality of service that the peer deserves. Based on the value of the counter, peer services are either upgraded or degraded and such service differentiation is enforced through the trusted OS.

The peer evaluation function of the framework requires the employ of a remote data possession verification protocol to check the fair and correct storage sharing between peers. The verifier for such protocol may correspond to a service implemented within the trusted "userland" part (not the kernel) of the trusted OS.

To be able to design the framework, we may use the cooperation counter as a context information modifying a dynamic access control policy: any change in the value of the counter would result in a change in the access rights to services granted to the user.

Alternatively, we may make use of the Flask security architecture [Spencer et al. 1999] as used in the SELinux (Security-enhanced Linux) operating system to enforce the security policy in a flexible way. Such a security architecture (see Figure 10) divides the responsibility for security into an object manager part and a security server part. The object manager controls every object invocation by checking every object request through the security server. This latter contains a complete representation of the security policy. With such an architecture, the security policy is consulted for every security decision, and thus can manage the revocation of previously granted access rights. For instance, let us consider a user that had previously access to a given service but who was in the meanwhile uncooperative. His cooperation counter will be

diminished until reaching the point where the access to the very service will be revoked to him in its subsequent object (service) request.

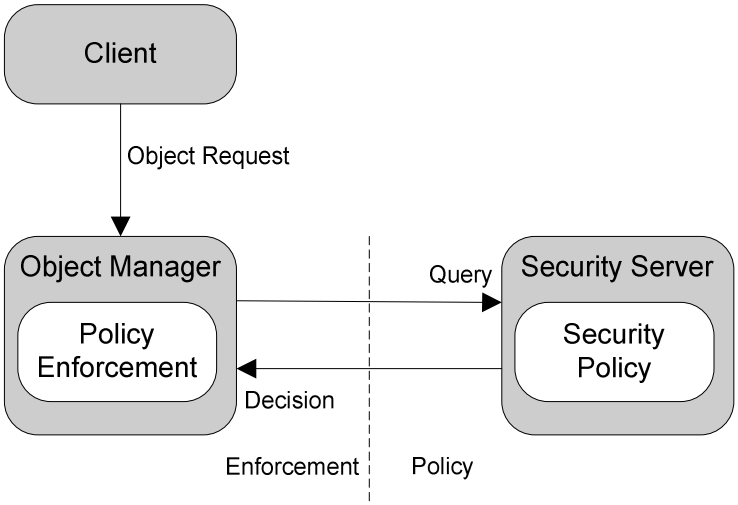


Figure 10 The Flask security architecture

2.5. Summary

This chapter describes the many elements of an architecture adapted to the secure P2P storage problem. The most important elements of this architecture are the overlay and trust management elements. We have identified techniques of management and implementation of the blocks that make up these layers and particularly how they may be enforced with a trusted environment.

The chapter describes several ways in which our cooperation incentive mechanisms may benefit from a trusted environment in order to improve peer behavior evaluation and motivate or enforce the cooperation of peers and the fairness of their contributions.

Finally, we described how it would be possible to design a multi-service framework based on trusted OSes for offering peers the opportunity to select the resources that they prefer to contribute in order to cope with the use of heterogeneous resources, capabilities, and needs. Although the use of trusted environment may make deploying our mechanism more costly, this would of course be mitigated by the deployment of more efficient security measures. The rest of the thesis studies how to ensure a correct operation of the P2P storage system by relying solely on peers, in particular based on remote data possession verifications.

Chapter 3

Remote data possession verification

Self-organizing data storage must ensure data availability on a long term basis. This objective requires developing appropriate primitives for detecting dishonest peers free riding on the self-organizing storage infrastructure. Assessing such a behavior is the objective of data possession verification protocols. In contrast with simple integrity checks, which make sense only with respect to a potentially defective yet trusted server, verifying the data possession remotely aims at detecting voluntary data destructions by a remote peer. These primitives have to be efficient: in particular, verifying the presence of these data remotely should not require transferring them back in their entirety; it should neither make it necessary to store the entire data at the verifier.

3.1. Problem Statement

This section describes the requirements that should be met by a self-organizing storage verification protocol.

We consider a self-organizing storage application in which a peer, called the data *owner*, replicates its data by storing them at several peers, called data *holders*. The latter entities agree to keep these data for a predefined period of time negotiated with the owner. Their behavior might be evaluated through the adoption of a routine check through which the holder should be periodically prompted to respond to a time-variant challenge as a proof that it holds its promise. Enforcing such a periodic verification of the data holder has implications on the organizational design, performance, and security of the storage protocol, which must fulfill requirements reviewed under the following three sections.

3.1.1. Organization

The self-organizing style of the P2P storage system entails specific features of the verification protocol. The protocol faces multiple requirements regarding the large storage capacity of the system and churn.

- *Scalability:* The verification protocol should scale to large populations of data owners. Verification information should be self-carried by the data verifiers although such data can be made available in the system in a self-organizing manner, within a distributed-hash-table (DHT) for instance. The latter alternative is more robust since the information essential to the protocol realization is reliably stored in the system rather than kept by a single entity.
- *Data redundancy:* The usual technique to achieve data reliability relies on disseminating the data in multiple copies to several peers (based on simple replication or erasure codes). Such data redundancy is initially managed by the owner; however further data rejuvenation may be initiated by the data owner or by other peers from the network. The potential detection of data destruction following their verification may trigger a restoration process of destroyed copies at new peers. The churn out characterizing peers may end in favor of the second option where some peers help in securing the storage of

other peers. The latter peers should also help in distributing the verification information required to periodically check the presence of data at the new holders.

- *Delegating data verification:* Self-organization addresses highly dynamic environments like P2P networks in which peers frequently join and leave the system: this assumption implies the need for the owner to delegate data storage evaluation verifiers, which should act as third parties ensuring a periodic evaluation of holders after his leave (see Figure 11). The need for scalability also pleads for distributing this verification function, in particular to distribute verification costs among several entities. Last but not least, ensuring fault tolerance means preventing the system from presenting any single point of failure: to this end, data verification should be distributed to multiple peers as much as possible; the data should also be replicated to ensure its availability, which can only be maintained at a given level if it is possible to detect storage faults.

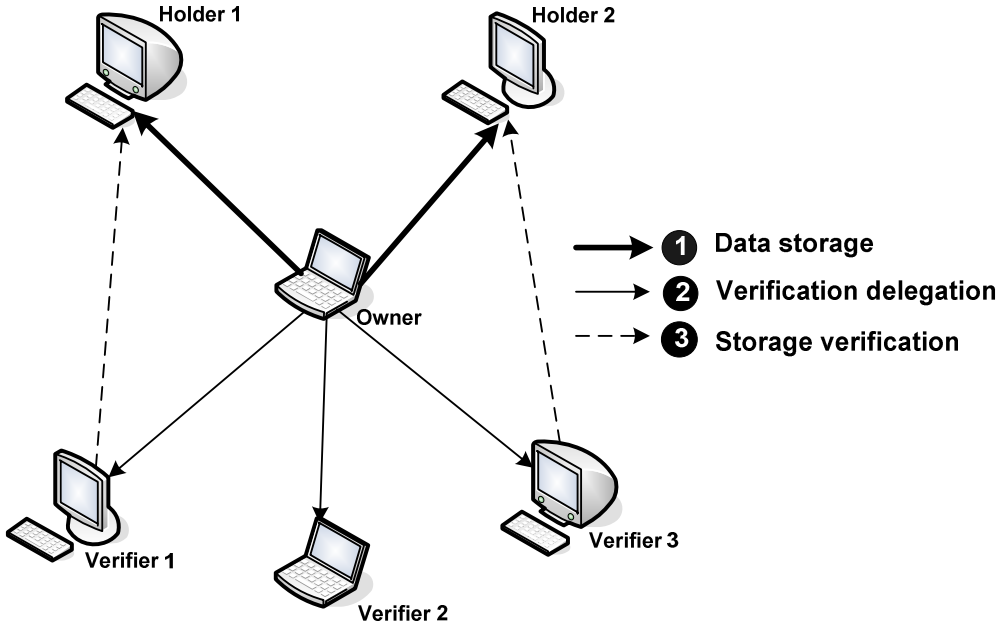


Figure 11 Verification protocol in 3 phases: (1) the owner requests storage from 2 holders, (2) owner delegates the verification of its data to 3 verifiers, and (3) the verifiers periodically check the behavior of holders.

3.1.2. Efficiency

The costs of verifying the proper storage of some data should be considered for the two parties that take part in the verification process, namely the verifier and the holder.

- *Storage overhead:* The verifier must store a meta-information that makes it possible to generate a time-variant challenge based on the proof of knowledge protocol mentioned above for the verification of the stored data. The size of this meta-information must be reduced as much as possible even though the data being verified is very large. The effectiveness of storage at the holder must also be optimized. The holder should store the minimum extra information in addition to the data themselves.
- *Communication overhead:* The size of challenge response messages must be optimized. Still, the fact that the proof of knowledge has to be significantly smaller than the data whose knowledge is proven should not significantly reduce the security of the proof.

- *CPU usage:* Response generation and response verification respectively at the holder and at the verifier should not be computationally expensive.

The use of a specific terminology (remote integrity checking [Deswarte et al. 2004], demonstration of data possession [Filho and Barreto 2006], proofs of data possession [Ateniese et al. 2007], or proofs of retrievability [Juels and Kaliski 2007]) has emphasized how the storage and communication overhead requirements differ between verification primitives for secure remote storage and classical proof of knowledge protocols.

3.1.3. Threat model

The verification mechanism must address the following potential attacks which the data storage protocol is exposed to:

- *Detection of data destruction:* The destruction of data stored at a holder must be detected as soon as possible. Destruction may be due to generic data corruption or to a faulty or dishonest holder.
- *Collusion-resistance:* Collusion attacks aim at taking unfair advantage of the storage application. Replica holders in particular may collude so that only one of them stores data, thereby defeating the purpose of replication to their sole profit.
- *Denial-of-Service (DoS) prevention:* DoS attacks aim at disrupting the storage application. Possible DoS attacks are:
    - **Flooding attack:** the holder may be flooded by verification requests from dishonest verifiers, or from attackers that have not been delegated by the owner. The verifier may be as well subject to the same attack.
    - **Replay attack:** a valid challenge or response message is maliciously or fraudulently repeated or delayed so as to disrupt the verification.
- *Man-in-the-middle attack prevention:* The attacker may pretend to be storing data to an owner without using any local disk space. The attacker simply steps between the owner and the actual holder and passes challenge-response messages back and forth, leaving the owner to believe the attacker is storing its data, when in fact another peer, the actual holder, stores owner's data. The replication may again be disrupted with this attack: since the owner may run the risk of storing the data in two replicas at the same holder.

The main security problem is the detection of data destruction combined with the risk of collusion between holders. We propose security primitives to handle this problem based on proofs of data possession and personalization mechanisms ([Caronni and Waldvogel 2003]).

Considering all these security and performance goals, we propose three different protocols: a probabilistic verification protocol and two deterministic ones. We consider for the three protocols an owner that wishes to store data generates individual replicas for holders and that delegates the verification to different peers (another method of deterministic verification based on the Diffie-Hellman problem is proposed in Appendix A). Prevention means against DoS attacks are presented in a refined version after basic verification protocols.

3.2. Probabilistic verification protocol

This section introduces a verification protocol that allows a peer to probabilistically verify whether a data holder still possesses the data he agreed to store for the originator using a secret key and based on challenge-response messages,. This protocol does not require the verifier to

keep data or pre-computed challenges, nor the holder to perform time-consuming computations to answer challenges.

3.2.1. Protocol description

The verification protocol is a three-party scheme in which the owner stores its data at the holder, then delegates the verification of the very data to a verifier that will periodically check whether the holder is still storing data.

The protocol requires two keys: the first one is used to encrypt data (encryption key), and the second one to check data possession (verification key).

The protocol manipulates a keyed function, denoted f_{KO}, where K_O is verification secret key only known from the owner, i.e., only the owner can compute for a given x, $f_{KO}(x)$. For instance, f_{KO} may be a symmetric encryption function or a keyed one-way hash function or even a one-way hash function with the message being concatenated with the key.

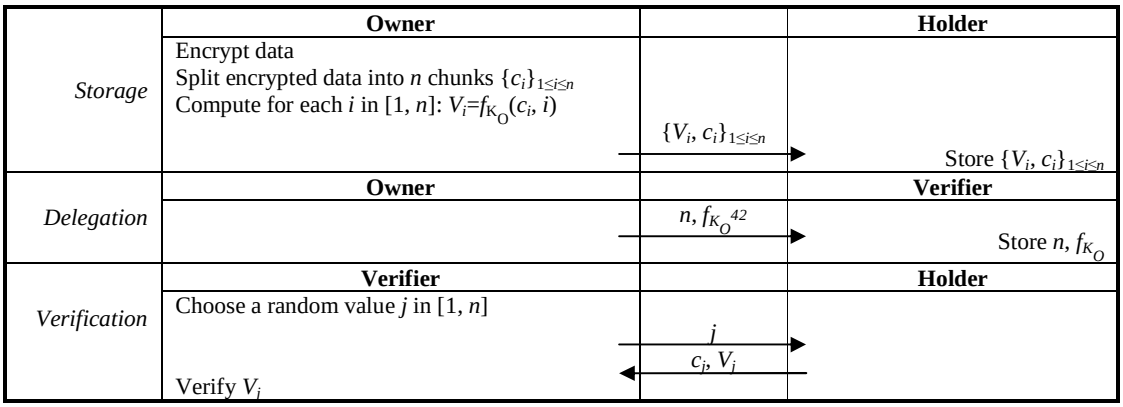


Figure 12 Probabilistic verification protocol

The protocol comprises the following three phases (see Figure 12):

- *Storage:* The owner first personalizes the data that will be stored at the holder by encrypting it concatenated to the identity of the holder using the encryption key. Personalization prevents the collusion between holders. The owner splits the encrypted data into n chunks $\{c_i\}_{1\le i\le n}$, and then it computes for each chunk (along with its index) its image with f_{KO}. The result is the set $\{V_i=f_{KO}(c_i, i)\}_{1\le i\le n}$ (the size of one chunk must be higher than the size of one generated verification tag to be cost-productive). Finally, the owner sends data chunks $\{c_i\}_{1\le i\le n}$ and the verifications tags $\{V_i\}_{1\le i\le n}$ for storage to the holder.
- *Delegation:* The owner appoints a verifier for checking data storage at the holder, and informs it of the number of chunks n stored at the holder and the function f_{KO} with its key K_O.
- *Verification:* the verifier randomly chooses a value j in [1, n] and sends it to the holder who responds with the corresponding couple (c_j, V_j). The verifier verifies if this couple is a valid one, using the key K_O.

[42] This message is not sent in clear, but encrypted with the session key shared between the owner and the verifier.

In this protocol, the holder proves that it is keeping a data segment for the owner, and provides an evidence that attests of its origin. The verification process requires computational resources consumed at the verifier, and additional storage space together with some computation at the prover. The extra storage at the holder is the price to pay for a verification process without data, or pre-computed information stored at the verifier. Keys do not need to be stored by the verifier if they can be generated based on a passphrase for instance. Such an approach, or the use of a token, would be required for a storage application in which the owner (or the verifier) may have completely crashed, thereby losing any secret stored there.

3.2.2. Security evaluation

In the protocol, if the result of the verification is positive, then the verifier is "probabilistically" assured that the holder is still storing the data. In reality, the verifier only checks that the holder is keeping the chunk c_j. Since j is chosen randomly[43], the holder has to keep all chunks and their images to answer correctly to challenges.

This section investigates how the probabilistic nature of the protocol makes it possible to enforce some security. We are making the following assumptions:

- The verifier is not in collusion with the holder.
- The verifier's random selection of indexes is uniform, i.e., for n chunks, the probability to pick any chunk is $1/n$.
- Index selections are independent events.
- The holder removes a fraction d of chunks from its storage; we term d the misbehavior rate of the holder.
- The verifier performs on average c challenges; $1 \leq c \leq n$.

The probability that the holder answers correctly to verifier's challenges all the time is described as:

$$p = (1 - d)^c$$

The probability that the verifier detects holder's misbehavior is given by $p_{detection}$:

$$p_{detection} = 1 - p$$

For a given probability of detection of misbehavior, it is possible to probabilistically determine the average number of challenges that the verifier should perform to attain this probability of detection. The number of challenges c can be derived as follows:

$$c = \lceil log_{1-d}(1-p_{detection}) \rceil$$

The required number of challenges to acquire a given probability of detection is most of the time not equal to 1. The verifier should therefore challenge the holder multiple times.

Figure 13 shows how the number of challenges c increases with the probability of detection. An appropriate value for c can be chosen based on the misbehavior rate of the holder that can be estimated thanks reputation computed from recent interactions. If the holder has bad reputation, fewer challenges are needed to likely detect holder's misbehavior. The opposite will be observed if the holder has good reputation; however, the owner will likely store more critical data at the holder and the verifier will challenge it more frequently thus compensating the higher number

[43] The selection of an index j, for a challenge, has no impact on its probability to be picked another time for another challenge.

of challenges required to detect a misbehaving peer (for a misbehavior rate of 0.1, the required number of challenges exceeds 50 to achieve a high probability of detection).

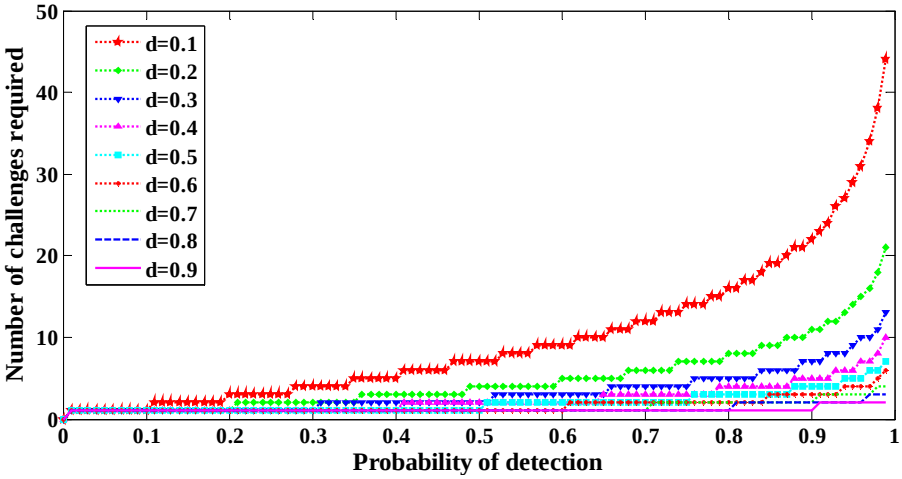


Figure 13 Number of challenges required to achieve a probability of detection of holder's misbehavior

In the proposed protocol, false negatives (i.e., verification is positive when the holder removes a given number of chunks) are possible, yet their occurrence can be reduced by increasing the number of challenges *c*. False positives (i.e., verification is negative when the holder has got all chunks) may occur and are associated with communication losses. The latter issue can be thwarted by usual measures like the retransmission of packets after a timeout.

While the use of a probabilistic approach might be seen as a weaker scheme, it should be noted for instance that multimedia data, like digital pictures or videos may support more degradation for some chunks such as image details, than for hunks with high-level description: these data, which promise to be one major area where in-the-field storage application will be required, may therefore tolerate less stringent protection mechanisms in exchange for more performance, as it will be demonstrated in the following section.

It is not a requirement for the verifier to be trusted by the owner, only in the case where function f_{K_O} is a symmetric function. In this case, if the verifier is not trustworthy, it may collude with the holder by divulging the verification key K_O and thus the holder may answer correctly to all verifiers appointed to it without having to keep data stored. However, if this function f_{K_O} is asymmetric (signature), the protocol does not require from the verifier to be trusted by the owner, it just requires from it to possess the public key of the function. The distribution of the verification task to several volunteer peers mitigates the misbehavior of the verifiers.

3.2.3. Performance evaluation

The following performance analysis shows that the probabilistic approach of the proposed protocol allows data possession verification to be less expensive for devices with limited resources. This approach indeed permits to trade some security, which can be measured probabilistically, in exchange of better performance.

The suggested verification protocol consists of three phases, only two of them are considered indispensable since the owner can be the verifier of the data (delegation phase is optional). The first phase corresponds to plain data storage. The verification phase comprises authentication

messages, then challenge-response messages. Thus, we only consider the verification part of the protocol for our analysis.

The verifier is not required to keep the whole data for verification, it only has to keep the verification key K_O. In counterpart, the holder keeps not only the data but some additional information in the form of verification tags. The challenge-response messages sent mainly correspond to c data chunks and their verification tags. Table 1 summarizes the discussed verification costs.

The probability of fault detection success increases with the number of chunks c verified per challenge sent by the verifier. However, the communication costs linearly increase with the number of chunks challenged because the holder has to send all requested chunks with their verification tags.

Table 1 Summary of resource usage consumed by the probabilistic verification protocol (variable *n* and *m* respectively correspond to data size and the number of chunks)

	Storage overhead	Computation complexity	Communication overhead
At holder	$O(n)$	O(1)	(upstream) $O(n/m)$
At verifier	$O(1)$	$O(n/m)$	(upstream) $O(1)$

3.2.4. Countering additional attacks

The described verification protocol permits to detect selfish holders that destroy the data they have promised to store. However, the protocol alone is not able to defend against other forms of misbehavior, such as denial of service attacks, man-in-the-middle attacks, or colluding replica holders.

A flooding attack can be launched by the verifier, by sending a large number of challenge messages to a victim data holder in order to slow it until it is unusable or crashes. Although this type of attack is unlikely to happen since the verifier performs computational operations for every challenge, it is possible to limit the number of challenges by imposing a quota on the frequency of challenges. This solution is proposed in [Lillibridge et al. 2003]. Moreover, it is possible to force the verifier to pay fees for every challenge it requests, for instance using a micropayment scheme. An alternative approach is to reciprocate in storing data, thereby performing symmetric verifications between the two peers, like in [Caronni and Waldvogel 2003].

A holder can pretend to be storing data while in fact proxying in front of another data holder. Then, the attacker simply passes data back and forth between the originator and the holder, making the data originator believe that it is the data holder, and the data holder that it is the originator of the data. This problem can be addressed by having the index j for each challenge randomly chosen by both parties as suggested in the random-read protocol presented in [Caronni and Waldvogel 2003] in which both parties randomly choose the offset of the block to be checked.

When using replication mechanisms to support the availability of data, replica holders may collude so that only one of them stores data, thereby defeating the purpose of replication to their sole profit. One way to counter this attack is to produce personalized replicas for each holder, as described in [Caronni and Waldvogel 2003], by using an encryption key (used to encrypt the data) derived from the identity of the holder.

3.3. Restricted deterministic verification protocol

This section presents a verification protocol that allows checking the presence of the whole data at a remote peer without excessive overhead though the protocol implies that the data checking supports only a limited number of verification operations per verifier. The proposed protocol relies on the uniqueness of the solution of the interpolation polynomial problem described in the following section. The subsequent sections present the protocol operation and its evaluation with respect to security and performance considerations.

3.3.1. Lagrange interpolation polynomial

The proposed protocol relies on the Lagrange interpolation polynomial. This section describes the polynomial interpolation problem and the Lagrange solution.

For a given set of (n+1) data points $\{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\}$ where all x_i are different, there is only one interpolation polynomial $P(.)$ of degree at most n that satisfies $P(x_i)=y_i$ for each $i \in [0, n]$.

The interpolation polynomial is computed using its Lagrange form which is the linear combination of Lagrange basis polynomials:

$$P(x) = \sum_{i=0}^{n} y_i \times l_i(x)$$

having for each $i \in [0, n]$:

$$l_i(x) = \prod_{j=0}^{n} \frac{x - x_j}{x_i - x_j}$$

3.3.2. Protocol description

The verification protocol involves the owner of some data, a potential holder of the data, and a verifier that is assigned to periodically check data presence at the holder. The protocol consists of the following three phases (see Figure 14):

- *Storage:* Data are first personalized to the holder by the owner. The data are then split into n chunks $\{c_i\}_{1 \le i \le n}$, such that all chunks have the same size k (the last chunk can be padded). Each chunk is mapped to a number of $\mathbb{Z}_p$ (with a bijection, i.e., one-to-one correspondence) where the size of p is also k. Chunks are sent to the holder for storage.
- *Delegation:* The owner generates a random number r. It derives m polynomials of degree n: $\{P_j(.)\}_{1 \le j \le m}$ (the random number r is chosen such that the generated polynomials are all of degree n). The polynomials are computed using Lagrange interpolation. For each j in [1, m], the polynomial $P_j(.)$ is the solution of the interpolation problem given the set of n+1 points $\{(0, r_i=Hash^j(r))\} \cup \{(i, c_i)\}_{1 \le i \le n}$ where $Hash$ is a pseudo-random one-way function and $Hash^j$ means that the function $Hash$ is executed j times. Finally, the owner computes $P_j(n+1)$ for each j in [1, m], and sends r and $\{P_j(n+1)\}_{1 \le j \le m}$ to the verifier. The number m corresponds to the maximum number of verification operations the verifier is able to perform with the verification metadata received from the owner.
- *Verification:* The verifier challenges the holder by sending a random number r_i. The random numbers $\{r_i=Hash^j(r)\}_{1 \le j \le m}$ are computed and periodically sent in the reverse order (i.e., starting from r_m to r_1). The holder derives the polynomial $P_j(.)$ using the points obtained from the received random number and all the stored data chunks. The holder may apply the Lagrange form to the interpolation polynomial. Finally, the holder

computes the point $P_j(n+1)$ and sends the result to the verifier. The verifier compares the holder's response to the stored value.

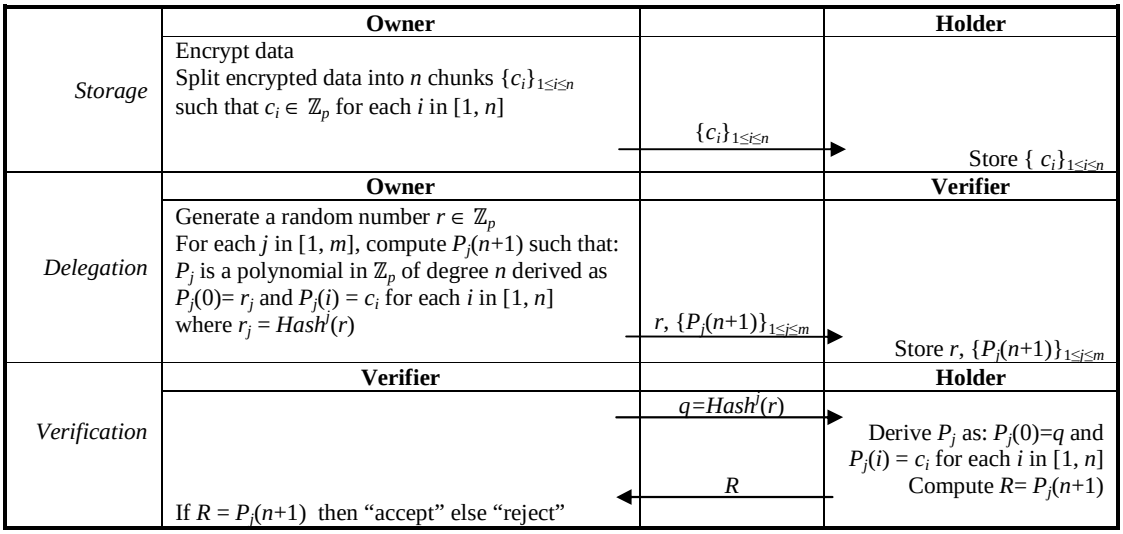


Figure 14 Restricted deterministic verification protocol

3.3.3. Security evaluation

The proposed verification protocol relies on the Lagrange form of the interpolation polynomial that shows the uniqueness of the polynomial. Based on the data set of $(n+1)$ points $\{(0, r_i=Hash^j(r))\}\cup\{(i, c_i)\}_{1\le i\le n}$, the interpolation polynomial is constructed. Since all points are distinct, there is only one polynomial as solution. To construct the polynomial, the holder must use all data points that include all the data chunks, which means that the verification of the stored data is deterministic. The generation of the response comprises the computation of a new data point from the constructed polynomial. Since this point is derived from the interpolation polynomial, it is then unique. The point is compared with a point computed in advance stored as a security metadata at the verifier.

The holder should store all data chunks to be able to compute the unique response to the verifier's challenge. The holder may store different forms based on composed versions of data chunks in order to accelerate the computation of the interpolated polynomial. These forms characterize a polynomial with order n, so these forms have the same number and size as the original data chunks. It is then still possible to recover data chunks from these forms.

3.3.4. Performance evaluation

The proposed protocol requires the verifier to keep m data points of the same size as the data chunks. Increasing m allows expanding the verification duration of the data stored, but at the same time it raises the storage overhead at the verifier. This increase can be compensated by reducing the size of data chunks (small k). However, very small-sized points may generate a large number of false positives which means that the holder has more chances to guess the response of the verifier's challenge without actually storing data chunks. The value k constitutes then a security parameter of the protocol that should be cared of.

Tuning *m* makes it possible to achieve a tradeoff between the availability of the protocol (number of verification reiteration) and the storage overhead at the verifier (refer to Table 2). The verifier is only able to perform *m* verifications of the data storage at the holder. A high value of the number *m* permits the verifier to check the data presence more often; even if the verifier must store a large set of data points. Moreover, if some of the stored pre-computed challenges are compromised (e.g., the holder may compute the reverse of a hash value), the verifier is still able to check the data with the remaining uncompromised values.

The owner should refresh the verifier metadata from time to time if it wants the latter to continue verifying the presence of the data at the holder. However, the generation of new metadata information for verification requires manipulating the data themselves; while this may be valid if the owner simply performed a backup (although he has to store the data personalization applied to a given holder), in more general storage cases, the owner has to retrieve a copy of the data before being able to compute new challenges. Since this obligation produces quite expensive communication costs, the number of pre-computed challenges *m* should be chosen as high as possible. The foremost burden of a high value of *m* is placed on the verifier with an increased requirement regarding the storage overhead. The storage of the verification metadata can however be rendered space efficient by relying on Bloom filters [Bloom 1970] for instance. Another alternative for alleviating the storage overhead at the verifier is to store hash values of the verification metadata instead. Storage costs then become quite modest (typically 128 to 160 bits). As discussed in [Ateniese et al. 2008], the storage overhead is less problematic in practice. For example, less than 6MB of metadata make it possible for the remote data to be checked every 15mn for 10 years.

The size of exchanged messages between the holder and the verifier are unrelated to the size of the data (they rather depend on the data chunk size). Additionally, the holder does not need to store a large extra-storage other than the data in chunks (some small information is needed e.g., the order of chunks).

Table 2 Summary of resource usage consumed by the restricted deterministic verification protocol (variable *n* and *m* respectively correspond to the number of data chunks and the number of pre-computed challenges)

	Storage overhead	Computation complexity	Communication overhead
At holder	$O(n)$	$O(n)$	(upstream) $O(1)$
At verifier	$O(m)$	$O(1)$	(upstream) $O(1)$

3.4. Deterministic verification protocol

The requirement of a cheap verification in terms of storage simply forbids the use of plain message integrity codes as a protection measure if verifier peers are to submit an unlimited number of challenges, since time-variant challenges based on such primitives cannot be constructed without the owner or without the verifier storing the entire data. This section presents a secure and self-organizing verification protocol exhibiting a low resource overhead. This protocol was designed with scalability as an essential objective: it enables generating an unlimited number of verification challenges from the same small-sized security metadata.

The security of the storage scheme relies on the hardness of specific problems in elliptic curve cryptography. The protocol is also especially original with respect to scalability: it both enables data replication while preventing peer collusion, and delegation of data storage verification to third parties.

The remainder of this section details the verification protocol incrementally: essential notions in elliptic curve cryptography and used hard problems are first introduced; two versions of the security protocol are then described.

3.4.1. Security background

The deterministic verification protocol relies on elliptic curve cryptography ([Koblitz 1987], [Miller 1986]). The security of the protocol is based on two different hard problems. First, given some required conditions, it is hard to find the order of an elliptic curve. Furthermore, one of the most common problems in elliptic curve cryptography is the Elliptic Curve discrete logarithm problem denoted by ECDLP.

Thanks to the hardness of these two problems, the deterministic verification protocol ensures that the holder must use the whole data to compute the response for each challenge. In this section, we formalize these two problems in order to further describe the security primitives that rely on them.

Elliptic Curves over $\mathbb{Z}_n$

Let n be an odd composite square free integer and let a, b be two integers in $\mathbb{Z}_n$ such that $\gcd(4a^3 + 27b^2, n) = 1$ ("gcd" means greatest common divisor).

An elliptic curve $E_n(a, b)$ over the ring $\mathbb{Z}_n$ is the set of the points $(x, y) \in \mathbb{Z}_n \times \mathbb{Z}_n$ satisfying the equation: $y^2 = x^3 + ax + b$, together with the point at infinity denoted O_n.

Solving the order of elliptic curves

The order of an elliptic curve over the ring $\mathbb{Z}_n$ where $n=pq$ is defined in [Koyama et al. 1991] as $N_n = \text{lcm}(\#E_p(a, b), \#E_q(a, b))$ ("lcm" for least common multiple, "#" means order of). N_n is the order of the curve, i.e., for any $P \in E_n(a, b)$ and any integer k, $(k \times N_n + 1)P = P$.

If ($a = 0$ and $p \equiv q \equiv 2 \bmod 3$) or ($b = 0$ and $p \equiv q \equiv 3 \bmod 4$), the order of $E_n(a, b)$ is equal to $N_n=\text{lcm}(p+1, q+1)$. We will consider for the remainder of the paper the case where $a = 0$ and $p \equiv q \equiv 2 \bmod 3$. As proven in [Koyama et al. 1991], given $N_n = \text{lcm}(\#E_p(a, b), \#E_q(a, b)) = \text{lcm}(p + 1, q + 1)$, solving N_n is computationally equivalent to factoring the composite number n.

The elliptic curve discrete logarithm problem

Consider K a finite field and $E(K)$ an elliptic curve defined over K. ECDLP in K is defined as given two elements P and $Q \in K$, find an integer r, such that $Q = rP$ whenever such an integer exists.

3.4.2. Protocol description: data-based version

The data, stored in the system, is uniquely mapped into a number $d \in \mathbb{N}$ in some publicly known way (for example, conversion from binary representation into decimal representation). In our context, the terms data file or data and the number d are often used interchangeably. The verification protocol consists of four phases (see Figure 15): Setup, Storage, Delegation, and Verification. The owner communicates the data to the holder at the storage phase and the meta-information to the verifier at the delegation phase. At the verification phase, the verifier checks the holder's possession of data remotely through an interactive process. This process may be executed an unlimited number of times.

- *Setup:* The phase is performed by the owner. From a chosen security parameter k ($k > 512$ bits), the owner generates two large primes p and q of size k both congruent to 2 modulo 3, and computes their product $n = pq$. It then considers an elliptic curve over the ring $\mathbb{Z}_n$ denoted by $E_n(0, b)$ where b is an integer such that $\gcd(b, n)=1$, to compute a generator P of $E_n(0, b)$. The order of $E_n(0, b)$ is $N_n = \text{lcm}(p+1, q+1)$. The parameters b, P, and n are published and the order N_n is kept secret by the owner.

- *Storage:* The owner stores its data at one or several holders, in a personalized form for each holder. To this end, the owner encrypts data using a keyed function f_s (s is a secret key known to the owner only) that takes in input the data and the identity of the holder and returns an identity-based encrypted version of data d' (we assume that peers are uniquely identified in the system).
- *Delegation:* The owner generates meta-information to be used by the verifier for checking the data possession at one holder. The meta-information is a reduced-size digest of the data stored at the holder and is computed as $T = (d' \bmod N_n)P \in E_n(0, b)$. This meta-information is sent for storage to the verifier.
- *Verification:* The verifier checks the presence of data at the holder. It generates a random number r and computes the point $Q = rP \in E_n(0, b)$ which is sent to the holder as a challenge. Upon its reception, the holder computes $R = d'Q \in E_n(0, b)$ with the data d' it is storing. The proof R is sent to the verifier. With this proof, the verifier checks if R is equal to rT, and decides if the holder's proof is accepted or rejected.

Setup	**Owner**		
	Generate two primes p and q of size k: $p, q \equiv 2 \bmod 3$ Compute $n = pq$ Compute $N_n = \text{lcm}(p+1, q+1)$ Generate random integer $b < n$, $\gcd(b, n)=1$ compute P a generator of $E_n(0, b)$ Public = (n, b, P), Secret = N_n		
Storage	**Owner**		**Holder**
	Compute $d'=f_s(d)$ send d'	d' →	Store d'
Delegation	**Owner**		**Verifier**
	Compute $T = (d' \bmod N_n)P$ send T	T →	Store T
Verification	**Verifier**		**Holder**
	Generate a random number r Compute $Q = rP$ Send Q	Q →	Compute $R = d'Q$ Send R
	If $R = rT$ then "accept" else "reject"	← R	

Figure 15 Deterministic verification protocol: data-based version

With the presented security primitives, the verifier keeps an extra-information (T) needed for the verification that is twice the size of n (< 2k) which is smaller than the size of the stored data (about 2Kb compared with 100Mb or 1Gb of data). For verification, the verifier has to compute two point multiplications with a small random number, in contrast with the holder that has to compute a point multiplication with the whole data.

3.4.3. Protocol description: chunk-based version

This section introduces an improved version of the protocol described above whereby the computation complexity at the holder is reduced.

In this version, the data are split into m chunks, denoted $\{d'_i\}_{1 \le i \le m}$, and the verifier stores the corresponding elliptic curve points $\{T_i = d'_i P\}_{1 \le i \le m}$. We assume that the size of each data chunk

is much larger than $4k$ where k is the security parameter that specifies the size of p and q, because the verifier must keep less information than the full data.

The owner proceeds in the setup phase like in the previous version. It personalizes the data, then splits the personalized data into m chunks of the same size (the last chunk is padded with zeroes): $\{d'_i\}_{1\le i\le m}$. At the delegation phase, the owner generates the curve points $\{T_i = d'_iP\}_{1\le i\le m}$ sent to the verifier. During the verification phase, the verifier generates a random number r and a random seed c (size of $c > 128$ bits). Then, it sends $Q=rP$ and the seed c to the holder. Upon reception, the holder generates m random numbers $\{c_i\}_{1\le i\le m}$ from the seed c (it is possible to generate the random numbers as $c_i=c^i$ for each i, or using a random number generator function). Then, it computes the point $R = \sum_{1\le i\le m} c_i d'_i Q$ that is sent to the verifier. To decide whether a holder's proof is accepted or rejected, the verifier generates the same m random numbers $\{c_i\}_{1\le i\le m}$ from the seed c and checks if R is equal to $r(\sum_{1\le i\le m} c_i T_i)$. The protocol is summarized in Figure 16.

Storage	**Owner**		**Holder**
	Compute $d'=f_s(d)$ Split d' in m chunks: $\{d'_i\}_{1\le i\le m}$ send $\{d'_i\}_{1\le i\le m}$	$\{d'_i\}_{1\le i\le m}$ →	Store $\{d'_i\}_{1\le i\le m}$
Delegation	**Owner**		**Verifier**
	Compute for each i in $[1, m]$: $T_i=(d'_i \bmod N_n)P$ send $\{T_i\}_{1\le i\le m}$	$\{T_i\}_{1\le i\le m}$ →	Store $\{T_i\}_{1\le i\le m}$
Verification	**Verifier**		**Holder**
	Generate a random number r and seed c Compute $Q = rP$ Send c, Q Generate $\{c_i\}_{1\le i\le m}$ from seed c	c, Q →	Generate $\{c_i\}_{1\le i\le m}$ from seed c Compute $R = \sum_{1\le i\le m} c_i d'_i Q$ Send R
	If $R = r(\sum_{1\le i\le m} c_i T_i)$ then "accept" else "reject"	← R	

Figure 16 Deterministic verification protocol: chunk-based version

Compared with the data-based version of the deterministic protocol where the data is considered as a whole, this new version makes the holder compute m point multiplications of the same elliptic curve point where the size of the scalar is the size of the data chunk instead of the full data. Also, the verifier has to keep m points instead of one point in the previous version. The number of chunks m is the ratio of tradeoff between the storage required at the verifier and the computation consumed at the holder (the case $m = 1$ corresponds to the previous version of the protocol).

3.4.4. Security analysis

This section analyzes the completeness and the soundness of the ECC based deterministic protocol (more specifically the chunk-based version since it is the generalized case) that are the two essential properties of a proof of knowledge protocol [Menezes et al. 1996]: a protocol is complete if, given an honest claimant and an honest verifier, the protocol succeeds with overwhelming probability, i.e., the verifier accepts the claimant's proof; a protocol is sound if, given a dishonest claimant, the protocol fails, i.e. the claimant's proof is rejected by the verifier, except with a small probability.

Theorem 1- The proposed protocol is complete: if the verifier and the holder correctly follow the proposed protocol, the verifier always accepts the proof as valid.

Proof: Thanks to the commutative property of point multiplication in an elliptic curve, we have for each i in [1, m]: $d'_i rP = rd'_i P$. Thus, the equation: $\sum_{1\le i\le m} c_i d'_i rP = r(\sum_{1\le i\le m} c_i d'_i P)$. □

Theorem 2- The proposed protocol is sound: if the claimant does not store the data, then the verifier will not accept the proof as valid.

Proof: If the holder does not store the data chunks $\{d'_i\}_{1\le i\le m}$, it may try first to collude with other holders storing the same data. However, this option is not feasible since data stored at each holder is securely personalized during the storage phase. Since f_s is a one-way function and the key s is secret, no peer except the owner can retrieve the original data d from d'. The other way to generate a correct response without storing the data relies on only storing $\{d'_i P\}_{1\le i\le m}$ (which is much smaller than the full data size) and retrieving r from the challenge rP in order to compute the correct response. Finding r is hard based on ECDLP. The last option for the holder to cheat is to keep $\{d'_i \bmod N_n\}_{1\le i\le m}$ instead of d' (whose size is very large). The holder cannot compute N_n based on the hardness of solving the order of $E_n(0, b)$. Thus, if the response is correct then the holder keeps the data correctly. □

3.4.5. Performance analysis

In the proposed protocol, challenge-response messages mainly each consist of an elliptic curve point in $\mathbb{Z}_n^2$. Message size is thus a function of the security factor k (size of $n \approx 2k$). Reducing communication overhead then means decreasing the security parameter.

The verification protocol requires the verifier to store a set of elliptic curve points that allows producing on demand challenges for the verification. Finally, the creation of proof and its verification rely on point multiplication operations.

The number of data chunks m can be used to fine tune the ratio between the storage required at the verifier and the computation expected from the holder: when increasing m, the verifier is required to keep more information for the verification task, but at the same time the holder is required to perform one point multiplication operation using much smaller scalars.

Table 3 Summary of resource usage of the deterministic verification protocol (variable *n* and *m* respectively correspond to data size and the number of chunks)

	Storage overhead	Computation complexity	Communication overhead
At holder	$O(n)$	$O(n/m)$	(upstream) $O(1)$
At verifier	$O(m)$	$O(1)$	(upstream) $O(1)$

3.4.6. Protocol refinement

The above verification protocols allow efficiently detecting data destruction by misbehaving holders. However, both are still weak against some security threats described in Section 3.1.3. This section refines the security protocol to address these additional attacks. Every peer in the framework is assumed to be uniquely identified by an identifier denoted by ID_P.

External DoS attacks. At each phase of the protocol, messages are authenticated with common signature algorithms such as RSA. Therefore each peer possesses a pair of public and private keys $\{PK_P, SK_P\}$.This authentication inherently prevents external Denial of Service (DoS) attacks whereby intruders generate some flooding attacks against holders. Only authorized verifiers are allowed to run the verification phase. In order to provide this security restriction, during the delegation phase, the owner provides each verifier an enabling credential for the verification phase. Therefore, verifiers generate a signature for each message at the verification phase and send this signature and their credentials together with the challenge

message. Since the stored data are assumed to be dense, the cost of storage is assumed to be much more expensive than the cost of verifying or generating a digital signature. Moreover, thanks to this technique message integrity is also provided.

Internal DoS attacks. In addition to external DoS attacks, some authorized verifiers might also generate some flooding attacks against holders. In this particular case, authentication is not a direct solution since verifiers are authorized to participate to the communication. We thus first propose to limit the number of verifiers that can challenge each holder. This number can be predefined or negotiated in the storage phase between the owner and the holder based on the capacity of the latter. We also propose to define a threshold value for requests originating from a verifier. Hence, the holder keeps a quota counter for each authorized verifier that is incremented at each new challenge. If this counter exceeds the given threshold value during a time interval, the verifier is not allowed to challenge the holder and the challenge message is automatically dropped.

Replay attacks. Replay attacks whereby valid challenge messages are maliciously repeated or delayed so as to disrupt the verification phase are also taken into consideration. To cope with this, the verifier only needs to send a newly generated nonce within the challenge message. Thanks to this well-known technique, the holder will be able to automatically detect replay attacks.

Man-in-the-middle attack. The holder may not be the actual holder of the data. It may play the owner pretending to store the data but in fact be performing a man-in-the-middle attack to step between the owner and the actual data holder. To prevent this type of attack, the actual holder H may, instead of sending the response *R* as the answer to the verifier's challenge, send the following message: $Hash(R \parallel ID_H)$; where *Hash* is a pseudo-random function and ID_H is the identifier of the holder ("||" means concatenation with elliptic curve points mapped to numbers). The attacker is not able to recreate such a response putting its own identity. And finally, the verifier is able to check the validity of the response.

3.5. Existing verification protocols

The security of distributed storage applications has been increasingly addressed in recent years, which has resulted in various approaches to the design of storage verification primitives.

The literature distinguishes two main categories of verification schemes: probabilistic ones that rely on the random checking of portions of stored data, and deterministic ones that check the conservation of a remote data in a single, although potentially more expensive operation. Additionally, some schemes may authorize only a bounded number of verification operations conducted over the remote storage (although the majority of schemes are designed to overcome this limitation).

Memory checking. A potential premise of probabilistic verification schemes originates from memory checking protocols. A memory checker aims at detecting any error in the behavior of an unreliable data structure while performing the user's operations. The checker steps between the user and the data structure. It receives the input user sequence of "store" and "retrieve" operations over data symbols that are stored at the data structure. The checker checks the correctness of the output sequence from the structure using its reliable memory (noninvasive checker) or the data structure (invasive checker) so that any error in the output operation will be detected by the checker with high probability. In [Blum et al. 1994], the checker stores hash values of the user data symbols at its reliable memory. Whenever the user requests to store or retrieve a symbol, the checker computes the hash of the response of the data structure and compares it with the hash value stored, and it updates the stored hash value if the user requested

to store a symbol. The job of the memory checker is to recover and to check responses originating from an unreliable memory, not to check the correctness of the whole stored data. With the checker, it is possible to detect corruption of one symbol (usually one bit) per user operation.

Authenticator. The work of [Naor and Rothblum 2005] better comprehends the remote data possession problem. It extends the memory checker model by making the verifier checks the consistency of the entire document in encoded version in order to detect if the document has been corrupted beyond recovery. The authenticator encodes a large document that will be stored at the unreliable memory and constructs a small fingerprint that will be stored at the reliable memory. Using the fingerprint, the authenticator verifies whether from the encoding it is possible to recover the document without actually decoding it. The authors of [Naor and Rothblum 2005] propose a construction of the authenticator where there is a public encoding of the document consisting of index tags of this form: $t_i=f_{seed}(i \; o \; y_i)$ for each encoded value bit y_i having f_{seed} a pseudorandom function with *seed* taken as secret encoding. The authenticator is repeatedly used to verify for a selection of random indices if the tags correspond to the encoding values. The detection of document corruption is then probabilistic but improved with the encoding process of the document. Moreover, the query complexity is proportional to the number of indices requested.

Provable data possession. The PDP (Provable Data Possession) scheme in [Ateniese et al. 2007] improves the authenticator model by presenting a new form of fingerprints $t_i=(hash(v||i) \cdot g^{y_i})^d \bmod N$, where *hash* is a one-way function, v a secret random number, N an RSA modulus with d being a signature key, and g a generator of the cyclic group of $\mathbb{Z}_N^*$. With such homomorphic verifiable tags, any number of tags chosen randomly can be compressed into just one value by far smaller in size than the entire set, which means that communication complexity is independent of the number of indices requested per verification.

Proof of retrievability. The POR protocol (Proof of Retrievability) in [Juels and Kaliski 2007] explicitly expresses the question of data recovery in the authenticator problem: if the unreliable data passes the verification, the user is able to recover the original data with high probability. The protocol is based on verification of sentinels which are random values independent of the owner's data. These sentinels are disguised among owner's data blocks. The verification is probabilistic with the number of verification operations allowed being limited to the number of sentinels.

Compact proofs of retrievability. [Shacham and Waters 2008] improves the POR protocol by considering compact tags (comparable to PDP) that are associated with each data chunk y_i having the following form: $t_i = \alpha y_i + s_i$ where α and s_i are random numbers. The verifier requests random chunks from the unreliable memory and obtains a compact form of the chunks and their associated tags such that it is able to check the correctness of these tags just using α and the set $\{s_1, s_2, \ldots\}$that are kept secret.

Remote integrity check. Remote Integrity Check of [Chang and Xu 2008] alleviates the issue of data recovery and rather focuses on the repetitive verification of the integrity of the very data. The authors described several schemes some of them being hybrid construction of the existing schemes that fulfill the later requirement. For instance, the unreliable memory may store the data along with a signature of the data based on redactable signature schemes. With these schemes, it is possible to derive the signature of a chunk from the signature of the whole data, thus allowing the unreliable memory to compute the signature of any chunk requested by the verifier.

Data chunk recovery. The majority of the probabilistic verification schemes require the recovery of one or multiple (in plain or compacted form) data chunks. For example, in the solution of [Lillibridge et al. 2003], the owner periodically challenges its holders by requesting a block out of the stored data. The response is checked by comparing it with the valid block stored at the owner's disk space. Another approach using Merkle trees is proposed by Wagner and reported in [Golle at al. 2002]. The data stored at the holder is expanded with a Merkle hash tree on data chunks and the root of the tree is kept by the verifier. It is not required from the verifier to store the data, on the contrary of [Lillibridge et al. 2003]. The verification process checks the possession of one data chunk chosen randomly by the verifier that requests also a full path in the hash tree from the root to this random chunk.

Erasure-correcting codes. Erasure-correcting codes are for great interest for probabilistic verification protocols, since they improve the probability of data recovery in case the probabilistic approach does not detect the destruction of some parts of the stored data. The scheme proposed in [Schwarz and Miller 2006] relies on algebraic signatures. The verifier requests algebraic signatures of data blocks stored at holders, and then compares the parity of these signatures with the signature of the parity blocks stored at holders too. The main drawback of the approach is that if the parity blocks does not match, it is difficult (depends on the number of used parity blocks) and computationally expensive to recognize the faulty holder.

Incremental cryptography. First step toward a solution to the deterministic verification problem comes from incremental cryptographic algorithms that detect changes made to a document using a tag, a small secret stored at a reliable memory that relates to the complete stored document and that is quickly updatable if the user makes modifications. [Bellare et al. 1995] proposes several incremental schemes where the tag is either an XORed sum of randomized document symbols or a leaf in a search tree as a result of message authentication algorithm applied to each symbol. These schemes provide tamper-proof security of the user document in its entirety; although they require recovering the whole data which is not practical for remote data verification because of the high communication overhead.

Deterministic remote integrity check. The first solution described in [Deswarte et al. 2004] allows the checking of the integrity of the remote data, with low storage and communication overhead. It requires pre-computed results of challenges to be stored at the verifier, where a challenge corresponds to the hashing of the data concatenated with a random number. The protocol requires small storage at the verifier, yet they allow only a fixed number of challenges to be performed. Another simple deterministic approach with unlimited number of challenges is proposed in [Caronni and Waldvogel 2003] where the verifier like the holder is storing the data. In this approach, the holder has to send the MAC of data as the response to the challenge message. The verifier sends a fresh nonce (a unique and randomly chosen value) as the key for the message authentication code: this is to prevent the holder peer from storing only the result of the hashing of the data.

Storage enforcing commitment. The SEC (Storage Enforcing Commitment) scheme in [Golle at al. 2002] aims at allowing the verifier to check whether the data holder is storing the data with storage overhead and communication complexity that are independent of the length of the data. Their deterministic verification approach uses the following tags that are kept at the holder along with the data: $PK=(g^x, g^{x^2}, g^{x^3}, \ldots, g^{x^n})$ where PK is the public key (stored at the holder) and x is the secret key (stored at the verifier). The tags are independent of the stored data, but their number is equal to two times the number of data chunks. The verifier chooses a random

value that will be used to shift the indexes of tags to be associated with the data chunks when constructing the response by the holder.

Table 4 A comparison of existing verification protocols (variable *n* and *m* respectively correspond to data size and the number of chunks)

	Detection	Delegation	Efficiency		
			Storage at verifier	CPU at holder	Communication overhead
[Juels and Kaliski 2007]: POR	Probabilistic Bounded	No	$O(1)$	$O(1)$ hash transformation	$O(1)$
[Blum et al. 1994]: Memory checker	Probabilistic Unbounded	No	$O(m)$	$O(n/m)$ chunk fetching	$O(n/m)$
[Naor and Rothblum 2005]: Authenticator	Probabilistic Unbounded	No	$O(1)$	$O(n/m)$ chunk fetching	$O(n/m)$
[Ateniese et al. 2007]: PDP	Probabilistic Unbounded	Possible	$O(1)$	$O(n/m)$ exponentiation	$O(1)$
[Shacham and Waters 2008]	Probabilistic Unbounded	No	$O(1)$	$O(n/m)$ exponentiation	$O(1)$
[Chang and Xu 2008]: based on redactable signatures	Probabilistic Unbounded	Possible	$O(1)$	$O(\log(n))$ signature construction	$O(\log(n))$
[Chang and Xu 2008]: RSAh solution	Probabilistic Unbounded	No	$O(1)$	$O(n/m)$ exponentiation	$O(1)$
[Lillibridge et al. 2003]	Probabilistic Unbounded	No	$O(n)$	$O(1)$ simple comparison	$O(1)$
Wagner in [Golle at al. 2002]	Probabilistic Unbounded	Possible	$O(1)$	$O(\log(n))$ hash transformation	$O(\log(n))$
[Schwarz and Miller 2006]	Probabilistic Unbounded	Possible	$O(1)$	$O(n/m)$ signature validation	$O(1)$
Our probabilistic solution	Probabilistic Unbounded	Yes	$O(1)$	$O(n/m)$ chunk fetching	$O(n/m)$
[Deswarte et al. 2004]: pre-computed challenges	Deterministic Bounded	No	$O(1)$	$O(n)$ hash transformation	$O(1)$
Our polynomial-based deterministic solution	Deterministic Bounded	Possible	$O(1)$	$O(n)$ polynomial interpolation	$O(1)$
[Bellare et al. 1995]: Incremental cryptography	Deterministic Unbounded	No	$O(1)$	$O(n)$ fetching	$O(n)$
[Caronni and Waldvogel 2003]	Deterministic Unbounded	No	$O(n)$	$O(n)$ hash transformation	$O(1)$
[Golle at al. 2002]: SEC	Deterministic Unbounded	No	$O(1)$	$O(n/m)$ exponentiation	$O(1)$
[Deswarte et al. 2004], [Filho and Barreto 2006]: RSA solution	Deterministic Unbounded	Possible	$O(1)$	$O(n)$ exponentiation	$O(1)$
[Sebé et al. 2007]	Deterministic Unbounded	Possible	$O(m)$	$O(n/m)$ exponentiation	$O(1)$
Our ECC-based deterministic solution	Deterministic Unbounded	Yes	$O(m)$	$O(n/m)$ point multiplication	$O(1)$
Our DH-based deterministic solution (Appendix A)	Deterministic Unbounded	Yes	$O(1)$	$O(n)$ operations of exponentiation	$O(\log(n))$

Homomorphic hash functions. The second solution described in [Deswarte et al. 2004] requires little storage at the verifier side and no additional storage overhead at the holder side;

yet makes it possible to generate an unlimited number of challenges. The proposed solution (inspired from RSA) has been also proposed by Filho and Barreto in [Filho and Barreto 2006]. It makes use of a key-based homomorphic hash function *H*. A construction of *H* is also presented as $H(m)=g^m \bmod N$ where *N* is an RSA modulus and such that the size of the message *m* is larger than the size of *N*. In each challenge of this solution, a nonce is generated by the verifier which the prover combines with the data using *H* to prove the freshness of the answer. The prover's response will be compared by the verifier with a value computed over *H*(data) only, since the secret key of the verifier allows the following operation (*d* for data, and *r* for nonce): $H(d + r) = H(d) \times H(r)$. The exponentiation operation used in the RSA solution makes the whole data as an exponent. To reduce the computing time of verification, Sebé et al. in [Sebé et al. 2007] propose to trade off the computing time required at the prover against the storage required at the verifier. The data is split in a number *m* of chunks $\{d_i\}_{1\le i\le m}$, the verifier holds $\{H(d_i)\}_{1\le i\le m}$ and asks the prover to compute a sum function of the data chunks $\{d_i\}_{1\le i\le m}$ and *m* random numbers $\{r_i\}_{1\le i\le m}$ generated from a new seed handed by the verifier for every challenge. Here again, the secret key kept by the verifier allows this operation: $\sum_{1\le i\le m} H(d_i + r_i)= \sum_{1\le i\le m} H(d_i) \times H(r_i)$. The index *m* is the ratio of tradeoff between the storage kept by the verifier and the computation performed by the prover. Furthermore, the basic solution can be still improved as described in [Chang and Xu 2008]; though the verification method is probabilistic. The holder will be storing tags of $t_i = g^{y_i+s_i}$ where s_i is a random number kept secret by the verifier. The holder periodically constructs compact forms of the data chunks and corresponding tags using time-variant challenge sent by the verifier. The authors of [Chang and Xu 2008] argue that this solution achieves a good performance.

Delegating verification. The authenticator and the memory checker perform verifications on behalf of the user; though they are considered as trusted entities within the user's platform. None of the presented schemes considers distributing the verification task to other untrusted peers; they instead rely on the sole data owner to perform such verifications. In a P2P setting, it is important that the owner delegates the verification to other peers in the network in order to tolerate the intermittent connection of peers and even the fact that a single point of verification constitutes a single point of failure. Some of the schemes presented above may allow delegating verification provided that the verifier is not storing any secret information because it may otherwise collude with the holder. Additionally, the amortized storage overhead and communication complexity should be minimized for this purpose. To our knowledge, our proposed verification protocols are the first work to suggest delegating the verification task to multiple peers selected and appointed by the data owner.

The main characteristics of the existing verification protocols seen in this section are summarized in Table 4.

3.6. Summary

This chapter presented three verification protocols that satisfy the performance, and security requirements of self-organizing storage applications with different levels. The security mechanisms which were developed in this paper make it possible to verify whether a data storing peer that responds to a challenge still possesses some data as it claims, and without sacrificing security for performance. This verification can also be delegated to third party verifiers, thereby fulfilling an essential architectural requirement of self-organizing storage.

Assessing the actual state of storage in such an application represents the first step towards efficiently reacting to misbehavior: active replication strategies, whereof we have presented

how we can achieve in a self-organized form, can be built based on such evaluations. Reactive replication strategies can be also envisaged as described in the next Chapter that particularly suggest that generation of new data copies does not require the participation of the data owner. Cooperation incentives may also benefit from peer evaluations. Stimulating peer cooperation is however more complex than assessing their instantaneous cooperation with the execution of a challenge-response protocol. The use of a cooperation stimulation scheme should ultimately make it possible to detect and isolate selfish and malicious peers. In Chapter 5, we suggest reputation-based and remuneration-based cooperation incentive mechanisms that both rely on the verification primitive in having a *quasi-punctual* evaluation of peer behavior, and thus we argue that they are better customized to the P2P storage problem than the existing literature on cooperation incentives.

3.7. Relevant publication

1. Nouha Oualha and Yves Roudier. Securing ad hoc storage through probabilistic cooperation assessment. 3rd Workshop on Cryptography for Ad hoc Networks, July 8th, 2007, Wroclaw, Poland. Electronic Notes in theoretical computer science, Volume 192, N°2, May 26, 2008, pp 17-29.

2. Nouha Oualha, Melek Önen, and Yves Roudier. A Security Protocol for Self-Organizing Data Storage. 23rd International Information Security Conference (IFIP SEC 2008), Milan, Italy, September 2008.

Chapter 4

Secure P2P data storage and maintenance

Data possession verification protocols allow the verifier to detect (either deterministically or probabilistically) whether the holder destroyed data. To ensure data security, the detection of data destruction should trigger regeneration of a new copy the data at another holder in order to maintain a high (or at least a minimum) replication rate. This task cannot be solely tackled by the owner, since it does not often participate in the verification process. Therefore, the task should be conferred to the other participants in the verification process: verifiers and holders should then cooperate in order to regenerate a new data replica that will be stored at a volunteer peer. This regenerated replica should be personalized such that personalization information is opaque to the new holder; however, the data should not be passed through a third party or a verifier because then it is transmitted one unnecessary time.

In this chapter, we introduce a new method that relies on the proposed deterministic data possession verification protocol described in the Section 3.4 and that additionally allows verifiers and holders to regenerate new data replicas even if the owner is absent (i.e., offline); though the method can be also applied to the probabilistic verification protocol in an analogous way. The novelty of the method is that it allows regenerating a replica with the help of verifiers and holders present in the network, in addition to performing personalization of the generated replica on the fly without the need of making the data transit via a verifier peer.

We will first review some existing techniques that may be used to realize the maintenance of the stored data with reliability, security and self-organization as essential objectives. Then, we will describe a new data storage and maintenance protocol that is resilient to several attacks that may pose a threat to the well operation of such protocol.

4.1. Threat model

The different attacks the P2P data storage and maintenance mechanism is exposed to are detailed in Section 3.1.3; there are also new kinds of threats related to the rejuvenation process:

- *Denial-of-Service (DoS) attacks:* Malicious verifiers may flood the network with useless rejuvenation messages. In order to prevent that attack, a threshold *k*' of honest verifiers is defined that must detect a storage fault before requesting the generation of a new data replica.
- *Data poisoning:* during the repair phase, holders may cheat by performing a bogus data rejuvenation. Verifiers may also play a part in constructing such bogus data. These verifiers may construct bogus metadata associated with the new holder that stores in its turn a bogus data; thus the malicious holder goes undetected by the honest verifiers.

4.2. An overview of existing approaches

A storage mechanism consists in mainly two phases which are data storage, whereby the owner stores some data at one peer, and data verification, whereby it verifies that the data is actually stored. However, in order to discuss all the requirements described above and to address all above threats, we further refine the storage service into five sequential phases (see Figure 17): during the *selection phase*, potential holders of the data are elected by the data

owner who later on stores its data at these holders during the *storage phase*. The owner then appoints verifiers for its remote data during the *delegation phase* and these verifiers periodically check the availability and integrity of the stored data during the *verification phase*. Whenever these verifiers detect any data destruction or corruption, the *repair phase* is activated, in which the verifiers generate a new copy of the data with the help of the remaining holders. These phases are described below in more detail.

4.2.1. Selection

Selection is the process through which peers that are asked to store data are elected. The goal of the selection phase is to choose a set of peers that can maintain data availability while consuming minimal bandwidth. There are two possible techniques for holder selection. A *discriminatory selection* determines specific peers chosen such that they satisfy some constraint (for example, they exhibit a correct behavior as described in [Dingledine 2000]) or such that they share similar characteristics with the owner like their on-line availability, or dedicated bandwidth (as illustrated in [Toka and Michiardi 2008]). In contrast, *random selection* is generally used for its simplicity since it is less sophisticated and since it consumes less bandwidth per peer. TotalRecall [Bhagwan et al. 2004] and our P2P storage cooperation incentives (which will be described in the next Chapter 5) rely on a distributed-hash-table (DHT) to randomly select data holders. The selection is realized by randomly choosing a value from the DHT address space and routing to that value. We claim that the random selection mitigates some type of pre-set collusion between these holders (see next Chapter 5). Similarly, [Godfrey et al. 2006] analyzes peer selection strategies and proves the positive effects of randomization through the study of a stochastic model of a P2P system under churn. After holders have been selected, the owner can directly contact them for data storage. To mitigate the problem of peers having multiple identities as first described as a Sybil attack in [Douceur 2002], peers joining the system may pay with computational, bandwidth or storage abilities, such as for example crypto-puzzles in [Vishnumurthy et al. 2003] (the reader may refer to [Levine et al. 2006] for an exhaustive survey of counter techniques to the Sybil attack).

4.2.2. Storage

Once peers which will store the data have been selected by the owner, the latter should send the data to these potential holders. Data availability can be ensured either by implementing some form of redundant storage, through either replication or erasure coding. With replication, a simple copy of the data is distributed to each selected peer. With erasure coding, a data is instead divided into several blocks and additional blocks are generated to ensure data reconstruction as soon as a given number of blocks are retrieved. Replication, which has been mostly used in DHTs, more seriously increases the storage overhead and maintenance bandwidth without a comparable increase in fault tolerance. In contrast, erasure codes offer a better balance between the storage overhead and fault tolerance achieved. Many storage systems like Wuala[44], AllMyData Tahoe[45], and TotalRecall [Bhagwan et al. 2004] rely on the latter. Erasure codes are more complex than replication and in particular, the maintenance of coded data blocks introduces additional computational costs since it requires performing the coding yet again. Communication costs are also needed to retrieve a minimum number of coded blocks from several holders. A tradeoff between storage requirement and data maintenance must be determined when considering the use of erasure codes or replication: [Weatherspoon et al.

[44] http://wua.la/en/home.html
[45] http://allmydata.org/

2005] for instance describes how quantitative simulation might help in doing so. Quite opposite results in [Courtes 2007] suggest with analytic models that simple replication may be less detrimental than erasure codes with respect to data dependability in several scenarios. Moreover, in the case of replication, since the size of the data can be very large, holders may not cooperate and cheat on storing the data. They may even collude and ensure that only one holder is storing the data for all the other selected ones. Data personalization has been introduced as a solution to this threat: the owner generates a single and different replicate of the data for each holder and ensures that the response to the challenge during the verification phase is also different (e.g., see Section 3.4 of Chapter 3). This type of collusion may also arise with erasure coding even though it becomes problematic only if the number of colluding holders exceeds the number of original data blocks.

4.2.3. Delegation

As previously described, the storage mechanism should ensure that the data is continuously available and that holders are rightly claiming to be storing the data assigned to them. The verification phase relies on specific challenge-response protocols that achieve remote data integrity verification. P2P networks being very dynamic, the owner cannot be assumed to be always online, in particular if the storage service is used for backup purposes. At times when the owner is not present in the network, data verification should still be ensured by the owner's delegates, that is, *verifiers*. The distribution of that verification also improves performance through load balancing.

4.2.4. Verification

P2P storage systems generally use timeouts/heartbeats to detect peer failures. A new type of challenge-response protocol has been proposed to tackle the problem of remotely proving the integrity of some data stored by a holder (see Section 3.5 of Chapter 3). Even though these new cryptographic primitives prevent the generation of correct responses, a cheating holder may simply not reply a verifier's challenge thereby pretending to be offline or crashed. Distinguishing permanent failures, malicious or not, from transient ones, in which case the holder may return with the data intact after some time, is difficult. This is generally handled through the use of a *grace period* during which the verifier waits for challenges to be answered before declaring the holder as faulty.

4.2.5. Repair

The repair phase can be triggered in active or reactive mode. Active repair can be performed in a regular-basis; though such operation may be either insufficient or expensive entailing considerable both storage and bandwidth overhead because its periodicity is generally not tailored to the actual status of the stored data. On the other hand, with reactive repair, detecting that one of the holders has cheated and does not store the data anymore should trigger a data repair operation in order to ensure data availability. The verifier in charge should select another peer to perform the required operations to store the data and to generate the corresponding security metadata. Given the dynamicity of P2P networks, such operations should not rely on the presence of the owner. Additionally, the cooperative behavior of the peers participating in the repair operations should be stimulated. The recovery may be triggered almost immediately after the detection of a cheating or delayed holder. Simulation results of [Bhagwan et al. 2004] demonstrate that delayed repair (lazy repair) is more efficient in terms of data availability and

overhead costs tradeoff than immediate repair (eager repair) for a large data size and a highly dynamic system. In Section 4.3, we propose an analytic model that studies the periodicity of the repair phase.

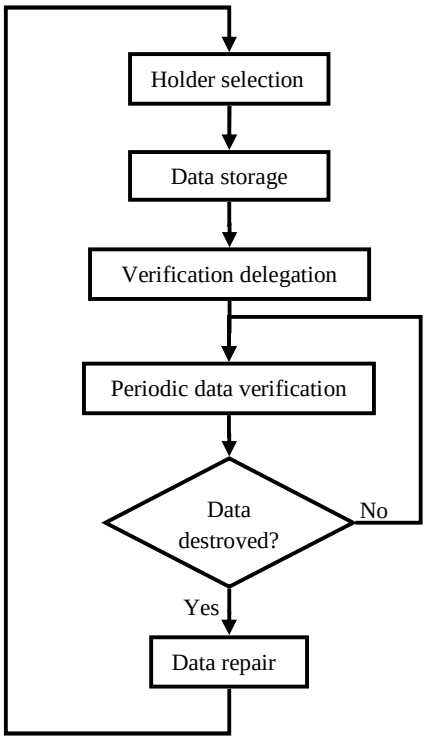


Figure 17 Data storage and maintenance phases

4.3. An erasure coding based data storage and maintenance protocol

This section presents a new data storage and maintenance protocol for P2P storage systems. The proposed protocol uses the verification protocol introduced in Section 3.4 of Chapter 3. To our knowledge, such verification protocol is the first to suggest distributing the task of verifying the remote data to multiple untrusted peers selected from the network. The security of the protocol relies on the hardness of two problems associated with the elliptic curve cryptography (defined in Section 3.4.1). However, the deterministic verification protocol does not instantiate a holder selection strategy, and specially does not suggest any repair method for the destroyed data that have been detected. The proposed solution that will be described in the following integrates such method.

Additionally, the following protocol opts for erasure coding rather than replication (that is rather suggested in Section 3.4) for better storage versus reliability tradeoff. It employs some type of erasure coding scheme, the random linear erasure coding [Acedański et al. 2005]. With such data coding, the entries of the generating matrix of the encoded data are chosen randomly.

4.3.1. Description

The description of the data storage and maintenance protocol concentrates on the four phases discussed in the previous section: storage (Figure 18), delegation (Figure 19), verification

(Figure 20), and repair (Figure 21) phases. The selection phase may however rely on a random selection.

In the following, the protocol is described phase by phase:

- **Storage:** As discussed in the previous section, data have to be stored at multiple peers in order to ensure data availability and reliability. Secure data storage with the simple replication technique has been proposed and evaluated in 3.4.5. Since the use of erasure coding technique provides the same level of reliability as replication but with much lower storage requirements at holders, a new storage mechanism based on erasure codes is proposed. At this phase, the data D is first divided into k blocks $\{d_i\}_{1\leq i\leq k}$. These blocks are then encoded to produce $k+m$ coded blocks $\{b_i\}_{1\leq i\leq k+m}$ based on the generating matrix G defined in $\mathbb{Z}$ as:

$$G = \begin{bmatrix} I_k \\ A \end{bmatrix}$$

where I_k denotes the $k\times k$ identity matrix and A denotes a $m\times k$ random matrix in $\mathbb{Z}$. Each coded block b_i is generated using the following linear operations in $\mathbb{Z}$:

$$b_i = \sum_{j=1}^{k} \alpha_{i,j} \times d_j$$

where the $\alpha_{i,j}$ is an entry of G at the i^{th} row and j^{th} column. The coded block b_i is finally sent by the owner to one distinct holder that is named the i^{th} holder.

1) Owner: divide D into k blocks $\{d_i\}_{1\leq i\leq k}$
2) Owner: generate random numbers in $\mathbb{Z}$ $\{\alpha_{i,j}\}_{1\leq i\leq k+m, 1\leq j\leq k}$
3) for each $1\leq i\leq k+m$
   Owner: compute $b_i = \sum_{j=1}^{k} \alpha_{i,j} \times d_j$
4) for each $1\leq i\leq k+m$
   Owner $\rightarrow$ i^{th} holder: b_i
5) i^{th} holder: keep b_i

Figure 18 Storage phase

Delegation & verification: These two phases integrate the solution from Section 3.4 of Chapter 3 whereby the metadata is computed using the coded block instead of the whole data. This verification only guarantees the storage of one block and therefore considered as partial. The delegation of verification uses a specific elliptic curve (defined in [Koyama et al. 1991]) such that the order of the curve is kept secret by the owner. The owner delegates the task of verifying one coded block to a number v of verifiers. The verifier assigned to the i^{th} holder receives from the owner a metadata information T_i such that $T_i=b_i.P$, where b_i is an integer that maps to the coded block stored at the holder, and P is a generator of the elliptic curve. Based on such metadata, the verifier is able to periodically check whether the holder stores block b_i. Indeed, it first sends a challenge message $Q=r.P$ to the holder where r is a freshly generated random number. Upon reception of the challenge, the holder computes $R=b_i.P$ and sends the product to the verifier. The verifier checks whether the equality $r.T_i=R$ holds. If the latter equality is not

met, the verifier detects that the block has been either corrupted or destroyed by the holder

1) Owner: generate a specific elliptic curve (refer to Section 3.4.1)
2) for each $1 \leq i \leq k+m$
   Owner: compute $T_i = b_i.P$
3) for each $1 \leq i \leq k+m$
   for each $1 \leq j \leq v$
   Owner → verifier: T_i
4) Verifier: keep T_i

Figure 19 Delegation phase

In reaction, this event should trigger the generation of a new block to replace the lost one. This operation is performed in the next phase.

1) Verifier: generate a random number r
2) Verifier: compute $Q = r.P$
3) Verifier → i^{th} holder: Q
4) i^{th} holder: compute $R = b_i.Q$
5) i^{th} holder → Verifier: R
6) Verifier: Check if $r.T_i = R$?
   If $r.T_i \neq R$ launch a repair phase

Figure 20 Verification phase

- **Repair:** To activate this phase, a fraction of verifiers assigned to a given holder consisting of at least *k*' peers has to detect the destruction of the block stored at a holder. They first select a random key altogether (e.g., the key is the XORed sum of the random numbers chosen by verifiers). The random key is used to select a new holder randomly. The new block is generated based on a coding operation over *k* blocks. The coding operation is executed by the new holder, who receives *k* verified blocks $\{b_{t_i}\}_{1 \leq i \leq k}$ from a randomly selected set of the remaining holders. The verifiers also agree on a seed *s* that will be sent to the new holder. The seed can be simply computed as a sum of random numbers each one of them chosen by each verifier. The seed allows to generate random coefficients $\{c_i\}_{1 \leq i \leq k}$. The new holder then computes the new block *b*' in $\mathbb{Z}$ as follows:

$$b' = \sum_{l=1}^{k} c_l \times b_{t_l}$$

The new generated block can be written as a linear combination of the original data blocks. Indeed, since each block transmitted by the holders participating in the regeneration process can be written as a combination of the original data blocks, then:

$$b' = \sum_{l=1}^{k} c_l \times b_{t_l} = \sum_{l=1}^{k} c_l \times \left(\sum_{j=1}^{k} \alpha_{t_l,j} \times d_j \right)$$

Thus,

$$b' = \sum_{j=1}^{k} \left(\sum_{l=1}^{k} c_l \times \alpha_{t_l,j} \right) \times d_j$$

As a result, the generated block is coded based on the random linear erasure coding scheme. [Acedański et al. 2005] demonstrates that any $k \times k$ sub-matrix of a random matrix is invertible with a high probability for a large field size; thus the property of erasure coding is still provided by the new block. Moreover, the new block is distinct from the lost block as well as from the remaining blocks stored in the system. The indexes of the holders involved in the redundant block generation process along with the seed *s* must be stored by the verifiers then handed out to the owner when he is available again, thus allowing it to update the generating matrix *G* of the erasure codes. If the block b_i has been destroyed, the update only affects the i^{th} row of the matrix: the new row is defined as $(\alpha'_{i,1}, \ldots, \alpha'_{i,k})$ where for each *j* in [1, *k*]:

$$\alpha'_{i,j} = \sum_{l=1}^{k} c_l \times \alpha_{t_l,j}$$

Each verifier assigned to the revoked holder keeps its role as a verifier for the new holder. However, it requires new metadata information *T*' for the new coded block that is computed as a linear combination over the metadata information stored at other verifiers (responsible of the holders that have been involved in the block generation) and using the same set of coefficients:

$$T' = \sum_{l=1}^{k} c_l \times T_{t_l}$$

The new metadata corresponds to the new block *b*' stored at the new holder; this is realized owing to the commutativity properties of elliptic curves ([Koblitz 1987], [Miller 1986]):

$$T' = \sum_{l=1}^{k} c_l \times T_{t_l} = \sum_{l=1}^{k} (c_l \times b_{t_l}).P = b'.P$$

1) Verifiers: generate a seed *s*
2) Verifiers: select *k* random holders
3) Verifiers → new holder: *s*, $\{b_{t_i}\}_{1 \le i \le k}$
4) New holder: generate random coefficients $\{c_i\}_{1 \le i \le k}$
5) New holder: compute $b' = \sum_{l=1}^{k} c_l \times b_{t_l}$
6) New holder: keep *b*'

(a)

1) Verifiers→ new holder's verifiers: *s*, $\{T_{t_i}\}_{1 \le i \le k}$
2) New holder's verifiers: generate random coefficients $\{c_i\}_{1 \le i \le k}$
3) New holder's verifiers: compute $T' = \sum_{l=1}^{k} c_l \times T_{t_l}$
4) New holder's verifiers: keep *T*'

(b)

Figure 21 Repair phase: (a) construction of a new coded block and (b) construction of the corresponding metadata.

4.3.2. Security evaluation

This section analyses the security of the proposed data storage and maintenance protocol with respect to the attacks discussed in Section 4.1.

Preventing data destruction: Each verifier checks the availability of one remote coded block. The destruction of any block is detected through the verification protocol that has been proven in Section 3.4.4 of Chapter 3 as being a proof of knowledge protocol. Indeed, the protocol is proved to be complete i.e., the verifier always accepts the proof as valid if the holder follows the protocol, and sound i.e., the verifier will not accept the proof as valid if the holder destroys or corrupts the data.

Collusion resistance: With erasure codes, the produced blocks that will be stored at holders inherently differ from each other, which ensures blocks are personalized. Even though collusion between k+1 or more holders may happen, we assume that such a collusion is unlikely because it requires sending k blocks (comparable in size to the original data) to one of the holders to encode the destroyed block. This entails considerable bandwidth and computation costs for each verification operation.

A newly generated block differs from the remaining stored blocks. This is guaranteed with the randomization added by the seed s that is chosen cooperatively by the verifiers to prevent potential collusions between one particular holder and an additional peer.

Preventing DoS attacks: A quota system can be introduced into the protocol to regulate the number of challenge messages the verifier is allowed to send to a given holder during a time frame. This allows mitigating a flooding attack against the holder launched by a malicious verifier.

The activation of a repair phase is made possible only after the agreement of at least k' verifiers. The use of independent verifiers mitigates the maliciousness of some of them that may flood the system with repair requests. The threshold value k' is a tradeoff factor between these two considerations: prevention against verifier collusion and also handling of peer churn and intermittent availability.

Preventing data poisoning: A holder or verifier may send bogus information to the concerned peers. We argue that this problem can be easily thwarted by including a signature with any information to provide proofs of origin and integrity for the recipient of such information. Each coded block or metadata is associated with some owner signature that attests its validity.

On the other hand, after a data repair phase, the new holder or the new verifier will keep track of a compilation of all necessary owner signatures that validates the newly generated block or metadata. For example, a new holder can keep along with the freshly coded block b', the seed s used to generate coefficients for the new block, and the following set of information:

$$\left\{b_{t_i}.P,\ sign_{owner}(b_{t_i}.P)\right\}_{1\le i\le k}$$

b' being coded based on blocks $\{b_{t_i}\}_{1\le i\le k}$ and $sign_{\text{owner}}(.)$ being the signature generated by the owner. When the owner reconnects to the system, it makes contact with the new holder, checks the validity of its signature compilation, and replaces all these with its own signature:

$$sign_{owner}\left(b'.P=\sum_{l=1}^{k}c_l\times\left(b_{t_l}.P\right)\right)$$

To simplify such process and in particular to reduce the signature overhead, the protocol may alternatively rely on any homomorphic signature (e.g., algebraic signatures as described in [Schwarz and Miller 2006]) providing the following property:

$$sign_{owner}\left(b' = \sum_{l=1}^{k} c_l \times b_{t_l}\right) = \prod_{l=1}^{k} sign_{owner}(b_{t_l})^{c_l}$$

The new holder or verifier can thus construct a valid signature for the newly generated block or metadata based on the signatures of the generating blocks or metadata and without having recourse to the owner.

Preventing man-in-the-middle attacks: The holders of a given data are selected randomly. As suggested in section 5.2.2 of the next chapter, a data owner cannot choose by itself the identities of its data holders. This means that the owner necessarily have a key ID in the DHT that is distinct from the key IDs of its holders. These holders are then contacted directly for data storage and verification. To prevent a man-in-the-middle attack, the response of a holder storing block b_i to a verifier's challenge may be constructed as a digest of the product $b_i.P$ along with the holder's identity *ID*: $R=hash(b_i.P, ID)$, *hash* being a pseudo-random one-way function. The peer's *ID* can correspond to the peer's IP address, which is forgeable but may still make it possible to establish a secure channel between two peers if we assume no attack on the routing protocol. With such a construction of the response, an attacker cannot trick a verifier by pretending to be storing the block and holding at the same time an identity different from *ID*.

4.3.3. Performance evaluation

In the proposed protocol, the performance of both the delegation and verification phases has been already evaluated in Section 3.4.5 (Chapter 3). Since metadata are only computed based on one block instead of the whole data, the performance in fact improves.

The proposed repair method requires the transmission of only *k* coded blocks (corresponding in total to the file size) for the regeneration of one block while at the same time providing personalized regenerated block to the new holder. Additionally, the bandwidth utilization for the regeneration is distributed between holders.

Furthermore, the communication overhead of the proposed repair method can be optimized by relying on hierarchical codes as proposed by [Duminuco and Biersack 2008]. With such erasure codes, the required number of coded blocks to repair a block is greater or equal to the number of children in the tree hierarchy and less or equal to *k*.

The communication overhead caused by verifier agreement and notification messages can be considered negligible owing to the fact that the data (or the block) are likely considerably larger in size.

The linear combination of blocks is operated in $\mathbb{Z}$ which may lead to an increase in the size of the produced block by at most *k* bits. We argue that this increase is insignificant given the original size of blocks.

4.4. An analytic model for P2P data storage and maintenance

This section introduces an analytic model that describes the P2P storage system inspired from the epidemic models in [Jones and Sleeman 1983] where peers are classified into groups depending on their state. We endeavor to determine the right periodicity for data maintenance with such model.

We consider an owner that replicates its data at r holders using a (k, r-k)-erasure coding. The original data can be generated from at least k coded blocks. The owner also delegates the verification of each coded block to v verifiers. These verifiers have the responsibility to periodically check the presence and integrity of the stored block at their assigned holder. Whenever at least k' verifiers detect the destruction or corruption of the block they decide to regenerate a new block and store it at a new holder. We assume that verifiers do not require to be replaced often during the data storage. The owner is supposed to connect to the P2P system from time to time in order to select new verifiers and to appoint them to the desired blocks.

4.4.1. Model of P2P data storage without data maintenance

Figure 22 depicts a state model for describing the presence of holders in the P2P system. Holders may disconnect or definitely leave the system. Peer disconnection and peer departure rates, which are respectively named λ and μ, are considered constant. Peers may reconnect at constant rate λ'. Holders may be in state "connected", "disconnected" or "left". Additionally, holders do not just leave the system after a crash but may also in state "left" if they destroy blocks they store. In this model, holders that have destroyed blocks are not replaced (no data maintenance). We designate the number of holders in states "connected", "disconnected", and "left" at time t by respectively $n_c(t)$, $n_d(t)$, and $n_l(t)$, the total number of peers at time t being:

$$n_c(t) + n_d(t) + n_l(t) = r$$

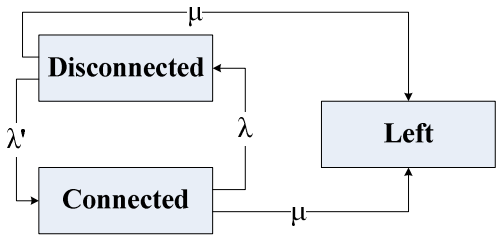


Figure 22 State model of data storage without maintenance

The number of holders in each state varies with time according to the differential equations derived from the state model:

$$\begin{aligned} \frac{dn_c(t)}{dt} &= \lambda' n_d(\mathrm{t}) - (\mu + \lambda) n_c(t) \\ \frac{dn_d(t)}{dt} &= \lambda m_c(\mathrm{t}) - (\mu + \lambda') n_d(t) \\ \frac{dn_l(t)}{dt} &= \mu(n_c(\mathrm{t}) + n_d(t)) \end{aligned} \qquad (4.4.a)$$

The solution of these equations gives the number of holders in each state at time t:

$$\begin{aligned} n_c(t) &= \frac{\lambda' r}{\lambda + \lambda'} e^{-\mu t} + \frac{\lambda r}{\lambda + \lambda'} e^{-(\mu+\lambda+\lambda')t} \\ n_d(t) &= \frac{\lambda r}{\lambda + \lambda'} e^{-\mu t} - \frac{\lambda r}{\lambda + \lambda'} e^{-(\mu+\lambda+\lambda')t} \\ n_l(t) &= r(1 - e^{-\mu t}) \end{aligned}$$

The number of holders in the system $n_c(t)+n_d(t)$ converges to zero with time. This means that there is a certain time t_0 at which the owner's data is not available any more (i.e., t_0 is the time limit for data availability), and there is another time t_1 at which the owner cannot retrieve its

data from the storage system (i.e., t_1 is the time limit for data reliability). The time limit t_0 is defined as:

$$n_c(t_0) = \frac{\lambda' r}{\lambda + \lambda'} e^{-\mu t_0} + \frac{\lambda r}{\lambda + \lambda'} e^{-(\mu+\lambda+\lambda')t_0} = k$$

To simplify the above equation, the exponential functions can be rewritten as infinite power series (in the form of Taylor series) that can be approximated to their first order. A solution is obtained for:

$$t_0 \sim \frac{1}{\mu + \lambda}(1 - k/r)$$

The time limit t_1 is obtained if:

$$n_c(t_1) + n_d(t_1) = re^{-\mu t_1} = k$$

This leads to:

$$t_1 = (1/\mu)\log(r/k)$$

4.4.2. Model of P2P data storage with data maintenance

If we consider that destroyed blocks are detected and regenerated at other new holders, we obtain a new state model (depicted in Figure 23). This model describes a repair phase for destroyed blocks during which new holders are introduced in the model at a constant rate γ ($1/\gamma$ is also the verification time period).

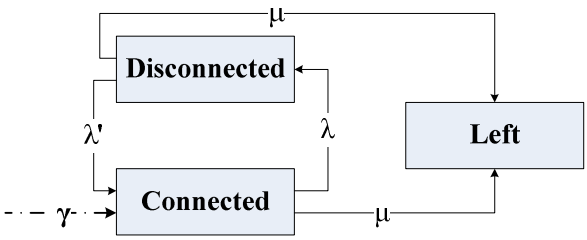


Figure 23 State model of data storage and maintenance

The number of holders in each state verifies the following differential equations derived from the model:

$$\frac{dn_c(t)}{dt} = \lambda' n_d(\mathrm{t}) + \gamma(r - n_c(\mathrm{t}) - n_d(\mathrm{t})) - (\mu + \lambda) n_c(t)$$

$$\frac{dn_d(t)}{dt} = \lambda n_c(\mathrm{t}) - (\mu + \lambda') n_d(t)$$

$$\frac{dn_l(t)}{dt} = \mu\big(n_c(\mathrm{t}) + n_d(t)\big)$$

$$\frac{d(n_c(t) + n_d(t) + n_l(t))}{dt} = \gamma(r - n_c(\mathrm{t}) - n_d(\mathrm{t}))$$

The solution to these differential equations gives the number of holders in each state:

$$n_c(t) = \frac{r}{\gamma + \mu}\left[\frac{\gamma(\mu + \lambda')}{\mu + \lambda + \lambda'} + \frac{\mu(\lambda' - \gamma)}{\lambda + \lambda' - \gamma} e^{-(\gamma+\mu)t} + \lambda\left(\frac{\gamma}{\mu + \lambda + \lambda'} + \frac{\mu}{\lambda + \lambda' - \gamma}\right) e^{-(\mu+\lambda+\lambda')t}\right]$$

$$n_d(t) = \frac{\lambda r}{\gamma+\mu}\left[\frac{\gamma}{\mu+\lambda+\lambda'} + \frac{\mu}{\lambda+\lambda'-\gamma}e^{-(\gamma+\mu)t} - \left(\frac{\gamma}{\mu+\lambda+\lambda'} + \frac{\mu}{\lambda+\lambda'-\gamma}\right)e^{-(\mu+\lambda+\lambda')t}\right]$$

$$n_l(t) = \frac{\mu r}{\gamma+\mu}\left[\gamma t + \frac{\mu}{\gamma+\mu}(1-e^{-(\gamma+\mu)t})\right]$$

We note that the equations in 4.4.a match the above equations for $\gamma = 0$. To be able to perform the recovery of dropped blocks, we should have $n_c(t) \geq k$ for each $t>0$. This leads to the following inequality that must be met:

$$\gamma \geq \frac{\mu}{\frac{r}{k}\left(\frac{\mu+\lambda'}{\mu+\lambda+\lambda'}\right)-1};\ \frac{k}{r} < \frac{\mu+\lambda'}{\mu+\lambda+\lambda'}$$

The above equation gives a precise bound on the data maintenance period T_{max}:

$$T_{max} = \frac{\frac{r}{k}\left(\frac{\mu+\lambda'}{\mu+\lambda+\lambda'}\right)-1}{\mu}$$

4.4.3. Numerical simulation

We simulated the above model of a P2P data storage system in different scenarios based on the equations developed earlier. In the simulation, peers join the system for an average lifetime of 2 weeks. Each peer stays online for 1 hour and connects on average 6.4 times in a day. Additionally, holders are assumed to destroy their blocks one time per day on average. Without data maintenance, the owner's data is not available after only 49 minutes and then cannot be recovered after less than 2 days. With maintenance however, the data is always available and retrievable. But data maintenance should be periodically performed 3.8 times per day.

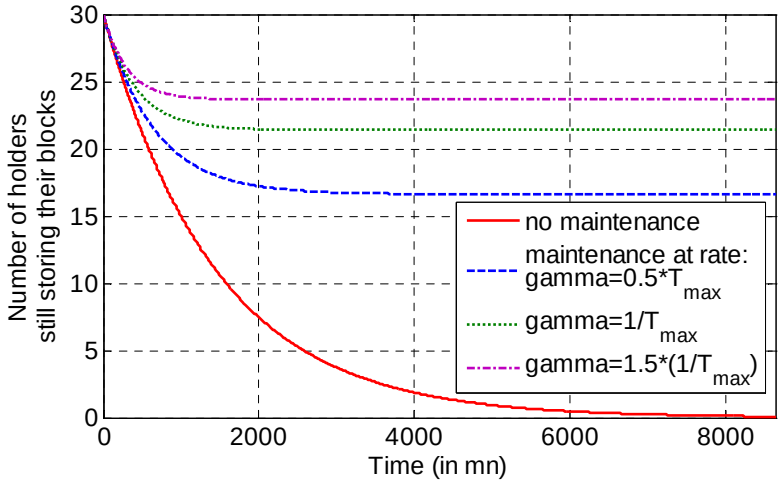


Figure 24 Number of holders. r=30, k=5, v=10, k'=7, d=6.94×10^{-4}, λ=0.0167, λ'=0.0044 (rates per minute (mn)).

Figure 24 shows the number of holders that are still storing data blocks with time in different settings. The figure illustrates the fact that without maintenance (γ=0) the number of holders decreases converging to zero. On the other hand, with maintenance ($\gamma \neq 0$), the number of holders converges to an equilibrium value that is not null. This value depends on the γ ratio: if $\gamma < 1/T_{max}$, then the value is lower than k; otherwise it is higher than k thus rendering the restoration of

destroyed blocks possible. The figure proves that with the data maintenance mechanism, the P2P storage system is able to achieve the survivability property for the stored data.

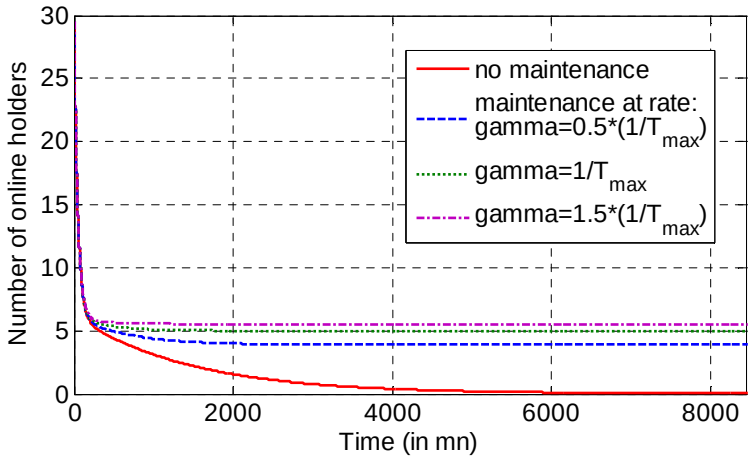


Figure 25 Number of online holders. r=30, k=5, v=10, k'=7, d=6.94×10^{-4}, λ=0.0167, λ'=0.0044 (rates per mn).

Figure 25 shows the number of online holders over time computed with and without a repair phase. The figure proves that considering maintenance at a rate $\gamma \geq 1/T_{max}$ and with k chosen in function of system parameters enables the system to work with high data availability. In the case without data maintenance however, blocks required to recover the data are not accessible anymore with time.

4.5. Summary

This chapter presented a P2P data storage and maintenance protocol in which the detection of data corruption and data rejuvenation are self-organizing functions. This protocol is innovative in that peers cooperate not only in providing storage resources, but also for ensuring their resilience. In particular, the main security and dependability functions of this protocol are distributed to multiple peers which makes it easier to mitigate non cooperative behaviors, while additionally coping with churn.

Chapter 5

Audit-based cooperation incentives

Cooperation is a central feature of P2P systems that is key to their scalability. However, cooperation to achieve some functionality is not necessarily an objective of peers that are not under the control of any authority and that may try to maximize the benefits they get from the P2P system. Cooperation incentive schemes have been introduced to stimulate the cooperation of such self-interested peers. They are diverse not only in terms of the applications which they protect, but also in terms of the features they implement, the type of reward and punishment used, and their operation over time. Cooperation incentives are classically classified into reputation-based and remuneration-based approaches. Cooperation enforcement, that is using cooperation incentives to ensure a proper operation of the P2P system, may rely on a dedicated and trusted coordinator or, in its purest form, constitute a self-organizing mechanism.

This chapter discusses the design of two cooperation incentive mechanisms for a P2P storage application and their application to thwarting misbehavior. The first one is reputation based, and relies on the evaluation of peers' past behavior to estimate how trustful they will be in upcoming interactions. The second one is remuneration (or payment) based, and features explicit rewards for a correct behavior. Both mechanisms rely on the security primitives discussed in previous chapters, notably on a protocol for the remote verification of data possession as a primitive for the continuous evaluation of behavior of storage peers.

5.1. Cooperation incentives for P2P storage

In a P2P storage system, individual peers join their efforts and cooperate for the correct operation of the application. It is generally suggested that cooperation will help entities to succeed better than via competition. [Buttyán and Hubaux 2003] demonstrated that the best performance in mobile ad-hoc routing is obtained when nodes are very cooperative. Devising mechanisms stimulating cooperation among peers should therefore receive a great deal of attention.

Shortcomings of existing approaches

The majority of existing approaches for stimulating cooperation have been introduced for immediate services like packet forwarding in ad hoc networks ([Michiardi 2004] and [Buttyán and Hubaux 2003]) or for transferring a data block in P2P file sharing networks (Napster[46], Gnutella[47], KaZaA[48], Morpheus[49], or BitTorrent[50]). In MANET routing for instance,

[46] http://www.napster.com/
[47] http://www.gnutella.com/
[48] http://www.kazaa.com/
[49] http://www.morpheus.com/
[50] http://www.bittorrent.com/

encouraging packet forwarding by increasing forwarder node's reputation or handing it tokens is an explicit and immediate counterpart for cooperation. Evaluating cooperation for a data storage service at the time of data retrieval is less easy, because pessimistic approaches make it necessary for a cooperating peer to wait for a long time before it gets rewarded for its cooperation, while optimistic approaches might make it possible for a not abuse the cooperation mechanism to achieve an immediate gain. Hence, there must be a cooperation incentive mechanism more adapted to distributed storage applications and that must support the periodic verification of stored data.

Incentives with multiple objectives

Coping with free riding or voluntary data destruction cannot be achieved by a simple tit-for-tat strategy like in BitTorrent [Piatek et al. 2007], but requires the owner together with the help of some volunteer peers, verifiers, to periodically check storage at holders and decide if some data needs to be replicated again in the network. The data stored can be periodically checked using one of the verification protocols discussed in previous chapters, including our own that suggests that verification should be mostly handled by verifier peers selected and appointed by the data owner to distribute the load of this task. Still, verifiers may themselves cheat like holders (indeed, "quis custodiet ipsos custodies"[51]); the P2P storage system therefore requires a cooperation incentive mechanism to be used to incite peers to cooperatively and fairly help in providing reliable and secure data storage, either-based on reputation or remuneration. Thus, a cooperation incentive mechanism adapted to distributed storage must at least serve a twofold objective: to incite peers to store data for other peers and to motivate others to verify these data.

The remainder of this chapter presents approaches whereby a good evaluation of peer behavior goes essentially by verifying the integrity of the data stored. Verification results serve to estimate the reputation of data holder peers in a reputation-based mechanism or to decide whether to reward or punish peers in a remuneration-based approach. Such an evaluation is additionally a way to indirectly establish trust.

5.2. Reputation-based approach

This section introduces a new reputation system for P2P storage applications that allows estimating the trustworthiness of peers based on experiences and observations of their past behaviors.

5.2.1. Threats

The adversaries that we consider for the reputation-based approach are peers that trick the reputation system for any perceived personal benefit. In the following, we examine ways which peers may use to subvert the reputation mechanism.

- *Lying:* a liar is a peer that disseminates incorrect observations on other peers ("rumor spreading") in order to either increase or decrease their reputation. Colluded liars may form a group of peers that conspires against one or more peers in the network by assigning unfairly low reputation to them ("bad mouthing") or unduly high reputation to members of their group.
- *Collusion between owner and holder:* The collusion aims at increasing the reputation of the holder at honest verifiers. Just lying to verifiers supposes that observations of peers

[51] Who watches the watchers?

rely on external recommendations. However without these recommendations, peers may still be vulnerable to lying using such type of collusion where the owner pretends storing bogus data at the holder.

- *Collusion between holder and verifier:* The aim of such collusion is to advertise the quality of holder more than its real value ("ballot stuffing") thus increasing its reputation at owner. But, still the owner may ultimately and opportunistically check by itself storage at holder to make its own view on the holder.
- *Whitewashing:* peers may repeatedly leave system and rejoin with new identity escaping the consequences of their bad actions, so that misbehaving or well behaving does not matter for them.
- *Sybil attack:* If peers are able to generate new identities at will, they may use some of them to increase the reputation of the rest of identities either by lying, or pretending to be the owner and holder or holder and verifier of some data.

5.2.2. Reputation-based storage

In the P2P storage system, we rely on the construction of groups in which we evaluate peer behavior. Peers store their personal data in their group. The security of data stored is the responsibility of group members, given that they are periodically verified by some group members for availability and no corruption.

Group construction and management

Peer groups are dynamic with members that join and leave the group at anytime. Such group-based architecture allows only intra-group interactions, and thus peers establish rapid knowledge of the trustworthiness of their group fellows. Moreover, the group ensures a minimum level of good behavior: whenever a peer misbehaves it is badly audited by a growing number of group members until becoming totally isolated from the group.

Peer groups are created either in a centralized or in a decentralized manner. Centralized managed groups can be constructed at outset by an authority like partnership in [Lillibridge et al. 2003] that may tackle also the task of distributing the group key to all members. The group key controls the access to the group, and ensures secure and private communication between its members. On the other hand decentralized groups are cooperatively formed at will by its members and they rely on collaborative group key agreement protocols (e.g., [Lee et al. 2006], [Lesueur et al. 2007]).

Group members are in a structured Distributed Hash Table (DHT). A DHT consists of a number of peers having each a key Key_{Peer} in the DHT space, which is the set of all binary strings of some fixed length. We assume that the DHT provides a secure lookup service (see [Sit and Morris 2002] and [Castro et al. 2002]): a peer supplies an arbitrary key (an element in the DHT space), and the lookup service returns the active node in the DHT that is the closest to the key.

In the group, peers have unique identities in the DHT. The risk of Sybil attacks can be mitigated by imposing a membership fee for peers willing to join a given group, or in a decentralized way constraining the number of invitations any group member possesses as proposed in [Lesueur et al. 2008].

Self-organizing peer selection

The audit-based P2P storage system allows peers to delegate the verification of their data to other volunteer peers, the verifiers, and also to only accept to store data of well-behaved peers.

Verifier selection. A data owner desiring to store a data replica in the system may randomly choose verifiers to whom it will send a verification request. The random selection of verifiers may be based on a random operation proper to the owner, for example the identity of the verifier *i* can be the closet key to the value $Key_{Verifier}=Hash(Key_{Owner}||nonce||i)$ where *Hash* is a pseudo-random function determined at group outset and *nonce* is a randomly chosen number protecting against a replay of the same operation ("||" means concatenation). From peers answering to this request, the owner selects *m* peers, and then acknowledges them including in the message the list of the *m* chosen verifiers. This information is a commitment from the owner to the verifiers' list.

Holder selection. To avoid collusion between the owner and the holder, holders may be chosen randomly in the DHT overlay in the same way as verifier selection (with a fresh new nonce). Alternatively, we may make also the selected verifiers choose altogether the holder for the owner. Each verifier *i* commits to a randomly chosen DHT key k_i (commitment can be as simple hash operation of the key) and then sends this commitment to the owner. The owner sends the digest of verifiers' commitments to each verifier. Upon the receipt of the owner's message, verifiers will send their chosen random keys to the owner. The selected holder is the peer with the closest key to the XORed sum of these random keys:

$$Key_{Holder} = k_1 \oplus k_2 \oplus \ldots \oplus k_m$$

The owner sends a digest of the messages received by verifiers containing their keys along with the identity of the chosen holder.

It is clear that the process of selecting holders requires several communication messages between the owner and verifiers that might be grouped in a single multicast message; nevertheless, this is the price to pay to obtain a consensus between the owner, the verifiers, and the holder, and particularly to avoid collusion between any participants in this agreement.

Interaction decision

We may rely on a simple trust model based on whitelisting (see Figure 26) similarly to the Tit-For-Tat (TFT) strategy in BitTorrent [Piatek et al. 2007]: peers that have correctly stored data they have promised to preserve are added to the whitelist of their observers (the data owner and its delegated verifiers).

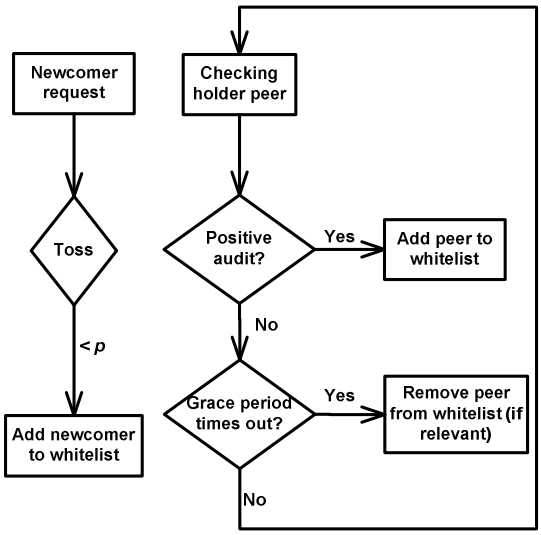


Figure 26 Whitelisting model.

Whenever a peer detects that another peer has destroyed data it has promised to store, the latter will be removed from the whitelist. We also propose a "grace period" during which "no response" from the challenged holder is tolerated until the period times out, thus avoiding abusively isolating cooperative holders with transient connection.

Newcomers to the system are probabilistically added to the whitelist. Newcomer acceptance probability may be computed based on the upload capacity of the peer and its whitelist size. This probabilistic process serves to bootstrap the storage system, but it also means that whitewashers may probabilistically gain some advantage of that. Other trust models can be adopted like for example the Additive Increase Multiplicative Decrease (AIMD), the Linear Increase Sudden Death (LISD), or blacklisting mechanisms.

A peer accepts to only serve peers pertaining to its whitelist: it stores their personal data or periodically verifies their data availability in the system. However, a peer may accept to store its data at peers that do not pertain to its whitelist.

5.2.3. Analytic evaluation

This sub-section discusses the potential of the audit-based approach in observing peer behavior through the study of an analytic model.

The trustworthiness of a peer can be estimated based on the observation of its behavior by third parties. The semantics of the information collected can be described in terms of direct (or local) or indirect (or system-wide) observations. Direct observation amounts to the compilation of a history of personal interactions by one peer towards another peer when being the owner of data stored at the peer or serving as verifier of this peer. On the other hand, indirect observation refers to any reputation information received from other peers in the system. There are substantial communication savings to be gained by limiting observations to just private interactions even though indirect observation may be only partially disseminated or piggybacked on ordinary messages. Besides, using only direct observation may delay the evolution of reputation.

A reputation-based approach for P2P storage applications generally allows estimating the trustworthiness of a given peer based on experiences and observations of its past behavior towards the actual estimator or other peers. Similarly, the audit-based approach, that we propose, relies on the estimation of the trustworthiness of this very peer based on experiences of the estimator, solely as a data owner or its observations obtained from audits of other peers' data, in the role of a verifier. The following gives an evaluation of both approaches based on an analytic model.

Model

This sub-section discusses how to compute the gain of choosing one way of observation reciprocity over the other in terms of the level of correctness of gathered reputation information.

Considering two peers p_1 and p_2, where p_1 desires to have correct observations on p_2. Peer p_1 may perform a correct observation itself or may receive observations from other peers in the system that may be correct or incorrect. Our model assumes that incorrect observations are received from dishonest peers only. Let η denote the fraction of dishonest peers in the total population.

We define a quality level for the estimated observation with two extrema: $\bar{o}$ and $\underline{o}$. An observation of quality $\bar{o}$ is correct, and an observation of quality $\underline{o}$ is incorrect. Observation may be null to refer to the situation where p_1 does not have any observation on peer p_2 (indistinguishably from the worst reputation).

First of all, the probability that p_1 knows about the p_2's behavior is computed (it must at least obtain the result of one interaction involving p_2); the estimated observation of p_1, denoted $\tilde{o}$, is then derived for two different cases:

- *Audits:* observations based on storage and verification results: p_1 only takes into account its personal interactions with p_2 as an owner storing data at p_2 or as a verifier for other peers' data stored at p_2.
- *Reputation:* observations based on peer's experiences and also recommendations: p_1 takes into account both its personal interactions and opinions expressed by other peers with respect to p_2. The reputation model is inspired from [Anceaume and Ravoaja 2006] where reputation computation is based on a subset of information provided by randomly chosen peers.

Audits: The probability that p_1 knows about the behavior of p_2 is equal to:

$$Prob[p_1\ knows\ p_2] = \theta_1 = 1 - \left(1 - \frac{\lambda r}{(n-1)}\right)(1 - \frac{\lambda r}{(n-1)} + \frac{\lambda r}{(n-1)}(1 - \frac{m}{(n-2)})^r)^{n-2}$$

λ being the average storage rate of peers and n being the number of peers (the considered time unit is the time period between two verification operations).

Since personal observations are always correct, the estimated observation quality may only take two values: correct observation or no observation.

$$Prob[\tilde{o}_1 = \bar{o}] = \theta_1$$
$$Prob[\tilde{o}_1 = \underline{o}] = 0$$
$$Prob[\tilde{o}_1 = 0] = 1 - \theta_1$$

On average, we have:

$$\tilde{o}_1 = \theta_1 \times \overline{o}$$

Reputation: The probability that p_1 knows about the behavior of p_2 is equal to:

$$Prob[p_1\ knows\ p_2] = \theta_2 = 1 - (1 - \theta_1)^{\gamma n}$$

γ being the fraction of the peer population to which the reputation is propagated. External observations may either originate from honest peers or from dishonest peers. Peer p_1 receives at best $(1-\eta)\times\gamma\times n$ observations from honest peers and $\eta\times\gamma\times n$ from dishonest peers. Observations from honest peers are all correct; and observations from dishonest peers are always incorrect. For k and k' not null observations respectively received from honest and dishonest peers, the average observation quality is denoted by $t_{k,k'}$ when p_1 has a direct observation, and by $t'_{k,k'}$ when p_1 does not have a direct observation:

$$t_{k,k'} = (1-w)\overline{o} + w\frac{(k\overline{o} + k'\underline{o})}{k+k'}$$
$$t'_{k,k'} = w\frac{(k\overline{o} + k'\underline{o})}{k+k'}$$

w being the weight that p_1 gives to averaged system-wide observations with respect to local observations. For $0 \leq k \leq (1-\eta)\times\gamma\times n$ and $0 \leq k' \leq \eta\times\gamma\times n$, we have:

$$Prob[\tilde{o}_2 = t_k] = (C^{k}_{(1-\eta)\gamma n}{\theta_1}^{k+1}(1-\theta_1)^{(1-\eta)\gamma n-k})(C^{k'}_{\eta\gamma n}{\theta_1}^{k'}(1-\theta_1)^{\eta\gamma n-k'})$$
$$Prob[\tilde{o}_2 = t'_k] = (C^{k}_{(1-\eta)\gamma n}{\theta_1}^{k}(1-\theta_1)^{(1-\eta)\gamma n-k+1})(C^{k'}_{\eta\gamma n}{\theta_1}^{k'}(1-\theta_1)^{\eta\gamma n-k'})$$
$$Prob[otherwise] = 0$$

The value $C^{k}_{(1-\eta)\times\gamma\times n}$ (respectively $C^{k'}_{\eta\times\gamma\times n}$) is the number of combinations of k (respectively k') peers from the set of honest (respectively dishonest) peers from which p_1 gathers observations. A certain probability of interaction is attached to the observations of both honest and dishonest peers. This is due to the fact that even though peers have to provide cryptographic proofs that they had interactions with p_2, even honest peers cannot always provide proofs of correct observation: for example, the observation of the absence of any response from p_2 cannot be proved; or the peer sending an observation may be in collusion with p_2.

Using the Vandermonde's identity, we have on average:

$$\tilde{o}_2 = \theta_1(1-w) + w((1-\eta)\times\overline{o} + \eta\times\underline{o})$$

Comparison

Seeking for simplicity, we choose quality observations such as: $\overline{o} = 1$, $\underline{o} = -1$. Thus, we have:

$$\tilde{o}_1 = \theta_1$$
$$\tilde{o}_2 = \theta_1(1-w) + w(1-2\eta)$$

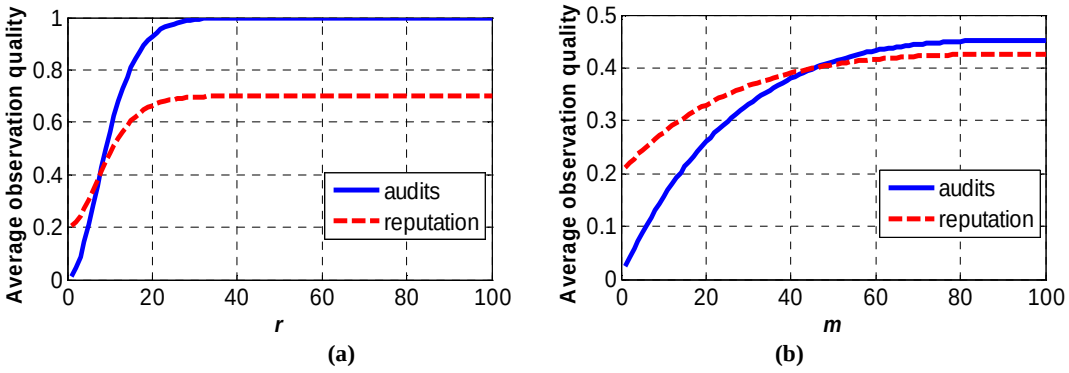


Figure 27 Average observation quality: (a) varying *r* with *m*=5 and (b) varying *m* with *r*=3. *n*=100, λ=0.2, γ=0.3, *w*=0.5, η=0.3.

The average quality of observations is computed in the two cases. Figure 27 shows that the best quality obtained depends very much on the replication rate.

If the replication rate is low (simple data redundancy), the reputation outperforms the audit-based approach; however, if the replication rate is high (more than 10 replicas using for example erasure codes), the audit-based approach is the best way to observe.

The number of verifiers has also an impact on both approaches: increasing *m* leads into an increase on the observation quality of the two approaches with a more significant increase of the audit-based approach.

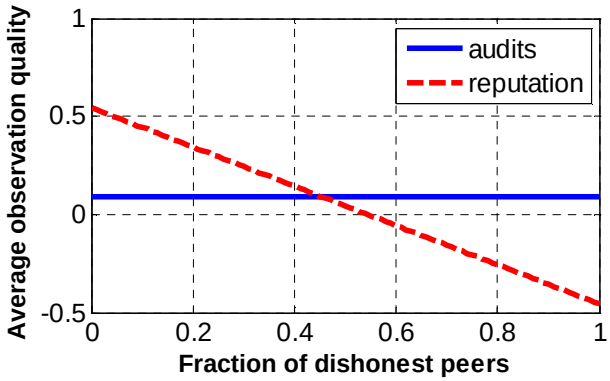


Figure 28 Average observation quality varying the fraction of malicious peers. n=100, λ=0.2, γ=0.3, r=3, m=5, w=0.5.

If the ratio of peers that send false observations increases, the quality of observation in the case with reputation linearly decreases with this ratio, however this quality is not affected in the case of audits, as it is depicted in Figure 28.

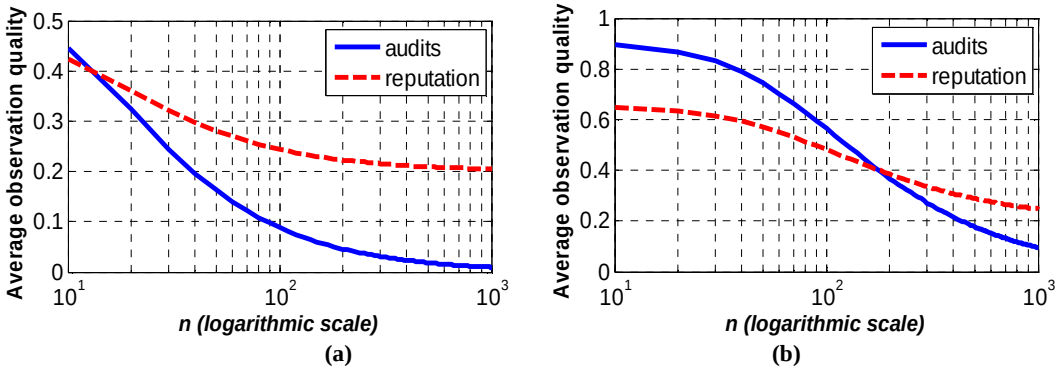


Figure 29 Average observation quality varying the number of peers for (a) r=3 and (b) r=10. λ=0.2, γ=0.3, m=5, w=0.5, η=0.3.

Figure 29 shows that increasing peer population *n* leads to a decrease in the quality of observations in both approaches, especially the audit-based one. Small peer populations are more in favor of audit-based approach than reputation; whereas large peer populations are more advantageous for reputation if the replication rate is small than for audit-based approach.

Discussion

The study of the analytic model demonstrates that the audit-based approach for observing peer behavior outperforms reputation if the data replication rate is high (e.g., erasure coding) and with small peer population. Moreover, the approach is robust against liars, and it does not require propagation of information which avoids the problem of rumor spreading.

Since the audit-based approach works better for small population, the analytic model validates the group-based architecture for the P2P storage system as more favorable to the audit-based approach for peer behavior observation than reputation.

5.2.4. Simulation experiments

To validate the ability of our audit-based P2P storage system to detect and punish selfish peers, we implemented a custom simulator whose framework is at first described, and then results of simulation are presented and analyzed.

Framework

The self-organizing storage system is modeled as a closed set of homogeneous peers. The storage system operation is modeled as a cycle-based simulation. One simulation cycle corresponds to the period between two successive verifications.

Churn: Peers arrive to the system in Poisson distribution: there are 100 newcomers per hour, for an average lifetime of 2 weeks. [Stutzbach and Rejaie 2004] shows that Gnutella peer uptime follows a power-law distribution. We will use the same distribution for peer uptime and downtime. In average, a peer stays online for 1 hour and connects in average 6.4 times in a day.

Storage: Peer storage space, file size, and storage duration are chosen from truncated log-normal distributions. The storage space of each peer is chosen from 1 to 100GB, with an average of 10GB. In each day of simulated time, 2.85 of files are stored per peer for an average period of 1 week. The average file size is 500MB. The stored files will be checked by verifiers each day.

User strategies: We consider three peer strategies: cooperation, passive selfishness (free-riding) and active selfishness.

- *Cooperative:* whenever the peer accepts to store data from another peer, it keeps them stored. Whenever the peer accepts to check the availability of some data at a storage peer, it will periodically perform verification operations on this peer as agreed. Such peers carefully apply the audit-based approach to their strategies.
- *Passively selfish:* the peer will never accept to store data and will never accept to verify the availability of some data stored for other peers. The peer is just consumer of the storage system. This type of behavior is also termed free-riding.
- *Actively selfish:* the peer probabilistically accepts to store data for other peers or to verify storage at other peers. Whenever it stores or verifiers for others, it will fulfill its promise only probabilistically. This type of behavior with an instability effect probabilistically alternating between cooperation and selfishness: probability of participation denoted p and probability of achieving promise denoted q.

Strategy with strangers: Cooperative peers accept to store or verify strangers' data only probabilistically. Such strategy bootstraps the system and allows peers to discover new peers with whom they may reciprocally cooperate; even though it also permits to whitewashers to

unfairly take advantage of peers' generosity. The probability of cooperation with strangers is denoted *P*.

Simulation results

The framework is simulated in different scenarios in order to analyze the impact of system parameters and choices on the convergence time of the storage system to a stable state where only cooperative peers are the active consumers of the storage in the system.

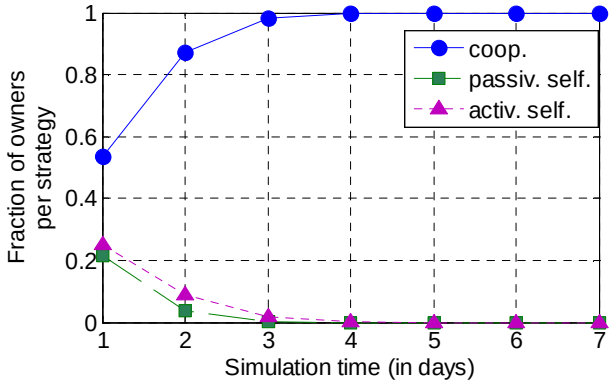


Figure 30 Averaged ratio of owners per strategy. *n*=300, *r*=3, *m*=5, *P*=0.01, *p*=0.2, *q*=0.2, 40% cooperators, 30% passively selfish peers, 30% actively selfish peers.

Exclusion of selfish peers: Figure 30 demonstrates that selfish peers have less capability over time to store data in the system; however, cooperative peers are becoming the majority of data owners in the storage system. Free-riders are excluded from storing data in the system before active selfish peers, because the latter cooperate at first by storing data then they destroy them which may slow their detection. The process of filtering out selfish peers from the system is made possible in a short time period of 3 days.

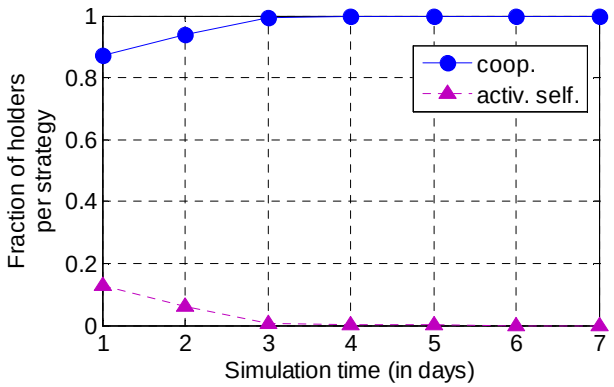


Figure 31 Averaged ratio of holders per strategy. *n*=300, *r*=3, *m*=5, *P*=0.01, *p*=0.2, *q*=0.2, 40% cooperators, 30% passively selfish peers, 30% actively selfish peers.

Figure 31 shows the fraction of holders per strategy over time. The figure demonstrates that the stored data will be exclusively held by cooperative peers after selfish peers have been detected (after 3 days). Selfish peers do not participate in the storage effort because they consider many owner peers as non cooperative towards them. These owner peers have

previously detected their selfish behavior and decided to stop cooperating to them. On the other hand, cooperative peers have probabilistically received contributions from many cooperative holders; therefore they return the favor by participating in the storage of the data of these holders.

Overhead: The bandwidth consumed for verification is dependent on the number, rather than the size, of files being stored. This is in fact a requirement on the verification protocol. Figure 32 shows the amount of control messages per stored file.

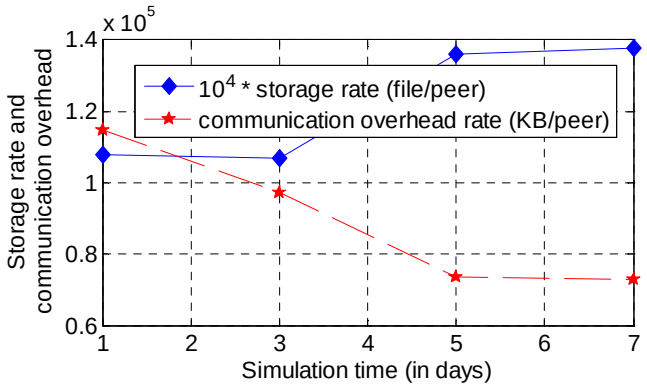


Figure 32 Average amount of control messages per file stored (in KB). *n*=1000, *r*=3, *m*=5, *P*=0.01, *p*=0.2, *q*=0.2, 40% cooperators, 30% passively selfish peers, 30% actively selfish peers.

The figure demonstrates that the bandwidth cost decreases since more and more peers are acquainted with other peers and thus their contributions increase. This explains the increase in the storage rate since cooperative peers cooperate at 100% with the peers they know rather than probabilistically.

Newcomer's acceptance: Figure 33 depicts the fraction of owners per strategy varying the probability *P* for newcomers' acceptance. This probability slows the participation of peers in the system; but, it insignificantly affects the convergence time of the system to a system free from selfish storage consumers.

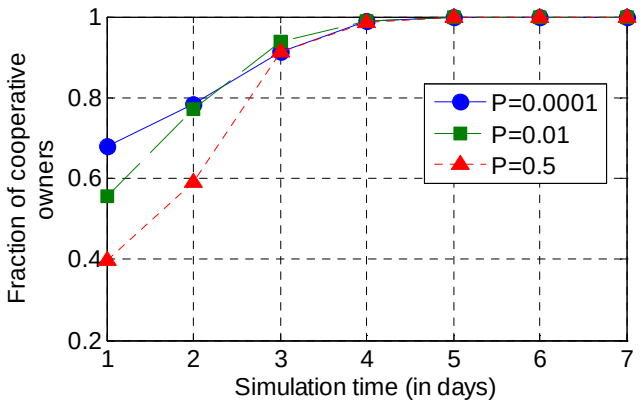


Figure 33 Fraction of cooperative owners varying the probability of newcomer's acceptance *P*. *n*=300, *r*=3, *m*=5, *p*=0.2, *q*=0.2, 40% cooperators, 30% passively selfish peers, 30% actively selfish peers.

Figure 34 illustrates the effect of the probability of newcomer's acceptance *P* on the storage rate. A very low value of the probability *P* (*P*=0.0001) realizes a very small storage rate because peers voluntarily participate less and then they are considered as selfish even if some of them are actually cooperative.

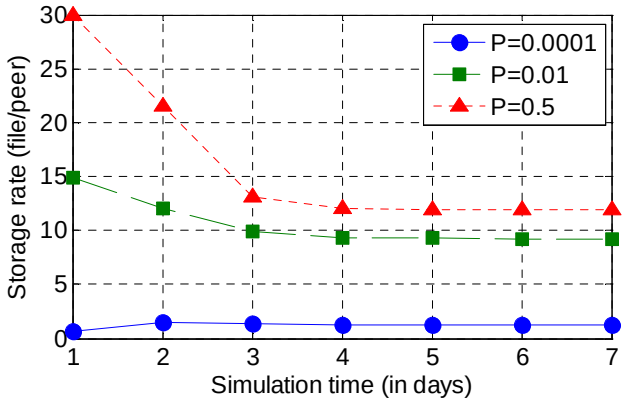


Figure 34 Average amount of data stored per peer varying the probability of newcomer's acceptance *P*. *n*=300, *r*=3, *m*=5, *p*=0.2, *q*=0.2, 40% cooperators, 30% passively selfish peers, 30% actively selfish peers.

High value of *P* is however more advantageous to the system since the storage rate is high; even though it decreases over time. Nevertheless, a system with a high value of *P* is vulnerable to the problem of whitewashing where peers defect then rejoin the system with new identities (not evaluated in the simulation). The theoretical study in Appendix B demonstrates that there is an optimal value of *P* that deters the effect of whitewashers while achieving a maximum societal welfare.

5.2.5. Security considerations

In this sub-section, we evaluate the robustness of the audit-based cooperation incentive mechanism against the attacks exposed in 5.2.1.

Lying observers have no impact on the auditing mechanism since estimations are based on verification results performed by the actual estimator; thus observations are objective. Collusions between the owner and its holder or a subset of its verifiers are mitigated by the random selection of holders and verifiers. Verifiers' selection relies on a pseudo-random function and a secure routing in the DHT that can be assessed by each verifier. And, holders are randomly selected by each verifier. So, collusion between any subset of participants is prevented.

The group-based architecture of the P2P storage permits controlling peers who are joining the storage system in order to mitigate Sybil attackers. This latter may still be able to take profit of peers that are probabilistically adding newcomers to their whitelist, still this probability can be adjustable depending on peer's confidence on the system. The architecture allows also a rapid knowledge about the behavior of group members, and then peers are able to refuse storage to non cooperating peers, hence limiting free-riders.

5.3. Remuneration-based approach

This sections introduces a mechanism that combines the monitoring of data storage on a periodic basis together with a payment scheme between the data owner, holders, and verifiers.

5.3.1. Threats

Cooperative storage relies on the interaction with unknown peers, hence under no prior trust relationships. Peers should participate to the system in compliance with the payment protocol; however peers may misbehave in various ways.

- *Sybil attack:* Sybil attack represents a potential vulnerability making it possible to generate new peers at will. The payment based mechanisms to be envisioned should therefore support some form of real world based authentication: this attack should at least be mitigated by imposing a real world monetary counterpart to membership for peers joining the storage system so that creating bogus identities cannot be a source of revenue for peers.
- *Impersonation:* every peer must know with whom it is dealing. Systems usually rely on a PKI (Public Key Infrastructure) where a certification authority issues certificates which bound an identity (peer's identity) with a public key.
- *Counterfeiting:* Peers are generally paid with tokens (virtual money, credit, cheque, etc). Counterfeiting is a fraudulent reproduction of a token. A token signed by the certification authority cannot be forged as long as the private key of the minter remains secret.
- *Double spending attack:* Double spending is a problem akin to digital cash where it is easy to spend a digital coin twice. There are two solutions to this problem: either making the payee verify that the coin is valid with the bank at the time of spending, or making spending a coin too many times reveal the identity of the double spender.
- *Fair exchange:* As mentioned in [Asokan et al. 1997], "many commercial transactions can be modeled as a sequence of exchanges of electronic goods involving two or more parties. An exchange among several parties begins with an understanding about what item each party will contribute to the exchange and what it expects to receive at the end of it. A desirable requirement for exchange is fairness. A fair exchange should guarantee that at the end of the exchange, either each party has received what it expects to receive or no party has received anything." Fair exchange protocols thus provide ways to ensure that items held by two or more parties are exchanged without one party gaining an advantage. In remuneration systems, obtaining an efficient cooperation incentive depends upon devising a protocol that enforces a fair exchange of the remuneration (virtual or not) against some task. This property can only be attained by intricately integrating the remuneration operation with the application functionality.
- *Starvation:* Starvation is the inability of a peer to participate in the cooperative system because it has not enough tokens to do so. Payment-based schemes generally suffer from starvation, e.g., see [Weyland et al. 2005].

5.3.2. Enabling mechanisms

Means of verification of remote storage and P2P peers fair payment must be supported by the payment-based incentive model, given that it relies on periodic fair exchange of credits between the contributing peers and consuming peers in the storage system, and periodic verification of remotely stored data at some holder peers by some verifier peers. We discuss in this section solutions that aim at providing these means.

Related work

There are several micropayment schemes that have been proposed in the past like PayWord, MicroMint [Rivest and Shamir 1996], and Millicent [Glassman et al. 1995] that particularly

present a centralized functionality marked by a central broker, the bank, that tracks each peer balance and payment transactions. In most of these schemes, the load of the bank grows linearly with the number of transactions; though hash chains in PayWord or the use of electronic lottery tickets [Rivest 1997] greatly reduce such cost. As example, the P2P micropayment system MojoNation[52] has also a linear broker's load and this system has gone out of work because the central bank constitutes also a single point of failure.

The scale of the P2P system makes it necessary to resort to a type of protocols termed optimistic protocols where the bank does not necessarily take part in the payment, but may be contacted to arbitrate litigations between peers. With such type of protocols, the bank's work is reduced. PPay [Yang and Molina 2003] is a lightweight micropayment scheme for P2P systems where the issuer of any coin is a peer from the system that is responsible for keeping trace of the coin. However, the bank comes into play when the issuer of a coin is off-line. In a very dynamic system, the probability of finding the original issuer of the coin on-line is very low. In this situation, PPay converges to a system with a centralized bank. Additionally, tamper resistant hardware (TRH) can be used to enforce payment protocols in a decentralized and optimistic fashion as illustrated by the TermiNodes [Buttyán and Hubaux 2001] and CASHnet [Weyland et al. 2005] projects.

To the best of our knowledge, the only fully-decentralized micropayment scheme that exists so far is KARMA [Vishnumurthy et al. 2003]. KARMA splits the bank functionality in different bank sets composed of peers selected and appointed randomly from the P2P system for each peer when it first joins the system. The KARMA payment scheme does not require any trusted infrastructure and is scalable. The scheme is described in more detail in the following section.

DHT-based payment framework

The KARMA framework [Vishnumurthy et al. 2003] proposes a payment protocol in which peers' balance and payment transactions are handled by a set of peers from the system network. KARMA proposes to substitute the bank by a set of peers randomly assigned within a distributed Hash Table (DHT) for each peer, called bank-set. The karma value, which constitutes the name of the currency, is maintained for each peer by its bank-set whose members are collectively responsible for continuously increasing and decreasing the karma value as peers contribute and consume resources from the P2P system (see Figure 35).

The bank-set is randomly assigned to each peer: the b closest peers to $HASH_B(Id(\text{Peer}))$ belong to the bank-set of that very peer ($HASH_B$ is a pseudo-random function publicly known). The bank-sets independently track the credits belonging to their assigned peers, and periodically agree on a given balance of credits with a majority rule. Even if there are inconsistencies in peer balances, transactions among peers correspond to tiny micropayments and thus do not produce considerable gains or losses to peers. Peers joining the system for the first time must solve a cryptographic puzzle in order to mitigate Sybil attacks against the storage system. The payment protocol in KARMA is similar to an online bank payment but with additional features that guarantee the consistence and synchronization of peer balances.

[52] MojoNation archived website.http://web.archive.org/web/20020122164402/%20http://mojonation.com/

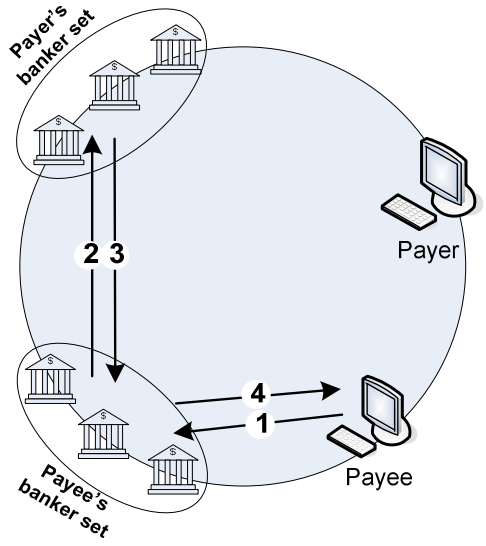


Figure 35 KARMA framework: 1) payee sends a transfer request to its banker set; 2, 3) after confirming the transfer from the payer's banker set, 4) payee's banker set will send back receipt to the payee.

The payment scheme proposed in this section relies on this framework to guarantee the fair exchange of payment against some storage space. The KARMA framework has also been applied to the file sharing problem described in [Vishnumurthy et al. 2003]. That application cannot be assimilated to a P2P storage application since in the former case, payments are immediately charged after the exchange of the file, whereas in the latter case, payments for storage or verification are by installment i.e., they are billed at a due date that corresponds to the confirmation (by verifications) of the good behavior of the holder or the verifier. Therefore, we will supplement the KARMA framework by an escrowing mechanism (described in detail later on) that guarantees the effective payment of credits promised by the owner towards a cooperative holder or a verifier.

Remote data verification

Our proposed scheme uses a verification protocol based on pre-computed challenges (see Figure 36). These challenges are generated by the owner and stored at the verifier. Each verifier metadata consist of the random numbers and their corresponding pre-computed challenges (the reader may refer to the first solution in [Deswarte et al. 2004] for details).

The number of verification operations is limited by the number of pre-computed challenges stored at the verifier. This limitation does not restrain our mechanism because payments of holders or verifiers should be in essence computable (in number and price); and also the number of verification operations should be proportional to such payments. Besides, we opted for this type of verification protocol because it does not require special cryptographic functions (just a hashing function), in addition to the fact that the computational and storage overhead from the verifier side and the holder side are optimized.

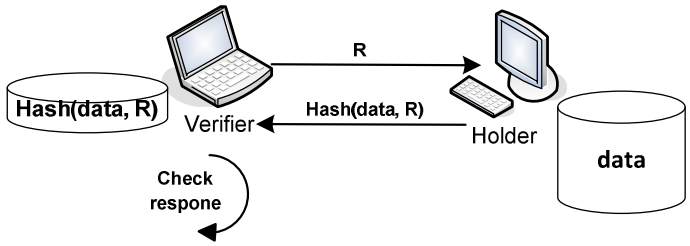


Figure 36 Used verification protocol

5.3.3. Payment-based Storage

In this section, we first give an overview of the payment scheme, to describe then the cryptographic protocol that achieves such scheme.

Overview of the payment scheme

We propose a mechanism that monitors data storage on a regular basis to determine the payments between data owners, holders, and verifiers. The payment mechanism allows a peer storing data for other peers to be paid for its service. It thus controls the storage functions seen above by rewarding cooperating peers.

Notations: Let B_P denote the bank-set of the peer P, pk_P the public key of a peer P, sk_P the private key of P, and sk_{B_P} the private key of the bank-set of P. A message M signed by some key K is denoted as $\{M\}_K$ (bank-set signature is explained in [Vishnumurthy et al. 2003]).

Let G be a finite cyclic group with n elements. We assume that the group is written multiplicatively. Let g be a generator of G. If h is an element of G then finding a solution x (whenever it exists) of the equation $g^x = h$ is called the discrete logarithm problem (DLP) and is assumed hard to solve.

Assumptions: A P2P system generally consists of altruistic peers, selfish peers, malicious peers, and others with behavior ranging in between. We will assume that there are a non-negligible percentage of the peers that are altruistic or at least correctly follow the protocol. Peers of the storage system are structured in a distributed Hash Table (DHT). A DHT consists of a number of peers having each a key *Key*(Peer) in the DHT space, which is the set of all binary strings of some fixed length. Each participant is assigned a secure, random identifier in the DHT identifier space: *Id*(Peer). We assume that the DHT provides a secure lookup service (see [Sit and Morris 2002] and [Castro et al. 2002]): a peer supplies an arbitrary key (an element in the DHT space), and the lookup service returns the active node in the DHT that stores the object with the key.

Peer selection: To avoid collusion between verifiers and the owner or verifiers and the holders, holders and verifiers should be randomly chosen. For instance, a *long-list* of randomly chosen potential holders and verifiers can be constructed using the DHT. The l_1 (respectively l_2) closest peers in the DHT identifier space to the key $HASH_H$(*Id*(Owner), timestamp) (respectively $HASH_V$(*Id*(Owner), timestamp)) constitute the potential holders (respectively verifiers) of the owner ($HASH_H$ and $HASH_V$ are pseudo-random functions publicly known). The random choice of holders and verifiers within the DHT allows the dissemination of storage requests, instead of for instance relying on network flooding.

Auction-based pricing: Payment-based schemes generally suffer from starvation, e.g., see [Weyland et al. 2005]. In our case, starvation means the inability of a peer to store data in the system because its account of credits is empty. We suggest an auction-based solution in order to mitigate the starvation phenomena. The solution aims at making peers holding small number of credits to contribute uppermost to the system in order to replenish their accounts. These peers may offer low price values for their storage allowing owners desiring to store data in the system to select them in priority. A peer seeking to store data initiates an auction by asking peers from the long-list that have been randomly selected to submit a bid to store the data in question. It then selects the lowest bidders, though other alternatives, such as second-price auctions are also possible. In the end, the owner has a *short-list* of n_h holders for its data. The same operation is repeated for n_v verifier selection.

Credit escrowing: Each peer has a personal account managed by a set of peers likewise KARMA [Vishnumurthy et al. 2003] that are called *bank-set*. Our payment scheme relies on digital checks. To prevent peers from emitting bad checks, the amount of credits that corresponds to a check value are escrowed, i.e., the necessary number of credits to pay check holder are locked by the bank-set. Consequently, bank-sets keep two types of peer balances: normal credits and locked credits. Credits are escrowed for some time-out (that corresponds to the check's expiry time), after which they are returned to the peer normal balance. The owner desiring to store data in the system must be able to pay its holders and verifiers with checks. That's why, it must escrow credits which are converted to digital checks (see Figure 37). These checks are then stored in a blinded version at the corresponding holders and verifiers. Checks include some random numbers that are generated by the owner and certified by its bank-set. The latter have a blinded version of these numbers too (to prevent collusion between one bank-set member and a holder or a verifier). Each blinded digital check has this form:

$$C(payer, payee, g^c) = g^c, id\ (payer), id(payee), price, seq, TTL, \sigma_{SK_{payer}}$$

c being a random number, *seq* being the check's sequence number, and *TTL* the check's expiry date. The payee's knowledge of *c* allows it to be paid credits of value *price*. The bank-set of the payer is not informed of this number *c*; but only a blinded version of it g^c. The verification operation allows both the verifier and the holder to extract the check in order to be able to present it to their bank-set to be paid in return. The holder must also escrow an amount of credits corresponding to the punishment it gets if it destroys data that it has promised to store. The escrowed credits of the holder are converted to one digital check that is certified by the holder's bank-set. The check is split into multiple shares each one will be stored at each verifier: a threshold number *k* of these shares allows reconstructing the full check. Shares of the digital check comprise the following numbers (in blinded version) $\{g^{s_i}\}_{1\leq i\leq n_v}$ which are shares of g^s if $\{s_i\}_{1\leq i\leq n_v}$ are shares of *s* [Desmedt and Frankel 1989]. If a threshold-based majority of verifiers agree that the holder has destroyed data, they can construct holder's check and present it to the owner such that this latter will be reimbursed.

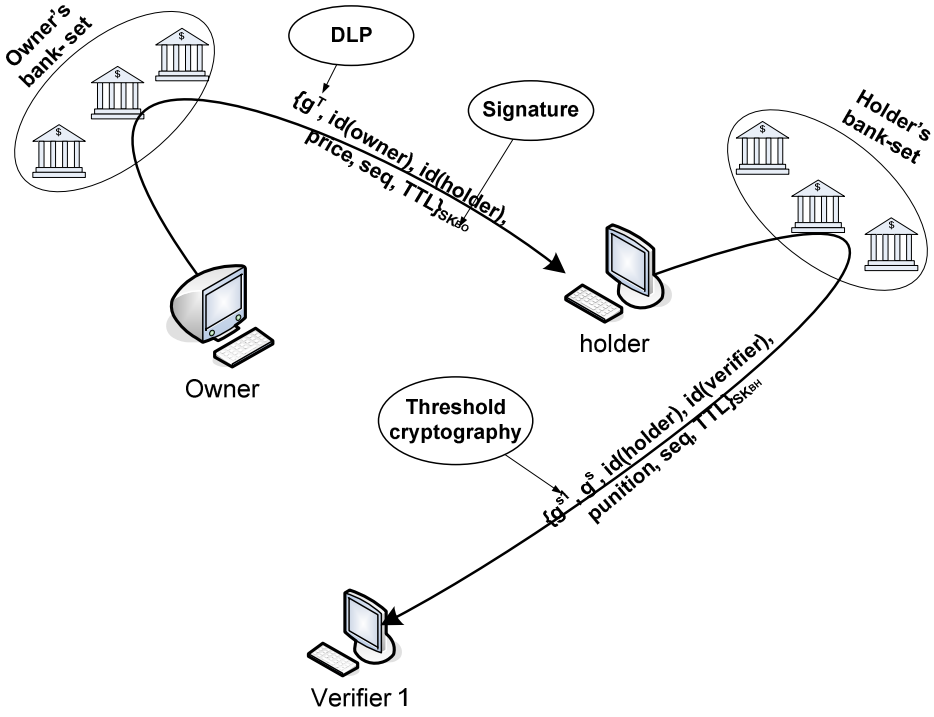


Figure 37 Escrowing credits

Data verification: Each verifier appointed by the owner periodically check storage of data stored at a holder. The verifier does not have the full challenge for the holder, but rather a share of the challenge: a threshold number of messages received by the holder from verifiers allows this latter to reconstruct the full challenge. Distributing the verification task to multiple verifiers prevents potential collusion between the holder and a verifier. The verification operation has three-fold objectives: it allows assessing the availability of stored data, it permits the verifier to remove the blinding factor of the stored digital check in order to get paid for verification, and finally it allows the holder to recover also its check for its payment too. Since, the verifier is paid exactly for each verification operation it actually performs, verification operations are executed in a defined number. Consequently, the payment scheme does not require a verification protocol where verifications are unlimited and may rely on pre-computed challenges for instance.

Payment protocol

In this section, we describe a protocol that provides a cryptographic implementation of the scheme.

We consider an owner denoted *O* that stores its data at a holder *H*. The integrity of such data is periodically checked by a verifier *V* on behalf of *O*. The proposed protocol consists in multiple steps described in the following Figure 38.

Credit escrowing	**(*O* escrows credits for the payment of *H* and *V*)** *O*: fix number of verification operations to *m* *O*: generate random numbers $\{R_i\}_{1\leq i\leq m}$, v_H, v_V *O*: compute for each $i\in[1, m]$ $T_i = HASH(HASH(d, R_i), v_H)$ $T'_i = HASH(HASH(d, R_i), v_V)$ $O \rightarrow B_O$:$\{g^{T_i}, price\}_{1\leq i\leq m}, id(H), \{g^{T_i}, price\}_{1\leq i\leq m}, id(V)$ $B_O \rightarrow O$: $\{C(O, H, g^{T_i})\}_{1\leq i\leq m}$, $\{C(O, V, g^{T'_i})\}_{1\leq i\leq m}$ **(*H* escrows credits to form its punition *p*)** *H*: generate a random number *s* *H*: generate $\{s_i\}_{1\leq i\leq n_v}$ shares of *s* $H \rightarrow B_H$: $\{g^{s_i}\}_{1\leq i\leq n_v}$, $g^s, id(O)$, *punition* B_H: check $\{g^{s_i}\}_{1\leq i\leq n_v}$ are shares of g^s $B_H \rightarrow H$: $\{C(H, O, g^{s_i})\}_{1\leq i\leq n_v}$
Data storage	**(*O* stores data *d* at *H*)** $H \rightarrow V$: $s_i, C(H, O, g^{s_i})$ $V \rightarrow H \rightarrow O$: $\{ACK\}_{SK_V}$ $O \rightarrow H$: *d*, $\{C(O, H, g^{T_i})\}_{1\leq i\leq m}$, v_H **(*O* delegates verification of *d* to *V*)** *O*: generate for each $i\in[1, m]$ $\{r_{ij}\}_{1\leq j\leq n_V}$ shares of R_i *H*: compute $\{HASH^2(d, R_i)\}_{1\leq i\leq m}$ ($HASH^2$: *HASH* is executed 2 times) $O \rightarrow V$: $\{HASH^2(d, R_i)\}_{1\leq i\leq m}$, r_i, $\{C(O, V, g^{T'_i})\}_{1\leq i\leq m}$, v_V
Data verification	**(*V* sends a share of the i^{th} challenge to *H*)** $V \rightarrow H$: i, r_{ij} **(*H* answers verifiers upon construction of challenge)** *H*: compute *Res*=*HASH*(*d*, R_i) $H \rightarrow V$: *Res* **(*V* checks *H*'s answer)** *V*: check *HASH*(*Res*) =? $HASH^2(d, R_i)$
Payment	**(*H* obtains its i^{th} payment)** *H*: compute $T_i = HASH(HASH(d, R_i), v_H)$ $H \rightarrow B_H$: T_i, $C(O, H, g^{T_i})$ $B_H \rightarrow B_O$: T_i, $C(O, H, g^{T_i})$ B_H: increase *H*'s balance B_O: decrease *O*'s balance **(*V* obtains its i^{th} payment)** *V*: compute $T'_i = HASH(HASH(d, R_i), v_V)$ $V \rightarrow B_V$: T'_i, $C(O, V, g^{T'_i})$ $B_V \rightarrow B_O$: T'_i, $C(O, V, g^{T'_i})$ B_V: increase *V*'s balance B_O: decrease *O*'s balance
Data retrieval	**(*O* retrieves *d* from *H*)** $H \rightarrow O$: *d* **(*H* unblocks its escrowed credits)** $O \rightarrow H \rightarrow B_H$: $\{ACK\}_{SK_O}$ If *TTL* times out: unspent escrowed credits are returned (respectively to *O* and *H*)

Figure 38 Payment protocol

Use cases

In this sub-section, we discuss the operation of the proposed payment storage-based scheme by reviewing several use cases.

Cooperative holder: A holder that has agreed to store data escrows credits from its bank-set that correspond to the punishment received in the case where it does not achieve its promise. Escrowed credits are converted to a digital check that is split into multiple shares and stored at the assigned verifiers. During data storage, the holder will receive digital checks in blinded version from the data owner certified by the bank-set of this latter. Each periodic verification results in revealing one digital check a time. The check will be then sent to the bank-set of the holder that contacts the owner's bank-set such that holder's balance increases with the amount of the agreed price and that of the owner decreases proportionally. At the time of data retrieval, the holder will send the data back to the owner, and receives in return an acknowledgement that when sent to the bank-set allows the holder to unlock its punishment escrowed credits.

Cooperative verifier: A verifier that accepts to periodically check the availability of data on behalf of the owner will receive first the holder's check shares corresponding to the punishment of the holder and then checks in blinded version from the owner corresponding to its payment. After each performed verification, the verifier receives the resolution of the blinded version of one check which enables it to increase its balance when the check is passed on to its bank-set. At the end of the data storage, if the holder returns the data to its owner, the verifier destroys holder's check share; otherwise it sends the check share to the owner in exchange for some small payment.

Selfish holder: A selfish holder destroys the data it has promised to keep. The (online) verifiers detect such selfishness and act accordingly by sending the holder's check shares to the owner. If at least a t threshold number of them do so, the owner is able to reconstruct the complete holder's check. The check will be cashed by the owner through its bank-set.

Blackmailing holder: The holder may decide not to send the data to the owner. The damage caused by such decision is mitigated thanks to the replication of the data at multiple holders. The blackmailer is not able to generate an acknowledgement to be sent to its bank-set, and then the check corresponding to its punition will be cashed by the owner. Indeed, the owner contacts the verifiers to receive the holder's check shares doomed to it.

Offline or selfish verifier: A verifier may be offline or just selfish and then neglect to perform the verification of its assigned holder's storage. Nevertheless, the holder is paid if at least a t threshold number of verifiers are honest and online. The number t should be then minimized in a way that takes into account the potential disconnection or selfishness of verifiers. But also, this number should be maximized to avoid possible collusions between a fraction of verifiers and the holder. Subsequently, the threshold t is a parameter that trades off the way of handling peers' churn and the way of mitigating their collusion. For a verifier that has been offline, it should (loosely) synchronize with the holder in order to send the right challenge index, whenever it reconnects to the system.

Selfish owner: If the owner does not send any acknowledgement message to the holder after correct reception of the data, the holder can request from verifiers to generate an acknowledgement message on behalf of the owner. The acknowledgement message is forwarded to the holder's bank-set to unlock the credits corresponding to the escrowed credits

for the punition. However, if there is litigation, i.e., the owner pretends not having received any data from the holder, then the verifiers may act as proxies for transmitting data between the holder and the owner (in exchange for small amount of payments): the holder splits data into chunks that will be conveyed to the owner through verifiers.

5.3.4. Simulation experiments

In order to validate the ability of our payment-based storage approach to detect and punish selfish peers, we developed a custom simulator of our payment scheme. This section first describes the framework of simulations, then presents and analyzes the results obtained.

Framework

The self-organizing storage system is modeled as a closed set of homogeneous peers. We consider the same simulation model as in the reputation-based approach. Newcomers arrive to the system in Poisson distribution at rate equal to 100 per hour and they stay in the system 2 weeks in average. Peers go online and offline in power-law distribution with average online period of 1 hour and average number of connections of 6.4 times per day. Peer storage space, file size, and storage duration are chosen from truncated log-normal distributions with average value equals respectively to 10GB, 500MB, and 1 week. There are 2.85 of files that are stored per peer and each one is verified each day.

User strategies: We consider three peer strategies: cooperation, passive selfishness (free-riding) and active selfishness.

- *Cooperative:* peers thoroughly follow the audit-based approach.
- *Passively selfish:* peers never contribute to the storage community or perform verifications.
- *Actively selfish:* peers are cooperative with respect to a given data with probability *p* (participation probability) and continue to be cooperative with respect to the very data with probability *q* (fully achieving promise probability). Otherwise, they are selfish.

Pricing: For the formulation of storage bid prices, we propose the following pricing function:

$$price(peer, t) = w \times \left(\frac{account(peer, t)}{account_0}\right) \times price_0 + (1 - w) \times price_0$$

Where *account*(*peer*, *t*) is the amount of credits held by the peer at time *t*, *w* is used for weighting the impact of the amount of credits owned by the peer, $price_0$ is the regular price for storage or verification, and $account_0$ is used for normalization.

Simulation results

Different scenarios were simulated to analyze the impact of several parameters on the payment mechanism. Simulation studies the transition phase of the network to a stable state where cooperative peers are the only active actors of the system.

Exclusion of selfish owners: Figure 39 demonstrates that selfish peers have less capability over time to store data in the system; on the other hand, cooperative peers are becoming the

majority of data owners in the storage system. Passive selfish peers are the first to be excluded from the system because they consume all their initial credits (all peers have a default number of credits when they join, in order to facilitate system bootstrap). Active selfish peers are also filtered out from the system because they cooperate only probabilistically.

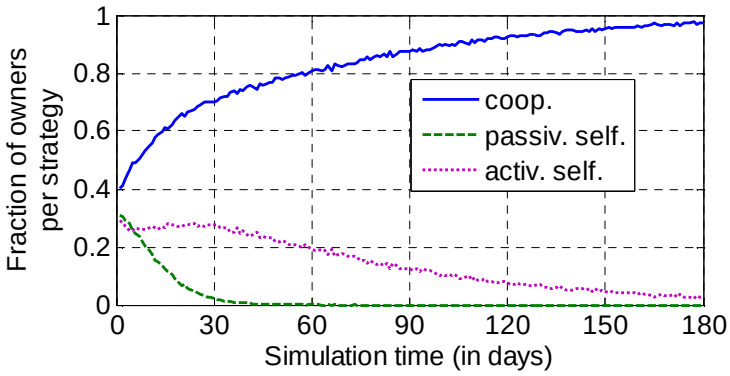


Figure 39 Averaged ratio of owners per strategy. *n*=1000, *r*=3, *m*=5, *w*=0.5, *p*=0.2, *q*=0.2, 40% cooperators, 30% passively selfish peers, 30% actively selfish peers.

The figure shows also that a decreasing fraction of these active selfish peers are still present in the system. Because they cooperate at some probability; they may temporarily gain some credits and then go without detection. These are considered as the false negatives of our detection scheme. But still, such false negatives are decreasing with time.

We may notice that 1 simulated month is sufficient to filter out passively selfish peers; however the filtering may take more than 3 months for actively selfish peers. Yet, this time period can be reduced by adaptively reducing the default initial income for newcomers.

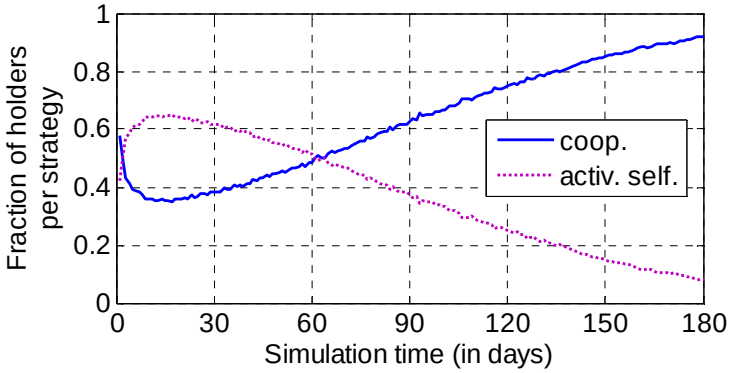


Figure 40 Averaged ratio of holders per strategy. *n*=1000, *r*=3, *m*=5, *w*=0.5, *p*=0.2, *q*=0.2, 40% cooperators, 30% passively selfish peers, 30% actively selfish peers.

Exclusion of selfish holders: Figure 40 depicts the fraction of cooperators and selfish peers in the population of data holders. The figure demonstrates that with time cooperative peers will make the majority of holders. This result is due to the fact that actively selfish peers are losing their credits and then becoming unable to escrow credits necessary for the storage of other peers' data; albeit the fact that they will propose small prices (this explains the small pick in the first simulated month).

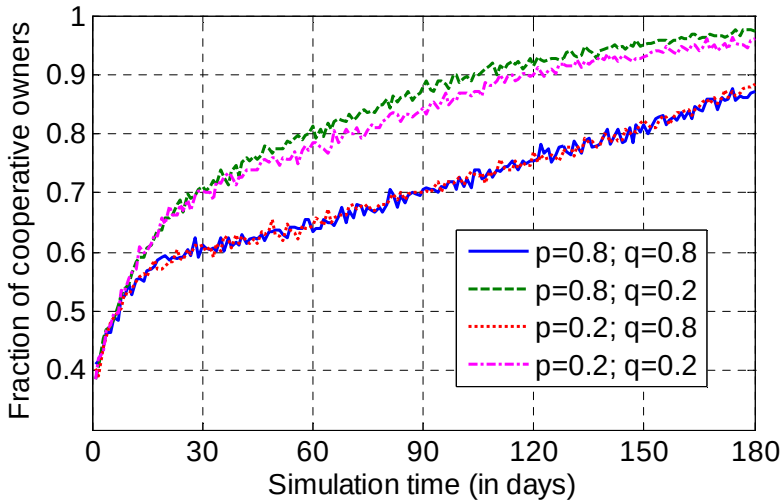
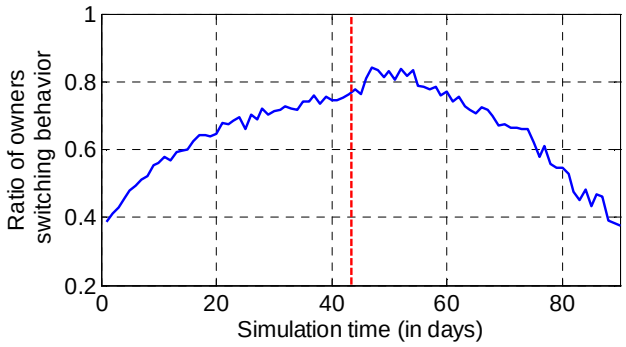


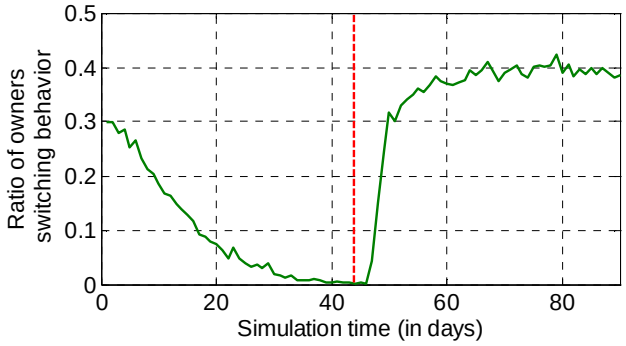
Figure 41 Averaged ratio of cooperative owners varying probability of participation *p* and probability of achieving promise *q* of actively selfish peers. *n*=1000, *r*=3, *m*=5, *w*=0.5, 40% cooperators, 30% passively selfish peers, 30% actively selfish peers.

Selfishness ratio: The penchant of actively selfish peers towards cooperation or selfishness is represented by the probabilities of participation *p* and achieving promise *q*: high probability *p* and *q* means that the actively selfish peer is cooperative most of the time. The probability *q* impacts more the convergence of the system to cooperative-only owners: for high *q* (=0.8), the system converges more quickly to 100% cooperative owners than for low *q* (=0.2), as illustrated in Figure 41. However, the probability of participation *p* has a less effect on the system because initially all actively selfish peers have enough credits to continue to be present in the system. Starting from the first month, the graph shows that there is a little increase in detecting actively selfish peers provided that they participate more (*q*=0.8).

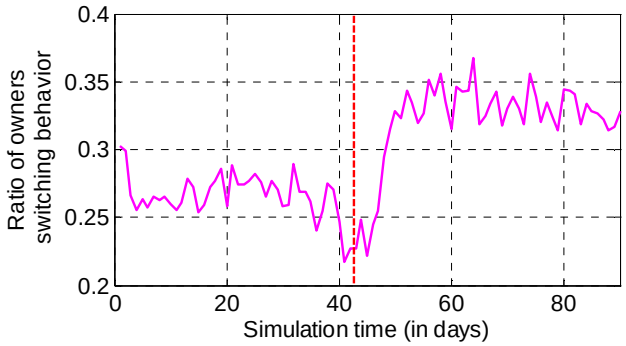
Dynamic strategies: If a peer changes its strategy from cooperation to selfishness, it is gradually deprived from storing its data as proven in Figure 42.a. On the other hand, selfish peers that change their strategies to cooperation, they are progressively permitted to store their data in the system (Figure 42.b and .c). Finally, these peers share the storage capability of the system with the rest of cooperative peers (at a ratio of 0.4). This result demonstrates the ability of the payment scheme to encourage peers to opt for cooperation instead of selfishness by working as a quota system that regulates the consumption of peers to their contribution.



(a)



(b)



(c)

Figure 42 Averaged ratio of owners that switch their strategy at time=45 days (marked by the red dashed line): (a) from cooperation to passive selfishness, or (b) from passive selfishness to cooperation, or (c) from active selfishness to cooperation. *n*=1000, *r*=3, *m*=5, *w*=0.5, *p*=0.2, *q*=0.2, 40% cooperators, 30% passively selfish, 30% actively selfish peers.

Overhead: Note that we only measure the communication overhead due to holder and verifier selection and storage verification. In particular, we exclude the cost of P2P overlay maintenance and storing/fetching of files, since it is not relevant to our analysis. In a further observation, the bandwidth consumed for verification is dependent on the number, rather than the size, of files being stored. This is in fact a requirement on the verification protocol. Figure 43 shows the amount of control messages per file. The figure demonstrates that the bandwidth cost decreases with time.

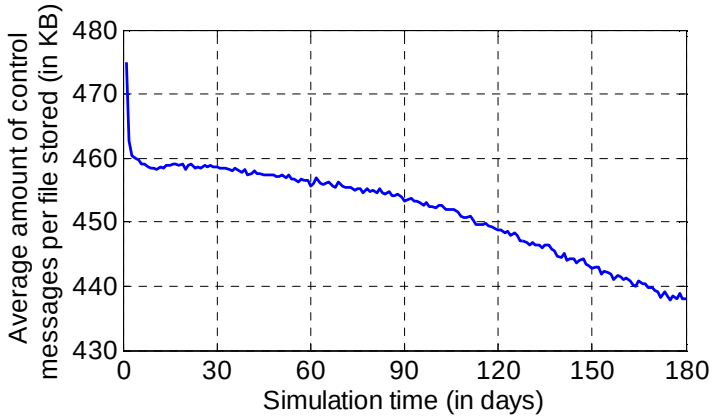


Figure 43 Average amount of control messages per file stored (in KB). *n*=1000, *r*=3, *m*=5, *w*=0.5, *p*=0.2, *q*=0.2, 40% cooperators, 30% passively selfish peers, 30% actively selfish peers.

Data reliability: Figure 44 shows that the rate of the amount of data injected into the storage system decreasing. This is due to several factors. First of all, there is the gradual exclusion of selfish peers that limits the number of peers able to store data in the system. Second, there are possible false positives of our detection system due to the starvation phenomenon where cooperative peers are not able to contribute because they are not chosen as holders or verifiers, and at the end they consume all their credits and get expelled from the system. The figure also depicts the rate of file loss that is falling down as low as zero, owing to the exclusion of selfish holders (explained earlier).

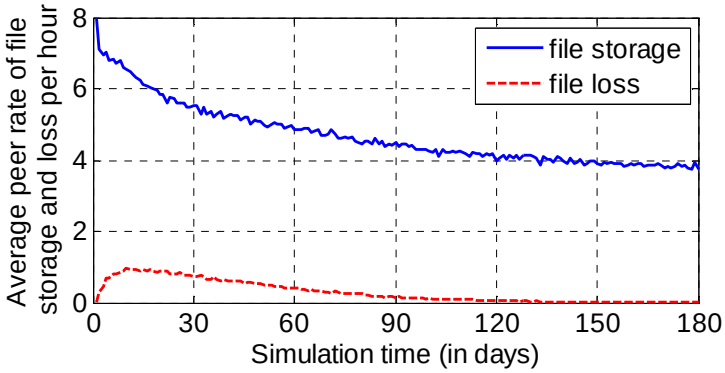


Figure 44 Average peer rate of file storage and loss per hour. *n*=1000, *r*=3, *m*=5, *w*=0.5, *p*=0.2, *q*=0.2, 40% cooperators, 30% passively selfish peers, 30% actively selfish peers.

Starvation: Figure 45 depicts the rate of file injection with time varying the value of the weight *w* in the price function. The figure shows that if the price does not take into account the amount of credits possessed by peers ($w = 0$), file storage rate decreases due to the phenomenon of starvation where peers are not able to store data due to a lack of credits. However, if the price is based on the factor of possessed credits ($w \neq 1$), the rate decreases at first then becomes stable for the rest of the time. So the consideration of the pricing function allows handling the starvation problem. Auctioning for holder and verifier selection is then helpful for starving peers.

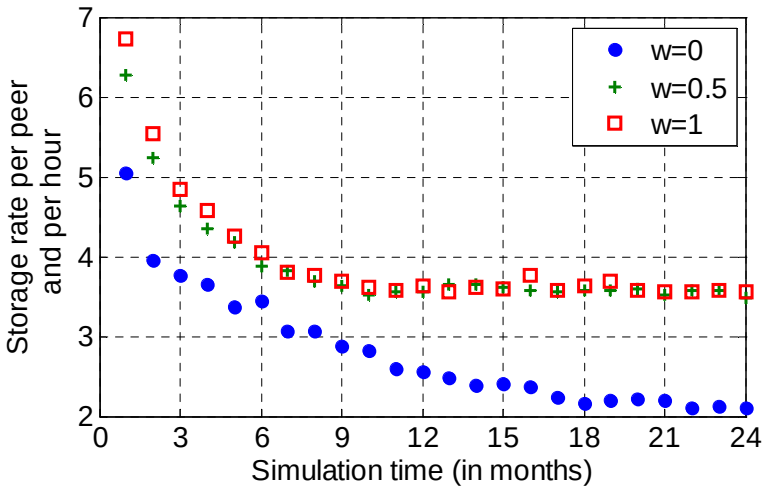


Figure 45 Averaged amount of data stored in the system varying the weight *w*. *n*=1000, *r*=3, *m*=5, *p*=0.2, *q*=0.2, 40% cooperators, 30% passively selfish peers, 30% actively selfish peers.

5.3.5. Security considerations

In this sub-section, we analyze the security of the protocol to prevent or at least mitigate the threats described in sub-section 5.3.1.

The security of our scheme relies principally on replication to deter peers that might try to subvert the protocol. It assumes that there are at least a given number of peers in the system at all times, and uses protocols to ensure that the system will correctly operate unless a substantial fraction of these peers are selfish or malicious.

Selfishness punishment: The proposed payment scheme works as a quota system: peers have to keep a given balance to be able to participate to the storage system. Peers that are passively selfish gradually consume all their credits for their data storage and when their accounts are exhausted they will not be able to use the storage system anymore. In the same way, actively selfish peers keep losing credits because they have been detected destroying data they have promised to store. These peers will also drain their accounts and with time will not able to use the storage system.

Collusion prevention: Holder and verifier selection is random which limits preset peer collusions. The digital check of the holder is shared among verifiers, thus mitigating also collusion between the owner and one or a small number of verifiers. Additionally, challenges sent to the holder are constructed cooperatively by verifiers to avoid collusion between the holder and one or a small number of verifiers. Finally, collusion between the owner and the holder is less probable because it does not generate any financial profit since the owner must pay verifiers to check holder's storage. The distribution of tasks to several verifiers limits collusion; but it is still feasible if at least *k* verifiers collude with the holder for instance. The probability of collusion can be computed as:

$$\sum_{i=k}^{n_v} \binom{n_v}{i} p^i (1-p)^{n_v - i}$$

where p is the probability that a given verifier is not honest (colluder). However such collusion probability is less than 0.1 for 60% of dishonest peers (i.e., p=0.6) in the system and with n_v=10 and k=8, or 80% of dishonest peers and with n_v=20 and k=18.

Ensuring fair-exchange: Holder and verifier payments are strongly related to the correct operation of data verification. This motivates holder to accept that its storage being verified and incites verifiers to perform this task periodically on behalf of the owner. The frequency of verifications is determined by the owner at the time of delegating verification. This frequency is the matter of all verifiers: the majority of verifiers use the determined frequency at which the holder collects a sufficient number of random numbers to compute the challenge; therefore very fast or very slow frequencies of some verifiers do not influence (with large probability) the actual frequency of computing the verification challenge. The holder or the verifier cannot cash their checks without verifying the stored data. This is due to two reasons. First of all, the secret random numbers included in the checks are only known to the actual payers since they are held in DLP-based blinded version at the payees and bank-sets too. Second, *HASH* is a one-way function, so knowing *HASH*(*m*) does not give any extra information on *m*. Therefore, the data verification operation strongly relates to the holder and the verifier payment operations. However, the existence of such relation is only guaranteed by the owner. So, if a verifier or a holder is still not paid even though it behaves well, it has the possibility to prove owner's misbehavior to the other participants (using the certified checks) and also to stop cooperating with the owner without being punished. Thus, the owner is encouraged to provide this type of relation to secure the future cooperation of peers handling its data. The bank-set comes into play to guarantee that payments are actually doable since the corresponding amounts of credits are locked to prohibit the payer from emitting bad checks.

Preventing remuneration-related attacks: Attacks on the payment scheme (such as double spending or impersonation) are handled by the KARMA framework. Moreover, the sequence number and the identity of the payee included in each payment receipt prevent replay attacks, because they impose that the digital check is only cashed by the payee one time. We assume that all exchanged messages are signed and encrypted by the keys of the involved parties in order to ensure the integrity of exchanged messages and even the security against man-in-the-middle attack for instance. Sybil attacks are mitigated a la KARMA by compelling peers to execute a cryptographic puzzle before joining the storage system, the result of which will be used to construct their identities.

5.4. Discussion

The proposed cooperation incentive mechanisms: reputation-based and remuneration-based suggest two distinct solutions to the P2P storage problem (Table 5 shows a number of dissimilarities between both approaches).

In the reputation-based approach, the periodic results of data holders' evaluation obtained by verifiers and data owners serve in computing the reputation value of these holders. Such results are not disseminated in the network to other peers, thus conferring to the approach a locality property. Therefore, the approach operates better in a group-based architecture where peers are only concerned with the reputation of their group members. In a group of modest size, peers are able to be acquainted about the behavior of other peers inside the group and then act accordingly.

On the other hand, the remuneration-based approach does not require dissemination of the verification information, it rather uses such information to decide in a self-organizing way if the verified holder deserves being rewarded or punished with financial incentives. Moreover, the security of the remuneration-based approach supposes that there is at all times a fraction of peers in the network that are honest in order to correctly function. The selfishness or

maliciousness of peers is mitigated by distributing the critical functions of the incentive mechanism to multiple peers in the same way as a Byzantine failure model. For that reason, large population of peers better suits the remuneration-based approach than a small or a group-based peer population.

Table 5 Comparison between the proposed reputation-based and remuneration-based approaches

	Reputation	**Remuneration**
Data resilience	+ random but uniform holder and verifier selection + no starvation of storage sites	+ random holder and verifier selection complemented by an auction ++ starvation of malicious or selfish storage sites
Cooperation stimulation	++ storage incentive: good reputation enables storage + verification incentive: accuracy of reputation estimation - Local view of peer behavior + limited collusions + Sybil attack mitigated by controlling entry to the group	++ storage and verification incentive: remuneration + global view of peer behavior + limited collusions + Sybil attack mitigated using a cryptographic puzzle
Architecture	- group formation (e.g., social networks)	- - requirement of a substantial fraction of honest peers

Reputation-based and remuneration-based approaches may be combined into one cooperation incentive mechanism to achieve a twofold objective: protecting the system from malicious peers and inciting the cooperation of the other peers. In this way, peers use the P2P storage system since they trust and rely on peers that are well-reputed. They are also motivated to contribute to the system thanks to financial rewards they gain from their cooperation. The security of the storage system that relies on such cooperation incentive mechanism is guaranteed. Indeed, selfish peers are generally considered as rational and therefore they prefer to obtain a compensation for their contributions instead of not cooperating. On the other hand, irrational peers that are malicious are detected and then acquire bad reputation leading to their gradual eviction from the system. Finally, irrational peers that are rather altruistic attain a high reputation and are then admitted to stay in the system. Additionally, we suggest that verifiers should receive a reputation value as holders. This is not proposed in the described audit-based reputation approach since only holders have a reputation value and verifiers are incited to perform their work in order to estimate that very reputation. It is possible that holders and verifiers are ranked with reputation. However reputation for storage and verification services should be decoupled to avoid free-riding peers that offer verification assistance to the data stored in the system as an alternative to utilizing their own storage resources. Therefore, peers should have two values for reputation: the first one concerns their aptitude to store and to preserve data of other peers, and the second one points out to their contributions in checking the presence and integrity of such data. However, we do not require distinguishing storage from verification services in the remuneration mechanism because generally payment are neutral and may even allow a multi-service framework as discussed in Section 2.4.2 of chapter 2.

5.5. Summary

This chapter described two new cooperation incentive mechanisms that are well suited for P2P storage applications. These approaches, whether they rely on reputation estimates or payments, allow a fast isolation of selfish peers, and prevent several further malicious behaviors that go beyond the absence of contribution to the system to data destruction or even malicious peer collusion to render data verification inoperative. Remuneration incentives have been shown

to be effective as cooperation incentives for P2P data backup [Toka and Maillé 2007] although their resistance to attacks beyond selfishness has not been really studied. The proposed cooperation incentive mechanisms are not only able to detect non cooperating peers in the storage system and to punish them, but they also aim at mitigating several potential attacks that may cause the system to fail in providing a reliable and fair storage for all peers. The choice of one mechanism over the other depends on the organization of peers inside the storage system. The first mechanism presented is based on local reputation that is appropriate for peers clustered in groups (like social networks). The second mechanism instead better fits larger networks.

The next chapter further and more formally investigates the capabilities of the proposed mechanisms as cooperation incentives through the use of game theoretical models.

5.6. Relevant publication

1. Nouha Oualha and Yves Roudier. Reputation and Audits for Self-Organizing Storage. In the 1st Workshop on Security in Opportunistic and SOCial Networks (SOSOC 2008), Istanbul, Turkey, September 2008.

2. Nouha Oualha and Yves Roudier. Designing Attack Resilient Cooperation Incentives for Self-Organizing Storage. 1er Workshop sur la sécurité des réseaux autonomes et spontanés organisé, Loctudy, France, 13-14 October, 2008.

3. Roberto Baldoni, Miguel P. Correia, Ludovic Courtes, Felicita Di Giandomenico, Fabrizio Grandoni, Marc-Olivier Killijian, Nuno Ferreira Neves, Nouha Oualha, Thea Peacock, David Powell, Adriano Rippa, Yves Roudier, Michel Raynal, Peter Ryan, Marco Serafini, Neeraj Suri, Sara Tucci Piergiovanni, Paulo Veríssimo. Resilience-Building Technologies: State of Knowledge: Part Algo – Resilience Algorithms and Mechanisms. ReSIST NoE: Resilience for Survivability in IST, Deliverable D12, September 2006.

Chapter 6

Evaluating cooperation incentives using game theory

Cooperation incentives prevent selfish behaviors whereby peers free-ride the storage system, that is, they store data onto other peers without contributing to the storage infrastructure. Remote data verification protocols are required to implement the auditing mechanism needed by any efficient cooperation incentive mechanism. In general, a cooperation incentive mechanism is proven to be effective if it is demonstrated that any rational peer will always choose to cooperate whenever it interacts with another cooperative peer. One-stage games or repeated games have been mostly used to validate cooperation incentives that describe individual strategies; in addition, the use of evolutionary dynamics can help describe the evolution of strategies within large populations.

This chapter proposes two theoretic game models of a P2P storage system that we use to show under which conditions an audit-based strategy wins over self-interested strategies. The contribution of this chapter is the validation of the security primitive particularly the probabilistic and the deterministic verification protocols, with respect to its cooperation enforcement function for data storage.

6.1. Preliminaries

Game theory offers valuable tools for the validation of cooperation incentive mechanisms as the study of selfish behavior and incentive measures strongly relate to rationality and decision making. Consequently, it has been used in several works that try to provide means to prevent selfishness and to enforce cooperation among self-interested individuals. In the following, essential definitions about game theory are first introduced, and then some approaches inciting resource sharing and applying game theoretical models that we deem to be interesting are reviewed.

6.1.1. Definitions

Game theory is a branch of applied mathematics that models interactions among individuals making decisions. It attempts to mathematically capture individual rational behavior in strategic situations where individuals' decisions are based on their preferences and also depend on the other individuals' choices. It then provides a language to describe, analyze, and understand strategic scenarios [Turocy and Stengel 2001].

Game: A game consists of:

- A set of *players* $\{p_1, \ldots, p_n\}$ which are the individuals who make decisions
- A set of *strategies* i.e., moves for each player S_i, i=1, …, n
- A specification of each player's *payoffs* which are the numeric values assigned to the outcomes produced by the various combinations of strategies. Payoffs represent the

preference ordering of players over the outcomes. Payoffs are expressed using player's *utility function* U_i:

$$U_i: S_1 \times S_2 \times \ldots \times S_n \rightarrow \Re$$

The game assumes that all players are *rational*; this means that they will always choose the strategy that maximizes their payoffs. Players are then participants in the game with the goal of choosing the actions that produce their most preferred outcomes.

Game types: A game can be one of two types: *non-cooperative* or *cooperative*. In the first type, players are selfish and are only concerned with maximizing their own benefit. In the second type, some players cooperate and form a coalition in order to achieve a common goal, and then the coalition and the rest of players play non-cooperatively the game. A game can be a *repeated game* that consists in a finitely or infinitely number of iterations of some one-stage game. In such one-stage game, players' choices are referred to as actions rather than strategies (term reserved to the repeated game) and these actions take into account their impact on the future actions of other players. *Evolutionary game* theory provides also a dynamic framework for analyzing repeated interactions. In such games, randomly chosen players interact with each other, and then the player with the lower payoff switches to the strategy of the player with the higher payoff i.e., players reproduce proportionally to their payoffs. Hence, strategies with poor payoffs eventually die off, while well-performing strategies thrive.

Game equilibria: Finding a solution of a game is trying to find equilibria in the game. In equilibrium, each player of the game has adopted a strategy that they are unlikely to change. Many equilibrium concepts have been developed in an attempt to capture this idea. The most famously one is the *Nash equilibrium*. Nash Equilibrium is the set of players' strategy choices where no player can benefit by changing its strategy while the other players keep their strategies unchanged. So, it is a set of strategies $\{\sigma_1 \in S_1, \ldots, \sigma_n \in S_n\}$, such that:

$$U_i(\sigma_1, \ldots, \sigma_i, \ldots, \sigma_n) \geq U_i(\sigma_1, \ldots, \sigma'_i, \ldots, \sigma_n), \ \forall\, i \in \{1, \ldots, n\} \text{ and } \sigma'_i \in S_i$$

Evolutionary stable strategy (ESS) is a refined version of the Nash equilibrium which captures the idea that a strategy that is a Nash equilibrium, if adopted by a population of players, cannot be invaded by any alternative strategy that is initially rare. For a two-player game with a strategy space S, a strategy $\sigma*$ is an ESS if and only if for any $\sigma' \neq \sigma*$, either one of the following two conditions holds:

a) $U(\sigma*, \sigma*) > U(\sigma', \sigma*)$
b) $U(\sigma*, \sigma*) = U(\sigma', \sigma*)$ and $U(\sigma*, \sigma') > U(\sigma', \sigma')$

Here, $U(.,.)$ is the payoff function of the associated two-player game.

6.1.2. Related work

To achieve a socially optimal equilibrium for a self-organizing system with autonomous peers, different incentive mechanisms have been proposed in the literature. These incentives include providing virtual or real payment incentives or establishing and maintaining a reputation index for every peer in the network.

Payment incentive modeling

One of the first studies that considered payment schemes in P2P systems is [Golle et al. 2001], which uses a game theoretical model to study the potential benefits of introducing micro-

payment methods into centralized P2P file-sharing systems such as Napster. In such type of systems, in order to catch the asymmetric aspect of interactions between peers, called agents, the strategies have two independent actions: sharing i.e. providing the service, and downloading i.e. acquiring the service. Without considering any incentives as it is the case with Napster, the outcome of the equilibrium analysis results in an unique equilibrium with nothing is shared and nothing can be downloaded. Even with some level of altruism in the system, all agents, either altruistic or free-rider, are not restrained from downloading and the whole cost then weights over the small number of altruistic agents. Therefore, the authors propose alternatives based on payment to overcome the free-riding problem. The first proposed payment scheme consists in charging agents for every download and rewarding them for every upload. The result of the equilibrium analysis shows that there is one unique and strict equilibrium where agents are extensively sharing and downloading files. This result validates the payment scheme; but still, the analysis does not take into account the fact that agents share diverse files and some of them may store files that are sufficiently rare thus unfairly receiving a large fraction of all the download requests for these files. For that reason, the authors propose a second payment-based alternative that continues to penalize downloads, but rewards agents in proportion to the amount of material they share rather than the number of uploads they provide. The equilibrium analysis demonstrates that two strict equilibriums may be reached either full file sharing or no sharing at all; though simulation experiments of the model demonstrate that the system converges to an equilibrium where all agents cooperate by sharing files.

[Toka and Maillé 2007] in the DisPairSe project took a different direction for defining peer utility function that becomes more centered on payment than the model of [Golle et al. 2001]. Actually, the authors of [Toka and Maillé 2007] modeled a P2P backup service as a non-cooperative game using an economic model. The parameters characterizing the profile of each user and, associated with the demand and supply functions turned to be playing a crucial role on justifying the use of a pricing scheme or imposed symmetry with respect to the optimal situation of the service that is maximizing the social welfare. Indeed, the theoretical study of the economic model shows that if users are homogeneous in terms of these parameters, then it is better to opt for imposed symmetric user contributions rather than a pricing scheme. However, heterogeneous user population, which is the general case of P2P networked peers, validates the use of a pricing scheme by which a monopoly is introduced to fix unit prices for buying and selling storage resources. The mathematical study presented by the authors attempts to defend the intrusion of the operator in fixing prices for a P2P backup system; albeit the fact that a profit-oriented intrusion whereby the operator strives to extract the maximum profit out of the business, reduces the social welfare of the system by ¼ times its maximum.

Reputation incentive modeling

Reputation schemes have received a great deal of intention for enforcing node cooperation in mobile ad hoc networks. Notably, [Michiardi 2004] proposed CORE as a collaborative reputation mechanism motivating nodes to forward packets, and used a game theoretical approach to assess the features and validate the mechanism. The author relied on a cooperative game that uses a two-period structure: players first decide whether or not to join a coalition, and then both the coalition and the remaining players choose their behavior non-cooperatively. Additionally, the model employs a preferential structure as suggested by the ERC-theory [Bolton and Ockenfels 2000]; even though the use of such theory for ad hoc networks is not argued in more detail by the author. The study of the model demonstrates that there is a Nash equilibrium where at least half of the total number of nodes cooperate. The authors also considered the case where nodes may have a continuous strategy space where they may choose their cooperation levels instead of discretely choosing just between cooperation and defection. If nodes have identical ERC preferences and are interested enough in being close to the equal share, the study reveals that the grand coalition is stable i.e., no player has an incentive to leave

the coalition. Such interesting analytical result is not very well defended by the author as a validation of the proposed mechanism since the assumption that the nodes will be very much interested in achieving equality of shares may not be met in practice.

Rather than to be expressly conceived for a specific mechanism, [Zhao et al. 2009] proposed a general and generic game theoretical framework to model and analyze cooperation incentive policies. The model studies, instead of game equilibriums, the game dynamics where strategies change according to two learning models: the current-best (CBLM) and the opportunistic (OLM) learning models. In CBLM, each peer chooses the strategy that has the highest payoff. In the second learning model OLM, each peer randomly chooses another peer as its *teacher*. If the teacher has a better payoff than the peer, the latter adapts to the teacher's strategy. OLM is similar to evolutionary game concepts where the so-called teacher is the co-player of the peer. The main parameter of comparison between these learning models is robustness: a system is robust if it stays at a high contribution level even with perturbation such as peer arrivals or departures from the network. The mathematical analysis demonstrates that a system with CBLM is less robust than with OLM; this latter being alike a typical evolutionary game model. Moreover, the analysis allows comparing two incentive policies: the mirror incentive policy under which a peer provides service with the same probability as the requester serves other peers in the system, and the proportional incentive policy whereby the peer serves the requester with a probability equal to the requester's contribution to consumption ratio. The study shows that the mirror incentive policy may lead to a complete system collapse, while the proportional incentive policy can lead to a robust system. This result demonstrates that a policy motivating fairness in terms of contributions and consumptions of resources achieves better stability than participatory incentives.

[Lai et al. 2003] opted also for an evolutionary study of applications in P2P systems. The authors proposed a model that they called a generalized form of the Evolutionary Prisoner's Dilemma (EPD). Though the model is very similar to the traditional EPD, they argued that the new model permits asymmetric transactions between a client peer and a server peer. Peers decide to cooperate or not based on a reciprocative decision function that sets the probability to cooperate with a given peer X to the ratio, rounded to a value in [0, 1], (cooperation X gave)/(cooperation X received), such function is comparable to the proportional incentive policy in [Zhao et al. 2009]. The authors simulated EPD under various situations and obtained several results. They showed that techniques relying only on private history, where solely peer experiences are taken into account, fail in inciting cooperation among peers as the population size increases. However, techniques based on shared history better scales to large populations. Additionally, results demonstrate that cooperation with strangers fails to encourage cooperation in the presence of whitewashers. Therefore, the authors proposed an adaptive policy in which the probability of cooperation with strangers becomes equal at time t+1 to $p_C^{t+1} = (1-\mu)\times p_C^{t} + \mu\times C_t$, where C_t=1 if the last stranger cooperated and =0 otherwise. Simulations validate the adaptive policy by demonstrating that incentives based on such policy make the system converge to higher levels of cooperation.

[Feldman et al. 2006] have studied in more depth the whitewashing problem in P2P systems using a game theoretical model that particularly takes into account heterogeneity of users' behavior. In order to sustain the system when the societal generosity is low, punishment mechanisms against free-riding users are required. The proposed punishment mechanism consists on imposing a penalty on free-riding behavior with probability (1-p). The optimal value for the probability p is defined by the maximum obtained performance of the system. Still, such mechanism can be undermined by the availability of cheap pseudonyms through which a free-rider may choose to whitewash. To measure the effect of whitewashing behavior, the authors computed system performance considering the cases of permanent identities and free identities, in addition to different turnover rates that represent user arrival and departure rates (arrivals and

departures are assumed *type-neutral* i.e., they do not alter the type distribution). Their study demonstrates that the penalty mechanism is effective when both the societal generosity and the turnover rate are low; otherwise a notable societal cost due to whitewashing is experienced (we will also come to such result in our study of an evolutionary game model of the audit-based incentives).

In the remainder of this chapter we will present several game theoretical models describing various features of our audit-based incentive mechanisms. We endeavor with such models to validate our mechanism as an influencing force exhorting rational peers to behave in a way that maximizes the common good of the P2P storage system.

6.2. Repeated signaling game of payment-based incentives

In this section, we model the P2P storage system as a game. For the sake of game symmetry, we assume the presence of just two players: the data holder and the data owner verifying the holder. These players are involved in the strategic process of deciding whether to cooperate or not on one hand, and to punish or reward on the other hand.

Although the considered games (that will be described in the following) model a payment-based incentive mechanism, the assumption of reputation incentives may also be sustained with such games given that the reward is the positive reputation gained by the holder and the punishment is a negative reputation; however the considered models imply also that the reward gained by the holder is deduced from the owner's outcome and the punishment inflicted to the selfish holder is reimbursed to the owner. So, the presented models are more adequate for reputation mechanisms that are based on a quota system.

Our game models show how incentives can be built based on the regular verification of the correct storage of data, as promised by holders. Cooperation incentives are expressed as payments: the holder is rewarded for a correct response while it is charged when responding incorrectly. The outcome of this modeling is the validation of the existence of cooperation equilibria after a series of verifications, and the evaluation of the parameters to be taken into account to design proper payment-based incentives. Two games are introduced that respectively model the holder's strategy and the owner's strategy.

6.2.1. Game elements

The essential elements of our game model are:

- *Players:* data owner denoted *O* and data holder denoted *H*.
- *Payoffs:* Payoffs represent the preference ordering of players over game outcomes.
- *Information:* information set for a player summarizes what the player knows when it gets to make a decision.
- *Chance:* probability distribution over chance events. We represent chance events by a random move of nature which is a pseudo-player whose actions are purely mechanical and probabilistic.

6.2.2. Game models

The P2P storage is modeled as a Bayesian game. In such a game, information about the characteristics of other players is incomplete, and nature is introduced as a player for modeling uncertainty.

Figure 46 illustrates the structure of our one-stage game in the extensive form (in the form of a tree where there is a complete description of how the game is played over time). A one-stage

game corresponds to the phase of one challenge conducted by *O* towards *H*. Notations used in figures are explained in Table 6. The parameters *G*, *R*, *R'* and *D*, in Table 6, are measured in the same units, e.g., the number of data bytes or data chunks stored. Also regarding data stored in a distributed fashion, we presume that the remote storage space has more value than local storage space, which explains that *G>R>D*.

Table 6 Notations

Notations		Explication
Players	O	data owner
	H	data holder
Errors	M	malfunction of H
	N	normal function of H
Types	C	H is cooperative
	S	H is selfish
	F	H is faulty
Signals	s	succeed O's challenge
	f	fail O's challenge
Actions	rw	reward H
	fg	do not do anything
	pn	punish H
Payoffs	G	distributed storage gained by O
	D	supplementary storage provided by H
	R	reward charge, such that R>S>0
	R'	punishment charge, such that G-R>R'>0
Chance	q	probability of challenge's success for a selfish holder H
	d	Probability of hardware failure (for H)

The game (depicted in Figure 46) models the fact that the holder *H* may follow two possible strategies, or in game theoretical terms, be of two types: cooperative, that is, it will store owner's data until its retrieval; or selfish, that is, it will destroy data chunks with probability 1-*q*.

These types are respectively referred as "C" and "S". If *H* chooses the type "C", it succeeds in answering a challenge requested by *O* as modeled by the emission of signal "s". However, it may fail because of a hardware crash or error for instance, which occurs with probability *d*, and is modeled by the emission of signal "f". The failure to answering a challenge is either an incorrect response to the challenge or, more frequently, no response at all (after some time-out). If *H* chooses type "S", we assume that it may successfully answer a challenge only with a probability equal to *q*(1-*d*). Otherwise, it will behave like a faulty peer. In addition, real faults may still happen with probability *d*.

This probability *q* of a correct answer from a selfish holder may be due to several reasons. The selfish holder may restrain from answering the owner pretending to be offline, and then the probability *q* is the probability that the owner is fooled with such "no response" (we assume that a holder of type "C" is always available for verifications). Moreover, the used verification protocol can be probabilistic such that the holder may destroy a portion of the data and still be able to answer verification relating to the remainder of the data. The verifier may be also another third peer that may neglect to perform the verification task, and then *q* is the probability that the verifier does not check the remote data that have been actually destroyed by the holder.

The owner *O* is not informed about *H*'s type, which is why *O* cannot distinguish between "C" and "S" despite the fact that *H*'s signal is seen by *O*. Such situations that cannot be discriminated belong to the same so-called "set of information". The two sets of information I and II depicted in the game diagram correspond respectively to success and failure signals.

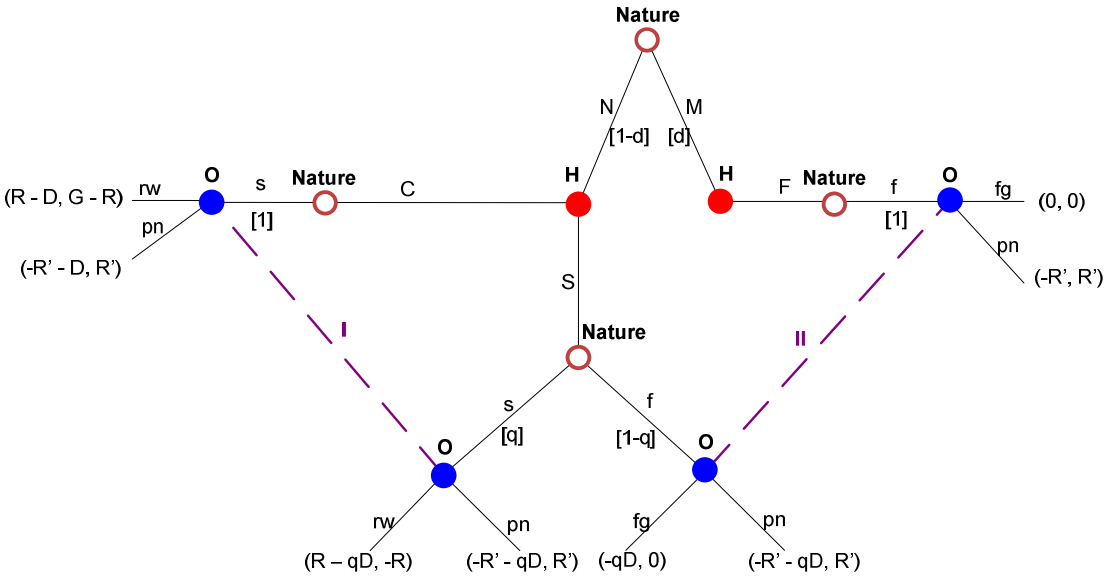


Figure 46 Modeling the holder strategy

In this section, we will consider a simplified version of the game of Figure 46, in which the risk of hardware failure for *H* is simply neglected (*d*=0). This simplification allows easier computations in the next sections, while focusing on holder strategies.

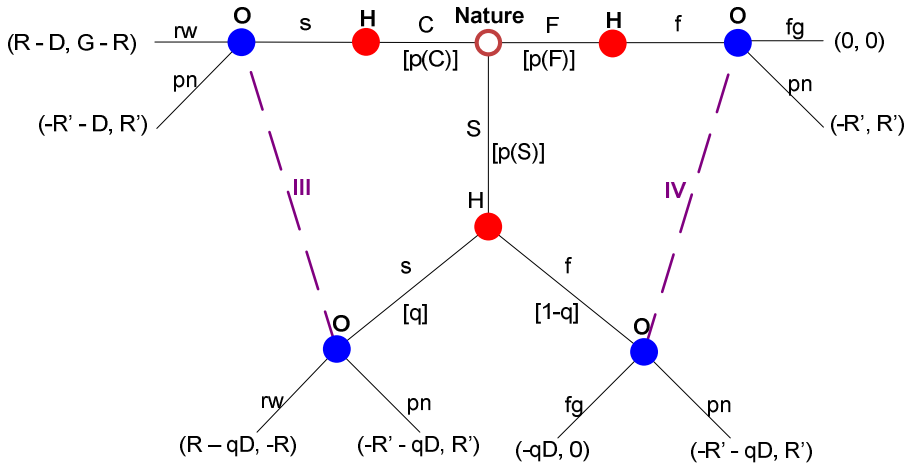


Figure 47 Modeling the owner strategy

The game model of Figure 46 is a sequential game with asymmetric distribution of information, since the holder *H* is informed about its type, but the owner *O* is not informed. However, *O* can probabilistically determine *H*'s type based on its prior beliefs, such beliefs typically reflecting *H*'s reputation. With every verification performed, *O* updates its beliefs according to Bayes' formula. To describe *O*'s prior beliefs about *H*'s type, we derive a second game model depicted in Figure 47. This model is a typical signaling game, that is, players have asymmetric information. The game is modeling the owner strategy: the game will use signals

based on *H*'s type as determined by the Nature. *H*, the informed player, has different types given by nature; while *H* knows its type, *O* does not. Based on the knowledge of its own type, *H* sends signals which *O* can observe but which do not provide perfect information about *H*'s type. In our model for instance, the set of information III may describe a cooperative or selfish *H*, and the set IV may describe a selfish or faulty *H*.

6.2.3. Equilibria

The solution of the game, which constitutes player's best response to the actions of the other player, is called an equilibrium. The following sections define the Nash equilibrium and the perfect Bayesian equilibrium of the game.

Nash Equilibrium: To define the Nash equilibrium of the game, the normal form of the game of Figure 46 (which lists each player's strategies and the payoffs that result from each possible combination of choices) is presented below in Table 7.

Table 7 Normal form of the game of Figure 46

		O's payoff	
		rw	**pn**
H's payoff	**C**	(*R - D, G-R*)	(*-R' - D, R'*)
	S	(*R - qD, -R*)	(*-R' - qD, R'*)

We assume that *G-R > R'*. If *H* chooses the type "C", then *O*, by strict dominance, chooses the action "rw" because the payoff associated to "rw" (= *G - R*) is higher than the payoff associated to "pn" (= *R'*). By choosing "rw", the better response by *H* is "S" because *R – D < R - qD*, and so, *O* will prefer to choose "pn" because *R' > 0 > -R*. At this point, neither *O* or *H* can have a benefit by changing to another strategy. So, ("S", "pn") is a Nash equilibrium. The normal form game leads to an equilibrium where non-cooperation is the best response for players.

Compared to the extensive form game, the normal form game lacks the information on whether *O* is informed or not about the type of *H*. The view of incomplete information is not represented within the normal form. Another equilibrium, the perfect Bayesian equilibrium, takes into account this view.

Perfect Bayesian Equilibrium: A perfect Bayesian equilibrium is a strategy profile σ^* = (σ_1^*, σ_2^*) and posterior beliefs $\mu(\cdot \mid m)$ such that:

1) $\forall$ type *t*, $\sigma_1^* \in argmax_{\sigma_1}(U_1(\sigma_1,\ \sigma_2^*,\ t))$
2) $\forall$ signal *m*, $\sigma_2^* \in argmax_{\sigma_2}(\sum_t \mu(t|m) U_2(m,\ \sigma_2,\ t))$
3) $\mu(t|m) = \frac{p(t)\sigma_1^*(m|t)}{\sum_{t\prime} p(t\prime)\sigma_1^*(m|t\prime)}$

Finding the perfect Bayesian Equilibrium of the game means finding the following probabilities ([Ghassemi 2006]):

$$\sigma_1^*(s|C) = 1 \qquad \sigma_1^*(f|C) = 0$$
$$\sigma_1^*(s|S) = q \qquad \sigma_1^*(f|S) = 1 - q$$
$$\sigma_1^*(s|F) = 0 \qquad \sigma_1^*(f|F) = 1$$
$$\sigma_2^*(rw|s) = u_1 \qquad \sigma_2^*(fg|s) = v_1 = 0 \qquad \sigma_2^*(pn|s) = w_1 = 1 - u_1$$
$$\sigma_2^*(rw|f) = u_2 = 0 \qquad \sigma_2^*(fg|f) = v_2 \qquad \sigma_2^*(pn|f) = w_2 = 1 - v_2$$

Thus, the belief update equations are as follows:

$$\mu(C|s) = \frac{p(C)}{p(C) + p(S)q} \qquad \mu(S|s) = \frac{p(S)q}{p(C) + p(S)q} \qquad \mu(F|s) = 0$$
$$\mu(C|f) = 0 \qquad \mu(S|f) = \frac{p(S)(1-q)}{p(S)(1-q) + p(F)} \qquad \mu(F|f) = \frac{p(F)}{p(S)(1-q) + p(F)}$$

H's payoffs corresponding to each type is given by:

$$U_1(\sigma_1, \sigma_2^*, C) = u_1(R + R') - R' - D$$
$$U_1(\sigma_1, \sigma_2^*, S) = q(u_1(R + R') + R'w_2 - R' - D) - R'w_2$$
$$U_1(\sigma_1, \sigma_2^*, F) = -R'w_2$$

Expected *O*'s payoffs for each signal sent by *H* is given by:

$$\sum_t \mu(t|s) U_2(s, \sigma_2, t) = u_1\left(G\frac{p(C)}{p(C) + p(S)q} - R - R'\right) + R'$$
$$\sum_t \mu(t|f) U_2(f, \sigma_2, t) = R'w_2$$

Finding the solution of the game depends on the sign of $\left(G\frac{p(C)}{p(C)+p(S)q} - R - R'\right)$ that corresponds to whether the following inequality holds or not:

$$p(C) \geq \frac{q(R+R')}{G-(1-q)(R+R')} \tag{6.2.3}$$

There are two case solutions:

- *Case 1:* if $p(C) \geq \frac{R+R'}{G}$, then σ_2* is maximized for u_1=1 and w_2=1. Because $R + R' - D > 0$, σ_1* is maximized for q = 1. The perfect Bayesian equilibrium is the strategy where:

$$\sigma_1^*(s|S) = 1 \qquad \sigma_1^*(f|S) = 0$$
$$\sigma_2^*(rw|s) = 1 \qquad \sigma_2^*(fg|s) = 0 \qquad \sigma_2^*(pn|s) = 0$$
$$\sigma_2^*(rw|f) = 0 \qquad \sigma_2^*(fg|f) = 0 \qquad \sigma_2^*(pn|f) = 1$$
$$\frac{p(S)}{p(C)} \leq \frac{G - R - R'}{R + R'}$$

  The equilibrium of the game leads to a strategy where *O* and *H* cooperate.

- *Case 2:* if $p(C) < \frac{R+R'}{G}$, then σ_2* is maximized for w_2=1 only. The choice of u_1 is dependent on q and vice versa. If u_1=0, then σ_1 is maximal for q=0, and for q=0, σ_2 is maximal for u_1=1, and for u_1=1, σ_1 is maximal for q=1, however, for q=1, σ_2 is maximal for u_1=0, and so on. There is no perfect Bayesian equilibrium for this case.

6.2.4. Repeated game

We analyze a class of repeated games in which the informed player's type is persistent and the history of actions is perfectly observable. This context rightly represents the periodic iteration of the verification protocol performed by the owner to assess whether the holder is still storing the data it promises to keep. The analyzed repeated game is the game of Figure 46 and Figure 47 iterated while maintaining *H*'s type. These games are played for finite times, but no

player knows the exact game termination time. The probability p captures the probability of "natural" termination of the repeated game (e.g., loss of connection between *O* and *H*). Additionally, the owner *O* has the possibility to stop the repeated game if it detects the selfishness or the failure of *H* (*H* is of type "S" or "F"). The payoff at the i^{th} period is designated by $g_i=(g_i^H, g_i^O)$. The sum of per-period payoffs is given by:

$$g = \left(\sum_{i=0}^{\infty}(1-p)^i\, g_i^H, \sum_{i=0}^{\infty}(1-p)^i\, g_i^O\right)$$

Action profiles

From the signals sent per-period by *H*, *O* may infer the type of *H*. There are three distinct possible action profiles:

d) (s, rw), (s, rw), (s, rw), …
e) (s, rw), (s, rw), …, (s, rw), (f, pn)
f) (f, pn)

If the signal is "s", then, the best response of *O* is to play "rw". If the signal changes from "s" to "f", *O* concludes that *H* is of type "S" and the action played is "pn". If the signal is "f" from the the first round, *O* infers that the type of *H* is either "S" or "F", for both cases it is better to play the action "pn".

Numerical evaluation

The games of Figure 46 and Figure 47 are iterated and evaluated within different scenarios. Games' parameters are measured in MB (Mega Bytes) unit (1 MB=106 bytes). We intend with such evaluations to define the impact of the probability of game termination p and also the requirements on the values of the reward and punishment to achieve full cooperativeness of the holder.

At first, we consider the repeated game of Figure 46. *H* chooses the strategy that maximizes its payoff. To make *H* choose the type "C" over "S", its outcome by choosing "C" must be higher than its outcome choosing "S". If the owner adheres to the action profiles presented earlier, the payoff of *H* if it chooses the type "C" is derived as:

$$g_{"C"}^H = \frac{R-D}{p}$$

H's payoff if it chooses the type "S" is:

$$g_{"S"}^H = \frac{qR + (1-q)(-R') - qD}{1-q(1-p)}$$

Cooperation is more advantageous for *H*, if the inequality $g_{"C"}^H > g_{"S"}^H$ holds for every p. From this inequality, we derive the lower bound of the probability p (for $q \neq 1$):

$$p > \frac{R-D}{-R\prime} \qquad (6.2.4.a)$$

Since, we assumed in the beginning that $R > D$, then $\frac{R-D}{-R\prime} < 0$; which means that the inequality (6.2.4.a) is always achieved for any value of p. So, choosing the type "C" results in a higher outcome than the type "S" for any probability of game termination p.

Figure 48 depicts *H*'s payoffs varying p. The figure shows that the gap between the payoff of *H* with type "C" and its payoff with type "S" increases inversely proportional to p: for low value of p the gap counts in hundreds to thousands MBs (the graph shows truncated $g_{"C"}^H$ for low p)

compared to a ten or so MBs with high value of *p*. The figure also demonstrates that the holder with type "S" always achieves a higher outcome $g^{H}_{"S"}$ with high value of q (e.g., q=0.9) then with low q (q=0.1).

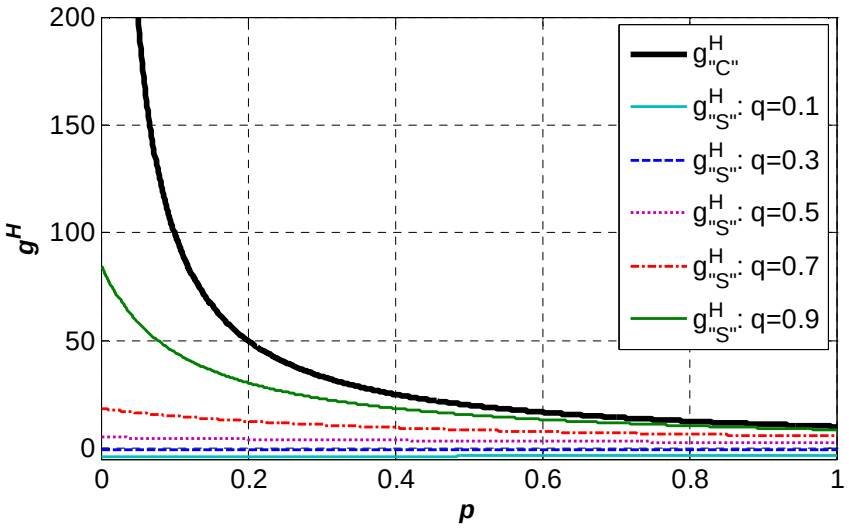


Figure 48 Payoffs of *H* with type "S" and "C" (truncated) varying *p* and *q*. *G*=30, *R*=20, *R'*=5, *D*=10.

Additionally, the outcome $g^{H}_{"S"}$ gets exponentially higher with low probability *p*. These results demonstrate that the repetition of the game (low *p*) motivates the holder *H* to cooperate since it obtains a high payoff with cooperation than with selfishness taking into account the fact that the owner follows the action profiles of 0.

Here, we consider the repeated game of Figure 47. The payoff of the owner *O* is dependent on whether the holder *H* has opted for the type "C" or "S". If the type of *H* is "C" then the payoff of *O*, if this latter follows the presented action profiles, is derived as:

$$g^{O}_{"C"} = \frac{G - R}{p}$$

However, if *H* chooses rather the type "S" then the payoff of *O* becomes:

$$g^{O}_{"S"} = \frac{q(-R) + (1-q)R'}{1 - q(1-p)}$$

The owner is faced with the alternatives of whether to stop the game by punishing the holder (playing "pn") or to continue rewarding the holder whenever this latter answers correctly to its challenge (playing "rw"). If the owner has prior beliefs on the probabilities that the holder may choose one type or the other, p(C) and p(S), the owner then may decide on these alternatives based on the following inequality:

$$p(C) \times g^{O}_{"C"} + p(S) \times g^{O}_{S} > R' \qquad (6.2.4.b)$$

This inequality (6.2.4.b) means that the average payoff of *O* if it commits to the game is higher than its payoff if it calls off the game (by directly punishing *H*). If this inequality is obtained then *O* chooses to continue the game; otherwise, it is more advantageous for *O* to stop the game. The inequality (6.2.4.b) is obtained when:

$$p(C) > V(p);$$

$$V(p) = p\left(\frac{q(R + pR')}{\big(1 - q(1-p)\big)G - (1-q)(R + pR')}\right)$$

The inequality of (6.2.3) presented previously in finding the perfect Bayesian equilibrium of the game corresponds to p(C) ≥ $V(1)$ (case of p=1).

Figure 49 illustrates the function $V(p)$ varying the probability of selfish holder success to challenges q. The figure gives the asymptotic lower bound (q=1) of the probability p(C) over which *O* deems acceptable to continue the game with *H*. Based on its prior beliefs about *H*, *O* can decide whether to play with *H* or not. The figure shows that the lower bound of p(C) increases with the probability p which means that the iterated version of the game is less risky for *O* than one-stage game. Iteration of the game then motivates the owner to be cooperative and to play the game with the holder; albeit the fact that its prior belief on the probability p(S) that this latter is selfish is not null. This observation is even sustained by the fact that if p=0 (i.e., the game never ends), then $V(0)$=0 which means that *O* always gains a higher payoff if it cooperates with *H* than the one obtained by not playing the game.

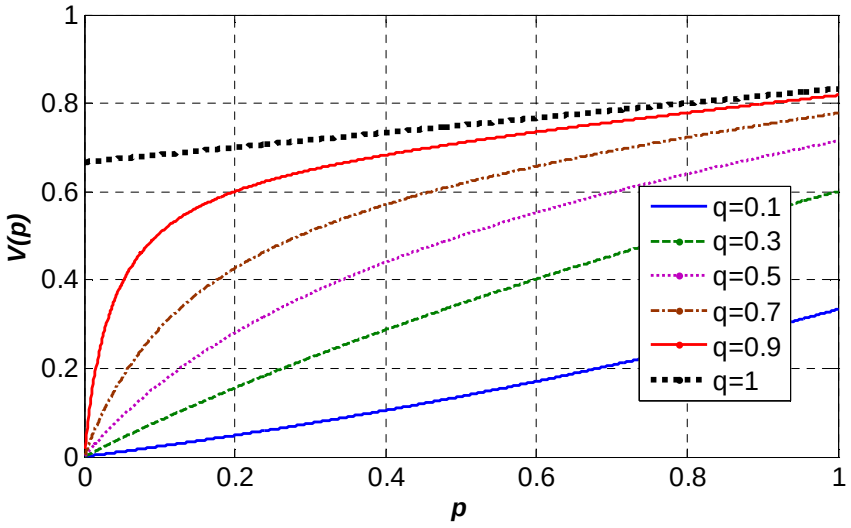


Figure 49 The minimum value for p(C) acceptable for *O* to continue the game varying *p* and *q*. *G*=30, *R*=20, *R'*=5, *D*=10.

The value of the reward gained by *H* if it successfully answers *O*'s challenge and also the value of the punishment lost by *H* if otherwise it sends an incorrect response to *O* have an impact on the minimum value of p(C). We can define such impact by computing the maximum asymptotic lower bound of p(C) (i.e., $V(1)$ for q=1) that becomes equal to:

$$V(1)_{q=1} = \frac{R + R'}{G}$$

This latter equality demonstrates that increasing *R* and *R'* increases the lower bound of p(C). So, increasing *R* and *R'* reduces the cooperativeness of the owner. This is a quiet an interesting result to find out that increasing the punishment that *O* acquires if *H* fails has a negative impact on the cooperation of *O*. This is because the punishment is also obtained by *O* if it does not cooperate and stops the game by declaring that *H* has been selfish. That's why, increasing *R'* increases also *O*'s payoff of non cooperation that may exceed its payoff of cooperation.

Figure 50 depicts the function $V(p)$ for different values of R and R'. The figure shows that increasing R increases the lower bound of p(C) that even attains the limit 1 for some given reward R (=35 and R'=0). Increasing R' also increases the lower bound of p(C) that significantly increases with p (compared to increasing R). We may notice that for low value of p, O's cooperation is better stimulated by increasing R' than R because we have a smaller raise in the lower bound of p(C) by increasing R' than R by the same value (=10); whereas for high value of p, the lower bound of p(C) considerably increases by increasing R' than R. This result is due to the fact that iteration of the game (low p) increases the chances of O to obtain the punishment value R' if it chooses to cooperate and if the type of H is "S" and this compensates the acquiring of this value by not cooperating (playing "pn" at the beginning of the game).

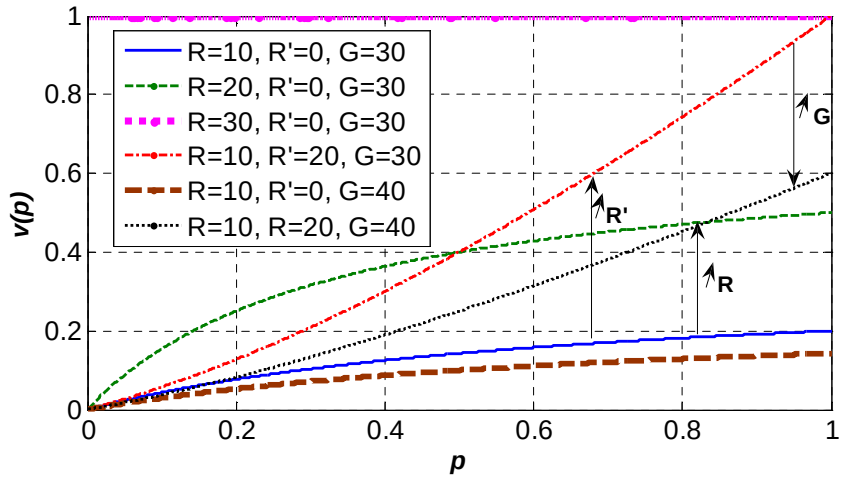


Figure 50 The minimum value for p(C) acceptable for *O* to continue the game varying *R* and *R'*. *G*=30, *D*=10, *q*=0.5.

Additionally, the figure illustrates the fact that increasing the gain G obtained by the owner if the holder is cooperative makes the lower bound of p(C) decreases, as a result of the fact that the gain is only obtained when the owner cooperates. In conclusion, owner's cooperation is better stimulated by minimizing the reward and the punishment values R and R' (R=D and R'=0) and maximizing the storage gain G.

Discussion

The repeated game of Figure 46 represents an interaction between a data owner and a data holder from a data holder perspective. For this repeated game, we aim to encourage the cooperation of the holder by making its cooperative behavior the best strategically choice to make. The result on the probability p shows that iteration of the game favors the cooperativeness of H. On the other hand, the repeated game of Figure 47 illustrates the interaction of a data owner with a holder from the owner perspective. For this repeated game, we aim, this time, to guide the owner in choosing the best response to holder actions based on the prior beliefs about this very holder (these prior beliefs may correspond to holder's reputation). We showed that the cooperativeness of the owner increases by iterating the game. We identified which actions the owner must follow for a given probability p(C). For this, we showed the inequalities that the reward R and the punishment R' should verify.

6.3. Evolutionary game model of reputation-based incentives

An evolutionary game model describes the evolution of strategies within large populations as a result of many local interactions, each involving a small number of randomly selected individuals. An individual plays only once; it plays in a one shot game against another randomly selected player with the goal of maximizing its utility (fitness) in that game.

This section presents an evolutionary game model of the P2P storage system with which it is demonstrated that peers using the audit-based reputation strategy will dominate the system. Audits are obtained from periodic checking of storage at holders based on a deterministic detection of data destruction.

6.3.1. Game model

In the proposed system, an owner stores data replicas at *r* holders. It appoints *m* verifiers for its data replica that will periodically check storage at holders. The system is modeled as an evolutionary game [Friedman 1998]: "an evolutionary game is a dynamic model of strategic interaction with the following characteristics: (a) higher payoff strategies tend over time to displace lower payoff strategies; (b) there is inertia; (c) players do not intentionally influence other players' future actions".

One-stage game

The one-stage game represents an interaction between one data owner, *r* data holders, and *m* verifiers randomly chosen. Thus, the considered game players are an owner, *r* holders, and *m* verifiers. The one-stage interaction consists of several phases:

- *Storage phase:* the owner stores data at the *r* holders. At this phase, holders may decide to keep data stored or to destroy them depending on their strategy (see next paragraph "Evolutionary game"). Holders that crash or leave the system without any notice are considered as defectors contrary to our previous work with the Bayesian game.
- *Delegation phase:* the owner sends verification information to the *m* verifiers in order to be able to periodically check data at holders. Whether to cooperate with the owner in verifying data is determined by each verifier's strategy (see next paragraph "Evolutionary game").
- *Verification phase:* a verifier can decide whether the holder has been cooperative based on the results of a verification protocol and take potential action depending on its strategy. A verifier whose strategy is to cooperate will send the owner the results it obtained by auditing the holder. A non-cooperative verifier may mimic a cooperative strategy by sending a bogus result. Verifiers are not more trusted than other peers and may lie about verification, for instance reporting an absence of response to a challenge for a cooperative holder. A verifier might also be framed by a malicious holder trying to make it appear as a non-cooperative verifier. Some verifiers may also crash or leave the system, and be unable to communicate results of verifications. The owner therefore cannot determine with certainty whether a verifier chose to adopt a cooperative strategy. One negative result from a verifier is also not enough for the owner to decide that the holder is non cooperative. Such a notification may however be used as a warning that the holder may have destroyed its data. Based on such a warning, the owner would replicate the endangered data, therefore maintaining or even increasing storage reliability to his advantage.

- *Retrieval phase:* the owner retrieves its data from the *r* holders. If one holder destroyed the data, the owner decides on potential action towards that holder depending on its strategy (see next paragraph "Evolutionary game").

Data storage is a long-term process during which several peers may have been storing data from multiple owners; we define the evolutionary game that models our P2P storage application as a sequence of a random number of such simultaneous one-stage interactions.

Evolutionary game

Our proposed game is similar to the game in [Brandt and Sigmund 2006] where players have either the role of the donor or the role of the recipient. The donor can confer a benefit *b* to the recipient, at a cost *-c* to the donor. We consider three roles in our game: owner, holder, and verifier; any peer may play several of these roles throughout the game. In a one-stage game, the owner is considered a recipient, the *r* holders and *m* verifiers are donors. The owner gains *b* if at least one holder donates at a cost *–c*; however if no holder donates then the owner gains βb if at least one verifier donates at a cost $-\alpha c$ ($\alpha \leq 1$) for each verifier (Figure 51 summarizes the model). The latter case corresponds to the situation where the cooperative verifier informs the owner of the data destruction, and then the owner may replicate its data elsewhere in the network thus maintaining the security of its data storage.

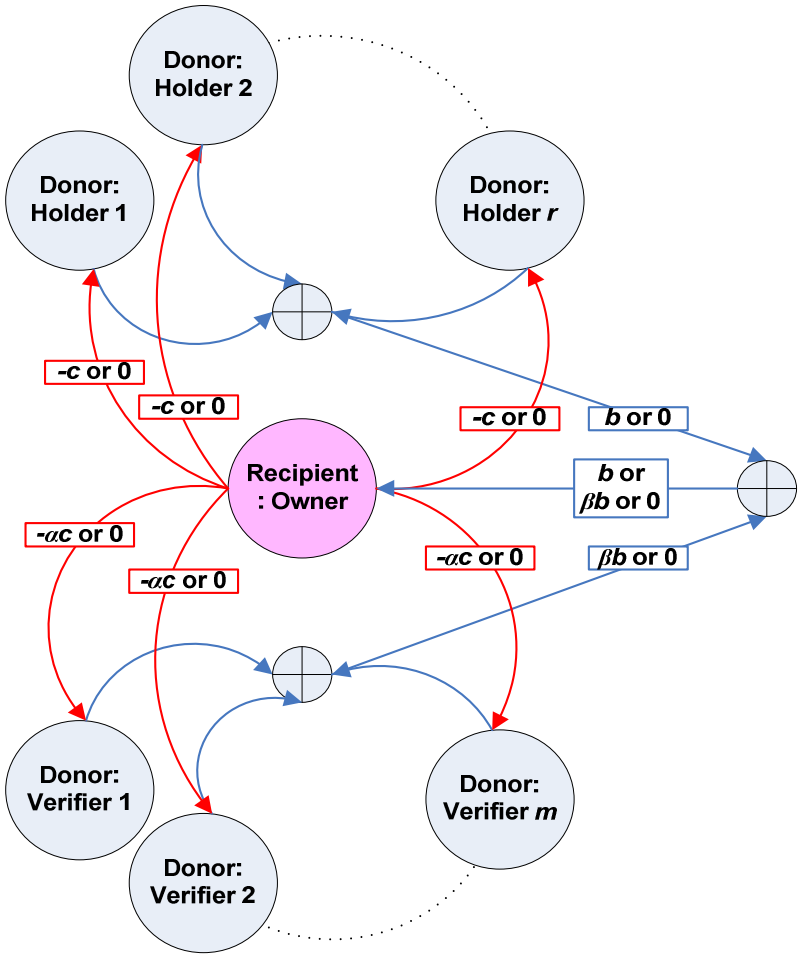


Figure 51 One-stage game model

Holders and verifiers have the choice between cooperating, which we call interchangeably donate, or defecting:

- Cooperation whereby the peer is expected to keep others' data in its memory and to verify data held by other peers on behalf of the owner.
- Defection whereby the peer destroys the data it has accepted to hold, and also does not verify others' data as it promised to.

Storage of data and their verification are two independent actions. Appendix C studies the behavior of peers that may defect in one of these actions independently of the other. The following instead considers peers with some determined behavior that take these two actions as falling under the same objective: either to cooperate or to shirk.

The peers' strategies that we consider for study are:

- Always cooperate (*AllC*): the peer always decides to donate, when in the role of the donor.
- Always defect (*AllD*): the peer never donates in the role of the donor.
- Discriminate (*D*): the discriminator donates under conditions: if the discriminator does not know its co-player, it will always donate; however, if it had previously played with its co-player, it will only donate if its co-player donates in the previous game. This strategy resembles Tit-For-Tat but differs from it in that both the owner (the donor) and its verifiers may decide to stop cooperating with the holder in the future.

6.3.2. Observations

Let us consider a scheme (see Figure 52) inspired from epidemic models which categorize the population into groups depending on their state [Jones and Sleeman 1983]. Two states are distinguished: "not known" and "known" states. Because of the random selection of holders and verifiers among all peers and given the presence of churn, there are always nodes potentially in the "not known" state.

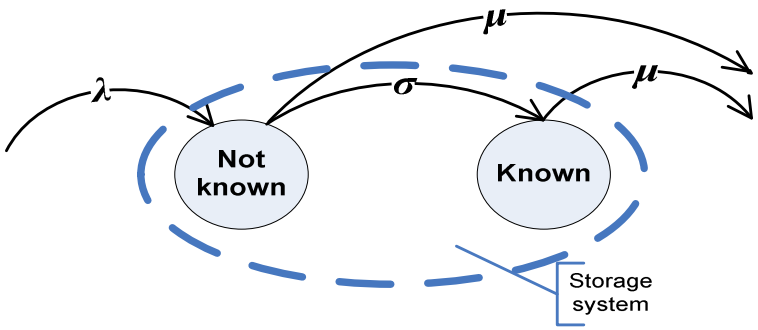


Figure 52 System dynamics

We denote the number of peers that a given peer in average does not know at a certain time t by D and the number of peers that it knows on average at time t by K. Peers that may join the system are peers who were invited by other members with a fixed invitation rate λ. Peers are leaving the system with a fixed departure rate of μ.

The rate σ designates the frequency of encounter between two peers, one of them being the holder (i.e., the probability that a peer knows about the behavior of another peer). The rate σ depends on the replication rate r and verification distribution m; indeed it is derived in average as:

$$\sigma = 1 - \left(1 - \frac{\gamma r}{N-1}\right) \times \left(1 - \frac{\gamma r}{N-1} + \frac{\gamma r}{N-1} \times \left(1 - \frac{m}{N-2}\right)\right)^{N-2}$$

γ being the average storage rate of peers and N being the total number of peers in the system. The formulation of the rate σ takes into account the probability that the observing peer chooses the observed peer as a holder of its data (the peer stores data at rate γ) and the probability that another peer from the N-2 remaining peers chooses the observed peer as a holder and the observing peer as a verifier for it.

We denote the total number of peers in the storage system - excluding the observing peer - as $n = D + K$. The dynamics of K and D are given by the following equations:

$$\frac{dD}{dt} = \lambda n - (\sigma + \mu)D$$

$$\frac{dK}{dt} = \sigma D - \mu K = \sigma n - (\sigma + \mu)K$$

Since $n = D + K$:

$$\frac{dn}{dt} = (\lambda - \mu)n$$

Let q be the probability that the discriminator knows what a randomly chosen co-player chose as a holder strategy in a previous one-stage game (the discriminator being an owner or verifier in that game). The probability q is equal to K/n, hence:

$$\frac{dq}{dt} = \frac{dK/dt}{n} - \frac{K dn/dt}{n^2}$$

Thus,

$$\frac{dq}{dt} = \sigma - (\sigma + \lambda)q$$

At time t=0, the set of peers in state K is empty. Over time, peers in state D enter state K with rate σ. A new peer joining the system is assigned state D meaning that initially $q(0)$=0. The result of the above differential equation is thus:

$$q(t) = \frac{\sigma}{\sigma + \lambda}(1 - e^{-(\sigma+\lambda)t})$$

The limit of $q(t)$ when $t \to \infty$ is $\sigma/(\sigma + \lambda)$. If we consider a system without churn (λ=0), the limit becomes 1.

6.3.3. Fitness

We respectively denote the frequency (i.e., fraction in the population of playing peers) of strategies *AllC* by *x*, *AllD* by *y*, and *D* by *z*. The expected values for the total payoff obtained by the three strategies are denoted by U_{AllC}, U_{AllD} and U_D, and the average payoff in the population by:

$$\overline{U} = x \times U_{ALLC} + y \times U_{ALLD} + z \times U_D$$

The average payoffs that are also called fitness for each strategy are defined in the following.

At time *t*, a participating peer will have *r* times more chances to be chosen as a holder and *m* times more chances to be chosen as verifier than to be chosen as an owner.

A peer playing the strategy *ALLC* will always cooperate: it will donate at a cost *–c* if it is chosen as a holder or at a cost *–αc* if it is chosen as a verifier. It will gain a benefit *b* if it is chosen as an owner and at least one of its data holders is not a defector, otherwise, it may gain a benefit *βb* if at least one of its verifiers is not a defector.

$$\begin{aligned} U_{ALLC} &= -rc - m\alpha c + b(1 - y^r) + \beta b\left(y^r(1 - y^m)\right) \\ &= -c(r + m\alpha) + b(1 - y^r + \beta y^r(1 - y^m)) \end{aligned}$$

A peer playing the strategy *ALLD* will never cooperate, so it will never donate. It will gain a benefit *b* if it is chosen as an owner and at least one of its data holders is not any of these types: a defector (type occurs with frequency, i.e., probability *y*) or a discriminator that knows the peer (type occurs with probability *qz* on average). Otherwise, the peer may gain a benefit *βb* if at least one of its verifiers is not of any of the former two types.

$$\begin{aligned} U_{ALLD} &= b(1 - (y + qz)^r) + \beta b\left((y + qz)^r(1 - (y + qz)^m)\right) \\ &= b(1 - (y + qz)^r + \beta(y + qz)^r(1 - (y + qz)^m)) \end{aligned}$$

A peer playing the strategy *D* will always cooperate if it does not know the recipient or the latter was cooperative in a previous interaction. It will donate at a cost *–c* if it is chosen as a holder or at a cost *–αc* if it is chosen as a verifier. It will gain a benefit *b* if it is chosen as an owner and at least one of its data holders is not a defector, otherwise, it may gain a benefit *βb* if at least one of its verifiers is not a defector.

$$U_D = -c(r + m\alpha)(1 - qy) + b(1 - y^r + \beta y^r(1 - y^m))$$

Strategies with higher fitness are expected to propagate faster in the population and become more common. This process is called *natural selection*.

6.3.4. Replicator dynamics

The basic concept of replicator dynamics is that the growth rate of peers taking a strategy is proportional to the fitness acquired by the strategy. Thus, the strategy that yields more fitness than average fitness of the whole system increases, and vice versa. We will use the well known differential replicator equations:

$$\begin{aligned} \frac{dx}{dt} &= x(U_{AllC} - \overline{U}) \\ \frac{dy}{dt} &= y(U_{AllD} - \overline{U}) \\ \frac{dz}{dt} &= z(U_D - \overline{U}) \end{aligned} \tag{6.3.4}$$

6.3.5. Evolutionary stable strategy

A Strategy is said to *invade* a population of strategy players if its fitness when interacting with the other strategy is higher than the fitness of the other strategy when interacting with the same strategy. An evolutionarily stable strategy (ESS) is a strategy which no other strategy can invade if all peers adopt it.

Case $x\neq0$, $y=0$, $z\neq0$: This case corresponds to a fixed point in the replicator dynamics, which means that a mixture of discriminating and altruistic population can coexist and are in equilibrium.

Case $x\neq0$, $y\neq0$, $z=0$: In this case, the replicator dynamics of both altruistic and defector populations are:

$$\frac{dx}{dt} = -xyc(r + m\alpha) \leq 0$$
$$\frac{dy}{dt} = xyc(r + m\alpha) \geq 0$$

The population of defectors wins the game and the ESS is attained at $x=0$ and $y=1$.

Case $x=0$, $y\neq0$, $z\neq0$: The dynamics of the populations of defectors and discriminators are derived as:

$$\frac{dy}{dt} = yz(c(r + m\alpha)(1 - qy) + b(f(y) - f(y + qz))$$
$$\frac{dz}{dt} = yz(-c(r + m\alpha)(1 - qy) + b(f(y + qz) - f(y))$$

where the function f is defined as follows:

$$f(u) = u^r - \beta u^r(1 - u^m)$$

The equilibrium point ($x=0$, $y=y_0$, $z=z_0$) for which defectors and discriminators may coexist corresponds to the solution(s) of the following equation:

$$\frac{dy}{dt} = \frac{dz}{dt} = 0$$

The equilibrium point is then defined as follows:

$$c(r + m\alpha)(1 - qy_0) = b\big(f(y_0 + qz_0) - f(y_0)\big)$$

Table 8 describes equilibrium values in some particular cases. More cases for equilibrium values will be examined in the next section.

Table 8 Finding the equilibrium for $x=0$, $y\neq0$, $z\neq0$.

Conditions	y_0	z_0
$r=1$, $m=0$, $b\neq c$, $q(t) \underset{t\to\infty}{\longrightarrow} \frac{\sigma}{\sigma+\lambda}$	$\min\left(\max\left(\frac{b\sigma - c(\sigma+\lambda)}{(b-c)\sigma}, 0\right), 1\right)$	$\min\left(\max\left(\frac{c\lambda}{(b-c)\sigma}, 0\right), 1\right)$
$r=0$, $m=1$, $b\neq c$, $q(t) \underset{t\to\infty}{\longrightarrow} \frac{\sigma}{\sigma+\lambda}$	$\min\left(\max\left(\frac{\beta b\sigma - \alpha c(\sigma+\lambda)}{(\beta b-\alpha c)\sigma}, 0\right), 1\right)$	$\min\left(\max\left(\frac{\alpha c\lambda}{(\beta b-\alpha c)\sigma}, 0\right), 1\right)$

Case $x\neq0$, $y\neq0$, $z\neq0$: There is one stationary point ($x=0$, $y=y_0$, $z=z_0$) for which defectors will exploit and eventually deplete all cooperators. The amount of defectors will first increase, and then converges to the equilibrium where there is either coexistence with discriminators, or winning over them, or losing to them depending on storage system parameters.

6.3.6. Numerical evaluation

The evolutionary game is simulated within a custom simulator using the differential equations of section 6.3.4. Simulations involve several scenarios with various storage system parameters in order to capture their impact on the convergence of the system to an equilibrium.

In our simulations, we consider that in each day of simulated time, 3 files are stored per peer with average file size of 500MB. The verification metadata corresponding to each file having an average size of 10KB is stored at the appointed verifiers. There are 10 newcomers to the storage system per month for an equivalent number of peers leaving it. These newcomers are detaining the same strategy as their hosts because we assume that the arrival and departures of peers are strategy-neutral i.e., they do not alter the strategy distribution (we assume that the dynamics of strategies solely depend on their payoffs as in the replicator dynamics of 6.3.4).

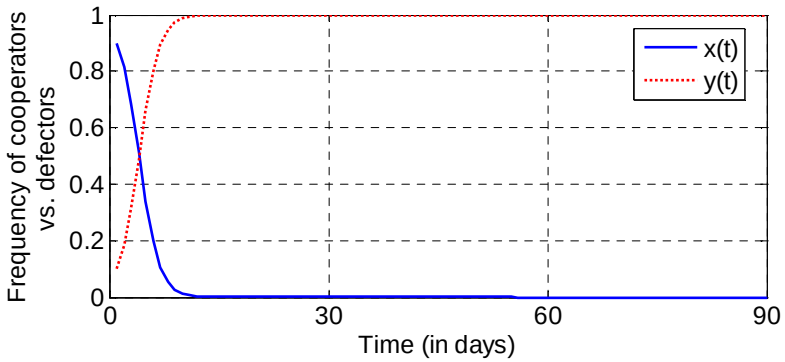


Figure 53 Frequency of cooperators vs. defectors over time. m=5, r=3, β=0.1, α=20.10^{-6}, λ=10/month, N=1000, γ=3 files/day, b=1, c=0.01, $x(0)$=0.8, $y(0)$=0.2, and $z(0)$=0.

Initial frequency of strategies: Figure 53 shows the frequency of cooperators and defectors over time, and demonstrates that with time cooperators will be eliminated from the system by these defectors. The presence of discriminators in the system does not prevent cooperators from being evicted from the system; however, discriminators and defectors will converge to an equilibrium where both coexist (see Figure 54).

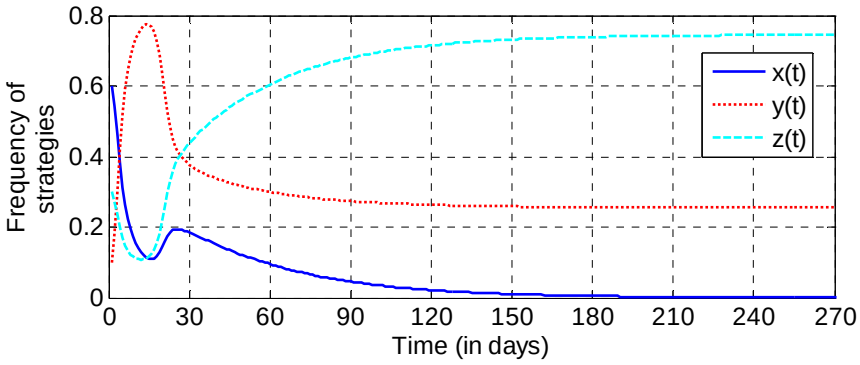


Figure 54 Frequency of the three strategies over time. m=5, r=3, β=0.1, α=20.10^{-6}, λ=10/month, N=1000, γ=3 files/day, b=1, c=0.01, $x(0)$=0.6, $y(0)$=0.1, and $z(0)$=0.3.

This equilibrium is perturbed by the injection of a large population of defectors, as illustrated in Figure 55 (by varying the initial frequency of z). If discrimination becomes a minor strategy in the population ($z(t) \leq 0.2$), it is completely eliminated from the system. However, if a small population of defectors is injected, discriminators still converge to the same equilibrium.

The minimum initial frequency for which the population of discriminators achieves an equilibrium where their frequency is not null is denoted $z_{min}(0)$ (~0.2). There are two equilibria that are determined by the initial population of discriminators: (x=0, y=1, z=0) if $z(0) \leq z_{min}(0)$ and (x=0, y=y_0, z=z_0) if $z(0) \geq z_{min}(0)$.

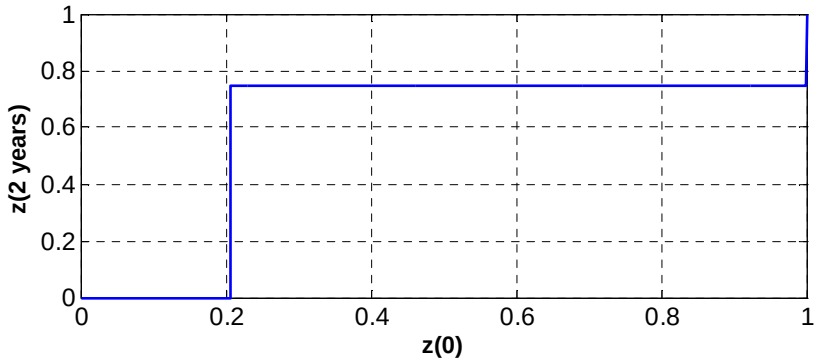


Figure 55 Frequency of discriminators at equilibrium varying $z(0)$. m=5, r=3, β=0.1, α=20.10^{-6}, λ=10/month, N=1000, γ=3 files/day, b=1, c=0.01, $x(0)$=0.

The discriminators do not win over defectors, because the latter may still have a good payoff if they interact with some discriminators that do not know them yet, for instance for discriminators that just entered the system, or defectors that just joined in. Additionally, defectors do not always win over the discriminators because there are discriminators that already know them and that always choose to defect with them. The figure shows also a little decrease in the frequency of discriminators before converging to the equilibrium. The decrease is due to the fact that discriminators act as cooperators in the beginning of the game since they do not know the behavior of defectors yet.

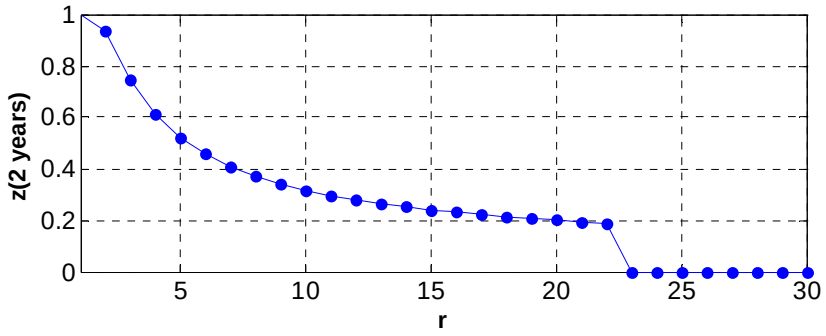


Figure 56 Frequency of discriminators at equilibrium varying r. m=5, β=0.1, α=20.10^{-6}, λ=10/month, N=1000, γ=3 files/day, b=1, c=0.01, $x(0)$=0, $y(0)$=0.5, and $z(0)$=0.5.

Number of verifiers and replicas: Varying the number r of data replicas or the number m of verifiers changes differently the equilibrium point. Increasing r favors defectors (see Figure 56). This is because the fitness gain of discriminating owners is overwhelmed by the fitness loss that results from data storage cost -c that is always paid by discriminating holders. Increasing r increases data reliability, thus increasing chances of having the benefit b. But, this benefit is

perceived by both populations of discriminators and defectors without favoring one over the other.

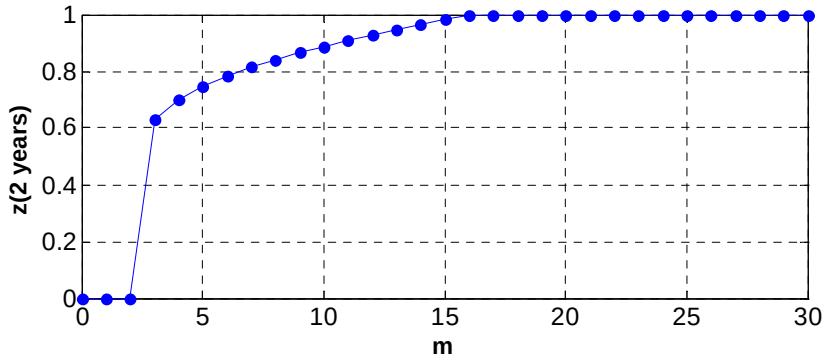


Figure 57 Frequency of discriminators at equilibrium varying *m*. *r*=3, *β*=0.1, *α*=20.10^{-6}, *λ*=10/month, *N*=1000, *γ*=3 files/day, *b*=1, *c*=0.01, *x*(0)=0, *y*(0)=0.5, and *z*(0)=0.5.

Increasing *m* increases the equilibrium value of discriminators frequency (see Figure 57). This is due to the fact that increasing *m* makes higher the chances to obtain a benefit *βb*. However, increasing *m* increases also the cost of data verification -*αc*. Even if this cost is just paid by discriminating verifiers, it is still modest compared to the benefit perceived in proportion ($\alpha << 1$).

Figure 58 also illustrates the fact that increasing the storage rate that in return increases the probability of encounter *σ* leads to an increase in the equilibrium value of discriminators' frequency because more discriminators get acquainted with more defectors. The figure defines he storage rate under which discriminators are eliminated from the system by defectors.

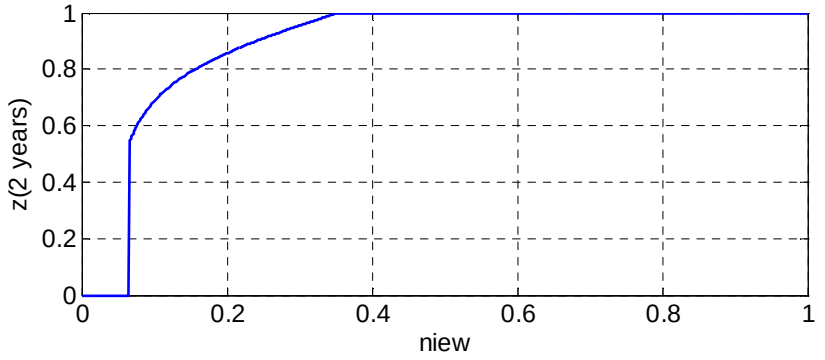


Figure 58 Frequency of discriminators at equilibrium varying the average storage rate *γ* in #file/hour. *m*=5, *r*=3, *β*=0.1, *α*=20.10^{-6}, *λ*=10/month, *N*=1000, *b*=1, *c*=0.01, *x*(0)=0, *y*(0)=0.5, and *z*(0)=0.5.

Churn: The peer arrival rate *λ* affects the probability *q*, and hence the equilibrium point of the game (see Figure 59). For a low churnout value (small *λ*), the frequency of discriminators at equilibrium is high; whereas for a high churnout value, the frequency at equilibrium decreases. For high churnout, peers are not able to get acquainted with all peers since there are always new peers in the system, and defectors may take advantage of the lack of knowledge of

discriminators about the system to gain benefit and remain in the game. For a system without churnout (λ=0), discriminators win against defectors that are eliminated from the game.

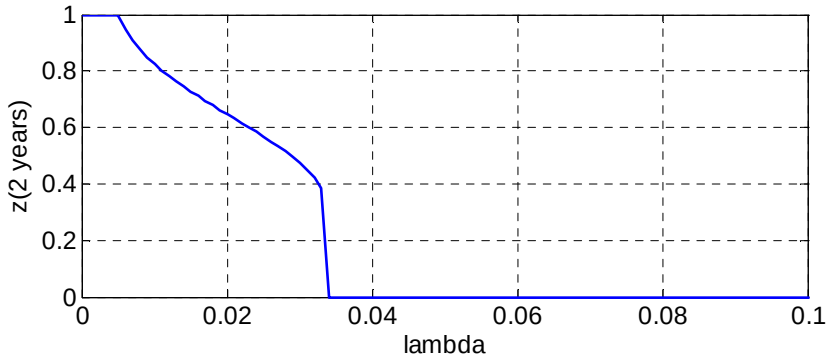


Figure 59 Frequency of discriminators at equilibrium varying the arrival rate λ in #newcomers/hour. m=5, r=3, β=0.1, α=20.10^{-6}, N=1000, γ=3 files/day, b=1, c=0.01, x(0)=0, y(0)=0.5, and z(0)=0.5.

Benefit and cost: Figure 60 depicts the impact of the benefit b and of the cost c on the frequency of discriminators at equilibrium. The figure shows that b and c have opposite effects on the equilibrium frequency of discriminators: increasing b increases the frequency whereas increasing c makes it decrease. If the storage cost is small, it will be compensated by the benefit. In contrast, if the storage cost is high ($c \geq 0.01 \times b$), discriminators cannot cope with this high cost and they will be eliminated from the system by defectors. Additionally, the figure shows that the equilibrium point varies in function of b and c.

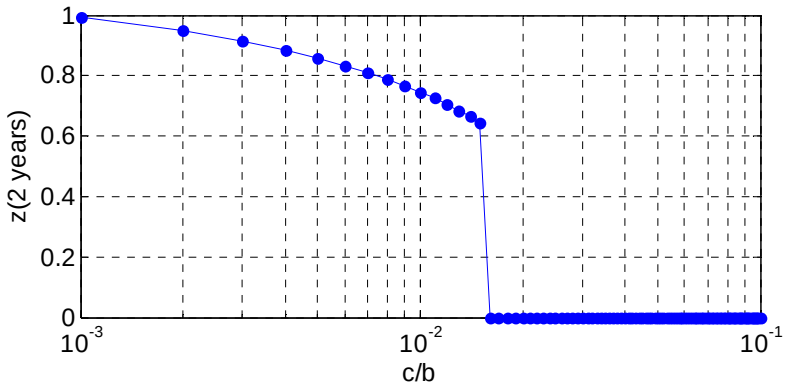


Figure 60 Frequency of discriminators at equilibrium varying the ratio c/b. m=5, r=7, β=0.1, α=0.001, λ=0.01, σ=0.05, b=0.05, x(0)=0, y(0)=0.5, and z(0)=0.5.

Discussion: Simulation results prove that there exist parameter values for which discriminators, who use an audit-based mechanism, may win against free-riding defectors. Discriminators are not hopeless when confronting defectors, even if the latter may dominate altruists (*always cooperate* strategy). At the equilibrium of the game, both discriminators and defectors may coexist if there is churn in the system otherwise discriminators will dominate. The number of verifiers m increases the frequency of discriminators at the equilibrium. Whereas, a costly storage or an increase of the replication rate r reduce this frequency.

In the proposed reputation-audit based approach of 5.2, the discriminators do not always cooperate with newcomers; they only cooperate with them with some probability p. We have

studied such type of discriminating behavior taking into account a new type of defectors that whitewash i.e., leave the system and rejoin with a new identity to escape its punishment. Results are presented in Appendix B. They demonstrate that the probabilistic cooperation with strangers of discriminators is not sufficient to fully dominate defectors. Thus, it is required to further prevent or at least mitigate the whitewashing behavior by controlling the peer entry into the system, for instance the joining of peers may be by invitation, or using a cryptographic puzzle or even imposing a fee.

6.4. Summary

In this chapter, we presented several game theoretical models of our audit-based cooperation incentive mechanism. The Bayesian game model illustrating the probabilistic verification protocol allows solutions where both parties of the game are cooperative for well identified payment parameters and repetition frequency. Additionally, the audit-based strategy that relies on a deterministic verification protocol wins over the free-riding strategy in a closed system; if not, with some particular conditions, it coexists with free-riders at a high frequency. Thus, using these game models, we validate the inherent incentive capability property of the proposed payment and reputation-based approaches that aim at steering peers towards cooperative behavior.

6.5. Relevant publication

1. Nouha Oualha, Pietro Michiardi, and Yves Roudier. A game theoretic model of a protocol for data possession verification. TSPUC 2007, IEEE International Workshop on Trust, Security, and Privacy for Ubiquitous Computing, June 18, 2007, Helsinki, Finland.

2. Nouha Oualha and Yves Roudier. Evolutionary game for peer-to-peer storage audits. In the 3rd International Workshop on Self-Organizing Systems (IWSOS'08), December 10-12, Vienna, Austria.

3. Nouha Oualha and Yves Roudier. A Game Theoretical Approach in Securing P2P Storage against Whitewashers. In the 5th International Workshop on Collaborative Peer-to-Peer Systems (COPS'09), June 29 - July 1, 2009, Groningen, Netherlands.

Chapter 7

Conclusion and future work

Peer-to-Peer (P2P) systems have emerged as an important paradigm for distributed data storage in the way they exploit and efficiently make use of untapped peers' storage resources. Outsourcing data from a single location to multiple peers in a network is probably the only solution for increasing data availability and fault-tolerance on a large scale while reducing if not suppressing storage maintenance costs. In this thesis, we addressed the security and cooperation issues that such an application is likely to be exposed to when effectively deployed in the wild.

Summary and contributions

We first discussed the security issues associated with P2P data storage. The correct operation of a P2P storage system relies on the fair and effective cooperation of peers. Unfortunately, peers may misbehave in various ways. Data holders may pretend to be storing some data which they in fact destroyed. With replication based approaches, peers may collude to store a single data replica thereby defeating mechanisms to ensure reliability. Collusion may not be the sole way to do so, since Sybil attackers may generate several identities and deceitfully use them.

We describe elements of a modular architecture for such a system encompassing the security and cooperation mechanisms necessary to ensure the correct and secure operation of a P2P data storage system. We describe how a trusted environment may make it easier to prevent some misbehaviors, in particular if peer identification, data integrity verification, and trust management may be assured by dedicated hardware or trusted platforms rather than performed by peers themselves.

Hidden actions of non cooperative peers can be revealed using a new type of protocol that we call data possession verification. Such protocols enable a verifier to detect whether some data that are stored remotely have been corrupted. We propose three different such protocols with different verification capabilities, in particular regarding delegation.

The behavior of data holders can be evaluated based on the results obtained out of such protocols. Such audits form the basic observation primitives of the cooperation incentive mechanisms that we propose for stimulating cooperation and inciting correct behaviors. The originality of the incentive mechanism stems from the optimistic peer behavior evaluation, following a very different approach compared with cooperation incentives in MANETs: while peer behavior can only be decided at the end of the storage period, audits can be performed on a regular basis and we consider that a peer behaves well as long as no data corruption is detected. We propose two incentive mechanisms, one reputation-based and the other remuneration-based. Both mechanisms are designed not only to incent to cooperative behavior but also to establish trust as well as to detect and punish misbehaving peers. These constitute essential features of a security mechanism for such applications given the possibility of purely malicious attacks.

The effectiveness of our security and cooperation achieved by our proposed audit-based mechanisms is demonstrated through non-cooperative game theoretical models. We first evaluate the effectiveness of our incentives with various observation primitives both probabilistic and deterministic. Evolutionary games are also introduced in order to evaluate the macroscopic equilibria achieved.

The following is a summary list of the contributions of this thesis:

- P2P data storage architecture: organization principles for security mechanisms at various layers of the system, and interest of introducing a trusted computing base as a security infrastructure.
- Cryptographic protocols for remote data possession verification
  - Probabilistic-based approach: realizes a good performance by conceding verification determinism, and allows open verifiability of the stored data.
  - Restricted deterministic approach: achieves an efficient verification trading off security and performance with verification periodicity (availability).
  - Deterministic-based approach: realizes a good performance to security tradeoff.
- P2P data storage and maintenance mechanism: introduction of a reactive data rejuvenation process in order to achieve storage reliability and availability on the long term. The process relies on the operation of an erasure code based data maintenance protocol.
- Cooperation incentive mechanisms: open and scalable reputation-based and remuneration-based mechanisms that do not require a trusted infrastructure, and are resilient to various attacks.
- Game theoretical models: validate the incentive property of proposed mechanisms at micro and macroscopic levels of granularity.

Perspectives

Our work presented primitives for evaluating the behavior of peers with respect to storage. The feedback resulting from such evaluations mainly serves cooperation incentive mechanisms. However, peers, in particular data owners, also need to adapt their storage strategies based on such evaluations. Detecting a storage fault should trigger a data regeneration process to ensure the long-term reliability of data storage. However, the effectiveness of such a process not only depends on the availability of enough holders, as we modeled it, but also on the time it takes to transfer data blocks between peers. A performance analysis of such a process would certainly bring more realistic estimations as to the bandwidth and churn requirements of a P2P storage application.

The security mechanisms developed in this thesis, and in particular cooperation incentives, are crucial in forecasting how trusted a peer can be and in stimulating its cooperation. Although they were tuned for P2P data storage in this work, other P2P applications (say for instance P2P IP telephony) would definitely benefit from such security and cooperation mechanisms. For instance, Internet providers are deploying Wifi relays for IP telephony with the cooperation of end-users that accept to configure their ADSL boxes to carry this service in exchange of the capability to use it. A finer grained yet self-organizing regulation of such infrastructures might be achieved with remuneration-based incentives in particular. Wuala for instance has started deploying its data storage infrastructure with such an approach. Remuneration-based cooperation incentives also pave the way for multi-service architectures that would then make it possible for heterogeneous platforms to cooperate efficiently and exchange some bandwidth for some storage for instance.

Protection against Sybil attackers and whitewashers is a central issue in many P2P applications. It should be noted that completely self-organized approaches can only mitigate such attacks while at the same time imposing a penalty on honest peers. We discussed the use of a trusted computing base, as provided by some tamper-resistant hardware, as a possible solution. Although costly in terms of deployment, it may indeed provide an interesting and scalable solution to this problem. In particular, the TCG architecture is increasingly deployed in corporate hardware, thus making it an interesting candidate. In particular direct anonymous

attestation mechanisms may link some data to a unique platform while preserving platform privacy. There is also an increasing trend to establish dynamic trust based on existing static trust relationships, as illustrated with the emergence of services based on social networks (e.g., Skype, Facebook, hi5, LinkedIn, MySpace). In such systems, small groups of peers may easily be established based on the graph of relationships. Dunbar's rule determines that a given peer can maintain stable social relationships with 150 other peers. This may mean that P2P applications developed in the future may exhibit topologies very different from those used in P2P file sharing in which a peer may connect with 3000 others, as witnessed within BitTorrent "swarms" for instance. Scalability will undoubtedly remain an important research challenge in such systems as well and may trigger the development of more efficient protocols for managing the interconnection of multiple groups of well connected peers.

Appendix A Diffie-Hellman based deterministic verification

We propose a second deterministic verification approach that is based this time on the hardness of the Diffie-Hellman problem: finding the value of g^{xy} given an element g a generator of a multiplicative group (typically a finite field or an elliptic curve group) and the values of g^x and g^y.

Tree-based number generation

We work in a group G of prime order p with generator g. The protocol relies on the idea that l number of values allows deriving n number of values where $n > l$. We employ a binary tree to generate these values in a top-down manner, where the values consist of the leaves, and the generator g is at the root.

The tree construction is defined as (see

Figure **61** for an example): at tree level i, the value of the child in the left is equal to the value of its parent, whereas the value of the child on the right is the value of its parent multiplied by g^{x^i} where x is a random number in $\mathbb{Z}^*_{p-1}$ (the value of x is chosen such that the values on the leaves are all distinct).

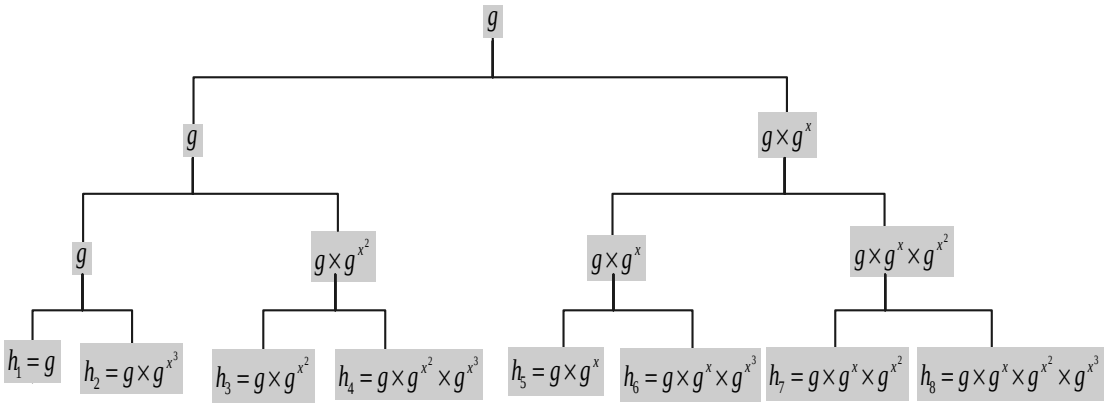


Figure 61 Tree-based number generation. $n=2^3$.

From the proposed tree construction, we are able to generate n distinct numbers just knowing g and x and performing $O(n)$ exponentiations.

An original property of the tree-based construction is that if we consider a generator of the form g^r, then we obtain numbers of the form h_i^r where h_i is a number generated from the tree at root g. This property is obtained due to the fact that the only operation that is carried out all along the tree is the multiplicative operation in the group G that is commutative.

Protocol description

The verification protocol comprises the following phases (summarized in Figure 62):

- *Storage phase:* the owner splits its data into n ($n=2^l$) blocks (not necessarily with the same size) of size less than $|p\text{-}1|$ ("$|M|$" means size of M). All blocks can be easily mapped to random numbers $\{d_i\}_{1\leq i\leq n}$ in $\mathbb{Z}^*_{p-1}$. The blocks are sent to the appointed holder for the data storage. This latter should keep the blocks stored until the time of their retrieval by the owner. The holder should also keep the order of block indexes unchanged.
- *Delegation phase:* the owner chooses a random number x that will be used to construct tree-based random numbers as explained in the previous section. Generated random

numbers are denoted $\{h_i\}_{1\leq i\leq n}$. Then, the owner computes the value $T=\Pi_{i=1}^{n} h_i^{d_i}$. This value is sent to the verifier assigned to the holder of $\{d_i\}_{1\leq i\leq n}$, along with the secret x and the generator g. The verifier should keep the random number x secret.

- *Verification phase:* the verifier generates a random number r, then it computes the numbers $\{(g^{x^i})^r\}_{0\leq i\leq l}$. where l is the height of the tree. These numbers are sent to the holder to be used to generate random numbers $\{h'_i\}_{1\leq i\leq n}$ based on the tree construction with root g^r that actually results in $h'_i=(h_i)^r$ for each i. The holder then computes the product $R=\Pi_{i=1}^{n} h'_i{}^{d_i}$ that will be sent back to the verifier. This latter has just to check whether the equality $R=(T)^r$ holds or not.

Storage	**Owner**		**Holder**
	Split data d in n chunks: $\{d_i\}_{1\leq i\leq n}$ send $\{d_i\}_{1\leq i\leq n}$	$\{d_i\}_{1\leq i\leq n}$ →	Store $\{d_i\}_{1\leq i\leq n}$
Delegation	**Owner**		**Verifier**
	Generate random number x Generate n numbers $\{h_i\}_{1\leq i\leq n}$ based on x and g Compute: $T= \Pi_{i=1}^{n} h_i^{d_i}$ Send T, x, g	T, x, g →	Store T, x, g
Verification	**Verifier**		**Holder**
	Generate random number r Compute $g'i =(g^{x^i})^r$ for each i in [1, l] Send $\{g'_i\}_{1\leq i\leq l}$	$\{g'_i\}_{1\leq i\leq l}$ →	Generate $\{h'_i\}_{1\leq i\leq n}$ from $\{g'_i\}_{1\leq i\leq l}$ Compute $R = \Pi_{i=1}^{n} h'_i{}^{d_i}$ Send R
	If $R=(T)^r$ then "accept" else "reject"	← R	

Figure 62 Deterministic verification protocol

Security analysis

The commutative property of the multiplicative group G produces random numbers from the generator g^r of this form $h'_i=(h_i)^r$ for each i in [1, n] ($\{h_i\}_{1\leq i\leq n}$ is the set of numbers produced from the generator g) which results in the equality that the verifier checks:

$$R = \prod_{i=1}^{n} h'_i{}^{d_i} = \prod_{i=1}^{n} ((h_i)^r)^{d_i} = \prod_{i=1}^{n} ((h_i)^{d_i})^r = T^r$$

Additionally, the holder is not able to compute the response to the verifier's challenge without knowing the data blocks. Actually, it cannot infer the random number r or the secret number x from the distinct received numbers $\{(g^{x^i})^r\}_{0\leq i\leq l}$ thanks to the Diffie-Hellmann problem. The tree-based approach produces distinct generated random numbers. The owner and after the verifier may check this property by choosing the right values for x and for r. Therefore, the holder receives distinct numbers that it cannot deduce from them the secret numbers x and r assuming the Diffie-Hellman problem hard to resolve.

Performance analysis

In the proposed verification protocol, the verifier should keep small verification information that consists of the secret x, the generator g and the number $T \in \mathbb{Z}_p$. The holder should keep the data blocks without any additional storage overhead.

The challenge message is composed of a set of l random numbers where l is the height of the tree then $l=log_2(n)$. The response message consists of a random number in $\mathbb{Z}_p$.

The main weakness of the protocol is the computation complexity. The verification process entails l exponentiations in $\mathbb{Z}^*_{p-1}$ and l exponentiations in $\mathbb{Z}_p$ at the verifier side. On the other hand, the tree-based generation of random numbers requires l exponentiations in $\mathbb{Z}^*_{p-1}$ and n exponentiations in $\mathbb{Z}_p$. Moreover, the holder performs other n exponentiations in $\mathbb{Z}_p$ using the data blocks.

Table 9 Summary of resource usage of the deterministic verification protocol (*n* corresponds to data size)

	Storage usage	**Computation complexity**	**Communication overhead**
At holder	$O(n)$	$O(n)$	(upstream) $O(1)$
At verifier	$O(1)$	$O(log(n))$	(upstream) $O(log(n))$

Appendix B Managing whitewashers

An inherent problem to a cooperation incentive mechanism implemented into a dynamic system where peers may join or leave at any time is the whitewashing problem. Whitewashers are peers that repeatedly misbehave then leave the storage system and rejoin with new identities thus escaping punishment imposed by the incentive mechanism. In order to deal with such whitewashers, the paper presents a penalty mechanism against strangers that attempts to counter whitewashers and it describes also a theoretical game that models such mechanism and attempts to capture the point of tradeoff between restricting whitewashers and encouraging newcomers to participate into the system.

Whitewashing problem

The proposed P2P storage system relies first and foremost on holder and verifier cooperation to properly function. Therefore, it may be exposed to several attacks due to peer misbehavior such as data destruction or corruption or even collusion between peers. Peer collusion can be mitigated through proper selection of data holders and verifiers. For instance, the random selection of peers within a structured P2P system limits pre-set collusions among these peers (for details refer to 0). On the other hand, peer participation and data preservation can be stimulated thanks to the use of cooperation incentive mechanisms.

Still, such mechanisms are vulnerable to whitewashers that repeatedly leave the storage system and rejoin with new identities thus escaping any punishment caused by their previous misbehavior. With new identities, peers have a clean record: good reputation rating or a default initial amount of credits without debts to pay.

Particularly in a so open and dynamic P2P system where peers are able to freely join, disconnect, reconnect, and leave the system, whitewashing becomes an eminent attack. Such attack undermines the operation of the cooperation incentive mechanism since whitewashers are not motivated to cooperate because otherwise they are not punished and they are eventually cutting down the utilization of their storage resources: they consume but do not contribute. Without peer cooperation, the system may collapse in the tragedy of the commons [Hardin 1968].

Penalty over newcomers

There are several solutions to the whitewashing problem. The first approach relies on a central trusted authority that assigns strong identities to peers (linked to real-world identities). Alternatively, the authority may impose the payment of membership fees. However, additionally to introducing a single point of failure, such approach reduces the decentralized nature of P2P systems.

Without a trusted third party, another option is to impose penalties on all newcomers: an insider peer may only probabilistically cooperate with newcomers (like in BitTorrent [Piatek et al. 2007]), or peers may join the system only if an insider peer with limited invitation tickets invites them [Lesueur et al. 2008]. This option seems to be detrimental to the scalability of the system; it has even been shown that this degrades the total social welfare [Feldman et al. 2006] because whitewashing behavior is not observable and thus the penalty affects all newcomers either cooperative ones or whitewashers.

This appendix studies the latter solution. The countering measure against whitewashing consists of a penalty mechanism. The imposed penalty is performed by each peer that does not cooperate with strangers with probability 1-p. The penalty may be also represented as service degradation by (1-p)-fraction imposed on each newcomer. The probabilistic strategy attempts to reach a point of tradeoff between restricting whitewashers and encouraging newcomers to join and participate into the system.

In the proposed P2P storage system, the penalty mechanism corresponds to making each peer accept to store or verify a newcomer's data only with some given probability *p*.

In the remainder of this appendix we will present a game theoretical model describing the features of a P2P storage system and capturing the whitewashing problem in such system. We endeavor with such model to discuss the ability of the strategy based on the probabilistic cooperation with strangers in coping with whitewashers.

Game model

We consider the evolutionary game of the audit-based approach described in 6.3. We may analogously make correspond a whitewasher to some defectors with probability *w*, $AllD^w$, and a probabilistic stranger strategy to a discriminator D^p that will only cooperate with peers that it does not know with probability *p*.

Strategies

Our study considers two types of strategies: the peers that follow the desired behavior in the P2P storage system and particularly use the penalty mechanism to deal with strangers, and the peers that defect and whitewash.

Discriminators are peers that adhere to the following strategy (corresponding to the audit-based strategy in 5.2):

- Discriminate and probabilistically cooperate with strangers (D^p): the discriminator donates under conditions: it donates with probability *p* with a stranger and probability 1 with a peer that previously donated. A discriminator may know that a peer has donated in a previous game in the case where that peer was a holder and the discriminator was its verifier or the owner of the data the peer was storing.

Defectors are peers that not only defect but also probabilistically whitewash to cover up their defection:

- Always defect and probabilistically whitewash ($AllD^w$): the peer never donates in the role of the donor and may be a whitewasher with probability *w* so that it is not identified by a discriminator. The value *w* may represent the average rate (per generation) at which defectors change identities.

Fitness

We respectively denote the frequency (fitness) of strategies $AllD^w$ by *y*, and D^p by *z*. The expected values for the total payoff obtained by the two strategies are denoted by $U_{AllD}{}^w$ and $U_D{}^p$, and the average payoff in the population by:

$$\overline{U} = y \times U_{ALLD^w} + z \times U_{D^p}$$

To simplify the formulation of the fitness for each strategy, we will use the following functions:

$$f(u) = -c(r + m\alpha) \times u$$
$$g(u) = b(1 - u^r + \beta u^r(1 - u^m))$$

The function $f(u)$ gives the cost paid by a peer for storing and verifying data for a fraction *u* of peers. On the other hand, the function $g(u)$ gives the benefit obtained if a fraction *u* of peers defect as holders and as verifiers of the peer's data.

Let q be the probability that the discriminator knows what a randomly chosen co-player chose as a holder strategy in a previous one-stage game (the discriminator being an owner or verifier in that game). The probability q is computed in 6.3.2.

A peer playing the strategy $AllD^w$ will never cooperate, so it will never donate. It will gain a benefit b if it is chosen as an owner and at least one of its data holders is not any of these types: a defector or a discriminator that knows the peer or that probabilistically defects because either it does not know the peer or the peer itself is a whitewasher. Otherwise, the peer may gain a benefit βb if at least one of its verifiers is not of any of the former two types.

$$U_{ALLD^w} = g\big(y + q(1-w)z + \big(w + (1-q)(1-w)\big)(1-p)z\big)$$
$$= g(1 - p(1 - q(1-w))z)$$

A peer playing the strategy D^p will cooperate if the recipient was cooperative in a previous interaction or will probabilistically cooperate if it does not know the latter. It will donate at a cost $-c$ if it is chosen as a holder or at a cost $-\alpha c$ if it is chosen as a verifier. It will gain a benefit b if it is chosen as an owner and at least one of its data holders is not a defector or a discriminator that the peer previously defects with it (the peer defects with a fraction p of discriminators that it does not know), otherwise, it may gain a benefit βb if at least one of its verifiers is not a defector or again a discriminator that the peer previously defects with it.

$$U_{D^p} = f\left(p\left((1-q)\big((1-w)y + z\big) + wy\right) + qpz\right) + g(y + (1-p)z)$$
$$= f\big(p(1 - q(1-w)(1-z))\big) + g(1 - pz)$$

The dynamics of strategies' fitness follow the differential replicator equations defined below:

$$\frac{dy}{dt} = y(U_{AllD^w} - \overline{U})$$
$$\frac{dz}{dt} = z(U_{D^p} - \overline{U})$$

The basic concept of replicator dynamics is that the growth rate of peers taking a strategy is proportional to the fitness acquired by the strategy. Thus, the strategy that yields more fitness than average fitness of the whole system increases, and vice versa.

Simulation experiments

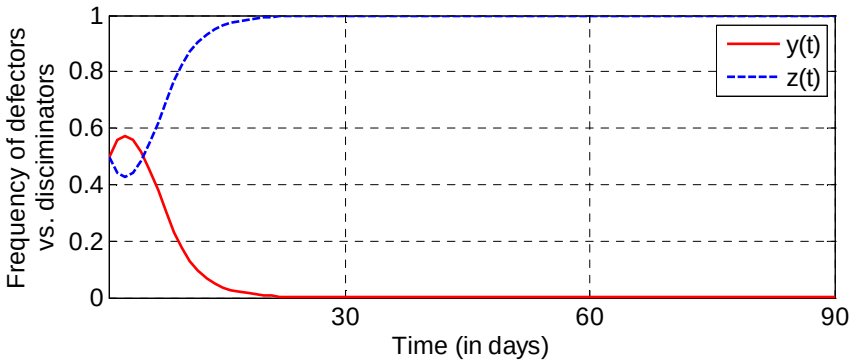


Figure 63 Frequency of defectors and discriminators. m=5, r=3, β=0.1, α=20.10^{-6}, λ=10/month, N=1000, γ=3 files/day, b=1, c=0.01, $y(0)$=0.5, and $z(0)$=0.5.

Using the above differential equations, the model is simulated within several scenarios to capture the impact of various parameters on the convergence of the system to an equilibrium.

We consider files with an average size of 500MB that are stored at a rate of 3 files per day and per peer. The verification metadata corresponding to each file has an average size of 10KB. Newcomers to the storage system arrive at a rate of 10 peers per month. These newcomers are assumed detaining the same strategy as their hosts.

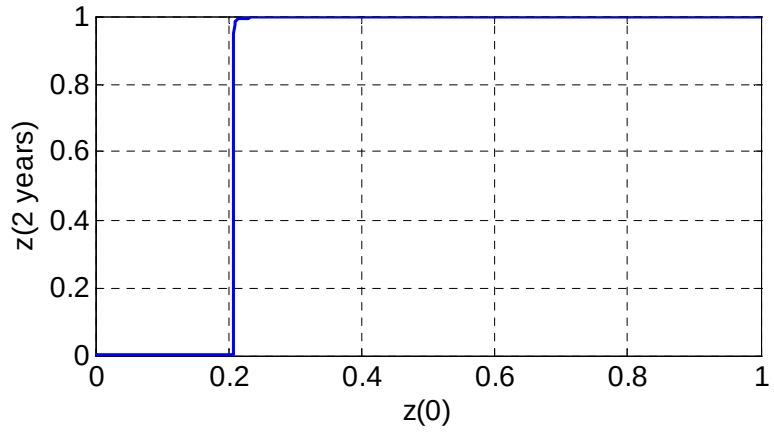


Figure 64 Frequency of discriminators at equilibrium varying their initial frequency. m=5, r=3, β=0.1, α=20.10^{-6}, λ=10/month, N=1000, γ=3 files/day, b=1, c=0.01.

Figure 63 shows the convergence of the storage of the system to an equilibrium where only discriminators are active. Defectors are totally eliminated by discriminators.

There is a little increase in the population of defectors in the beginning of the evolutionary game due to the fact that discriminators are still not able to distinguish between a discriminator and a defector. However, with time they have a good knowledge of discriminators (fraction p of them) and defectors (fraction (1-w) of them).

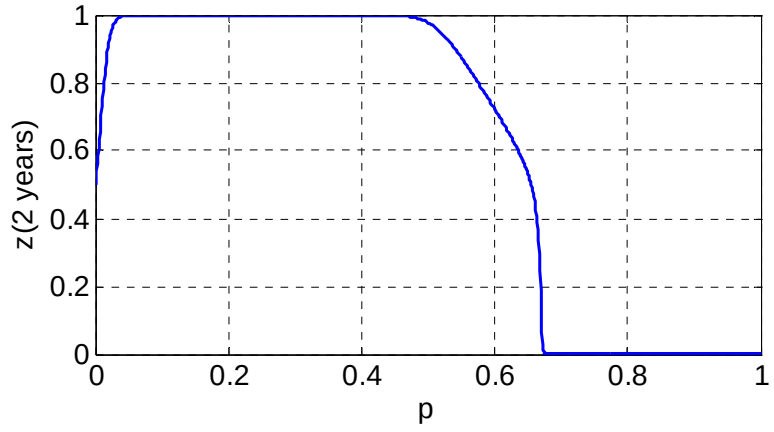


Figure 65 Frequency of discriminators at equilibrium varying their probability of cooperation with strangers p. m=5, r=3, β=0.1, α=20.10^{-6}, λ=10/month, N=1000, γ=3 files/day, b=1, c=0.01, $y(0)$=0.5, and $z(0)$=0.5.

Varying the probability of whitewashing *w* in the system affects also the frequency of discriminators at equilibrium. For sufficiently high *w*, defectors invade the population of discriminators and win the game. For instance, if all defectors are whitewashers, discriminators are eliminated from the game.

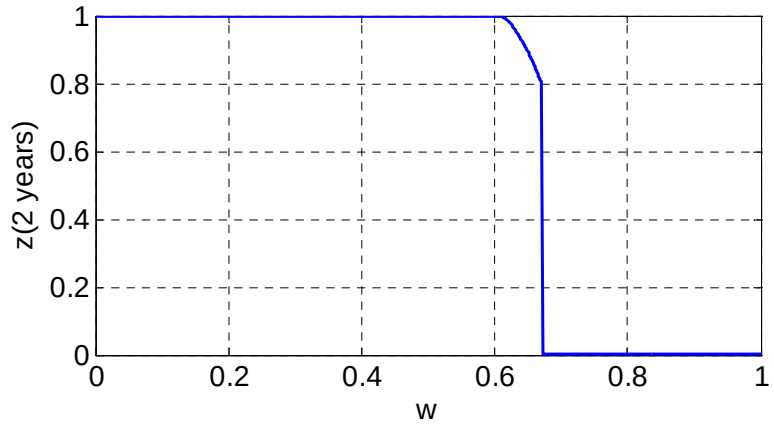


Figure 66 Frequency of discriminators at equilibrium varying the probability of whitewashing *w*. *m*=5, *r*=3, *β*=0.1, *α*=20.10^{-6}, *λ*=10/month, *N*=1000, *γ*=3 files/day, *b*=1, *c*=0.01, *y*(0)=0.5, and *z*(0)=0.5.

Figure 64 depicts the frequency of discriminators varying their initial frequency. The figure demonstrates that the equilibrium where only discriminators are present in the system is only achieved if there is enough population of discriminators in the system. Otherwise, the defectors win the game by eliminating discriminators.

The equilibrium where only discriminators are active depends also on the probability of cooperation of discriminators with strangers. Figure 65 demonstrates that if this probability is sufficiently high, the frequency of discriminators decreases and may attain zero.

The social welfare is the total sum of peer payoffs. It illustrates the well-being of the community of peers as a whole. Figure 67 shows that this welfare is maximized for a defined value of the probability of cooperation of discriminators with strangers *p* ($0.5<p<0.9$) and if the discriminators are not eliminated from the system (probability of whitewashing $w<0.7$).

Discriminators are the only contributors to the game therefore their presence increases the payoff of peers. Their cooperation may be undermined by the presence of defectors that use the system without contributing and particularly whitewashers that defect and go without being detected by the discriminators. Increasing *p*, it certainly increases the benefit for all peers but at the same time it increases the cost due to the presence of defectors. Figure 67.c demonstrates that there is an optimal value for *p* that achieves the highest social welfare and this optimal depends on *w*.

Figure 68 depicts the variation of the social welfare with the replication rate *r* and the verification distribution factor *m*. The figure shows that there is an optimal value for the replication rate *r* for which the social welfare is maximized ($r\sim3$). Exceeding this value, the social welfare decreases until reaching the value zero i.e., the system collapses. Increasing *r* makes the benefit obtained by the owner increase since the chances to select a cooperative holder are improved; however the replication rate *r* affects also the cost of cooperation that is solely paid by discriminators.

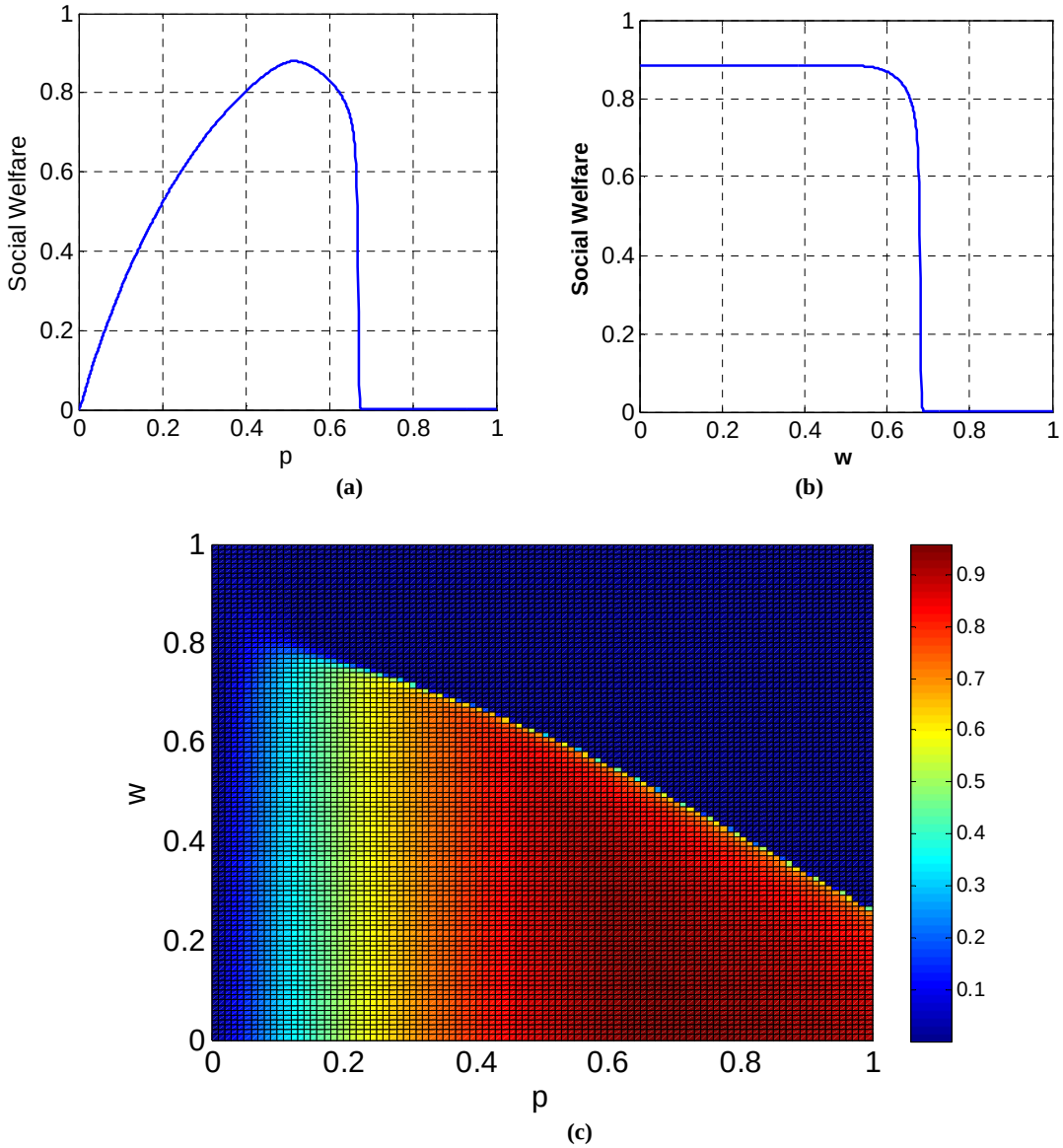


Figure 67 Social welfare at equilibrium varying (a) the probability of cooperation *p*, (b) probability of whitewashing *w*, and both of them. *m*=5, *r*=3, *β*=0.1, *α*=20.10^{-6}, *λ*=10/month, *N*=1000, *γ*=3 files/day, *b*=1, *c*=0.01, *y*(0)=0.5, and *z*(0)=0.5.

Varying *m* has less impact on the social welfare because the cost charged on discriminators is minimized by the significantly low unit cost value αc (α=20.10^{-6}). The social welfare increases by increasing *m* (small increase) since a high value of *m* means better chances to have a verifier that is discriminator and then gain a benefit βb if all holders are defectors.

Figure 69 demonstrates that there is a maximum value for tolerable churn. If peers arrive in the system at a high rate, discriminators may not be able to distinguish sufficiently quickly defectors and they may then be eliminated from the system. Churn can be tolerated until a given rate identified in the figure (λ~0.9) for the considered system parameters.

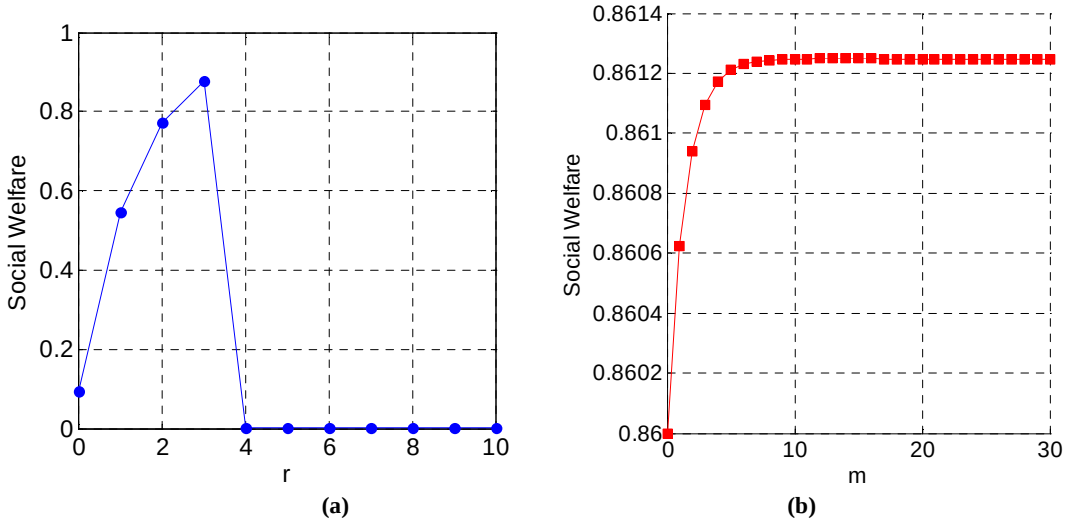


Figure 68 Social welfare at equilibrium varying (a) replication rate r (m=5) and (b) verification distribution factor m (r=3). β=0.1, α=20.10^{-6}, λ=10/month, N=1000, γ=3 files/day, b=1, c=0.01, p=w=0.5, y(0)=0.5, and z(0)=0.5.

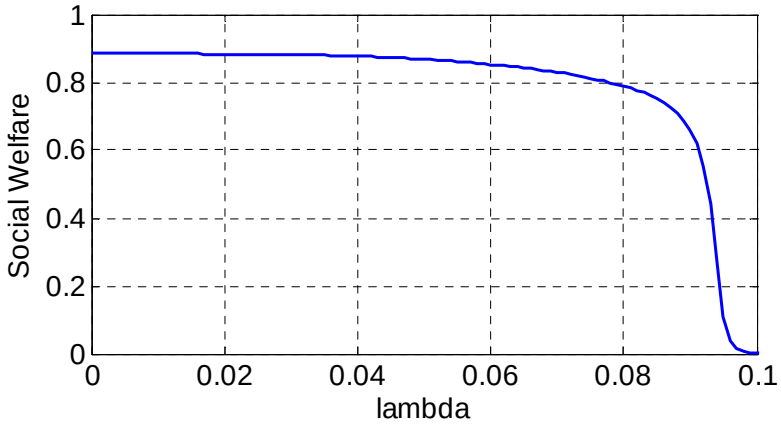


Figure 69 Social welfare at equilibrium varying the churn λ. m=5, r=3, β=0.1, α=20.10^{-6}, N=1000, γ=3 files/day, b=1, c=0.01, p=w=0.5, y(0)=0.5, and z(0)=0.5.

Discussion

Simulation results demonstrate that discriminators are not hopeless in front of defectors and that even they may win over them for a judicious choice of system parameters, notably the fraction of discriminators in the system should be initially not null, the replication rate and the churn sensed in the system should not be considerably high.

The results show also that there is an optimal probability p for the penalty mechanism that achieves a high social welfare for the whole P2P storage system. However, a non-zero welfare is only obtained if the whitewashing phenomena is restricted to a given fraction of defectors. For instance, if all defectors are whitewashing discriminators are entirely eliminated and the system collapses. This result motivates the requirement to supplement the proposed penalty

mechanism with other means that prevent or at least limit the whitewashing behavior such as controlling the peers that join the system using a cryptographic puzzle [Vishnumurthy et al. 2003] the payment of a membership fee. Another solution is to force or motivate peers to stay online a minimum amount of time in the system like in Wuala[53] (1/*w* is then increased) because peer connection time must be taken into consideration.

[53] http://wua.la/en/home.html

Appendix C Dissymmetric peer defection

Peers in the P2P network are autonomous and may behave in various ways. It is interesting to consider strategies where a peer wants to make others believe that it behaves well while minimizing its network cost. A peer following such strategy will properly store assigned data, will even correctly answer to verifiers, but to save network bandwidth, it will not verify other's data.

From the evolutionary game model described in Section 6.3.1 of Chapter 6, we may think of the interactions of three types of populations:

- Storage defectors (denoted *SD*) are peers that have reduced storage resources and do not consume them within the P2P storage application (they rather prefer defecting); even though they cooperate in verifying others' data.
- Verification defectors (denoted *VD*) are peers that have more interest in minimizing their bandwidth consumption than optimizing their storage resources. Therefore, they correctly store other peers' data but defect when being verifiers of some others data.
- Discriminators (denoted *D*) are peers that only cooperate with peers that either they do not know yet or peers that were previously cooperative holders. If they cooperate, they correctly keep the data that they have promised to store and also periodically check other peers' data.

We designate the fitness of *SD*, *VD*, and *D* strategies respectively *x*, *y*, and *z*. Their respective payoffs can be respectively derived U_{SD}, U_{VD}, and U_D. The total payoff is given as:

$$\overline{U} = x \times U_{SD} + y \times U_{VD} + z \times U_D$$

We employ the following function g that gives the benefit gained by a peer having in average a potential fraction u of peers that do not store its data and a potential fraction v of peers that do not verify its data:

$$g(u,v) = b(1 - u^r + \beta u^r(1 - v^m))$$

Storage defectors are only charged the costs of verification. They may gain a benefit b from the storage application if their co-players are not storage defectors or discriminators that know that they are defectors; otherwise they may gain a benefit βb if their co-players are not verification defectors or again discriminators that know their type.

$$U_{SD} = -cm\alpha + g(x + qz, y + qz)$$

The costs paid by verification defectors are storage costs. They may gain a benefit if their co-players are not defectors.

$$U_{VD} = -cr + g(x,y)$$

Discriminators pay the storage and verification costs relative to all peers except storage defecting peers that have been detected. They may gain a benefit if their co-players are not defectors.

$$U_D = -c(r + m\alpha) + g(x,y)$$

To study the dynamics of their strategies, we rely on the replicator dynamics as seen in Section 6.3.4 of Chapter 6. First of all, we consider simple cases where they are only one type of defectors at a time.

Case $x(0)=0$: we obtain the following differential equation:

$$\frac{dy}{dt} = yz \times cm\alpha > 0$$

The above inequality means that verification defectors always win over discriminators because they pay small costs compared to discriminators (only storage costs) and because also they are not punished for their defection (punishment concerns only non cooperative holders).

Case $y(0)=0$: the payoff of discriminators is the same as 6.3.3 of Chapter 6 but with $m=0$. The payoff of storage defectors resembles to that of defectors of 6.3.3 even though the chances to obtain the benefit βb are improved. Storage defectors and discriminators may coexist at a certain equilibrium that depends on system parameters.

Case $z(0)=0$: the following differential equation is obtained:

$$\frac{dx}{dt} = xy \times c(r - m\alpha) > 0$$

Since the verification costs are generally less important than storage costs (the size of the metadata needed for verification is smaller than the data), the above inequality is generally held. This means that storage defectors win over verification defectors. Tough these costs are exclusively paid by each population in this particular case. Therefore, the relative costs may be perceived differently (e.g., $r < m\alpha$) and then the above inequality may not be obtained.

Case $x(0)\neq0$, $y(0)\neq0$, and $z(0)\neq0$: Figure 70 depicts a simulation of the dynamics of the three strategies with non null initial frequencies (the same system parameters are taken as 6.3.6). The figure demonstrates that storage defectors at first are the most reproductive (their frequency increases more importantly than the other strategies). They even eliminate verification defectors and reduce the frequency of discriminators, but these later catch up thanks to the growth of their knowledge about the behavior of defectors. With time, discriminators are able to distinguish defectors from cooperators and subsequently refuse to cooperate with these defectors. Their costs are then reduced, which allow them to increase in frequency at the expense of storage defectors.

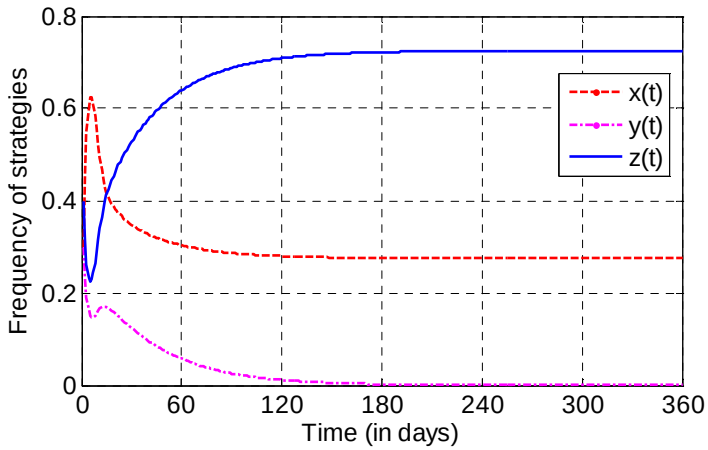


Figure 70 Frequency of strategies over time. m=5, r=3, β=0.1, α=20.10^{-6}, λ=10/month, N=1000, γ=3 files/day, b=1, c=0.01, $x(0)$=0.3, $y(0)$=0.3, and $z(0)$=0.4.

The study of dissymmetric peer behavior shows that peers that focus on reducing their bandwidth utilization may win over cooperative peers since their contributions incur lower

costs. Although, cooperative peers rely on direct experiences to compute reputation, they may be affected by this type of defectors in ensuring the security properties of their remote data. The data stored in the system will only be periodically verified by their owners at game equilibrium. On the other hand, peers that do not contribute with storage resources are detected and punished by the reputation system.

Bibliography

[Acedański et al. 2005] Szymon Acedański, Supratim Deb, Muriel Médard, and Ralf Koetter. How good is random linear coding based distributed networked storage?. In Proceeding of 1st Workshop on Network Coding, WiOpt 2005 Riva del Garda, Italy, April 2005.

[Anceaume and Ravoaja 2006] Emmanuelle Anceaume and Aina Ravoaja. Incentive-Based Robust Reputation Mechanism for P2P Services. Research Report PI 1816 (2006), IRISA, http://hal.inria.fr/inria-00121609/fr/

[Asokan et al. 1997] N. Asokan, Matthias Schunter, and Michael Waidner. Optimistic Protocols for Fair Exchange. In Proceedings of the 4th ACM Conference on Computer and Communications Security, Zurich, April 1997.

[Asokan et al. 1998] N. Asokan, Victor Shoup, and Michael Waidner. Asynchronous protocols for optimistic fair exchange. In Proceeding of the IEEE Symposium on Security and Privacy, 1998, 3-6 May, p. 86-99, Oakland, CA, USA.

[Ateniese et al. 2007] Giuseppe Ateniese and Randal Burns and Reza Curtmola and Joseph Herring and Lea Kissner and Zachary Peterson and Dawn Song. Provable data possession at untrusted stores. In Proceedings of the 14th ACM conference on Computer and communications security, ACM, 2007, 598-609.

[Ateniese et al. 2008] Giuseppe Ateniese, Roberto Di Pietro, Luigi V. Mancini and Gene Tsudik. Scalable and Efficient Provable Data Possession. In Proceedings of the 4th International Conference on Security and Privacy in Communication Networks (SecureComm'08).

[Balfe et al. 2005] Shane Balfe, Amit D. Lakhani and Kenneth G. Paterson. Trusted Computing: Providing security for Peer-to-Peer Networks. In Proceedings of the 5th International Conference on Peer-to-Peer Computing (P2P), 2005.

[Beigl 2000] Michael Beigl. MemoClip: A location-based remembrance appliance. Personal and Ubiquitous Computing Vol. (4), 230-233, Springer-Verlag, September 2000.

[Bellare et al. 1995] Mihir Bellare, Oded Goldreich and Shafi Goldwasser. Incremental Cryptography and Application to Virus Protection. In Proceedings of the 27th annual ACM symposium on Theory of computing, p.45-56, May 29-June 01, 1995, Las Vegas, Nevada, United States.

[Bennett et al. 1994] Frazer Bennett, Tristan Richardson, and Andy Harter. Teleporting - making applications mobile. In Proceedings of IEEE Workshop on Mobile Computing Systems and Applications, pages 82-84, Santa Cruz, California, December 1994. IEEE Computer Society Press.

[Bhagwan et al. 2004] Ranjita Bhagwan, Kiran Tati, Yu-Chung Cheng, Stefan Savage, and Geoffrey M. Voelker. TotalRecall: System Support for Automated Availability Management. In Proceedings of the 1st ACM/USENIX

Symposium on Networked Systems Design and Implementation (NSDI), San Francisco, CA, March 2004.

[Bishop 2003] Matt Bishop. Computer Security: The Art and Science. Addison-Wesley, Jun 2003.

[Bloom 1970] Burton Bloom. Space/time trade-offs in hash coding with allowable errors. In Communications of the ACM, 13(7), pp. 422-426, July 1970.

[Blum et al. 1994] Manuel Blum, William S. Evans, Peter Gemmell, Sampath Kannan, and Moni Naor. Checking the Correctness of Memories. In 32nd Annual Symposium on Foundations of Computer Science, pages 90-99, San Juan, Puerto Rico, 1-4 October 1991.

[Bolton and Ockenfels 2000] Gary E Bolton and Axel Ockenfels. ERC: a theory of equity, reciprocity, and competition. American Economic Review 2000, vol. 90, pp. 166-193.

[Brandt and Sigmund 2006] Hannelore Brandt and Karl Sigmund. The good, the bad and the discriminator--errors in direct and indirect reciprocity. Journal of Theoretical Biology, Volume 239, Issue 2, 21 March 2006, Pages 183-194.

[Buttyán and Hubaux 2001] Levente Buttyan and Jean-Pierre Hubaux. Nuglets: a virtual currency to stimulate cooperation in self-organized ad hoc networks. Technical report, EPFL, 2001.

[Buttyán and Hubaux 2003] Levente Buttyan and Jean-Pierre Hubaux. Stimulating Cooperation in Self-Organizing Mobile Ad Hoc Networks. ACM/Kluwer Mobile Networks and Applications (MONET) Special Issue on Mobile Ad Hoc Networks, vol. 8, no. 5, October 2003.

[Carbone et al. 2003] Marco Carbone, Mogens Nielsen, and Vladimiro Sassone. A Formal Model for Trust in Dynamic Networks. BRICS Technical Report RS-03-4, University Aarhus, 2003.

[Caronni and Waldvogel 2003] Germano Caronni and Marcel Waldvogel. Establishing Trust in Distributed Storage Providers. In Proceedings of 3rd IEEE International Conference on P2P Computing, Linkoping, Sweden, September 2003.

[Castro et al. 2002] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron and Dan S. Wallach. Secure routing for structured peer-to-peer overlay networks. Symposium on Operating Systems and Implementation, OSDI'02, Boston, MA, December 2002.

[Chang and Xu 2008] Ee-Chien Chang and Jia Xu. Remote Integrity Check with Dishonest Storage Server. 13th European Symposium on Research in Computer Security (ESORICS), 2008.

[Cohen 2003] Bram Cohen. Incentives build robustness in BitTorrent. In Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems, June 2003.

[Courtès 2007] Ludovic Courtès. Cooperative Data Backup for Mobile Devices. PhD Thesis, November 2007.

[Cox and Noble 2002] Landon P. Cox and Brian D. Noble. Pastiche: making backup cheap and easy. In Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation, Boston, MA, December 2002.

[Cox et al. 1995] Benjamin Cox, J. D. Tygar, and Marvin Sirbu. Netbill security and transaction protocol. In Proceedings of the 1st USENIX Workshop in electronic commerce. 77–88, 1995.

[Desmedt and Frankel 1989] Yvo G. Desmedt and Yair Frankel. Threshold Cryptosystems. In Proceedings on Advances in cryptology, Santa Barbara, California, United States, pages: 307 – 315, 1989.

[Deswarte et al. 2004] Yves Deswarte, Jean-Jacques Quisquater, and Ayda Saïdane. Remote Integrity Checking. In Proceedings of 6th Working Conference on Integrity and Internal Control in Information Systems (IICIS), 2004.

[Dey and Abowd 2000] Anind K. Dey and Gregory D. Abowd. CybreMinder: A context-aware system for supporting reminders. In Proceedings of 2nd International Symposium on Handheld and Ubiquitous Computing, HUC 2000, pages: 172-186, Bristol, UK, September 2000. Springer Verlag.

[Dingledine 2000] Roger R. Dingledine. The Free Haven project: Design and deployment of an anonymous secure data haven. Master's thesis, MIT, June 2000.

[Douceur 2002] John R. Douceur. The Sybil attack. In Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02). MIT Faculty Club, Cambridge, MA, 2002.

[Druschel and Rowstron 2001] Peter Druschel and Antony Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. In Proceedings of the 8th Workshop on Hot Topics in Operating Systems, p.75, May 20-22, 2001.

[Duminuco and Biersack 2008] Alessandro Duminuco and Ernst W. Biersack. Hierarchical codes: how to make erasure codes attractive for peer-to-peer storage systems. The 8th IEEE International Conference on Peer-to-Peer Computing (P2P'08), September 8th-11th, 2008, Aachen, Germany.

[Feldman and Chuang 2005] Michal Feldman and John Chuang. The Evolution of Cooperation under Cheap Pseudonyms. In Proceedings of the 7th International IEEE Conference on E-Commerce Technology (CEC'05), July 2005.

[Feldman et al. 2006] Michal Feldman, Christos Papadimitriou, John Chuang and Ion Stoica. Free-Riding and Whitewashing in Peer-to-Peer Systems. Selected Areas in Communications, IEEE Journal on, Vol. 24, No. 5. (2006), pp. 1010-1019.

[Filho and Barreto 2006] Décio Luiz Gazzoni Filho and Paulo Sérgio Licciardi Messeder Barreto. Demonstrating data possession and uncheatable data transfer. Cryptology ePrint Archive, Report 2006/150, 2006. http://eprint.iacr.org

[Franklin and Reiter 1997] Matthew K. Franklin and Michael K. Reiter. Fair exchange with a semi-trusted third party. In Proceedings of The 4th ACM Conference on computer and communications security, T. Matsumoto, Ed. Zurich, Switzerland, 1–6, 1997.

[Friedman 1998] Daniel Friedman. On economic applications of evolutionary game theory. Journal of Evolutionary Economics 8, pp. 15–43, 1998.

[Ghassemi 2006] Farhad Ghassemi. Signaling games, 2006. Available: http://www.cs.ubc.ca/~kevinlb/teaching/cs532a%20%202006/Projects/FarhadGhassemi.pdf.

[Glassman et al. 1995] Steve Glassman, Mark Manasse, Martín Abadi, Paul Gauthier, and Patrick Sobalvarro. The millicent protocol for inexpensive electronic commerce. In Proceedings of the 4th International World Wide Web Conference, December 1995.

[Godfrey et al. 2006] Brighten Godfrey, Scott Shenker, and Ion Stoica. Minimizing churn in distributed systems. Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM), 147-158, 2006.

[Goldschlag et al. 1999] David Goldschlag, Michael Reed, and Paul Syverson. Onion Routing for Anonymous and Private Internet Connections. Communications of the ACM, vol. 42, num. 2, February 1999.

[Golle at al. 2002] Philippe Golle, Stanislaw Jarecki, Ilya Mironov. Cryptographic Primitives Enforcing Communication and Storage Complexity. In Proceeding of Financial Cryptography, pages: 120-135, 2002.

[Golle et al. 2001] Philippe Golle, Kevin Leyton-Brown, Ilya Mironov. Incentives for Sharing in Peer-to-Peer Networks. In Proceedings of the 3rd ACM conference on Electronic Commerce, October 2001.

[Hardin 1968] Garrett Hardin. The Tragedy of the Commons. Science, Vol. 162, No. 3859, Issue of 13, December, 1968, pp. 1243-1248.

[Huang at al. 1999] Andrew C. Huang, Benjamin C. Ling, Shankar Ponnekanti, and Armando Fox. Pervasive computing: What is it good for?. In Proceedings of the ACM International Workshop on Data Engineering for Wireless and Mobile Access, pages 84-91, Seattle, WA, August 1999. ACM Press.

[Jakobsson et al. 2003] Markus Jakobsson, Jean-Pierre Hubaux, and Levente Buttyan. A Micro-Payment Scheme Encouraging Collaboration in Multi-Hop Cellular Networks. In Proceedings of Financial Cryptography, La Guadeloupe, January 2003.

[Jones and Sleeman 1983] Douglas Samuel Jones and B. D. Sleeman. Differential Equations and Mathematical Biology. London: Allen & Unwin, 1983.

[Jøsang and Ismail 2002] Audun Jøsang and Roslan Ismail. The Beta Reputation System. In Proceedings of the 15th, Bled Electronic Commerce Conference, Bled, Slovenia, June 2002.

[Jøsang et al. 2005] Audun Jøsang, Roslan Ismail, and Colin Boyd. A Survey of Trust and Reputation Systems for Online Service Provision. In Proceedings of Decision Support Systems, 2005.

[Juels and Kaliski 2007] Ari Juels and Burton S. Kaliski PORs: Proofs of retrievability for large files. Cryptology ePrint archive, June 2007. Report 2007/243.

[Kamvar et al. 2003] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The EigenTrust Algorithm for Reputation Management in P2P Networks. In Proceedings of the 12th International World Wide Web Conference, Budapest, May 2003.

[Koblitz 1987] Neal Koblitz. Elliptic curve cryptosystems. Mathematics of Computation, Volume 48, pages: 203-209, 1987.

[Koyama et al. 1991] Kenji Koyama, Ueli Maurer, Tatsuaki Okamoto, and Scott Vanstone. New Public-Key Schemes Based on Elliptic Curves over the Ring Zn. Advances in Cryptology - CRYPTO '91, Lecture Notes in Computer Science, Springer-Verlag, vol. 576, pp. 252-266, August 1991.

[Kubiatowicz et al. 2000] John Kubiatowicz, Davic Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, Ben Zhao. OceanStore: An architecture for global-scale persistent storage. In Proceedings of the 9th international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000), November 2000.

[Lai et al. 2003] Kevin Lai, Michal Feldman, Ion Stoica, and John Chuang. Incentives for Cooperation in Peer-to-Peer Networks. In Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems, UC Berkeley, Berkeley, California, USA, June 2003.

[Lee et al. 2006] Patrick P. C. Lee, John C. S. Lui and David K. Y. Yau. Distributed collaborative key agreement and authentication protocols for dynamic peer group. IEEE/ACM Transactions on Networking, 2006.

[Lesueur et al. 2007] François Lesueur, Ludovic Mé, Valérie Viet Triem Tong. Contrôle d'accès distribué à un réseau Pair-à-Pair. 2nd joint conference on Security in network architectures and information systems (SAR-SSI'2007), Annecy, France, June 12-15, 2007.

[Lesueur et al. 2008] François Lesueur, Ludovic Mé, and Valérie Viet Triem Tong**.** A Sybilproof Distributed Identity Management for P2P Networks In Proceedings of the 13th IEEE Symposium on Computers and Communications (ISCC) 2008, IEEE Computer Society, Marrakech, Morocco.

[Levine et al. 2006] Brian Neil Levine, Clay Shields, and N. Boris Margolin. A Survey of Solutions to the Sybil Attack. Technical Report 2006-052, University of Massachusetts Amherst, Amherst, MA, October 2006.

[Li and Dabek 2006] Jinyang Li and Frank Dabek. F2F: reliable storage in open networks. In Proceedings of the 5th International Workshop on Peer-to-Peer Systems (IPTPS), February 2006.

[Liang et al. 2006] Jian Liang, Rakesh Kumar, and Keith W. Ross. The FastTrack overlay: A measurement study. Computer Networks, vol. 50, no. 6, pp. 842-858, April 2006.

[Lillibridge et al. 2003] Mark Lillibridge, Sameh Elnikety, Andrew Birrell, Mike Burrows, and Michael Isard. A Cooperative Internet Backup Scheme. In Proceedings of the 2003 Usenix Annual Technical Conference (General Track), pp. 29-41, San Antonio, Texas, June 2003.

[Marmasse and Schmandt 2000] Natalia Marmasse and Chris Schmandt. Location-aware information delivery with ComMotion. In Proceedings of 2nd International Symposium on Handheld and Ubiquitous Computing, HUC 2000, pages 157-171, Bristol, UK, September 2000. Springer Verlag.

[Menezes et al. 1996] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. Handbook of Applied Cryptography. CRC Press, 1996. Available free (for personal use) on line from: http://www.cacr.math.uwaterloo.ca/hac/

[Michiardi 2004] Pietro Michiardi. Cooperation enforcement and network security mechanisms for mobile ad hoc networks. PhD Thesis, December 14th, 2004.

[Miller 1986] Victor Miller. Uses of elliptic curves in cryptography Advances in Cryptology. In Proceedings of Cryptology Conference (CRYPTO'85), Lecture Notes in Computer Science, 218 (1986), Springer-Verlag, 417-426.

[Mui et al. 2001] Lik Mui, Mojdeh Mohtashemi, Cheewee Ang, Peter Szolovits, and Ari Halberstadt. Ratings in Distributed Systems: A Bayesian Approach. In Proceedings of the Workshop on Information Technologies and Systems (WITS), 2001.

[Naor and Rothblum 2005] Moni Naor and Guy N. Rothblum. The Complexity of Online Memory Checking. In Proceeding of 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), pp. 573-584.

[Obreiter and Nimis 2003] Philipp Obreiter and Jens Nimis. A Taxonomy of Incentive Patterns - the Design Space of Incentives for Cooperation. Technical Report, Universität Karlsruhe, Faculty of Informatics, 2003.

[Page at al. 1998] Larry Page, Sergey Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical Report, Stanford Digital Library Technologies Project, 1998.

[Pham et al. 2000] Thai-Lai Pham, Georg Schneider, and Stuart Goose. Exploiting location-based composite devices to support and facilitate situated ubiquitous computing. In Proceedings of 2nd International Symposium on Handheld and Ubiquitous Computing, HUC 2000, pages 143-156, Bristol, UK, September 2000. Springer Verlag.

[Piatek et al. 2007] Michael Piatek, Tomas Isdal, Thomas Anderson, and Arvind Krishnamurthy. Do incentives build robustness in BitTorrent?. In Proceedings of the ACM/USENIX 4th Symposium on Networked Systems Design and Implementation (NSDI 2007), 2007.

[Popescu et al. 2004] Bogdan C. Popescu, Bruno Crispo and Andrew S. Tanenbaum. Safe and Private Data Sharing with Turtle: Friends Team-Up and Beat the System. In 12th International Workshop on Security Protocols, Cambridge, UK, April 2004.

[Ratnasamy et al. 2001] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In Proceedings of ACM/SIGCOMM, San Diego, CA, Aug. 27–31, 2001.

[Rivest 1997] Ronald L. Rivest. Electronic lottery tickets as micropayments. In Proceeding of Financial Cryptography, Lecture Notes in Computer Science vol. 1318, Springer Verlag (1997), pp. 307–314.

[Rivest and Shamir 1996] Ronald L. Rivest and Adi Shamir. Payword and micromint: two simple micropayment schemes. In Proceeding of Security Protocols Workshop, Lecture Notes in Computer Science, Vol. 1189, pp. 69--87, Springer-Verlag, 1997.

[Rowstron and Druschel 2001] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In Proceeding of the IFIP/ACMInternational Conference on Distributed Systems Platforms, Heidelberg, Germany, November 2001.

[Sandhu and Park 2003] Ravi Sandhu and Jaehong Park. Usage Control: A Vision for Next Generation Access Control. The 2nd International Workshop on Mathematical Methods, Models and Architectures for Computer Networks Security, 2003.

[Schwarz and Miller 2006] Thomas Schwarz, and Ethan L. Miller. Store, forget, and check: Using algebraic signatures to check remotely administered storage. In Proceedings of the IEEE Int'l Conference on Distributed Computing Systems (ICDCS '06), July 2006.

[Sebé et al. 2007] Francesc Sebe, Josep Domingo-Ferrer, Antoni Martínez-Ballesté, Yves Deswarte, and Jean-Jacques Quisquater. Efficient Remote Data Possession Checking in Critical Information Infrastructures. IEEE Transactions on Knowledge and Data Engineering, 06 Aug 2007. IEEE Computer Society Digital Library. IEEE Computer Society, 6 December 2007 http://doi.ieeecomputersociety.org/10.1109/TKDE.2007.190647

[Shacham and Waters 2008] Hovav Shacham and Brent Waters. Compact Proofs of Retrievability. In Proceedings of Asiacrypt 2008, Lecture Notes in Computer Science, Vol. 5350, pp. 90-107, Springer-Verlag, 2008.

[Sit and Morris 2002] Emil Sit and Robert Morris. Security Considerations for P2P Distributed Hash Tables.In Proceeding of the 1st Int'l Workshop Peer-to-Peer Systems (IPTPS), 2002.

[Solomon and Chapple 2005] Michael G. Solomon and Mike Chapple. Information Security Illuminated. Jones and Bartlett Publishers, Inc., USA, 2005.

[Spencer et al. 1999] Ray Spencer, Stephen Smalley, Peter Loscocco, Mike Hibler, David Andersen, and Jay Lepreau. The Flask security architecture: System support for diverse security policies. In Proceeding of the 8th USENIX Security Symposium, Washington, DC, USA, page 123139, USENIX Association, Berkeley, CA, USA, Aug 1999.

[Srivatsa and Liu 2005] Mudhakar Srivatsa and Ling Liu. Countering Targeted File Attacks using LocationGuard. In Proceedings of the 14th USENIX Security Symposium (USENIX Security), August 1 - 5, 2005, Baltimore, MD. pp. 81-96.

[Stoica et al. 2001] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In Proceedings of ACM SIGCOMM 2001, San Deigo, CA August 2001, pp. 149-160.

[Stutzbach and Rejaie 2004] Daniel Stutzbach and Reza Rejaie. Towards a Better Understanding of Churn in Peer-to-Peer Networks. Technical Report CIS-TR-04-06, University of Oregon, November 2004.

[Toka and Maillé 2007] Laszlo Toka and Patrick Maillé. Managing a peer-to-peer backup system: does imposed fairness socially outperform a revenue-driven monopoly?. 4th International Workshop on Grid Economics and Business Models (GECON 2007), August 28, 2007, Rennes, France, pp 150-163.

[Toka and Michiardi 2008] Laszlo Toka and Pietro Michiardi. Analysis of user-driven peer selection in peer-to-peer backup and storage systems. GameComm 2008, 2nd ACM-Valuetools International Workshop on Game theory in Communication networks, October 20, 2008, Athens, Greece, pp 428.

[Turocy and Stengel 2001] Theodore L. Turocy and Bernhard von Stengel. Game theory. Cdam Research Report lse-cdam-2001-09, London School of Economics, October 2001.

[Vishnumurthy et al. 2003] Vivek Vishnumurthy, Sangeeth Chandrakumar and Emin Gun Sirer. KARMA: A Secure Economic Framework for P2P Resource Sharing. In Proceedings of the Workshop on the Economics of Peer-to-Peer Systems, Berkeley, California, June 2003.

[Vogt et al. 2001] Holger Vogt, Henning Pagnia, and Felix C. Gärtner. Using Smart Cards for Fair Exchange. In Proceedings of the 2nd International Workshop on Electronic Commerce, Lecture Notes In Computer Science, Vol. 2232, p. 101 - 113, Springer-Verlag, 2001.

[Weatherspoon et al. 2005] Hakim Weatherspoon, Byung-Gon Chun, Chiu Wah So, and John Kubiatowicz. Long-term data maintenance in wide-area storage systems: A quantitative approach. Technical Report (2005) UCB/CSD-05-1404, EECS Department, University of California, Berkeley.

[Weyland et al. 2005] Attila Weyland, Thomas Staub and Torsten Braun. Comparison of Incentive-based Cooperation Strategies for Hybrid Networks. 3rd International Conference on Wired/Wireless Internet Communications (WWIC 2005), pp 169-180, ISBN: 3-540-25899-X, Xanthi, Greece, May 11-13, 2005.

[Yang and Molina 2003] Beverly Yang and Hector Garcia-Molina. PPay: Micropayments for Peer-to-Peer Systems. ACM Conference on Computer and Communications Security (CCS '03), Washington, DC, USA, October 2003.

[Yu et al. 2006] Haifeng Yu, Michael Kaminsky, Phillip B. Gibbons, and Abraham Flaxman. SybilGuard: defending against sybil attacks via social networks. IEEE/ACM Transactions on Networking (ToN), Volume 16, Issue 3, pp. 576-589, June 2008.

[Zhao et al. 2000] Ben Y. Zhao, John Kubiatowicz, and Anthony D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB//CSD-01-1141, University of California, Berkeley, April 2000.

[Zhao et al. 2006] Wenrui Zhao, Yang Chen, Mostafa Ammar, Mark Corner, Brian Levine, and Ellen Zegura. Capacity Enhancement using Throwboxes in DTNs. IEEE International Conference on Mobile Ad hoc and Sensor Systems (MASS), Vancouver, Canada, October 2006.

[Zhao et al. 2009] Bridge Q. Zhao, John C. S. Lui, Dah-Ming Chiu. Analysis of Adaptive Protocols for P2P Networks. IEEE INFOCOM, Rio de Janeiro, Brazil, 2009.

[Zhong et al. 2008] Ming Zhong, Kai Shen, Joel I. Seiferas. The Convergence-Guaranteed Random Walk and Its Applications in Peer-to-Peer Networks. IEEE Transactions on Computers, Vol.57, 2008, p. 619 - 633.

List of publications

2009

[1] Nouha Oualha and Yves Roudier. A Game Theoretical Approach in Securing P2P Storage against Whitewashers. In the 5th International Workshop on Collaborative Peer-to-Peer Systems (COPS'09), June 29 - July 1, 2009, Groningen, Netherlands. **Best Paper Award**.

2008

[1] Nouha Oualha and Yves Roudier. Evolutionary game for peer-to-peer storage audits. In the 3rd International Workshop on Self-Organizing Systems (IWSOS'08), December 10-12, Vienna, Austria.
[2] Nouha Oualha and Yves Roudier. Designing Attack Resilient Cooperation Incentives for Self-Organizing Storage. 1er Workshop sur la sécurité des réseaux autonomes et spontanés organisé, Loctudy, France, 13-14 october, 2008.
[3] Nouha Oualha and Yves Roudier. Reputation and Audits for Self-Organizing Storage. In the 1st Workshop on Security in Opportunistic and SOCial Networks (SOSOC 2008), Istanbul, Turkey, September 2008.
[4] Nouha Oualha, Melek Önen, and Yves Roudier. A Security Protocol for Self-Organizing Data Storage. 23rd International Information Security Conference (SEC 2008), September 2008, Milan, Italy.

2007

[1] Nouha Oualha, Yves Roudier. Securing ad hoc storage through probabilistic cooperation assessment. WCAN '07, 3rd Workshop on Cryptography for Ad hoc Networks, July 8, 2007, Wroclaw, Poland.
[2] Nouha Oualha, Pietro Michiardi, Yves Roudier. A game theoretic model of a protocol for data possession verification. TSPUC 2007, IEEE International Workshop on Trust, Security, and Privacy for Ubiquitous Computing, June 18, 2007, Helsinki, Finland.

2006

[1] Nouha Oualha and Yves Roudier. Securing cooperative backup for mobile systems. Programme Initiative du GET, « Réseaux autonomes et spontanés », Novembre 16- 17 2006, Rennes, France.