Coding for resource optimization in large-scale distributed systems

Nicolas Le Scouarnec

Thèse

THESE INSA Rennes
sous le sceau de l'Université européenne de Bretagne
pour obtenir le titre de
DOCTEUR DE L'INSA DE RENNES
Spécialité : Informatique

présentée par
Nicolas Le Scouarnec
ECOLE DOCTORALE
Matisse

Codage pour l'optimisation de ressources dans les systèmes distribués

Thèse soutenue le 08.12.2010
devant le jury composé de :

Claude Jard
Professeur des Universités à l'ENS Cachan Bretagne, Rennes / *examinateur*
Pascal Felber
Professeur à l'Université de Neuchatel / *rapporteur*
Laurent Viennot
Directeur de recherche à l'INRIA Rocquencourt / *rapporteur*
Pablo Rodriguez
Directeur scientifique à Telefonica, Barcelone / *examinateur*
Mary-Luc Champel
System Project Leader à Technicolor, Rennes / Tuteur industriel
Anne-Marie Kermarrec
Directrice de recherche à l'INRIA Rennes / Directeur de thèse

Codage pour l'optimisation de ressources dans les systèmes distribués

Coding for resource

optimization in large-scale

distributed systems

Nicolas Le Scouarnec

En partenariat avec

Résumé

Introduction

Le large déploiement des technologies réseaux a rendu les systèmes informatiques fortement dépendants des communications. L'interconnexion des ordinateurs a permis la construction de systèmes distribués pouvant effectuer de nombreuses tâches de façon collaborative. Ces systèmes distribués sont devenus populaires du fait de la généralisation de l'accès à Internet qui a accru la demande en service tout en permettant le déploiement de nombreuses applications à l'échelle mondiale (applications pair-à-pair par exemple). D'une manière générale, les systèmes distribués à large échelle peuvent être construits en inter-connectant de nombreux serveurs au sein de centres de données (*cloud computing*) ou alors en exploitant les ressources disponibles sur les ordinateurs des utilisateurs pour offrir, à un coût faible, des services qui s'adaptent à la demande (Skype, Bittorrent...). Ces différents services exploitent le système distribué ainsi formé pour différentes tâches qui peuvent être la diffusion de contenu à de nombreux clients, le stockage de données ou encore des tâches plus complexes de calcul distribué. Dans cette thèse, nous nous intéressons aux systèmes de diffusion, qui diffusent une donnée initialement disponible auprès d'une seule source à l'ensemble des pairs du système, et aux systèmes de stockage, qui stockent les données de façon fiable malgré les défaillances qui peuvent toucher le système.

Cependant, malgré leurs propriétés séduisantes, les systèmes distribués tendent à être difficilement prévisibles et peu fiables. En effet, la nécessité de concilier performances et systèmes à large échelle implique d'abandonner les garanties globales usuelles au profit de l'efficacité. Cela introduit de l'imprévisibilité dans le système. Par exemple, dans les systèmes à large échelle, il n'est pas raisonnable d'exiger que chaque composant maintienne une connaissance totale de l'état de tous les autres composants. Ainsi, la plupart des algorithmes proposés sont locaux et ne s'appuient que sur une connaissance partielle de l'état d'un petit sous-ensemble des composants du système. Les algorithmes locaux s'appuient souvent sur des décisions aléatoires ce qui conduit les algorithmes à avoir un comportement global non-déterministe. De plus, les systèmes distribués sont peu fiables. En particulier, les systèmes à large échelle étant composés de nombreux composants, les défaillances d'au moins un composant sont fréquentes. Dans le cas particulier des systèmes pair-à-pair, déployés sur les ordinateurs des utilisateurs, les pairs peuvent se déconnecter tous les jours, augmentant ainsi le nombre de défaillances observées. L'ensemble de ces défaillances réduit les garanties (disponibilité, durée de vie, performances...) qui peuvent être offertes. Cependant, les systèmes distribués supportent aujourd'hui de nombreux services exigeant une haute qualité de service (stockage, téléphonie, téléchargement de mises a jour...). La non-prévisibilité et la faible fiabilité des composants d'un système distribué sont à priori incompatibles avec les garanties nécessaires au déploiement des services sus-cités.

Ainsi, afin de permettre une utilisation efficace des systèmes distribués, des mécanismes spécifiques doivent être prévus afin de masquer la non-prévisibilité et la faible fiabilité des composants constituant le système distribué. L'approche la plus simple consiste à effectuer chaque tâche plusieurs fois. Dans le contexte de la diffusion, cela signifie envoyer chaque paquet de données plusieurs fois. Dans le contexte du stockage, il suffit de stocker chaque paquet plusieurs fois sur des composants distincts. Cependant, ces approches basiques conduisent à des taux de redondance dans la communication ou le stockage très élevés. Par exemple, la conception d'un système de stockage distribué capable de tolérer jusqu'à deux défaillances simultanées conduit à stocker chaque paquet de données sur 3 composants différents.

Nous explorons, dans cette thèse, une alternative moins coûteuse qui consiste à envoyer ou stocker des paquets de données capable de remplacer n'importe quel autre paquet de données. Ainsi la redondance ajoutée n'est pas limitée à la protection d'un paquet de données mais protège l'ensemble des données. Dans le cas du stockage, décrit précédemment, il suffit d'ajouter deux paquets de redondance à l'ensemble du système pour tolérer deux défaillances de n'importe quels paquets alors que l'utilisation de la réplication conduit à ajouter deux paquets de redondance par paquet de données. Cette redondance générique est généralement construite sur le bases des codes. Ainsi, nous étudions l'utilisation de codes dans la prise en charge des défaillances. Nous nous intéressons à deux types de codes. Les plus simples sont les codes correcteurs d'effacements (qui sont des codes canaux). Ces codes transforment un ensemble de k paquets natifs en n paquets encodés de telle sorte que n'importe quel sous ensemble de k paquets encodés permet de retrouver les k paquets natifs. Certains de ces codes, les codes fontaines, sont particulièrement adaptés à une utilisation au sein des systèmes distribués puisqu'ils sont capables de produire un nombre infini de paquets encodés, autorisant les différents composants à encoder sans aucune coopération. Cependant, les codes canaux imposent des restrictions fortes sur les composants capables d'encoder. En effet, seuls les composants ayant une connaissance complète des données natives peuvent encoder. Ainsi, nous considérons une seconde classe de codes, les codes réseaux, qui permettent aux composants d'encoder même s'ils n'ont connaissance que d'un petit sous-ensemble (non-décodable) des paquets.

Comme décrit ci-dessus, les codes sont très efficaces dans la réduction des surcoûts de communication et de stockage. Dans l'exemple du stockage, la tolérance de deux défaillances requiert $k + 2$ composants en utilisant des codes alors qu'elle requiert $3k$ composants lorsque l'on utilise la réplication. Nous choisissons donc d'étudier les approches s'appuyant sur les codes pour leur habilité à masquer le manque de fiabilité des composants pour un surcoût en terme de communication et de stockage très faible. Cependant malgré leurs nombreuses qualités, les codes sont souvent délaissés au profit de solutions comme la réplication. Une des raisons principales est que les codes introduisent des coûts annexes dans les systèmes. Lorsque le codage est utilisé pour effectuer des diffusions, les critiques se concentrent sur les coûts de décodage associés, en particulier lors de l'utilisation de codes réseaux. En effet, l'implémentation la plus classique de ces derniers s'appuie sur un codage aléatoire pour lequel la complexité de décodage est $\mathcal{O}\left(k^3\right)$ ou $\mathcal{O}\left(mk^2\right)$ (k est la longueur du code et m est la taille des données. Lorsqu'il est utilisé pour du stockage, un des surcoûts les plus important est lié à la réparation du code. En effet, en cas de défaillance d'un composant, le bloc qui était stocké sur ce composant doit être régénéré ce qui implique de télécharger puis décoder l'ensemble du fichier. Ces surcoûts conduisent donc les systèmes pratiques à adopter des schémas de gestion de la redondance beaucoup plus simples (réplication, diffusion multiples) qui sont moins efficaces que les codes correcteurs d'effacements ou les codes réseaux.

Dans ce document, nous défendons la thèse que les codes bien que considérés comme coûteux peuvent apporter beaucoup aux systèmes distribués en permettant de se passer de réplication et en améliorant l'efficacité en terme de communication et de stockage. Afin de les rendre plus séduisants, nous proposons de réduire leurs inconvénients, les coûts annexes, en proposant trois approches centrées autour de la réduction des coûts de décodage pour la diffusion et de la réduction des coûts de réparation pour le stockage.

Diffusion et codes fontaines

Contexte

Les protocoles épidémiques [38, 39, 55] sont reconnus comme une solution efficace pour la diffusion dans les systèmes à large échelle. Dans ces protocoles, chaque composant (pair) fait suivre chaque paquet reçu pour la première fois à f (*fanout*) pairs choisis aléatoirement. Dans ces diffusions, on peut identifier deux phases : *(i)* une phase de croissance exponentielle au cours de laquelle les messages reçus sont plutôt utiles puisque la probabilité que la donnée diffusée arrive sur un pair qui ne l'a pas encore reçue est élevée ; *(ii)* une phase de décroissance au cours de laquelle la vitesse de diffusion diminue puisque la probabilité que la donnée soit envoyée à un pair qu'il l'a déjà reçue augmente rapidement. Cette seconde phase augmente le coût de diffusion mais reste nécessaire pour garantir que tous les pairs reçoivent l'ensemble des données.

L'utilisation de codes canaux est une alternative à ces réceptions multiples pour chaque paquet de données transmis. Les codes transforment k paquets de données natives en n paquets de données encodées de sorte que n'importe quel sous ensemble de k paquets encodés permet de récupérer les données originales. Ainsi, cela permet aux pairs d'obtenir toutes les données, même s'ils ne reçoivent pas tous les paquets diffusés. Les codes fontaines [73] sont particulièrement intéressants puisqu'ils permettent de

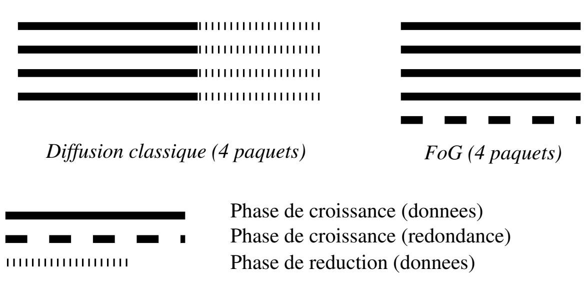

Diffusion classique (4 paquets)
FoG (4 paquets)
Phase de croissance (donnees)
Phase de croissance (redondance)
Phase de reduction (donnees)

FIGURE 1: *FoG conserve uniquement les phases de croissance exponentielle*

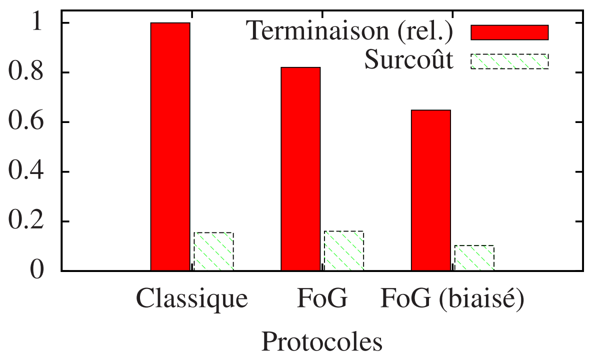


FIGURE 2: *Temps de terminaison relatif et surcoût en communication réseau*

générer un très grand nombre de paquets de données encodées ($n \to \infty$) et des implémentations efficaces (LT Codes [66], Raptor Codes,...) existent. Une de leurs utilisations proposées dans le cadre des fontaines numériques [12] est le téléchargement simultanées depuis plusieurs sources. Puisque le nombre de paquets encodés est très grand, deux sources génèreront des paquets encodés différents sans avoir besoin de se coordonner, ce qui est particulièrement adapté aux systèmes distribués que nous étudions.

FoG

Au sein de cette thèse, nous proposons un nouveau protocole de diffusion épidémique qui exploite les codes fontaines. FoG vise à remplacer la phase de décroissance propre à chaque paquet diffusé par des diffusions additionnelles de données de redondance, pouvant remplacer n'importe quelle donnée manquante, comme illustré par la Figure 1. Dans ce but, FoG interrompt les diffusions prématurément (en limitant le nombre de retransmission possibles pour un paquet), de façon à ne conserver que les phases de croissance exponentielle. Afin de garantir que, malgré cela, tous les pairs reçoivent toutes les données, FoG ajoute des diffusions additionnelles de données encodées. Ces données encodées pouvant remplacer n'importe quelle autre donnée manquante, la diffusion devient plus efficace.

Une propriété importante des codes fontaines est que seuls les pairs ayant reçu toutes les données peuvent décoder et ensuite produire de nouveaux paquets encodés utiles pour tous les autres pairs. Il est important de distinguer les pairs complets des pairs incomplets. Ainsi, FoG construit un réseau logique qui prend en compte cette information afin de construire un cœur de réseau composé des pairs incomplets fortement connecté et alimenté par les pairs complets qui injectent des nouveaux paquets encodés dans ce réseau.

De plus, les pairs complets sont plus utiles que les pairs incomplets. En effet, si les seconds se contentent de faire suivre des données reçues, les premiers sont capables d'encoder et de produire de nouveaux paquets qui seront forcement utiles puisque diffusés par aucun autre pair. Cependant, l'évolution des pairs dans un système classique est relativement homogène. La plupart des pairs vont terminer en même temps ce qui implique qu'ils ne vont pas vraiment s'entraider. Dans FoG, nous prenons le parti de d'introduire un biais afin de favoriser certains pairs. Ces pairs vont terminer beaucoup plus tôt et vont pouvoir encoder pour produire de nouveaux paquets plus utile que des paquets re-transférés.

Résultats

Les évaluations portant sur FoG (Figure 2) montrent que l'utilisation conjointe de codes fontaines et d'un protocole de diffusion prenant en compte la différence d'état entre les pairs capable d'encoder et les autres permet d'obtenir un gain aussi bien en terme de temps nécessaire à la terminaison qu'en terme de coûts réseaux.

De plus, l'ajout d'un biais favorisant certains pairs afin qu'ils terminent plus rapidement et se mettent à encoder plus tôt permet de raccourcir encore le temps de terminaison tout en réduisant les coûts réseaux.

Codes réseaux basse complexité pour la diffusion

Contexte

Le codage réseau [2] permet un gain en efficacité dans les applications de diffusion [4,33, 43,44,56]. En effet, au lieu de simplement transférer les paquets reçus, les composants (pairs) combinent les paquets déjà reçus pour former de nouveaux paquets encodés. Cette opération de recodage augmente la diversité en diminuant ainsi la probabilité qu'un paquet reçu par un pair soit inutile. Contrairement au codage canal exploité dans la proposition précédente, le codage réseau autorise les composants n'ayant reçu qu'une partie (non-décodable) des paquets à recoder. Le schéma général du codage réseau est présenté sur la Figure 3. La combinaison des paquets disponibles dans les structure de données locale permet de produire un nouveau paquet encodé. Ce paquet est ensuite transmis sur le réseau. Le destinataire doit alors décoder l'ensemble des paquets qu'il reçoit. Nous nous intéressons à l'opération de recodage permettant de construire un paquet encodé.

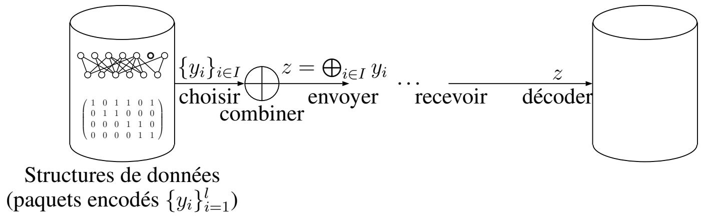
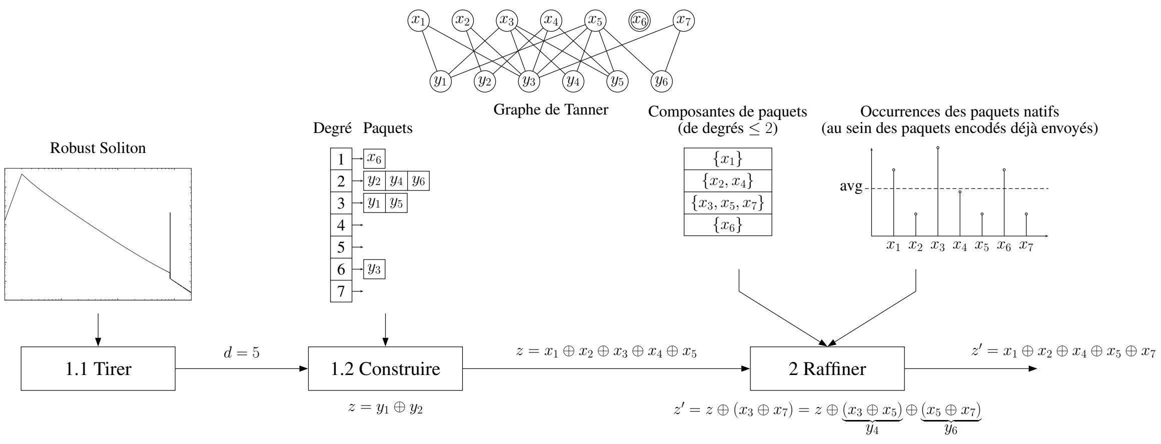


FIGURE 3: *Schéma général de codage réseau.*



Graphe de Tanner
Robust Soliton
Degré Paquets
Composantes de paquets (de degrés ≤ 2)
Occurrences des paquets natifs (au sein des paquets encodés déjà envoyés)
avg
1.1 Tirer
$d = 5$
1.2 Construire
$z = y_1 \oplus y_2$
$z = x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5$
2 Raffiner
$z' = z \oplus (x_3 \oplus x_7) = z \oplus \underbrace{(x_3 \oplus x_5)}_{y_4} \oplus \underbrace{(x_5 \oplus x_7)}_{y_6}$
$z' = x_1 \oplus x_2 \oplus x_4 \oplus x_5 \oplus x_7$

FIGURE *4: Vue d'ensemble de LTNC (*$\mathbf{k} = \mathbf{7}$*).*

L'implémentation habituelle de codage réseau repose sur les codes réseaux linéaires aléatoires [49] où les paquets transférés à partir d'un pair sont des combinaisons linéaires aléatoires de paquets connus par ce pair. L'absence de structure dans le code résultant implique que le décodage doit passer par une élimination de Gauss qui a une complexité élevée ($\mathcal{O}(mk^2)$ ou $\mathcal{O}(m^3)$, k étant le nombre de paquets natifs et m étant la taille d'un paquet). La complexité de décodage des codes réseaux linéaires aléatoires est souvent présenté comme le principal obstacle à leur déploiement [68, 101].

Nous souhaitons proposer de nouveaux codes réseaux décodables par propagation de croyance avec une complexité plus faible ($\mathcal{O}(mk \log k)$ ou $\mathcal{O}(k \log k)$). Nous nous appuyons sur les LT codes [66] qui ont des propriétés particulières permettant un tel décodage. Cependant, l'algorithme d'encodage des codes LT nécessite d'avoir une connaissance complète des paquets natifs, alors que la construction de codes réseaux requiert une méthode de recodage capable de combiner des paquets encodés entre eux.

LT Network Codes

Nous souhaitons définir une méthode de recodage qui respecte la structure des codes LT de telle sorte que le code résultant soit décodable par propagation de croyance. Cette propriété repose sur la faible densité du graphe de Tanner associé au code. Le graphe de Tanner (Figure 4) est un graphe bipartite où le premier ensemble de nœuds correspond aux paquets natifs et le second ensemble correspond aux paquets encodés ($y_1 = x_1 \oplus x_3 \oplus x_5$). L'algorithme d'encodage des codes LT garantit que le degré des paquets encodés suit une distribution particulière, le Robust Soliton, et que les paquets natifs ont tous à peu près le même degré.

Nous étendons ainsi les codes LT avec une méthode recodage en trois étapes, illustrées par la Figure 4 : *(i)* un degré d est tiré suivant la distribution Robust Soliton ; *(ii)* un paquet encodé de degré d est construit en combinant des paquets encodés suivant un algorithme proche d'une solution gloutonne au problème du sac à dos ; cette étape assure que le degré des paquets encodés suit la distribution Robust Soliton ; *(iii)* ce paquet est ensuite modifié, sans en changer le degré, en effectuant des substitutions de paquets de façon à remplacer les paquets natifs fréquemment inclus dans les paquets encodés envoyés par des paquets natifs rarement inclus ; cette étape s'appuie sur une structure de données que nous définissons et assure que le degré de tous les paquets natifs est à peu près le même.

Résultats

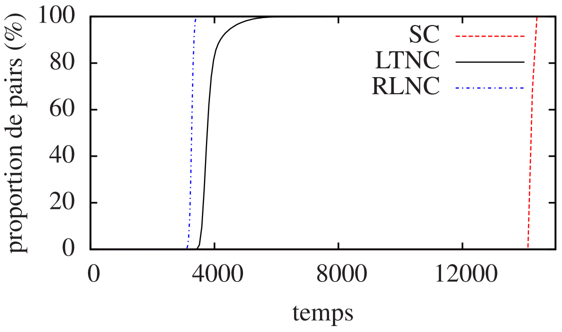


FIGURE 5: *Convergence*

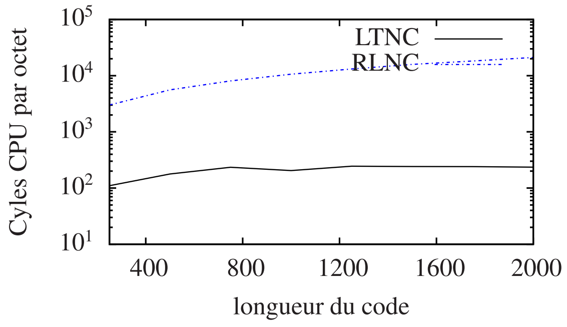


FIGURE 6: *Décodage des données*

Nous évaluons nos codes réseaux LT par simulation d'une application de diffusion épidémique en pair-à-pair. On considère un réseau constitué de 1000 pairs et un code de longueur $k = 2048$. Nous nous comparons à une diffusion n'utilisant pas de codes (SC) et à des codes réseaux linéaires aléatoires (RLNC). Ces derniers représentent la limite de performance pour un schéma de codage parfait.

Comme illustré par la Figure 5, nos codes réseaux LT (LTNC) permettent d'approcher les performances des codes RLNC et surpassent largement les schémas n'utilisant pas de codes. De plus, nos évaluations soulignent l'intérêt des codes réseaux LT. En effet, la complexité plus faible de l'algorithme de décodage employé permet de réduire considérablement les coûts de décodage associés. Sur la Figure 6, on observe un gain de l'ordre de 99% en terme de coûts de décodage.

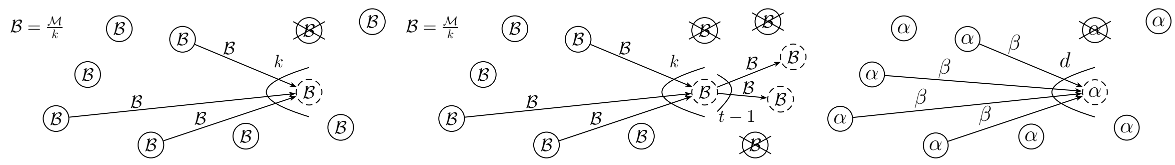
Les codes réseaux LT offre une alternative intéressante aux codes réseaux linéaires aléatoires avec une efficacité légèrement réduite en terme de communication mais un gain important en terme de coûts de calcul. Ces codes élargissent ainsi le champs des applications possibles pour le codage réseau.

Stockage optimal à base de codes réseaux

Contexte

Durant la dernière décennie, la croissance des données numériques et un large accès aux réseaux a favorisé le développement de systèmes de stockage distribués. Ces systèmes sont généralement construits par agrégation de plusieurs composants afin de fournir un espace de stockage important et résistant aux défaillances. Afin de palier aux défaillances, ces systèmes ajoutent de la redondance aux données stockées. Les deux principaux types de redondance sont *(i)* la réplication où chaque donnée est copiée plusieurs fois, et *(ii)* les codes correcteurs d'effacements où les données sont encodées offrant ainsi un système requérant moins de stockage additionnel.

Cependant, au fil du temps, les défaillances détruisent peu à peu la redondance introduite initialement. Afin d'éviter que les données ne soient définitivement perdues, il est nécessaire de mettre en place un mécanisme d'auto-réparation qui régénère les données de redondance perdues afin de maintenir le système dans un état sain. Un tel mécanisme se décompose en trois fonctions principales : *(i)* la détection des composants défaillants, *(ii)* le choix de composants de rechange et le déclenchement de la réparation, *(iii)* la réparation en elle-même, à savoir la re-création des blocs de données de redondance perdus. Nous nous intéressons à ce troisième point dans un contexte ou les données stockées sont encodées. Les stratégies de réparation de redondance employés dans les systèmes basés sur des données encodés sont décrits sur la Figure . Le système stocke un fichier de taille $\mathcal{M}$ et est composé de différents composants stockant une certaine quantité de données (α ou $\mathcal{B}$). Au cours de la réparation d'un composant (en pointillés), le composant réparé télécharge des données (β ou $\mathcal{B}$) depuis d ou k autres composants.


$\mathcal{B} = \frac{\mathcal{M}}{k}$
$\mathcal{B}$
k
$\mathcal{B} = \frac{\mathcal{M}}{k}$
$t-1$
α
β
d

(a) Code correcteur d'effacements (réparation immédiate) (b) Code correcteur d'effacements (réparation paresseuse) (c) Codes régénérants

FIGURE 7: *Réparation dans un systèmes à n composants. Dans ces exemples, $k = 3$, $d = 4$, $\mathcal{B} = 1$, $\alpha = 1$, and $\beta = 1/2$.*

L'utilisation de données encodées à l'aide de codes correcteurs d'effacements (Figure 7a) permet une très bonne efficacité vis a vis du stockage mais induit des coûts de réparation élevés [62, 86, 102]. En effet, dans ce cas, la réparation passe par le téléchargement et le décodage du fichier complet, même pour régénérer un seul bloc. Afin de réduire ces coûts, il a été proposé d'effectuer les réparations de façon paresseuse [7, 29, 30]. En effet, le coût de téléchargement du fichier est fixe que l'on souhaite régénérer un ou plusieurs blocs. Ainsi, attendre afin d'effectuer plusieurs réparations de façon simultanée permet de factoriser ce coût (Figure 7b). Plus récemment, l'application des codes réseaux aux systèmes de stockage a permis de réduire le coût de réparation en permettant de ne pas décoder [1, 33, 34, 54, 63]. Cette seconde approche a été étudiée en détail par Dimakis *et al.* [31, 32]. Ces articles définissent les codes régénérants (Figure 7c) qui correspondent aux compromis optimaux entre le coût de

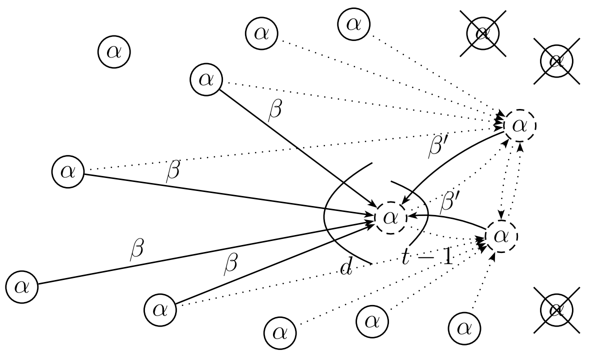


FIGURE 8: *Réparation de codes régénérants coordonnés.*

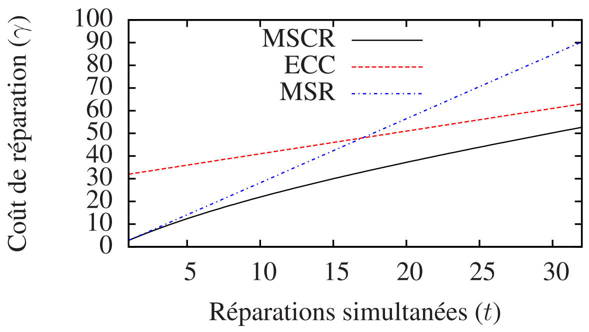


FIGURE 9: *Coût de réparation total $t\gamma$ pour $d = 48$ et $k = 32$.*

réparation et le coût de stockage. Cependant, ces coûts ne sont définis que pour des réparations indépendantes et pour des systèmes statiques.

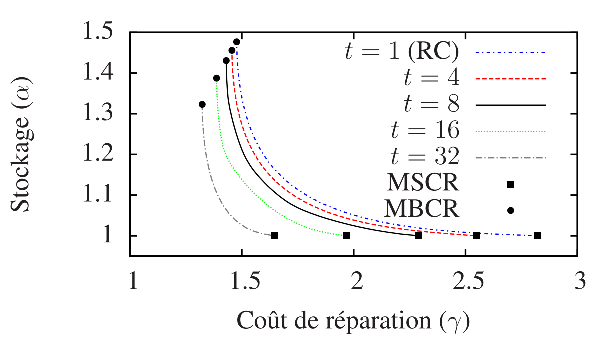
Au sein de cette thèse nous proposons les codes régénérants coordonnés (CRC) afin de supporter les réparations simultanées de plusieurs composants à un coût moindre que les codes régénérants existants et nous proposons aussi les codes régénérants adaptatifs (ARC) qui sont capables de s'accommoder de systèmes très dynamiques.

Codes régénérants coordonnés

Nous proposons des codes régénérants coordonnés, qui permettent d'effectuer les réparations de façon optimale même lorsque plusieurs réparations doivent être effectuées simultanément. Par rapport aux codes régénérants classiques, notre proposition ajoute une étape dans la réparation (Figure 8). Celle-ci se déroule en deux étapes : *(i)* la collecte, similaire aux codes régénérants existants, au cours de laquelle le composant en cours de réparation récupère β données depuis d composants en bon état ; *(ii)* la coordination au cours de laquelle les t composants en cours de réparation échangent β' données deux à deux. Nous étudions les valeurs optimales de β et β' permettant d'offrir le meilleur compromis entre le coût de stockage α et le coût de réparation $\gamma = d\beta + (t-1)\beta'$.

Ces compromis optimaux sont donnés sur la Figure 10. Il apparait clairement que l'utilisation d'une étape de coordination améliore les performances par rapport aux codes régénérants (RC) existants à partir du moment où $t > 1$. Par ailleurs, la courbe fait apparaitre deux points particuliers. Les MSCR (Codes régénérants coordonnés à stockage minimal) correspondent aux codes offrant le plus faible stockage possible. L'évolution du coût de réparation associé à ces codes (MSCR) en fonction de t est illustré par la Figure 9 et comparé aux codes régénérants existants (MSR) et aux codes correcteurs d'effacements (ECC). Au sein de cette thèse, nous déterminons et prouvons les quantités de données à stocker et transférer :

$$\alpha = \frac{\mathcal{M}}{k} \qquad \beta = \frac{\mathcal{M}}{k}\frac{1}{d-k+t} \qquad \beta' = \frac{\mathcal{M}}{k}\frac{1}{d-k+t}$$



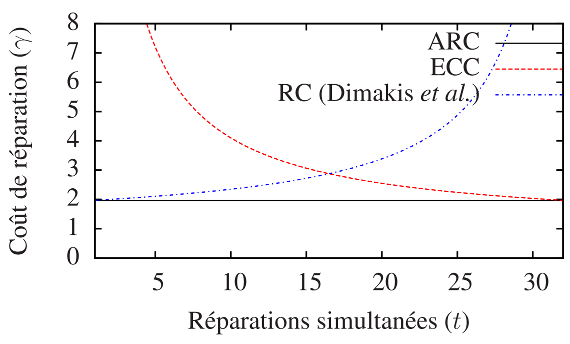


FIGURE *10: Compromis optimaux pour $k = 32$ and $d = 48$. Coûts normalisés par $\mathcal{M}/k$.*

FIGURE *11: Coût de réparation moyen γ pour $n = 64$ and $k = 32$.*

Les MBCR (Codes régénérants coordonnés à bande passante minimale) correspondent aux codes offrant la plus faible utilisation du réseau.

$$\alpha = \frac{\mathcal{M}}{k} \frac{2d + t - 1}{2d - k + t} \qquad \beta = \frac{\mathcal{M}}{k} \frac{2}{2d - k + t} \qquad \beta' = \frac{\mathcal{M}}{k} \frac{1}{2d - k + t}$$

Codes régénérants adaptatifs

Les codes régénérants existants [32] ainsi que ceux que nous définissons s'appuient sur des hypothèses contraignantes vis à vis des paramètres du système. En effet, ils supposent que le nombre d de composants contactés à chaque réparation et le nombre t de composants réparés simultanément restent invariants pendant toute la durée de vie du système. Nous relâchons cette contrainte, sous la condition $\alpha = \frac{\mathcal{M}}{k}$, et définissons des codes régénérants adaptatifs capable de choisir, à chaque réparation, les valeurs de t et d les plus adaptées.

Comme montré sur la Figure 11, nos codes (ARC) ont un coût moyen de réparation constant et inférieur aussi bien aux codes correcteurs d'effacements (ECC) qu'à une construction adaptative basée sur les codes régénérants existants (RC) [32].

Les codes que nous définissons sont plus adaptés à des systèmes pratiques que les codes existants. En effet, les codes régénérants coordonnés permettent d'effectuer plusieurs réparations simultanément et la forme adaptative de ces codes permet un gain de souplesse en choisissant automatiquement les paramètres de réparation les plus intéressants.

Conclusion et perspectives

Les systèmes distribués sont construits sur une base peu fiable et difficilement prévisible. Les codes canaux ou réseaux ont été largement utilisés pour masquer le non-déterminisme apparents des composants. Cependant, même s'ils offrent des performances séduisantes en terme de transmission ou de stockage, ils sont souvent délaissés au profit de mécanismes moins efficaces mais plus simple comme des transmissions multiples ou de la réplication. Les principaux arguments contre les codes sont que les gains sont souvent dépassés par les coûts annexes qui sont les coûts de calcul dans le cas des codes réseaux utilisés pour la diffusion et les coûts de réparation dans le cas du stockage.

Dans cette thèse, nous avons proposé de réduire les coûts de calcul en diffusion en utilisant des codes ayant une complexité de décodage basse. Par ailleurs, nous avons aussi étudié les gains apportés par le codage réseau en terme de coût de réparation au sein d'un système de stockage. Ces travaux ont conduit à différents protocoles permettant une utilisation plus aisée des codes dans les systèmes distribués grâce à des coûts fortement réduits.

Enfin, ces travaux suggèrent diverses perspectives, principalement centrées sur l'utilisation des codes dans les systèmes de stockage distribués. Au sein de cette thèse, nous avons défini des codes régénérants optimaux sans apporter de construction spécifique. Ces codes peuvent être implémentés à l'aide de codes réseaux linéaires aléatoires (à complexité de décodage élevée). Cependant, il serait beaucoup plus intéressant de disposer de constructions déterministes ou au moins permettant un décodage basse complexité.

List of publications

Chapter 2 describes a dissemination protocol presented at the SSS conference.

FoG : fighting the achilles' heel of gossip protocols with fountain codes
Mary-Luc Champel, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. In *SSS'2009 : 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, 2009.

Chapter 3 describes LT Network Codes presented at the following events.

LT Network Codes (Best paper award)
Mary-Luc Champel, Kévin Huguenin, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. In *IEEE ICDCS'2010 : 30th International Conference on Distributed Computing Systems*, 2010.

LT Network Codes: Low Complexity Network Codes (Short paper)
Mary-Luc Champel, Kévin Huguenin, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. In *ACM CoNEXT'09 (Student Workshop) : 5th ACM International Conference on emerging Networking EXperiments and Technologies*, 2009.

The following publications correspond to other works done during my PhD.

Phosphite: Guaranteeing out-of-order download in P2P Video on Demand
Mary-Luc Champel, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. In *P2P'2009 : 9th IEEE International Conference on Peer-to-Peer Computing*, 2009.

Phosphite: Incitation à la collaboration pour la vidéo la demande en P2P
Mary-Luc Champel, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. In *Algotel'2009 : 11ème Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*, 2009.

Contents

Contents **xix**

1 Codes and Distributed Systems **7**
1.1 Information and Coding Theory . . . 8
1.2 Applying Codes to Distributed Systems . . . 18

2 Fountain Codes in Dissemination **25**
2.1 Motivation . . . 25
2.2 Design of FoG . . . 27
2.3 FoG . . . 30
2.4 Evaluation . . . 33
2.5 Related Work . . . 38
2.6 Summary . . . 39

3 LT Network Codes for Dissemination **41**
3.1 Motivation . . . 41
3.2 Context . . . 42
3.3 LT Network Codes . . . 43
3.4 Evaluation . . . 53
3.5 Related Work . . . 59
3.6 Summary . . . 60

4 Network Codes for Storage **63**
4.1 Motivation . . . 63
4.2 Context . . . 65
4.3 Coordinated Regenerating Codes . . . 69
4.4 Adaptive Regenerating Codes . . . 82
4.5 Related Work . . . 86
4.6 Summary . . . 87

5 Conclusions and Perspectives **89**
5.1 Context . . . 89
5.2 Contributions . . . 89

5.3 Perspectives 92

Bibliography **97**

Introduction

The wide deployment of networking technology has made communication an essential facility for computer systems. It has allowed the interconnection of computers to form *distributed systems* which can perform common tasks in a collaborative way. Distributed systems have become popular through the generalization of fast Internet access, increasing the demand for services and allowing applications (peer-to-peer applications, for example) to be deployed world-wide. Distributed systems are either built by combining commodity hardware into large data-centers to allow cheaper processing (cloud services) or by combining end-users computers to provide, at low cost, peer-to-peer services (Skype, Bittorrent. . .) that scale with the demand. These services rely on distributed systems for various tasks such as disseminating data across large groups of computers, storing data and even performing distributed computations. In this thesis, we focus on dissemination and storage systems. Dissemination systems spread data initially available at a source to all other peers in the system while storage systems aim at storing various data safely (i.e., preserving data from failures).

Yet, in spite of their appealing properties, distributed systems tend to be hardly predictable and unreliable. Indeed, to conciliate the need for performance and the scale of distributed systems, distributed algorithms trade global guarantees for efficiency thus introducing unpredictability in distributed systems. For example, in large-scale distributed systems, it may not be reasonable for every single device to maintain a complete knowledge of every other device. Hence, most proposed algorithms are *local* and run using only a partial knowledge of the whole system. Local algorithms generally also rely on randomized decisions thus leading to non-deterministic algorithms. Moreover, distributed systems are unreliable due to temporary or permanent failures. More specifically, if we consider large scale distributed systems, built from commodity hardware, the occurrence of failures may be frequent. Moreover, if we consider peer-to-peer systems, deployed on end-users computers, peers may disconnect on a daily basis. These failures obviously reduce the guarantees (availability, durability, performance. . .) that can be offered. Yet, these distributed systems are now used for running services requiring a high quality of service (storage, telephony, software updates. . .). Clearly, the unreliability and unpredictability of the underlying devices of a distributed system are not trivially compatible with required guarantees for running such services on top of it.

Hence, to allow an efficient use of distributed systems, specific mechanisms must be added to cope with the unpredictability and unreliability of the underlying computers. The simplest approach consists in running each task several times. In the context of dissemination, it means sending each piece of data several times. In the context of storage, it means storing each piece of data on several different devices (i.e., replication). Yet, such naive approaches may lead to unacceptable communication overhead or storage overhead. For example, tolerating two simultaneous random failures in a storage system using replication implies that each block must be stored on three different devices.

A possible alternative that we explore in this thesis is to send or store additional pieces of data that can replace any of the previous pieces of data in case of a failure. For example, tolerating the same two simultaneous failures in a storage system now requires only two additional devices for the whole system (instead of two per block). This is typically what *codes* do and, in this thesis, we explore the usage of codes to deal with failures. We consider two class of codes. The simplest are erasure correcting codes (i.e., channel codes) which expand a set of k native packets into a larger set of n encoded packets, such that any subset of k encoded packets allows recovering the native packets. Some specific channel codes, namely fountain codes, are especially interesting for distributed systems for they can produce an infinite number of encoded packets, thus allowing devices to operate without coordination. Yet, channel codes pose severe restrictions on which devices can encode: only devices knowing all native packets can encode. Hence, we also consider network codes, which allow all devices to encode even if they only know a small (i.e., non decodable) subset of encoded packets.

As illustrated by the small example above, codes are very efficient in terms of overhead: they tolerate 2 failures using $k + 2$ devices while replication requires $3k$ devices. Hence, we choose to study code-based approaches for they allow hiding the unreliability of underlying computers at a low communication and storage overhead. However, regardless of how appealing codes seem, they are often left aside because of side costs they introduce in systems. First, coding in dissemination systems, especially network coding, is widely criticized for the high processing costs due to the high complexity of the decoding procedure it implies. Effectively, current implementations of network coding use a rather inefficient decoding algorithm whose complexity is $\mathcal{O}\left(k^3\right)$ or $\mathcal{O}\left(mk^2\right)$ where k is the code length (i.e., number of blocks/packets) and m is the block/packet size. Second, coding in storage systems is also criticized for high communication costs due to the repair procedure (repairing a lost block in storage systems relying on erasure correcting codes implies gathering and decoding the whole data). Hence, simple schemes relying on sending or storing several times the same pieces (i.e., replication) are often used in practice in place of more efficient erasure correcting codes or network codes.

Contributions of this thesis

While they are often considered as expensive, in this document, we defend the thesis that codes are appealing in distributed systems for they provide huge opportunities to reduce the need for pure replication thus increasing both the storage efficiency and dissemination efficiency. We argue that we can reduce their drawbacks to make them more appealing. To support this thesis, we propose three approaches to limit and reduce the side costs.

Adapting dissemination protocols to leverage the use of fountain codes

We consider the problem of disseminating some data from one source to a set of peers. We adopt an epidemic push-based dissemination algorithm. In epidemic dissemination, at each communication cycle, each peer randomly selects another peer and forwards (i.e., pushes) some packet of data to it. In a first phase, very few peers have received data, therefore forwarded packets tend to be useful. In a second phase, most of the peers have already received data, hence forwarded packets tend to be redundant with already received packets. The second phase is undesirable as it generates a lot of overhead. Yet, it is required to ensure w.h.p. that eventually, all peers have received all packets.

In FoG, we propose to replace the second phases that generate most of the overhead by additional dissemination (first phase-only) of encoded data that can replace any non-received data. We limit ourselves to erasure correcting codes (fountain codes), for which efficient implementations (LT or Raptor codes) exist. Fountain codes allow transforming a set of k native packets into an infinite set of encoded packets so that receiving any subset of k packets allows recovering the native packets. In FoG, the disseminated packets are encoded so that peers do not need to receive all packets but only a large enough subset. In FoG, we adapt the protocol so that it leverages fountain codes by taking into account the fact that peers that have recovered the native packets are more useful since they can produce newly encoded packets being different from all packets being forwarded by other peers. Therefore, we build an overlay that is able to take into account the current level of completion of each peer by introducing a bias so that some peers finish earlier to help other peers. Through the adaptation of the protocol, FoG is able to reduce the time to complete by 33% and the overhead by 50% over a protocol that disseminates encoded data while not taking into account the differences between peers that can only forward and peers that can encode.

FoG is presented in details in Chapter 2.

Reducing the decoding complexity in network codes for dissemination

Yet, channel codes (fountain codes) are limited by the fact that only sources are able to encode. Indeed, the encoding can only take place once the device has acquired the full knowledge of data (i.e., received all data and decoded it). Recently, it has been shown that pushing the paradigm of coding further by allowing intermediate devices to recode with only partial knowledge of the content allows significant gains [2]. A natural improvement over FoG, is that the dissemination algorithm can be simpler as there is no need to distinguish complete (full knowledge) devices from non-complete (partial knowledge) devices. The dissemination also becomes more efficient as disseminated packets tend to be more diverse [44]. In spite of these appealing improvements, network codes have a major drawback in their usual implementation (random linear network codes) that suffers a high decoding complexity [68, 101].

We propose to build new network codes decodable using a low complexity belief-propagation decoding procedure. To this end, we extend LT codes [66] into *LT network codes* by defining a recoding method. The main challenge in building our LT network codes consists in defining a recoding method that maintains the key properties of LT Codes. Indeed, the ability to decode LT codes by belief propagation relies on the degrees of encoded packets (i.e., the number of native packets that are xor-ed to form an encoded packet) and on the degrees of native packets (i.e., the number of encoded packets that encompass a particular native packet). While efficient algorithms exist for generating packets from the full knowledge, generating encoded packets verifying key properties of LT codes from partial knowledge is not addressed. With our algorithm, generating a fresh encoded packet consists of three main operations: *(i)* a degree satisfying the Robust Soliton is picked; *(ii)* an encoded packet of said degree is generated, thus ensuring the first key property; *(iii)* the encoded packets are refined by performing substitutions of native packets thanks to efficient algorithms and data structures we define, thus ensuring the second key property. Our codes offer significant gains on the decoding costs (up to 99%) while increasing only reasonably the communication costs (20%). The gains observed reflect the change in the complexity of the decoding algorithm. Therefore, LT network codes are an alternative to random linear network codes and offer an interesting tradeoff between communication efficiency and decoding costs.

Our LT network codes are presented in Chapter 3.

Reducing the repair cost in storage, using network codes

Finally, we consider a different use of codes in distributed systems: apart from dissemination, codes have been widely used in distributed storage. In distributed storage, the k native blocks are turned into n encoded blocks thanks to erasure correcting codes. The different encoded blocks are then distributed over the disks. It prevents data losses since as long as a large enough (k) subset of disks has not crashed, the native blocks can be recovered. Yet, to ensure continuous operations and to avoid the number of available blocks going bellow a critical threshold, whenever a failure is detected it must be repaired (i.e., the lost block must be regenerated). When the encoded blocks are produced by erasure correcting codes, regenerating the lost block implies downloading, decoding the whole file and encoding again. Downloading all the blocks to regenerate a single block appears to imply a huge communication cost. Hence, code-based approaches are often replaced by replication or hybrid solutions that offer very low repair cost at the price of a higher storage cost. In this context, Dimakis *et al.* [31, 32] have recently shown that the repair cost of code-based approaches can be significantly reduced by applying network coding to distributed storage. In their work, they determine the minimum amounts of data that must be stored or transfered to ensure that the repairs are correct.

We build upon previous work to add more flexibility to regenerating codes, which only support single failure repairs and very static systems. We propose *coordinated regenerating codes*, which support multiple failures. We also propose *adaptive regenerating codes*, which can support dynamic systems more efficiently than regular regenerating codes. We prove the correctness and optimality of the codes we propose. Interestingly, the class of our coordinated regenerating codes encompasses all existing repair strategies for code-based systems. Furthermore, our codes allow us to show that deliberately delaying repairs cannot improve the performance of regenerating-like codes. Finally, the added flexibility of our adaptive regenerating codes simplifies their use in practical systems by reducing the number of parameters to adjust and by allowing them to self-adapt to handle dynamic settings.

More details including optimal storage and repair costs, the associated proofs and a construction based on random linear network codes that achieves said costs are given in Chapter 4.

Other contribution

Incentive for out of order download in P2P Video On Demand

Besides having been used for dissemination and storage, the coding and information theories can be leveraged for security. For example, a public-key cryptosystem, the McEliece cryptosystem, is built upon error correcting codes [72], and Vernam ciphers [93, 100] also share interesting properties with erasure correcting codes. In the latter case, the ciphering and deciphering operations match the encoding and decoding operations of a simple parity code $(x_1, x_2, x_1 \oplus x_2)$. Similarly, network coding has also been studied as a way to offer security in networks. The two kind of attacks considered are wiretap attacks [105], in which an attacker can listen to a limited subset of communication links, and eavesdropping [61], in which each node is curious about the information. All these approaches are based on the principle that if you do not know the code structure or if you do not have enough information, decoding (and deciphering) is not possible [93].

We used this principle to provide an incentive to download out-of-order in P2P Video On Demand in [21, 22]. Many P2P VoD systems rely on Bittorrent-like protocols [15, 24, 26, 77] that are modified to download a large part of their content in-order (i.e., the block to be played first is downloaded first) so as to allow playing while downloading.

However, in all protocols, a small amount of bandwidth must remain dedicated to out-of-order (i.e., randomly by opposition with in-order) download so that the Local Rarest First policy of Bittorrent can ensure that no block becomes rare. However, there is no built-in incentive to obey the protocol and effectively dedicate some amount of bandwidth to out-of-order download. Hence, peers may behave selfishly by only downloading in order, thus reducing the overall performance as out-of-order download is crucial to efficient systems. In Phosphite, we address this point by ensuring that a peer cannot use a block if it does not download some content out-of-order.

The problem comes down to delivering a block of data only if the other peer owns another block of data downloaded out-of-order. The problem could have been solved by requiring the peer to provide a proof of possession [5, 76, 90]. However, such schemes involve complex computations, add a communication overhead, or are subject to requests of proofs being forwarded. In our system, we build a challenge to ensure that the peer cannot use the provided block if it does not download out-of-order. The challenge is based on parity erasure correcting codes. In short, the protocol works as follows. Whenever a peer requests a block x_i, the peer answering the request may, with a low probability, challenge it by sending $x_i \oplus x_j$ with $j > i$. The requesting peer cannot use $x_i \oplus x_j$ until it knows x_j (thus being able to decode): this acts as an incentive. The requesting peer must therefore request x_j and the protocol applies recursively. Phosphite is secure against selfish attacks and ensures that there is a strong incentive in dedicating some bandwidth to out-of-order download. Moreover, our proposition has very low deciphering/decoding costs (at most 1 xor per block).

More details about this protocol are available in the following publications [21, 22].

In Chapter 1, we briefly present information theory, codes and their uses in distributed systems. Chapters 2, 3, and 4 present our contributions supporting the thesis that side costs of codes in distributed systems can be reduced, thus making codes more practical. Finally, we detail our conclusions and the perspectives of this thesis in Chapter 5.

Chapter 1

Codes and Distributed Systems

In this chapter, we give some insights into important results in information and coding theory. We then present some applications of coding and information theory to distributed systems. This chapter is not meant to be exhaustive and focuses on general results. Hence, a more detailed description of the context needed to understand our contributions as well as discussions concerning closely related works are also given as a section in each chapter.

Large scale distributed systems aggregate numerous devices into a single system that can perform collaborative tasks. The tasks that can be performed by such systems range from performing computations to disseminating and storing data. The large number of devices (i.e., peers) that are aggregated in such systems makes it impossible for each peer to monitor every other peer. Hence, most large scale distributed systems systems are defined so as to rely on local algorithms (i.e., they only require a partial knowledge of the global state) [64]. Algorithms relying only on local information are found in most peer-to-peer systems such as Bittorrent and similar systems [26, 44], Distributed Hash Tables [87, 96], Gossip algorithms [52, 55, 70].... These algorithms include a randomized process (such as the selection of the peers they communicate with or the packets they exchange) allowing them to converge and avoid pathological cases. Overall the combination of local knowledge and randomized decisions implies that the global iterations of the algorithm cannot be predicted.

The underlying devices of distributed systems tend to be unreliable. Moreover, the large number of such devices implies that the failure of at least one device is likely to happen often [89]. Furthermore, when considering peer-to-peer networks, individual peers are likely to temporarily fail (i.e., disconnect for a short period of time) thus increasing the number of failures [10, 45]. Devices can also adopt malicious behaviors leading to further unreliability, for example, deleting data they store.

The randomized behavior of distributed algorithms combined with the unreliability of individual devices implies that the bare distributed systems appear as hardly predictable and exhibit randomness. The distributed applications deployed on such systems are expected to work and fully hide the underlying failures thus requiring mechanisms for preventing failures. A way to deal with unpredictable transmission channels is to use codes. We study, in the next section, how codes allow performing reliable transmissions over unreliable channels. In the last section of this chapter, we present some of the applications of codes to distributed systems and highlight criticisms, due to high side costs, that have prevented a wide use of codes.

1.1 Information and Coding Theory

In 1948, Claude Shannon introduced a mathematical theory of communication [92]. This theory applies to many systems including those built upon unreliable components. In a communication system, the signal sent over a channel may be randomly altered by noise. The simplest channel is the erasure channel[1] illustrated on Figure 1.1. In this channel model, a transmitter sends some information and the bits of information sent are lost with probability $1 - p$.

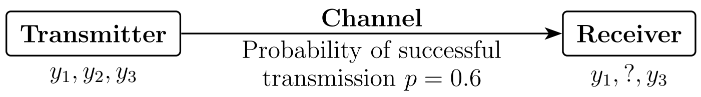


Figure 1.1: The erasure channel.

Shannon shows that using some construction known as codes, it is possible to build a reliable communication system upon unreliable channels. He determines the achievable upper bound for the transmission rate of reliable communication, the capacity of the channel C. If the transmission rate of the unreliable channel is R and the probability of a successful transmission is p, then $C = pR$. As shown on Figure 1.2, a reliable communication system encodes the information before transmitting it over the channel.

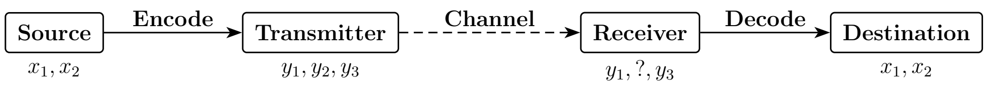


Figure 1.2: A reliable communication system over the unreliable erasure channel.

An alternative way to handle erasures on the channel is to rely on a retransmission scheme (i.e., acknowledge/request scheme). As shown on Figure 1.3, the receiver can

[1]Shannon also described other kinds of channels, for example channels that corrupt the transmitted bits (in Figure 1.1, the receiver would have received $y_1, \bar{y_2}, y_3$). Since our study concerns applicative protocols, we can safely assume that error detection is handled at lower levels and that we only need to deal with erasures: a packet is either received without errors or not received.

send some information to the transmitter through a feedback channel. In particular, it can acknowledge the correct transmission of a packet of information or ask for retransmission. Yet, such an approach requires a feedback channel and is not appropriate for one-to-many communications. Moreover, it adds delays and makes the transmission more complex.

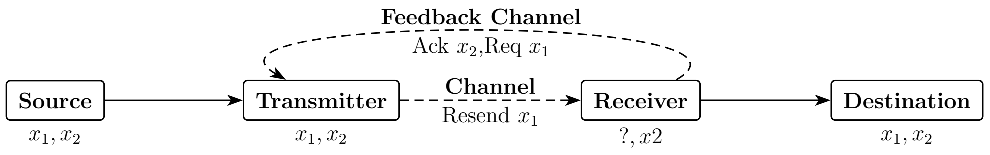


Figure 1.3: A reliable communication system over the unreliable erasure channel using a retransmission scheme.

Even if Shannon defined the fundamental properties of communication systems and showed that codes may transmit information reliably at the capacity of the channel, he left aside the design of practical codes. His proof suggests the use of random linear codes (detailed hereafter) of infinite length, but such codes are highly impractical due to the complexity of the decoding operation. Since then, research has focused on trying to achieve channel capacity while keeping decoding costs within reasonable bounds.

In the rest of this section, we present chronologically, some of the most important codes.

1.1.1 *Classical* Coding Theory

Codes are a construction allowing the detection of errors, the correction of errors (corruptions), and the correction of erasures (losses). A code can be described as a set of codewords (vectors) that are valid. The codewords are chosen so that they all differ from each other in at least d position. d is the Hamming [47] distance between two codewords and define the corrective power of a code. A code can correct up to $t = \lfloor d/2 \rfloor$ errors and d erasures. As stated earlier, we focus on erasure correction.

Decoding a corrupted/partial codeword consists in finding the closest valid codeword. For example, consider a code that contains four valid codewords $(0000, 0110, 1011, 1101)$. Upon reception of a corrupted codeword 0011, we can deduce that the closest (i.e., the most likely) valid codeword is 1011. Similarly, upon reception of partial codeword 0?10, we can deduce that the closest valid codeword is 0110. In the case of erasure correcting codes, decoding is equivalent to solving a system of equations (i.e., performing a Gauss reduction).

The design of codes that are optimal with respect to corrective power comes down to maximizing the minimum Hamming distance between every two codewords. We cannot expect from a (n, k) code (i.e., a code that expands vectors of k original packets into vectors of n encoded packets) to correct more than $d = n - k$ erasures.

This is known as the Singleton Bound [95]. Codes achieving this bound are called *Maximum Distance Separable* codes: all their codewords differ from each other in exactly $d = n - k$ positions. The codes we present in this subsection are designed to ensure good worst case performance by having good distance properties (i.e., being Maximum Distance Separable if possible).

Random Linear Codes

Random linear codes, described hereafter, are used in the proof of the capacity of channels by Shannon [92]. These linear codes are efficient only when the size of codes is high (i.e. the code length k is large). Yet, they are of interest in explaining the principle behind linear codes. We show an example of random linear codes on Figure 1.4. The encoded packets y_1, y_2 and y_3 are computed as random linear combinations of the native packets x_1 and x_2. y_1 and y_3 are transmitted. Each data block transmitted is accompanied by an header describing the coefficient of the random combination. By solving the resulting system of equations (i.e., performing a Gauss reduction), the native packets can be recovered. Any linear erasure correcting code can be described by their encoding matrix and decoded by a Gauss reduction.

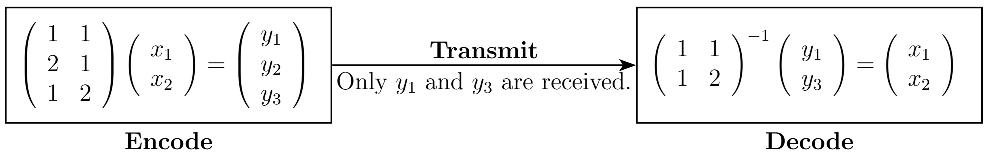

$\begin{pmatrix} 1 & 1 \\ 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$
Transmit
Only y_1 and y_3 are received.
$\begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix}^{-1} \begin{pmatrix} y_1 \\ y_3 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$
Encode
Decode

Figure 1.4: Random Linear Codes

Hamming Codes

In 1950, Richard Hamming proposed one of the first codes able to correct errors [47]. These codes were designed to prevent random read errors from stopping batch computations. His codes are able to correct a single error or two erasures.

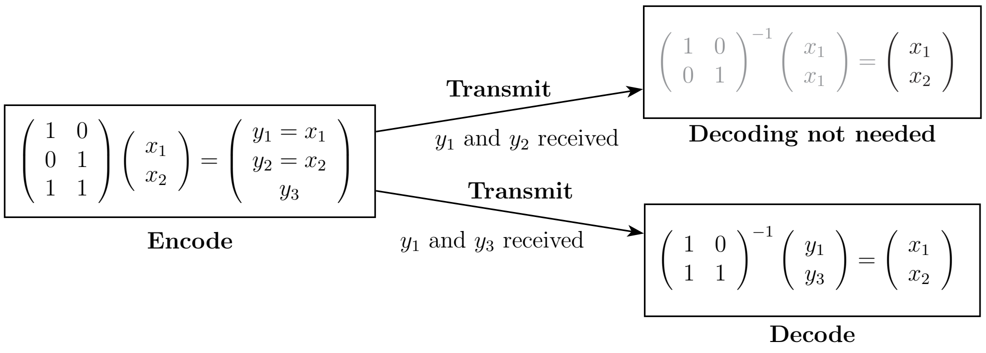
He defines $(2^m - 1, 2^m - m - 1)$ codes for all $m > 2$. Such codes are built by defining the parity checks (i.e., a parity check is an equation checking the parity of several symbols such as $y_1 + y_2 + y_6 = 0$). As shown on Figure 1.5, for defining a $(n = 2^m - 1, k = 2^m - m - 1)$ Hamming code, the parity check matrix H that contains all possible $(2^m - 1)$ vectors of length m as lines is re-arranged by swapping lines so that $H^T = [I_m|P]$. Then, the generator matrix for encoding is $G^T = [I_k|P^T]$. The first matrix defines the checks that are satisfied when no errors occurs during transmission (i.e., $Hx = 0$ if no error occurs). As these codes are linear erasure correcting codes, decoding can be performed as explained previously (using G^{-1}).

An interesting property of these codes is that they are systematic. It means that the set of encoded packets includes the set of native packets (i.e., the code generator matrix contains the identity matrix). For example, in Figure 1.6, the two first encoded

$$H = \left(\begin{array}{ccc} \mathbf{1} & 0 & 0 \\ 0 & \mathbf{1} & 0 \\ 0 & 0 & \mathbf{1} \\ \hline 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{array}\right) \qquad G = \left(\begin{array}{cccc} \mathbf{1} & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & \mathbf{1} \\ \hline 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{array}\right)$$

Figure 1.5: $(7,4)$ *Hamming Code Matrices*

packets are equal to the native packets. Hence, if the transmission succeeds, no decoding operation is needed. However, if the transmission is partial, a decoding allows recovering all native packets.



$\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} y_1 = x_1 \\ y_2 = x_2 \\ y_3 \end{pmatrix}$
Encode
Transmit
y_1 and y_2 received
Transmit
y_1 and y_3 received
$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} x_1 \\ x_1 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$
Decoding not needed
$\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}^{-1} \begin{pmatrix} y_1 \\ y_3 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$
Decode

Figure 1.6: Systematic Codes

Reed-Solomon Codes

Yet, the two previous codes do not achieve the Singleton Bound. Reed-Solomon Codes [83], defined in 1960, are the most widely used erasure correcting codes for dealing with short codes (small k) as they have an optimal correcting power (i.e., they are Maximum Distance Separable).

The data to be transmitted $(x_1 \dots x_k)$ defines a polynom $P(z) = x_k z^{k-1} + \dots + x_1$ over a finite field F of size p. The n encoded packets $(y_1, y_2 \dots y_n) = (P(0), P(\alpha^0), \dots, P(\alpha^{n-2}))$ are computed by evaluating the polynom P at n points (powers of a primitive root of F). Any subset of k encoded packets allows recovering the polynom by Lagrange interpolation. Therefore, (n,k) Reed-Solomon codes can tolerate $d = n - k$ erasures and are Maximum Distance Separable. The length of Reed-Solomon codes is limited by the size p of the finite field used since $n < p$. The size of the finite field for which the arithmetic can be implemented limits the codes to short length. In the case of the erasure channel, Cauchy-based Reed-Solomon [9] and

Vandermonde-based Reed-Solomon [85] represent the encoding operation as a matrix thus allowing simpler algorithms (similar to the ones presented for linear codes).

1.1.2 Fountain codes

The codes described previously, in particular Reed Solomon, have been widely used at low levels (ECC Memory, CDROM) where they are required to detect and correct errors in hardware. Yet, widely-used large-scale packet networks have created a new set of applications for erasure correcting codes. These applications match the digital fountain framework [13, 73] we describe in the first part of this section. Classical codes cannot support efficiently such applications. In the second part of this section, we present fountain codes, specially designed for such problems.

The codes described in this part are designed to handle very large numbers of native and encoded packets. Therefore, a requirement is that they must be encodable and decodable by low complexity algorithms. This is achieved by relying on sparse codes, thus losing ideal distance properties (i.e., their worst case performance is bad) yet offering good correcting performance on average.

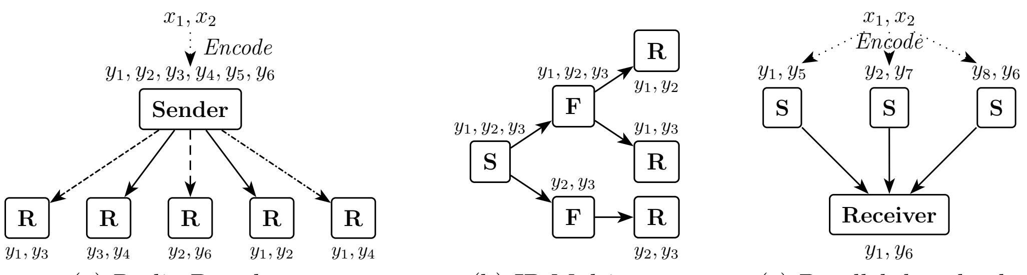
The motivation behind the digital fountains concept and the term fountain codes is the following: collecting data should be as simple as collecting water. Indeed, when you want to fill a cup at a fountain you do not care about which drops go in the cup, you just care about the cup being filled. If you could simply care about the number of packets you receive without having to care about which packets you receives, the protocols would be greatly simplified, and feedback would not be needed. Hence, the set of encoded packets should be infinite so that all packets are different and equivalently useful.

Digital Fountains

The typical examples of problems that can be addressed with fountain codes are the followings. First, let us consider the broadcast of software updates to millions of receivers through a satellite as depicted on Figure 1.7a. The receivers are not synchronized and have very different connectivity properties. A naive approach sending the update cyclically is inefficient since a receiver missing some packet has to wait a full cycle before it gets another opportunity to receive the packet it missed. The use of codes in these systems allows each receiver to randomly pick encoded packets: since n is very large, they are likely to be different. Once it has received enough different packets, it can decode and recover the native packets (x_1, x_2).

Second, TCP/IP protocol relies on retransmissions which are efficient in unicast communications even if they introduce a delay. However, sending some content to millions of receivers requires the use of multicast, as shown on Figure 1.7b. In this case, the receivers suffer the losses on all channels between them and the source. As losses reported by the receivers differ, the source receives a large set of different retransmission requests. By adding erasure correcting codes, encoded packets are transmitted and the losses are simply handled by decoding the received packets to recover the native packets instead of requiring retransmission from the source.

Third, erasure correcting codes can also ease cooperation between servers. In the case of parallel downloads, shown on Figure 1.7c, a single receiver gets the data from multiple senders. If non-encoded data is transmitted, the senders must coordinate to avoid the receiver from receiving several times the same packet. Thanks to erasure correcting codes, the packets to transmit can be picked at random in a very large set of encoded packets, thus avoiding conflicts without any need for coordination.



(a) Radio Broadcast (b) IP Multicast (c) Parallel downloads

Figure 1.7: Examples of uses of fountain codes

The works in [13, 73] advocate the use of large block codes that are rateless (i.e., $k/n \to 0$), or at least have a very low rate: with such codes, the set of encoded packets that can be transmitted is much larger than the set of decoded packets thus reducing the probability that two received packets are identical and hence useless for that matter. For example, the decoding may be possible once the receiver has acquired 1,000 different packets from a set of 10,000 encoded packets. Previously mentioned codes (Reed Solomon, Random Linear Codes. . .) cannot be used in such cases as they either limit the total number of encoded packets that can be generated or rely on high complexity encoding and decoding procedures.

Sparse codes [41, 69] are good candidates to address such settings. Sparse codes differ from traditional codes as they do not have ideal properties in terms of Hamming distance but exhibit good performance on average. Their sparsity allows these codes to be decoded by a simple message-passing algorithm (typically, belief propagation) at a lower complexity than the previously mentioned codes. The first sparse codes designed for the aforementioned problems are Tornado Codes [67] in 1997. In 2002, a major breakthrough was performed by Luby by defining LT Codes [66]. LT codes where extended in a third proposition of fountain codes, Raptor codes [94], in which a message is first pre-encoded by some code and then encoded by an LT code. We briefly present LT codes since one of our contributions relies on them.

LT Codes

LT codes [66] transform a set of k native packets into encoded packets such that any $k(1+\epsilon)$ encoded packets allows recovering the native packets (ϵ is a low overhead). The encoding process is randomized and consists in picking a degree d according to a Robust Soliton distribution (described in [66] and Figure 1.8). The encoded packet is then built by xor-ing d native packets chosen uniformly at random.

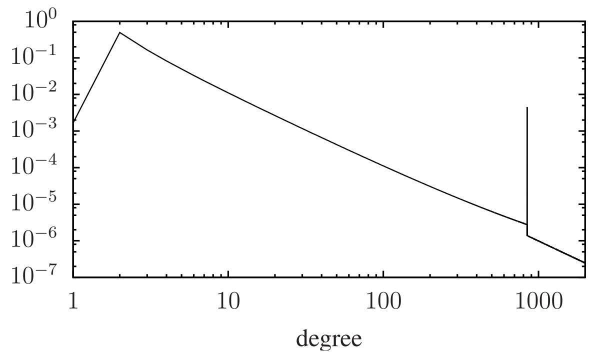


Figure 1.8: A Robust Soliton distribution for $k = 2000$ (i.e., probability of picking each degree).

LT encoded packets are organized into a specific data structure named a *Tanner graph* [98]. A Tanner graph is a bipartite graph where the nodes in the first set are native packets and the nodes in the second set are the encoded packets received. There exists an edge from a native packet x to an encoded packet y if x is involved in the linear combination forming y. The degree of a packet is the number of edges originating from (resp. pointing to) this particular node and is denoted by $d(\cdot)$. Figure 1.9 depicts an example of a Tanner graph.

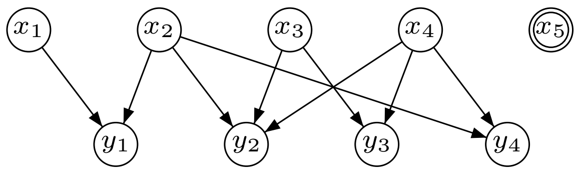


Figure 1.9: An example of a Tanner graph: $y_4 = x_2 \oplus x_4$, x_3 has degree 2 and y_2 has degree 3. x_5 has been decoded.

In LT codes, the specific degree distributions of the corresponding Tanner graph (Figure 1.9) allows decoding by belief propagation, with a complexity of $\mathcal{O}\left(mk \cdot \log k\right)$ operations. The decoding algorithm is the following. Every time a native packet x is received (i.e., an encoded packet of degree 1) or decoded, every encoded packet y involving x (i.e., to which x points) is xor-ed with x and the edge between x and y is deleted. Thanks to the Robust Soliton distribution, one of the updated encoded packet has now a degree 1. Hence, the decoding process can continue until all native packets are recovered. This algorithm is illustrated on a small example by Figure 1.10.

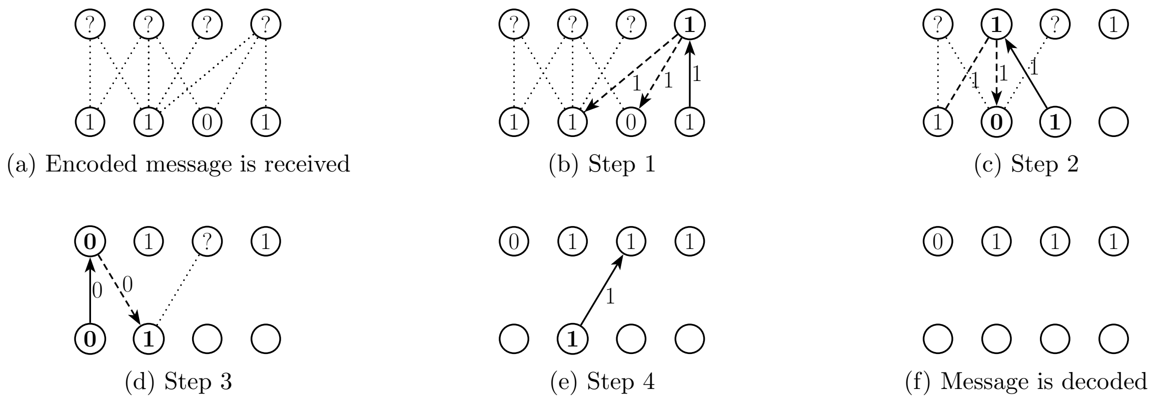

(a) Encoded message is received (b) Step 1 (c) Step 2
(d) Step 3 (e) Step 4 (f) Message is decoded

Figure 1.10: Decoding an LT code by belief propagation.

The performance of LT codes is directly linked with the degree distributions. On the one hand, it is clear from the previous paragraph that the belief propagation decoding algorithm requires at least one encoded packet of degree one. More generally, the lower the degree of the encoded packets the faster the decoding. On the other hand, the higher the degree of the encoded packets, the less redundant the sent packets. It is shown in [66] that the optimal distribution of degrees for the encoded packets is the Robust Soliton, depicted in Figure 1.8. The Robust Soliton distribution is composed of approximately 50% of encoded packets of degree 1 or 2 allowing to bootstrap belief propagation, and an average degree of $\log k$ resulting in low complexity decoding. Secondly, to ensure optimal decoding, all native packets must have roughly the same degree.

An interesting property of LT codes is that they are rateless. This means that an infinite set of encoded packets can be generated as opposed to Reed Solomon codes where the number of encoded packets (code words) is finite. Therefore, a major difference is that any packet can belong to an LT code, while for a Reed Solomon code, only a few specific possible combinations belong to the code. Random linear codes, presented earlier, share this same property as any random combination can belong to the code; hence the number of encoded packets that can be generated is infinite. This property allows multiple source to encode simultaneously without coordinating (as done in Chapter 2). Yet, this is also of interest for building RLNC (Section 1.1.3) and LTNC (Chapter 3).

In spite of their appealing properties these codes have some limitations. Indeed, to produce a new encoded packet, the full knowledge of native (decoded) packets is required. Therefore, in Figure 1.7b, the intermediary devices can simply forward what they receive without performing any coding operation. This becomes a limitation if the structure of the dissemination is not a tree but a graph. In Chapter 2, we propose a dissemination algorithm that takes this into account by ensuring a non-uniform progression of devices, so that some devices decode and become source more quickly than the majority of other devices. It increases the performance as sources send newly encoded packets that are much more likely to be useful than forwarded packets.

If we consider a global communication system, a network, the results by Shannon on individual channels are not sufficient to build a globally optimal transmission scheme. To this end, the information theory has been extended to apply to whole graphs instead of single edges. Network information flows [2] allow defining the achievable capacity for arbitrary networks (graphs).

1.1.3 Network Codes

In 2000, network coding [2] has been proposed as an alternative to forwarding at intermediary devices. The basic intuition is that by allowing encoding operation to occur within the network at each intermediary device, the global throughput of the network can be increased. More precisely, the authors have shown that as opposed to coding only at the source, network coding allows multicasting at the maximum flow of the graph. In the example of Figure 1.11, if the intermediary device $F*$ is not allowed to encode, the middle edge carries either x_1 or x_2 and the average flow that reaches the two destinations is 1.5 in spite of the maximum flow of this graph being 2.

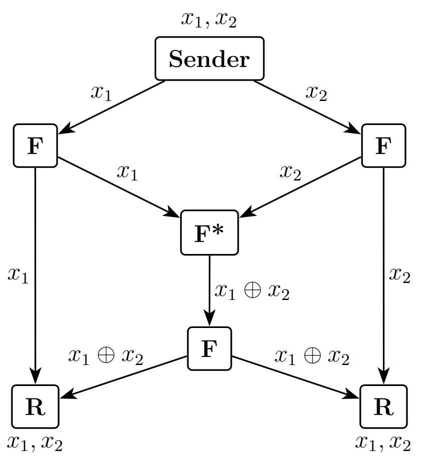


Figure 1.11: Network Coding allows reaching the maximum flow.

Random Linear Network Codes

Furthermore, linear codes have been shown to be sufficient to achieve the maxflow [57, 60]. In particular, random linear network codes [49] are a straightfoward solution for implementing network coding in a decentralized setting. In random linear network codes, the encoded packets correspond to random linear combinations of native packets. Contrary to channel coding, network coding allows intermediary devices to generate fresh encoded packets from only a partial knowledge of the native packets. Figure 1.12 shows an example of random linear network coding. The source encodes by forming random linear combinations of native packets (y_1, y_2, y_3 and y_4). The intermediary devices recode by forming random linear combinations of encoded packets ($y_5 = 2y_1 + y_4$). An important property of this scheme is that the resulting recoded packets are also random linear combinations of native packets ($y_5 = 3x_1 + 2x_1 + 3x_3$) and y_5 is a valid codeword. The coefficients of the combinations are carried as headers of encoded packets, so the receiver can decode, by a Gauss reduction, and recover the native packets.

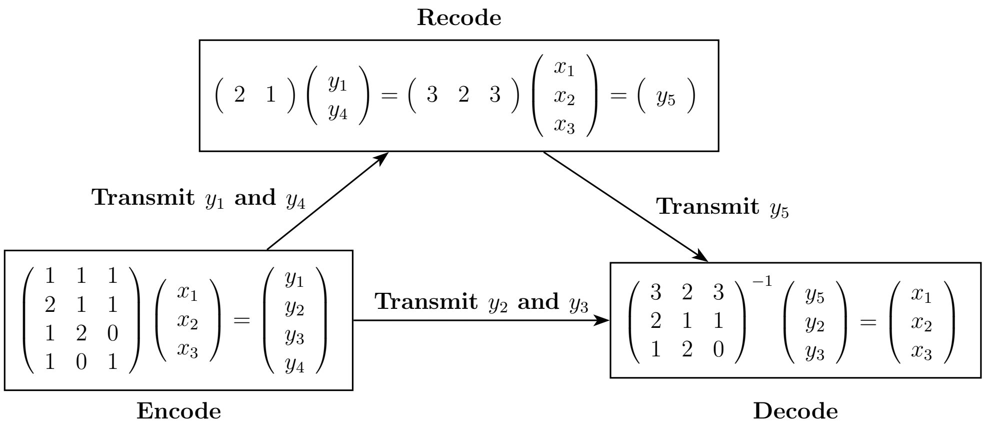

Recode
$\begin{pmatrix} 2 & 1 \end{pmatrix}\begin{pmatrix} y_1 \\ y_4 \end{pmatrix} = \begin{pmatrix} 3 & 2 & 3 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} y_5 \end{pmatrix}$
Transmit y_1 and y_4
Transmit y_5
$\begin{pmatrix} 1 & 1 & 1 \\ 2 & 1 & 1 \\ 1 & 2 & 0 \\ 1 & 0 & 1 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix}$
Transmit y_2 and y_3
$\begin{pmatrix} 3 & 2 & 3 \\ 2 & 1 & 1 \\ 1 & 2 & 0 \end{pmatrix}^{-1}\begin{pmatrix} y_5 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$
Encode
Decode

Figure 1.12: Random Linear Network Coding.

A major concern about random linear network codes is that their lack of structure implies a high complexity decoding (i.e., Gauss reduction). According to [68, 101], the costs of decoding clearly over-take gains obtained from using network coding. Some variations of random linear network codes have been proposed to allow lower encoding or recoding costs using generations, sparse encoding and recoding matrices [68, 71, 99, 101], or to allow partial recovery of packets by using specific (triangular for example) recoding matrices [63]. Yet, all these implementations still relies on Gauss reduction for decoding. In Chapter 3, we tackle this issue with new low complexity network codes based on belief propagation decoding.

Distributed Source Coding

Distributed source coding [3, 4, 33, 46, 54, 81] is a related topic in which the native packets are distributed over a set of sources and sources must cooperate to form encoded packets. This scheme is less powerful than network coding as it does not allows devices with encoded packets to take part into the encoding process. In the rest of this document, we do not consider distributed source coding due to its limitations. However, some of these works, closely related to LT codes [66], are discussed and compared to our LT network codes in Section 3.5. The global objective of such schemes is to transform a set of devices, connected as a graph and containing native packets, into a (larger) set of devices still connected as a graph yet containing encoded packets. The resulting set of devices can handle failures since it stores encoded packets.

The various codes we presented differ in their properties. Overall, they all expand a set of k native packets into a larger set of n encoded packets such that any subset of k encoded packets allows recovering the k original packets.

Erasure correcting codes (Channel codes) are the less powerful codes we presented (Reed-Solomon codes, fountain codes such as LT codes). They allow a device with the full knowledge (i.e., knowing all native packets) to encode. Devices with partial knowledge cannot perform any coding operation and can simply forward. Yet, in many communication systems, which map to single channels or trees, they allow to achieve the capacity (i.e., transfer optimally).

Distributed sources codes are slightly more powerful as they allow devices to cooperate to form encoded packets even though each has a partial knowledge of data (provided that what they know are native packets).

Network codes are much more powerful as they allow devices with partial knowledge (including devices that know only encoded packets) to form new freshly encoded packets by combining the information they have.

Yet, even if network codes could be used in all cases where channel codes are used, they have a major disadvantage, that is that their implementation involves complex calculations, while efficient implementations of channel codes exist (LT codes for example). Therefore, the choice of a code comes down to finding an appropriate tradeoff between the computational power required and the desired power of the code. Overall, these various codes allows coping with the unreliability of underlying systems to offer reliable services and are very efficient for this.

1.2 Applying Codes to Distributed Systems

We now present dissemination and storage systems. We also examine how codes described above, designed to transmit information over channels, map to distributed systems. We study how they have been used to enhance dissemination over random systems and how they have been used to build reliable storage from unreliable devices. Both erasure correcting codes and network codes, which are especially adapted to systems that map to graphs, are considered since they have been used to enhance distributed systems. This section does not present an exhaustive list of all uses of codes in distributed system but focus on some propositions.

1.2.1 Dissemination

Dissemination consists in sending some data to a large set of devices (peers) in a collaborative way. On controlled networks, it can be performed efficiently using IP multicast. Yet, IP multicast is often not available thus requiring applicative-level solutions. Since a central server (source) can quickly be overwhelmed if the number of client devices is large, it has been proposed to adopt distributed dissemination algorithms (i.e., peer-to-peer). Such algorithms leverage the bandwidth available at each device (peer) so that peers upload blocks to each other, thus relieving the source.

Structured push

Various approaches have been explored to perform dissemination. A first approach has been to structure dissemination by organizing peers as a tree [17] or as a ring [96]. The resulting structure can then be used to disseminate the data to peers with a minimum overhead. For example, Scribe [17] organizes peers in a tree. The messages are then disseminated along tree branches, thus ensuring that all peers receive the data exactly once. To disseminate large content SplitStream [16] has proposed to enhance Scribe and to use multiple structures (i.e., multiple independent trees). This enhancement also allows handling the unreliability: in Scribe, the failure of a peer impacts its whole sub-tree while, in SplitStream, codes allow coping with the failure of a subset of trees (the failures are independent) thus allowing reliable dissemination over unreliable structures. Other similar approaches have been proposed using the same basis.

Yet, these schemes rely on fragile structures (ring, trees. . .) and, in highly dynamics systems, maintaining the structures increase the costs.

Unstructured pull

The use of unstructured algorithms reduces the impact of failures. This approach has been widely used in peer-to-peer systems as one of the most widely used peer-to-peer application, namely Bittorrent, rely on an unstructured network of peers. Bittorrent [26] uses bidirectionnal communications to indicate which blocks can be sent and to request particular blocks to other peers. This is similar in spirit to using an Acknowledge/Request scheme. Other protocols using bidirectionnal communication are found in [40, 88].

There has been some proposals to extend Bittorrent with codes. First, Avalanche [43, 44] proposed to use network coding to encode packets. The file is split in *generations* and each generation is split in native packets. The native packets within a generation can be combined together using random linear network codes. Collaborating peers can then request each other recoded packets for generations that are not yet complete (i.e., generation for which some packets are missing). Network coding allows increasing diversity thus making cooperation between peers easier leading to a better usage of available bandwidth. This same goal is also explored in [65] where the authors use channel codes (here, random linear codes) to encode data.

Unstructured push

Finally, unstructured push-based approaches have some advantages such as simpler algorithms (no need for coordination). In this thesis, we mainly consider such approaches and more specifically epidemic dissemination (i.e., gossip) [38, 39, 55, 70]. In push-based epidemic dissemination, at each cycle, each peer selects (randomly) another peer and forwards data to it. Each peer forwards any packet it receives for the first time to f fanout peers. If f is sufficiently high then, in the end, all peers in the network receive the packet w.h.p.

In Chapters 2 and 3, we consider epidemic push-based dissemination as used in gossip protocols for their simplicity and explore the use of fountain codes over such protocols.

1.2.2 Storage

In the previous subsection, we have described how codes have been used to transmit data in space (i.e., over the network) while preserving data from errors. Another use of codes is to transmit data over time (i.e., guarantee that despite of failures that can affect some data, the whole data will remain available in the future). Codes are used for this purpose in storage systems.

A storage system consists of several devices (disks, peers...) [7, 28, 42, 84] that are aggregated to store large amounts of data. Yet, to be useful, a storage system must guarantee that any stored data will remain persistent. Hence, a major challenge in storage is to allow data to be stored safely, thus preventing a single device failure from disrupting the whole system. This has been studied at the hardware level for aggregating a small number of disks and at the software level for aggregating devices at a larger scale through the network to build large scale distributed storage systems.

Raid : Redundant Array Of Inexpensive Disks

Since the 1980s, codes have been applied at the hardware level, and at a small scale, for providing reliable storage systems from unreliable storage disks. The most widely used reliable storage is Raid [78]. A Raid can be described as a small distributed system with an efficient communication network (i.e., the local bus) between devices (i.e., disks). The methods used in Raid are also applied to larger distributed systems: the list of proposed methods is rather exhaustive thus describing many possible candidates for large scale distributed storage systems. Raid offers a hardware-supported way to aggregate several disks and tolerate single or double disk failures. There are currently 7 levels of Raid which offer various functionalities.

Raid 0 offers large storage by aggregating several disks to offer a single *virtual* disk whose storage capacity is the sum of the individual disks. This level does not provide any tolerance to failures but simplifies the management of disks and offers the ability to store files larger than one single physical disk.

Raid 1 offers tolerance to one failure through replication (Figure 1.13b). It uses $2k$ disks. Each data stored is replicated (mirrored) on another disk so that the failure of a single disk does not provoke permanent losses of data.

Raid 2 offers tolerance to errors (i.e., disks reading incorrect values without noticing). By using Hamming Codes and laying out the data on several disks, it allows tolerating one incorrectly read bit. The number of disks used depends on the Hamming Codes used. This level is deprecated as disks now already have an internal error correction/detection mechanism so that disks either read data correctly or fail at reading data. Disks behave more like an erasure channel; hence, Raid main use is to tolerate full disk crashes and not bit errors.

Raid 3, 4, 5 offer tolerance to one failure through erasure correcting codes (Figure 1.13c). They use $k+1$ disks. They rely on a simple parity erasure correcting code (i.e., the additional disk stores the XOR of the other disks). The three schemes slightly differ in the way the erasure correcting code is applied.

Raid 6 is an extension of Raid 5 that can tolerate up to two failures using erasure correcting codes. It uses k+2 disks to tolerate 2 failures.

Moreover, these levels can be combined to form hybrid schemes by applying successively several levels of Raid. In Raid 1+0, the individual disks are first mirrored and then the groups of two disks are aggregated to provide a large volume that can tolerate at least one failure. In Raid 0+5, the disks are first aggregated and then, the groups are organized to offer tolerance to the failure of the disks of one group. Moreover, other complex schemes have been designed to be able to handle more disks or more failures [80].

To sum up, redundancy can be implement either through replication (Raid 1), or through erasure correcting codes (Raid 5 or Raid 6). Erasure correcting codes are much more efficient as they can tolerate t failures with $k+t$ disks while replication needs $(t+1)k$ disks. This is also illustrated on Figure 1.13 where replication uses 6 disks to tolerate one failure when erasure correcting codes need only 4 disks.

Such systems are designed to offer the best reading/writing throughput. Indeed, the disks are accessed through the local bus: coding operations must be very efficient so as not to be a limiting factor. This has lead to various designs focusing on high performance coding, efficient load balancing and efficient writes [80]. Moreover, failures remain exceptional thus reducing the importance of the way failures are handled.

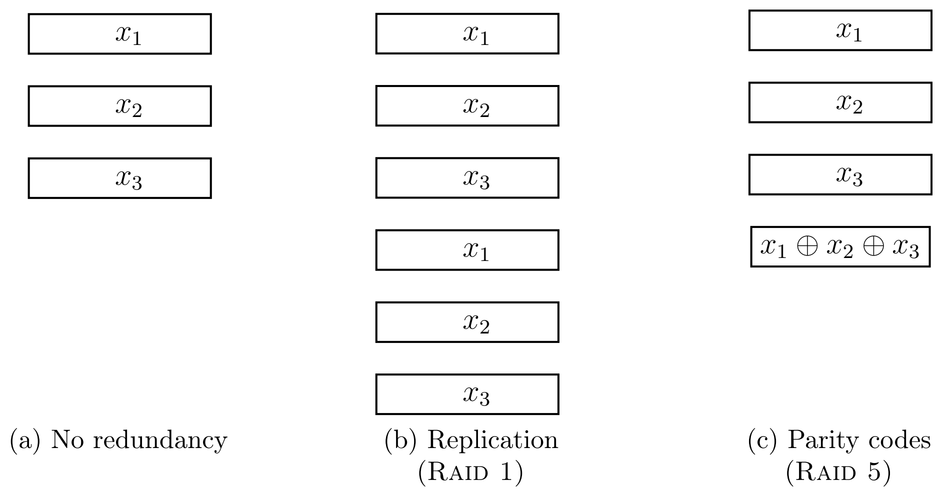


*Figure 1.13: Redundancy in storage systems (*RAID*)*

1.2.3 Large Scale Distributed Storage

In the 2000s, cheap networking technology and widely available Internet access have pushed the deployment of distributed systems that aggregate several disks through networks. These systems rely on the very same tools as Raid systems (replication, erasure correcting codes, hybrid schemes) to provide reliable distributed storage systems from unreliable devices, but with different objectives. The two main differences with Raid systems are an increased dynamicity, and a more limited interconnection (network instead of local bus). It worth be noted that Raid and large scale distributed storage are complementary as the first handles disk failures, and the second handles network or computer failures. Hence, as we explain hereafter, a low repair cost is a required feature of distributed storage systems while the performance of coding operations becomes less important.

To handle the increased dynamicity (i.e., increased failure rate due to the large number of disks and temporary network failures) when compared to Raid, a higher replication factor, or a higher number of correctable erasures is required [62,86,102]. Yet, simply adding redundancy when storing the file is not enough: the redundancy must be maintained. To ensure the persistence of data, the storage system must self-repair so that the redundancy does not fade off over time. When a failure is detected, a spare device joins the system and regenerates the lost redundancy. The regenerating procedure depends on the kind of redundancy used. For replication, an additional copy of the lost block is made. For erasure correcting codes, the encoded blocks cannot be copied from another stored block. The joining device must download and decode the whole file before encoding the lost block again. Clearly, the repair cost of erasure correcting codes is much higher than replication since downloading the whole file implies transferring k blocks when replication transfers only one block. Therefore, replication has a high storage overhead and a low repair cost while erasure correcting codes have a low storage overhead but a high repair cost. An ideal solution for redundancy should have both a low storage overhead and a low repair cost.

To reduce the global maintenance cost of distributed storage systems, it has been proposed to repair only permanent failures. Such approaches require to distinguish between permanent and temporary failures. To this end, it has been proposed to use a timeout to mark a device as failed only after a significant amount of time [7, 8]. Moreover, devices that come back into the system must be reintegrated, with their data, into the system to avoid non-necessary repairs [25]. Yet, these mechanisms only reduce the frequency of repairs; the repair cost of permanent failures and incorrectly classified temporary failures remains high. Hence, it is still crucial to offer a redundancy mechanism that has a low repair cost.

Several strategies have been proposed to reduce the repair cost with respect to network communications.

Hybrid schemes, mixing replication and erasure correcting codes, allow performing some repairs by simply copying the missing block from its replica [86]. In a first scheme, a single device may maintain a full copy of the file thus being able to encode to produce missing blocks. In an alternative scheme, each encoded block is replicated over several devices so that if at least one of the replicas is not lost, the repair can be performed without downloading and encoding. These schemes add complexity to the system as the repair must be performed differently if all replicas have failed. Moreover, they are not optimal with respect to storage.

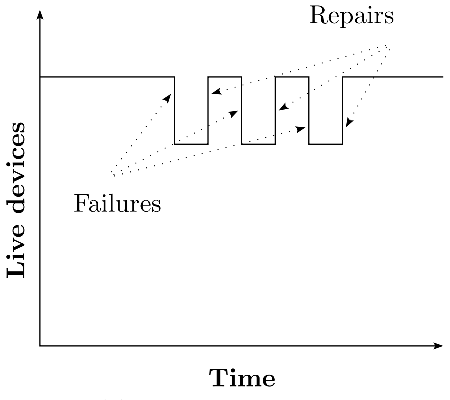
Specially structured codes allow recovering missing blocks by combining only a small subset of encoded blocks [36, 75]. For example, the block $x1 \oplus x3$ can be rebuilt by combining only 2 blocks $x_1 \oplus x_2$ and $x_2 \oplus x_3$ of the code of length $k = 3$ $(x_1 \oplus x_2, x_1 \oplus x_3, x_2 \oplus x_3, x_1 \oplus x_2 \oplus x_3)$. However, these schemes are neither optimal with respect to storage, nor optimal with respect to repair cost.

Lazy/delayed repairs consist in performing several repairs simultaneously instead of independently so as to the reduce the repair cost [7, 29, 30]. When using erasure correcting codes, the repair consists of two phases: *(i)* downloading and decoding the whole file, *(ii)* encoding to produce the lost blocks. Performing repairs individually is costly (Fig. 1.14a). Hence, waiting for the number of failure to go above a threshold and grouping repairs allows factoring most of the costs due to the first phase (Fig. 1.14b). However, these schemes are not optimal in term of repair cost and add more parameters (typically, the threshold for repairing) to the system.

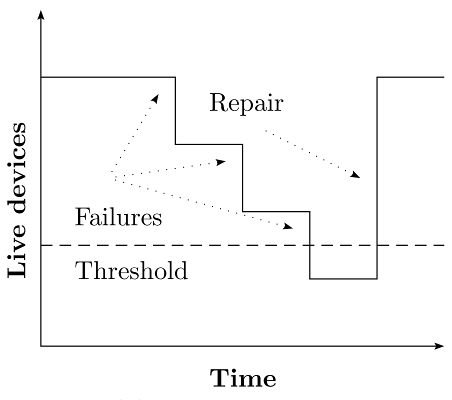
Network codes (regenerating codes) allow offering both a low storage overhead and a low repair cost. Network coding has been used in various systems (including sensor networks) to ensure the persistence of stored data [1, 33, 34, 54, 63]. One of the most important contribution in this area has been made by Dimakis *et al.* [31, 32]. They provide theoretical bounds on the amounts of information that must be transfered and stored to guarantee that data does not get lost. An optimal coding scheme achieving their bounds can be implemented with random linear network codes [49]. These results are described in more details in Chapter 4 (Section 4.2.3, Page 67).

Clearly, regenerating codes defined by Dimakis *et al.* outperform other approaches. Yet, they suffer several limitations. First, they can repair optimally only single failures. Second, they assume a static system where all repairs (during the whole lifetime of the system) are performed in the same conditions. In Chapter 4, we build upon the work by Dimakis *et al.* to offer new regenerating codes that are more flexible and support repairing multiple failures. The added flexibility offered by our codes makes them very good candidates to implement a distributed storage system offering an optimal tradeoff between storage and repair costs.

In spite of the ability of codes to correct errors and enhance dissemination, codes are often left around because of the high decoding complexity (dissemination) or the high repair cost (storage). In the next chapters, we present our contributions aiming at making codes more appealing by limiting the side costs linked with the use of codes. To this end, we propose to either increase gains, either reduce the decoding complexity, or reduce the repair cost.



(a) Immediate repairs



(b) Delayed repairs

Figure 1.14: Delaying repairs allows performing multiple repairs at once.

Chapter 2

Fountain Codes in Dissemination

In this chapter, we propose an epidemic dissemination protocol that leverages fountain codes properties. Fountain codes are interesting candidates for coding in distributed systems since there exist efficient fountain codes implementations with low complexity decoding. Yet, the dissemination protocol needs to be tailored to the use of codes so as to best exploit them. The use of codes allows avoiding multiple receptions due to the randomness of the dissemination process.

This work as been published [20] in the proceedings of the 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems in 2009.

2.1 Motivation

Gossip protocols are now recognized as a solid and robust approach for disseminating content in large scale-systems. They provide an efficient alternative to tree-based approaches, typically fragile in highly dynamic environments. In gossip protocols, peers periodically push contents to f (called the fanout in the sequel) peers picked uniformly and randomly among all peers. A message is first disseminated from a source. When receiving the message for the first time, a peer forwards it to f other random peers until the message reaches only peers that have already received it. In gossip, two phases can be distinguished in the dissemination: *(i) an exponential growth phase* during which the message spreads exponentially fast since initially the probability to reach a peer that has not yet been reached is very high; and *(ii) a shrinking phase* where the rate of dissemination diminishes, as eventually the probability to reach already informed peers increases drastically [55]. This phase typically generates a large number of duplicates. Yet, it is necessary to achieve a reliable dissemination.

This shrinking phase is generally recognized as the Achilles' heel of gossip protocols: this is a fatal pitfall but also the price to pay for achieving reliability in highly dynamic systems. Moreover, the costs associated with the shrinking phase becomes an increasingly serious issue, overhead wise, as the size of the content to disseminate increases. This typically explains the recent success of video streaming protocols which relies on a push gossip protocol for locating the content with small messages while the content is subsequently pulled [58, 107].

In this chapter, we propose to combine the use of gossip protocols with fountain codes to benefit fully from the exponential growth phase of gossip protocols while avoiding suffering from the shrinking phase. Fountain codes typically encode the full content so that encoded packets embed redundancy [13, 73]. However, the use of fountain codes precisely removes the notion of useless redundancy. Typically, a fountain code generates infinity of encoded packets from an original content of k packets. Any $k(1+\epsilon)$ packets, with $\epsilon \approx 0.04$ for long enough codes ($k > 1000$), are enough to decode the full content. Even though network codes [2] are usually considered as more powerful than fountain codes, we choose to rely on fountain codes which have a lower complexity[1]. Their lower complexity makes them more practical to deploy in environments where the amount of data to be encoded is important and the computation resources are scarce.

We propose the design and evaluation of FoG (Fountain codes On Gossip) a dissemination gossip protocol relying on fountain codes. FoG fully leverages the potential of gossip protocols through the following principles.

- First, FoG removes the need for the shrinking phase (i.e., gossip stops before reaching the shrinking phase) by simply increasing the number of exponential phases. This is due to a clever use of content encoded through fountain codes. Typically k gossips are turned into $z > k$ shorter gossips[2] where only the exponential growth is considered. The basic intuition behind this is that peers that have been missed during the exponential phase for one dissemination recover by being reached over the subsequent ones. Eventually, all peers receive the z packets required to decode the full content. This increases the marginal utility of any packet's dissemination. This scheme is illustrated on Figure 2.2.

- Second, FoG implements a biased dissemination to favor the most advanced peers. The bias allows some peers to complete more quickly so that they become sources that have the ability to help other others by producing new innovative packets by re-encoding the content as soon as possible. This exploits the fact that peers can receive the z distinct packets from any source. All encoded packets are equally useful, and two encoders produce distinct and independent

[1]Network codes involves a Gaussian elimination $O(k^3)$ for decoding while practical fountain codes such as LT Codes [66] exploit belief propagation decoding $O(k \ln k)$.

[2]z is automatically determined by the protocol which disseminates as long as some peers have not completed.

encoded packets thanks to the infinite size of the set of encoded packets. To this end, FoG gradually builds a split-graph overlay by sampling the system over all non-completed peers. This means that the f gossip targets on each peer are chosen randomly among the peers that have not yet received the full content.

The rest of the chapter is structured as follows. The design rationale is given in Section 2.2. The details of FoG are provided in Section 2.3. Experimental results are reported in Section 2.4. Finally, we review some related works in Section 2.5.

2.2 Design of FoG

In this section, we first present the system model and then explain the design of FoG.

2.2.1 System model

We consider a system of n peers in which peers cooperate to disseminate a file. Each peer is supposed to cooperate as specified: byzantine and selfish behaviors are out of the scope of this chapter. A peer that has a full copy of the file is called *complete*, *incomplete* otherwise. In the figures of this chapter, the complete peers are black and the incomplete ones are white.

The n peers are connected by an unstructured overlay network. The overlay network is maintained by a gossip-based peer sampling protocol [52]. Such protocols tend to build overlay network the topology of which is close to a d-regular random graph where d is the out-degree of peers. To this end, each peer maintains a view of size d, constantly updated by the gossip peer sampling protocol. In FoG, the complete peers have no incoming edges and never appear in samples provided by the peer sampling service. This means although complete peers remains connected by maintaining a view, they are absent of any views and they are never picked during the dissemination.

The dissemination protocol runs over the network of n peers following a push protocol. Initially, the source is assumed to be the only complete peer. A dissemination is considered finished, when all the n peers have a copy of the file. The file is divided in k small packets. Those packets are disseminated independently. The source sends packets periodically to one of its neighbors. When a peer receives a new packet, it forwards it to f *(fanout)* neighbors. This scheme ensures a complete dissemination with high probability as long as $f = O(\log n)$. To limit flooding, we introduce a *TTL* (*time to live*, which defines the number of hops after which a packet is not forwarded anymore) that significantly reduces the number of peers that receive the packet thus reducing multiple receptions and removing the shrinking phase. However, removing the shrinking phase results in significantly more peers not receiving the packet.

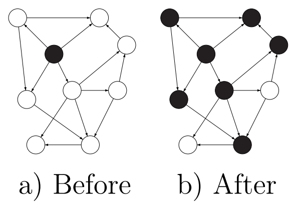


Figure 2.1: Exponential phase

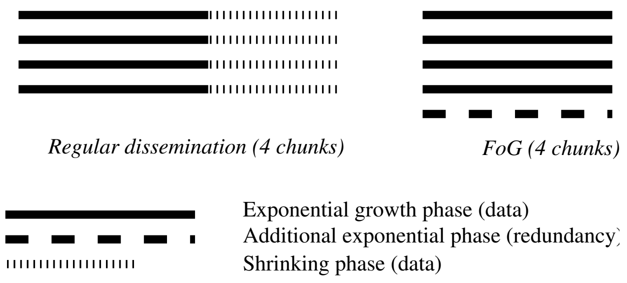

Regular dissemination (4 chunks)
FoG (4 chunks)
Exponential growth phase (data)
Additional exponential phase (redundancy)
Shrinking phase (data)

Figure 2.2: FoG keeps only growth phases

2.2.2 Design rationale

As mentioned earlier, gossip dissemination is implicitly characterized by a first phase, called exponential growth, where the data is spread exponentially fast, followed by a second phase, called the shrinking phase, where a lot of redundancy is introduced (i.e. peers which have already received the message receive it again) but is required to reach the remaining peers. This is due to the random selection of the peers to gossip to. As illustrated on Figure 2.1, without adding mechanism to ensure full dissemination, some peers are not able to receive the data. The idea behind FoG is to fully exploit the strength of gossip dissemination, namely the exponential growth phase, without suffering from its weakness, the shrinking phase. FoG achieves this by increasing the number of disseminations, but limiting them to the exponential growth phase. Obviously, a gossip dissemination stopping after the exponential growth phase misses many peers. We therefore use fountain codes so that the number of sources is infinite and the number of disseminations can be infinite. As explained in Section 1.1.2, as long as a peer receives any k different encoded packets being disseminated, it can recover the native packets. The termination of the dissemination is naturally detected overall as the complete peers are gradually removed from the views.

FoG replaces the shrinking phases by additional exponential growth phases. To illustrate, consider a content of k packets to disseminate to a set of peers. packets are disseminated in parallel by independent exponential-growth phases. At the end of each exponential-growth phase, $\frac{n}{2}$ peers have received the disseminated packet. Note that this is achieved by associating an empirically determined finite TTL to each packet, enough to reach $\frac{n}{2}$ peers. At each dissemination, $\frac{n}{2}$ peers miss the data disseminated. To ensure that all peers complete regardless, additional disseminations of encoded data are performed. We "factor out" the shrinking phases of all disseminations in a few disseminations of redundant content, as shown on Figure 2.2.

To keep the shrinking phase, $TTL = \infty$ so that we would reach all n peers at the end of each dissemination and no peer would miss the data with high probability.

The dissemination protocol disseminates data to peers that are provided by a peer sampling protocol [52] that builds an overlay network. FoG structures the overlay network as a split-graph (Figure 2.3) that is a composition of a random

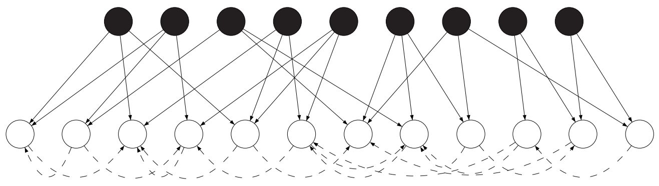


Figure 2.3: A split-graph overlay.

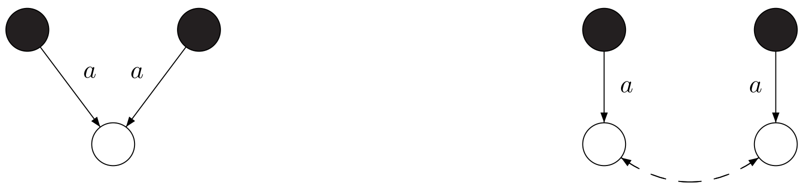
graph connecting incomplete peers and a bipartite-graph connecting complete peers to incomplete peers. This structure matches the dissemination progress. This enables complete peers to act as additional sources and speed up the dissemination: they encode the content and inject new encoded packets in the clique (the random graph connecting incomplete peers). Incomplete peers cooperate to disseminate the encoded packets to each other so that no bandwidth is lost.

In order to ensure termination, FoG keeps disseminating as long as some peers miss some packets they require in order to be able to decode and recover the original data. Complete peers remove all their incoming links. This is achieved simply by filtering out complete peers during merges and not propagating their information between incomplete peers. They leave the clique of the split-graph and do not receive redundant data once they have finished. The overlay maintenance protocol also automatically detects termination. A complete peer, acting as a source, stops sending data when its view becomes empty: it means that it does not know any incomplete peer. Therefore, the protocol eventually terminates.

The proposed protocol needs all packet disseminations to be independent and equally useful so that any subset of $k(1 + \epsilon)$ encoded packets received during exponential growth phases can be decoded to recover the original data. This is achieved in FoG through the use of fountain codes.

As explained in the previous chapter, fountain codes are erasure correcting codes that allow building an infinite set of encoded packets from k native packets. As soon as $k(1 + \epsilon)$ encoded packets are received, it is possible to recover the k native packets. Yet, fountain codes [12,13,73], as all erasure correcting codes [69], require the complete original data so as to generate new encoded packets. Therefore, only peers that hold the complete data can encode and produce new encoded packets. Network codes, also presented in the previous chapter bypass this limitation and allows all peers to produce new encoded packets.However, contrary to fountain codes, the implementations of network codes are rather inefficient as they rely on complex decoding algorithms. Hence, we limit ourself to fountain codes.

In FoG, fountain codes allow many distinct sources to start distinct disseminations without coordination. All the disseminations distribute different encoded packets and never conflict. Without fountains codes, we would suffer redundancy between disseminations thus reducing the cooperation efficiency. Figure 2.4 shows two simple cases where the black nodes send the same information to white nodes. If the black nodes had sent different information (a, b) instead of (a, a), then the overall dissemination would have been more efficient.



(a) The incomplete peer receives the packet of data a twice

(b) Incomplete peers cannot cooperate as they both receive a.

Figure 2.4: Difficulties in cooperation.

Fountain codes allow sources to start new disseminations as long as some peers have not completed. All these disseminations are independent and useful regardless of the packets the remaining peers miss.

Eventually, as only peers that hold the whole data can produce new encoded packets using fountain codes, the peers that hold the whole data are much more useful than the other peers are. Therefore, we bias the dissemination to deliberately favor some peers that will be useful to other peers as soon as possible. As incomplete peers are not able to encode and generate new innovative packets, the dissemination favors complete peers. This point differs from many peer-to-peer protocols where peers tend to progress at the same speed and finish all together.

2.3 FoG

The FoG dissemination protocol relies on random (uniform) samples of incomplete peers provided by a gossip-based peer sampling service. The peer sampling service is based on a *gossip-based peer sampling service* [52] that builds a d-regular random graph. We modified the protocol to systematically exclude complete peers from samples. Both the dissemination protocol and the peer sampling service are biased for a more efficient dissemination and are presented below.

2.3.1 FoG peer sampling: Split-graph overlay

FoG relies on a peer sampling service that creates a split-graph overlay network upon which the dissemination is implemented (Figure 2.3). Each peer maintains a *view*, which is a sample of the system provided by a peer sampling protocol. A view is an aggregate of information about at most s peers. Each peer entry in the view contains its IP, its completion level and its age. The peer sampling service in FoG ensures

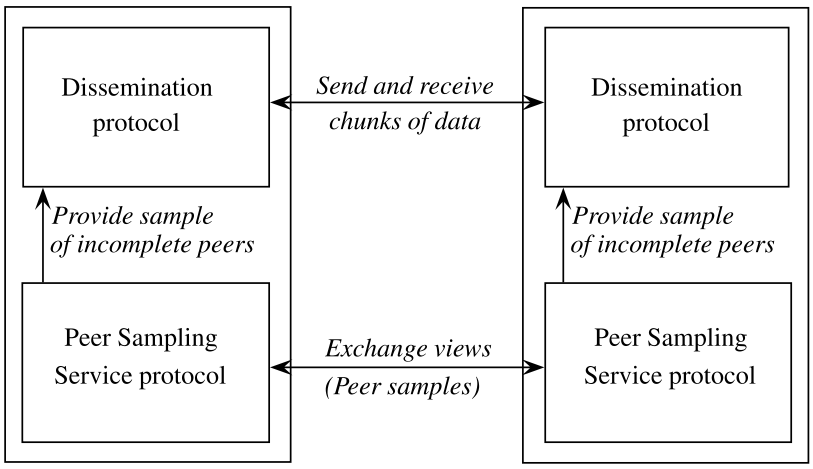

Dissemination protocol
Send and receive chunks of data
Dissemination protocol
Provide sample of incomplete peers
Provide sample of incomplete peers
Peer Sampling Service protocol
Exchange views (Peer samples)
Peer Sampling Service protocol

Figure 2.5: Organisation of protocols

that each peer p has a view v_p that corresponds to a uniform random sample of the incomplete peers in the system. The peer sampling service provides this sample of peers to the dissemination protocol as shown in Figure 2.5.

In the split-graph overlay, the incomplete peers are connected through a random graph while the complete peers are connected to some incomplete peers but not reciprocally. This is achieved by ensuring that the view of each peer, be it complete or incomplete, contains only references to incomplete peers. The structure of the overlay emerges naturally as complete peers have only outgoing edges and as only incomplete peers have incoming edges. Therefore, complete peers are forcibly kept outside of the clique.

More specifically, the peer sampling protocol runs two threads on each peer p_{myself} as in [52]. An active thread periodically chooses a random peer p_{random} and p_{myself} sends its view to p_{random}. A passive thread receives views and handles them. When p_{random} receives the view v, it replies by sending its own view to p_{myself}. Finally, both peers merge the view they have received with the view they had before and get a new view of size s. p_{random} merges v_{random} with v. First, any duplicates are removed. All references to complete peers are removed to ensure a *split-graph* structure. The view is then truncated to match the view size following the protocol of [52]. According to this protocol, the H oldest entries are removed to ensure *self-healing*. S entries that were just sent to the other peer are removed to ensure *low information losses*. Finally, if needed, some random entries are removed to get back to a view of size s.

Incomplete peers and complete peers differ in the view they send during the exchange. As complete peers have no incoming edges, they receive less up-to-date information. An incomplete peer applies the protocol as defined in [52]. However, complete peers apply a slightly different protocol when exchanging views. Instead of sending their own view, they send only information about themselves. The goal is to avoid overwriting an incomplete peer's up-to-date information with out-dated information coming from a complete peer; yet allowing complete peers to inform their partners about their completion level.

2.3.2 FoG dissemination protocol

We first present the basic dissemination protocol followed by the biased dissemination protocol towards the most advanced peers.

Dissemination and Foutain codes FoG aims at disseminating a whole file to all peers in the system. The dissemination is complete when all peers have completed. FoG relies on fountain codes (rateless erasure-correcting codes) to get rid of the unnecessary redundancy of gossip dissemination protocols as explained in the design rationale. To this end, a *TTL* is attached to every packet. As in a standard dissemination protocol, each peer contributes to the dissemination process. The peers that hold the full content can encode and send these encoded packets of data to some random peers. The other peers buffer at most the B most recent received packets and try to forward those packets to f *(fanout)* other randomly chosen peers from their views. Each peer forwards a given packet once; if a packet is received twice, it is simply dropped. In order to avoid the shrinking phase of gossip, the packets are forwarded over at most *TTL (time to live)* hops, enough to implement the exponential growth phase.

Due to the random nature of the algorithm, a peer may miss some packets. However, as the view of peers permanently evolves, the probability that a peer misses m consecutive packets decreases exponentially. We leverage this property in FoG by assuming that the peers not reached by a given dissemination, have a high probability to be reached by the following disseminations.

In FoG, the roles of peers differ depending on their being complete or not. As soon as a peer is complete, it can become a new source since it has the ability to generate new packets. Complete peers run a permanent task that produces new encoded packets using fountain codes and start disseminating. Note that incomplete peers also collaborate to the dissemination by forwarding packets but they cannot encode the content as they still miss some packets and the whole file is needed to encode using fountain codes.

Biasing the dissemination Increasing the diversity enhance the performance of the dissemination as explained in Chapter 1. Moreover, the more complete peers, the more new independent encoded packets are injected in the system. To benefit even further of complete peers as additional sources, FoG favors the most advanced peers with respect to the completion so that they can help the system as soon as possible.

This is achieved simply by adding a bias in the dissemination. Basically we aim at allowing the progression of peers to be different amongst peers. Typically, some peers should progress much faster so that they can help the rest of the system once they have completed. The bias is implemented as follows: when a complete peer chooses to serve some incomplete peer, it serves the one that has the highest completion first. Incomplete peers also have a bias in the dissemination to also ensure a non uniform progression within incomplete peers.

The dissemination is biased by adapting the probability to choose a given peer to gossip depending on its level of completion. The information about completion is included in the views exchanged during random peer sampling. All peers keep a reference to the last peer *last* to which they have sent data and keep a view v of other peers. To send data, a peer *self* chooses another peer *dest*. *self* builds a set of candidates $C = \{c || c \in v \cup \{last\} \wedge c.completion < self.completion\}$. Then, it chooses *dest* to be a peer $c \in C$ with a probability proportional to the completion level of c.

More formally, the choice is performed as explained bellow. Let j be the peer running the algorithm, l_j be the completion of peer j, v_j be the view of j and $last_j$ be the last peer to which j sent data. Let $c_{i,j}$ be the completion of peer i from the point of view of j. m_j is the highest completion $c_{i,j}$ with $i \in v_j \cup \{last_j\}$ such that $m_j < l_j$. For each peer, we compute $d_{i,j} = m_j - c_{i,j}$ if $m_j - c_{i,j} > 0$ or $d_{i,j} = 1$ else.

Complete peer j chooses to send data to peer i of its view with a probability $p_i = \frac{(1-(d_i))^e}{\sum_{k \in v_j \cup \{last_j\}} (1-(d_k))^e}$ that depends on the completeness of the peer so that the higher the completeness, the higher the probability. The parameter e expresses the level of aggressiveness of the bias.

Incomplete peers sort the peers in increasing order of the value $d_{i,j}$ and choose the first peer to which chosen packet of data has not been sent. Therefore, they send packets to peers that are the most complete but that are not more complete than they are. This avoids the creation of cycles in the dissemination process.

The peers and the active links of the overlay look like a multi-layered graph where peers move from bottom to top slowly. There is a little subset at the top that progresses quickly and a large pool of peers at the bottom. It should exhibit a long tail distribution for the completion of peers: the majority of peers have a low completion while a few peers have a high completion.

2.4 Evaluation

We performed extensive simulations of FoG and some reference protocols. The protocol has been simulated in an event-based simulation using PeerSim [53].

We simulated a 25,000 peer network, peers maintain a view of size 21. During the selection process, the healing parameter H was set to 8 while the swapping parameter S was set to 12. These values are chosen to quickly remove peers that have completed or have left the network as explained in [52]. We disseminate a file of $k = 1000$ packets. The upload bandwidth was constrained as follows: each peer can send a packet every unit of time. It also exchanges a gossip message to update the overlay every 10 units of time.

The source and encoder peers continuously encode, using a fountain encoder, and send packets of data. The incomplete peers buffer at most 48 packets of data. The fanout (f), the number of peers a received packet is forwarded to, is set to 6 while the number of hops a packet is forwarded (TTL) over is set to 6. Forwarder peers forward data until their buffer becomes empty. The level of aggressiveness e (a parameter of the bias described in Section 2.3.2) is set to 20 for the biased dissemination protocol.

We are mainly interested in the termination of the dissemination. A file can only be used once all the packets are available. Therefore, we plot the average value of completion (packets received over packets needed) for all peers. When the completion is 1.0, all peers have a full copy of the file and the dissemination is complete.

We compare our protocol, FoG, with and without the biased dissemination against these two following variants of traditional push-based dissemination protocols along termination time and overhead in term of useless messages. The two protocols are traditional protocols over which we have simply added fountain codes.

- **Peers stay in the overlay after completion.** Those peers receive more packets of data even if they do not need to receive them. Keeping the peers in the overlay helps to maintain connectivity. However, they can be on the paths between the source and peers that still need data. In this case, the dissemination slows down when almost all peers have completed, as complete peers do not forward data. It is shown as *Regular (nodes stay)* in the plots.

- **Peers leave the overlay gracefully.** The peers leave the overlay by excluding themselves when exchanging views. As they continue to maintain the overlay, the overlay should not be split. Thanks to the self-healing property, the peers are removed progressively. However, they do not contribute to the system once they have completed. In this case, the download does not slow down as the diameter of the network decreases and peers that do not have completed are close to the source. It is shown as *Regular (nodes leave)* in the plots.

Since these two traditional protocols are also enhanced by disseminating encoded information, comparing our protocol to these allows us to assess that adding foutain codes is not enough to get good performances: one need to modify the protocol to leverage the coding ability.

The graph on Figure 2.6 presents the completion results. FoG with or without overhead is faster than the reference protocols. As shown on Figure 2.6, biased FoG achieves 95% completion at time 2667 while unbiased FoG achieves same completion at time 3333. The two traditional protocols (Regular) are slower as they achieve the same completion at time 4000 if peers leave when they complete and at time 4400 if peers stay in the overlay once they have completed. Finally, biased FoG achieves full completion at time 2800, unbiased FoG achieves full completion at 3733 and the traditional protocol where peers leave the overlay once complete achieves full completion at time 4667. We observe a slight slowdown if peers depart gracefully while the progress becomes slow if peers do not leave as it can be seen at time 4000. Biased FoG completes almost twice earlier than a traditional dissemination protocol

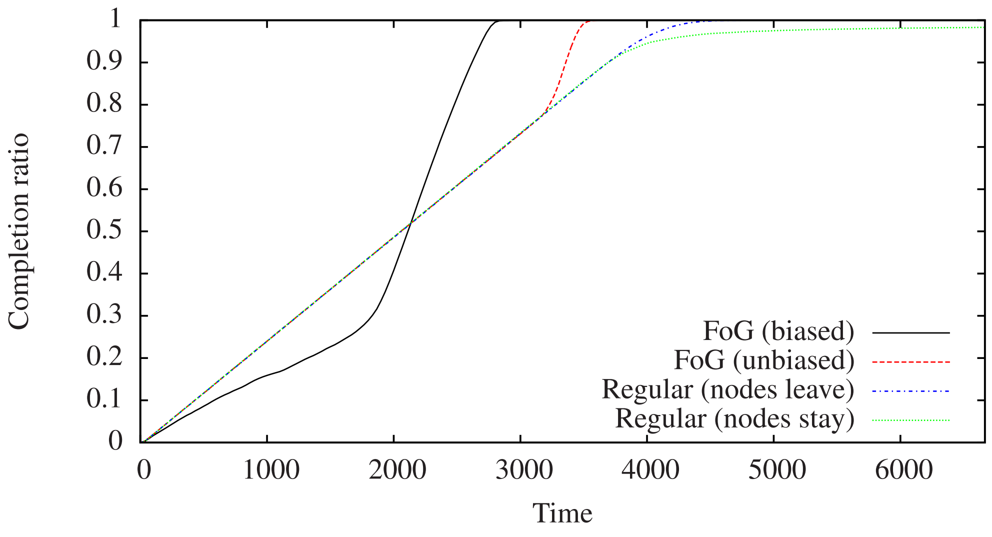


Figure 2.6: Average completion

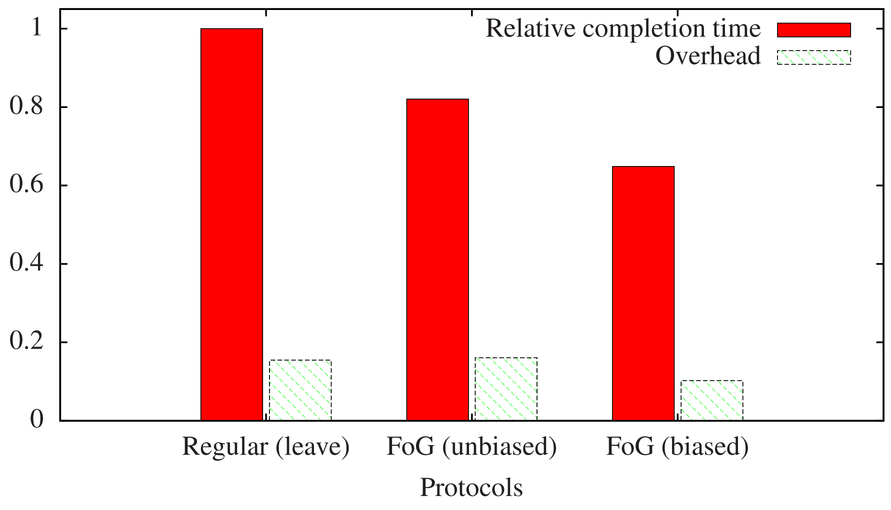
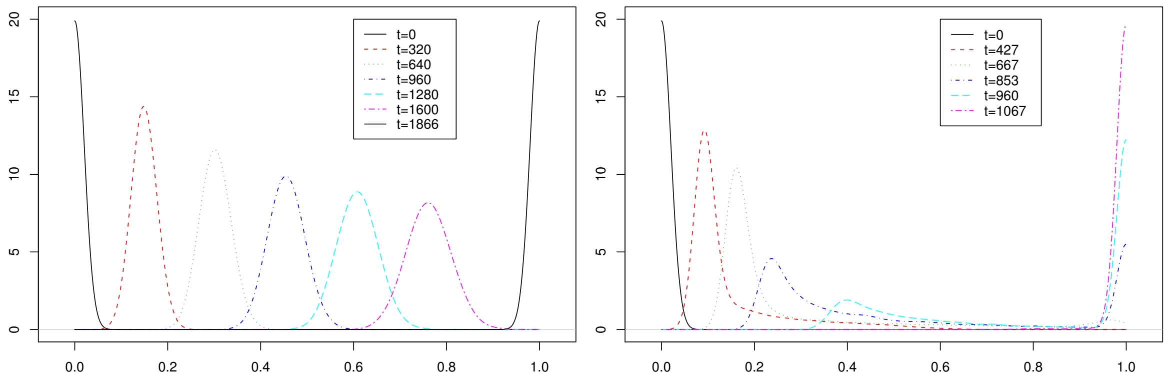


Figure 2.7: Completion time and overhead

(the one where peers leave once they have completed) and it even offers a slightly lower overhead as shown on Figure 2.7.

Based on these results, we can conclude that FoG completes faster than the two variants of traditional push protocols. We observe that an unbiased FoG is not much faster than competitors as the number of complete peers acting as encoders increases only in the end, at time, when some peers complete. This confirms the relevance of introducing a bias to the dissemination.



(a) Unbiased dissemination (b) Biased dissemination

Figure 2.8: Density of probability (vertical axis) that a peer has a completion c (horizontal axis) at time t. Unbiased FoG shows a uniform progression of peers while biased FoG exhibits a rather diverse progression of peers.

2.4.1 Impact of the biased dissemination

We observe on Figure 2.6 that the dissemination in unbiased FoG speeds up at time 3333 when the download process is almost over. This is mostly due to peers completing and acting as additional sources late in the process. Introducing a bias towards the most advanced peers enables to exploit earlier complete peers. At time 2000, biased FoG becomes faster than unbiased FoG. Consequently, the delay for all peers to get the whole content is reduced from 3733 to 2800.

We have studied the density of completion at different times for the two, unbiased and biased, protocols. The probability density function at time t, $d_t(x)$, is defined such that the probability that, at time t, a peer has a completion rate $c \in [a, b]$ is $p_t = \int_a^b d_t(x)dx$. The density functions at different times are plotted for each protocol on Figure 2.8. As an example, at time $t = 960$, in the unbiased version, all peers have a completion of approximately 50%. On the other side, at the same time, the biased version shows that some of the peers have a completion of 40% while half of them have already completed and a few of them are between 40% and 100%. The unbiased version of the protocol exhibits a density that is bell-shaped while the biased version has a density function that has a long-tail. When the dissemination is unbiased, all peers complete at the same speed. Therefore, no peer is able to encode data and help other peers. When a bias is introduced in the dissemination, a few peers behave differently from the majority and complete early resulting in the biased version outperforming the unbiased one.

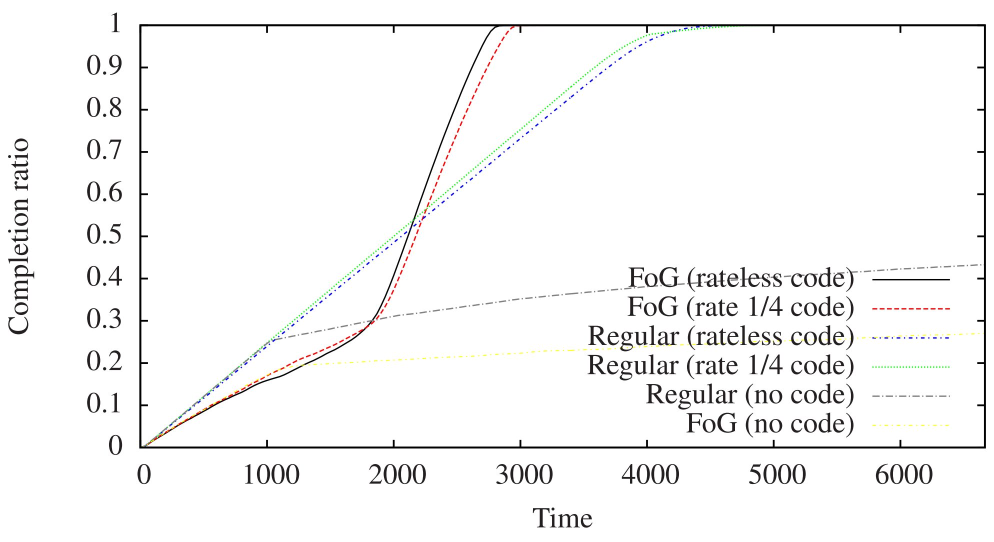


Figure 2.9: Encoding enhances the performance of dissemination algorithms.

2.4.2 Impact of coding

In the previous experiments, we disseminate data encoded with fountain codes. During the definition of our protocol, we asserted that using fountain codes (rateless erasure-correcting codes) was needed. We check this assertion by comparing our protocol with rateless codes (fountain codes), with fixed-rate ($\frac{1}{4}$) codes or without codes. We also study the impact of using codes with a traditional dissemination protocol where peers simply leave the overlay when they complete.

When disseminating with fixed rate codes, sources send encoded packets cyclically in order. When disseminating non-coded data, the native packets are sent cyclically by sources. If a new source is created, the starting point in the set of packets is chosen randomly.

The results are shown on Figure 2.9. The graph shows that encoding improves the dissemination in all settings. Clearly, the protocols that use some form of coding (either rateless codes or fixed-rate codes) outperform the protocols that do not use codes. Completion becomes slow at time 1000 when we use non-coded data: this time matches the time at which the source will loop over and start disseminating again packets that have already been disseminated.

The better performance of the protocols using codes can easily be explained. Peers complete faster as cooperation is easier and as redundancy in the packets they receive is lower. Therefore, our assumption that we need coding holds. Moreover, fountain and fixed rate codes both result in curves having similar shapes. This shows that even though we provide an infinite amount of redundancy through the use of fountain codes, a finite amount ($4k$) is enough for our protocol to work: the additional redundancy only brings very little enhancement. However, low complexity fixed-rate codes use the same decoder as fountain codes and have the same decoding complexity. Therefore, there is no reason to drop the more elegant design of using fountain codes for using fixed-rate erasure correcting codes.

2.5 Related Work

Dissemination protocols can either be streaming protocols where peers exchange data within a window of interesting data or be file swarming protocols where all peers cooperate to get a full copy of a file. The streaming case is widely used in live streaming of video or audio streams. In such protocols, losses of some packets of data, even if not desirable, are possible. File swarming protocols are more useful to get a full copy of a file where no losses are allowed. FoG is used to download a full copy of a file, similarly to file swarming protocols, through push protocols. Encoding content adds the constraint that the content need to be entirely downloaded and decoded before it can be used. However, the content we disseminate need to be entirely downloaded before being used. Therefore, encoding data does not constrain more the use that can be done of our protocol.

There has been many works in peer-to-peer based dissemination, we focus on systems using coding to enhance the performance. Various coding techniques have been used for enhancing dissemination process. *Avalanche* [43, 44] uses network coding [2] in order to reduce redundancy in exchanged packets. Simulations of *Avalanche* result in the conclusion that it can make throughput 2 or 3 times better than transmitting unencoded packets over *Bittorrent*-like networks [26]. However, as random linear network codes decoding involves complex computation ($O(k^3)$ Gauss reduction), some people [58] compared network coding in *Avalanche* to systems that do not use coding and had less optimistic conclusion. They claim that network coding is too complex and does not enhance the performance enough to justify its use [68, 101]. Locher and al. [65] propose to add source coding to *Bittorrent* instead of relying on costly network coding. As source coding increases diversity, it enhances the performance of Bittorrent and helps the tit for tat mechanism. Their work is based on random linear codes that have high decoding complexity, but their work can directly be extended to any other code with lower complexity such as LT codes [66]. In this chapter, we decided to avoid network coding and use lower complexity ($O(k \cdot \ln k)$) source codes such as LT.

A protocol [11, 70, 79] is said to push data if it sends data to a peer without being requested to do so. A protocol pulls data if it sends data to a peer after being requested. Push protocols have lower control overhead as they do not involve a query prior to pushing data. Moreover, push protocols have lower delay, because, as soon as a peer has some data, it can immediately push it to other peers. Pull protocols have higher delay as peer regularly poll for their neighbors to get new data and new data is only sent once the other peer asked it. Pull protocols have lower overhead as they allow the receiver to choose the data it gets, it avoids getting the same packet twice. Our choice was to implement a push dissemination protocol in order to reduce the amount of control messages needed. The article by Karp and al. [55] studies push protocol and their behavior. They study the performance of push-only and pull-only scheme before proposing a push&pull scheme that tries to take the best of two.

2.6 Summary

In this chapter, we presented FoG, a protocol that aims at keeping the strength of gossip protocols, exponential-growth phases, while removing the need for their weakness, shrinking phases. To this end, we ensure completion through additional disseminations of content encoded using fountain codes instead of relying on the shrinking phases of gossip based dissemination that produce a high level of redundancy.

In this context, we build an overlay (a split-graph) that matches dissemination progress. The overlay is structured so that only incomplete peers have incoming edges. Therefore, peers do not receive more data once they have completed. Moreover, peers that complete can act as new sources. In order to leverage further completed peers as additional sources, we bias the dissemination towards the most advanced peers. This enables peers to complete earlier and exploit further fountain codes. Experiments show that FoG outperforms by 50% for the overhead and by 30% for the termination time traditional push protocols.

Clearly, the fact that incomplete peers cannot encode but only forward is a central problem in the design of this protocol. Hence, network coding, by allowing incomplete peers to recode, has the potential to offer similar or better performance with a simpler protocol. Yet, the complexity of the decoding of random linear network codes ($\mathcal{O}\left(k^3\right)$ for the control and $\mathcal{O}\left(mk^2\right)$ for data) limits their field of applications. Therefore, in the next chapter, we build new network codes with a low complexity ($\mathcal{O}\left(k \ln k\right)$) decoding to allow the use of network coding in constrained environments.

Chapter 3

LT Network Codes for Dissemination

In this chapter, we propose new low complexity network codes. Network coding has allowed very efficient dissemination protocols fully alleviating failures and randomness in the dissemination. Yet, current implementations of network coding are based on random linear codes and suffer the high decoding complexity of the Gauss reduction decoding. Our LT network codes trade the communication efficiency for the decoding cost thus increasing the range of possible applications for network coding.

This work first appeared as a poster [18] at the Student Workshop of the 5th ACM International Conference on emerging Networking EXperiments and Technologies (ACM Conext Student Workshop) in 2009. It has since been published [19] in the proceedings of the 5th IEEE International Conference on Distributed Computing Systems (ICDCS) in 2010.

3.1 Motivation

In the previous chapter, we limited ourselves to fountain codes because of their appealing low complexity decoding. Yet, network coding, initially proposed by Ahlswede *et al.* [2], has proven to be a powerful paradigm significantly improving the throughput in epidemic dissemination. This has been successfully applied both in wired systems (e.g., p2p file sharing with Avalanche [43, 44]), and wireless systems (e.g., sensor networks with [4, 33] and mesh networks with [56]). While unencoded epidemic approaches provide robustness in the dissemination by implementing a lot of redundancy (when forwarding packets), network coding techniques introduce a smarter redundancy scheme, providing at a lower cost an optimal solution with respect to dissemination. Such schemes rely on a recoding procedure achieved at each step of the dissemination chain, where devices involved in the dissemination recode content in the form of linear combination of received packets. Despite the success of network coding, some works have shown that one of the limitations in current forms of network coding, namely random linear network codes (RLNC), is that they require a high complexity decoding process. Some optimizations [68, 99, 101] have been proposed. Yet, none removes entirely the complexity of the decoding method, namely $\mathcal{O}(m \cdot k^2)$ of Gauss reduction in RLNC (typical network codes) where the content disseminated is split in k native packets of size m.

In this chapter, we propose LTNC, a novel approach to build network codes from low complexity rateless erasure codes, namely LT codes [66], alleviating high complexity decoding procedure at the devices. This is of the utmost importance when devices have limited computational power typically in sensor networks composed of low capability devices. LT codes enable a low-complexity decoding thanks to the belief propagation decoding scheme, which recovers native packets in $\mathcal{O}(m \cdot k \log k)$. Yet, such schemes are extremely sensitive to specific statistical properties of encoded packets. Such properties are challenging to implement in fully decentralized settings where devices have only *some encoded* packets available when recoding. To the best of our knowledge, LTNC is the first network coding technique based on LT codes, thus enabling the use of belief propagation for decoding. There has been other attempts to distribute encoding of LT Codes or propose distributed encoding scheme using belief propagation for decoding method [3, 4, 33, 54]. However, other attempts build encoded packets only by combining native ones, limiting the range of applications that could benefit from such schemes. Instead, our scheme enables to recode fresh encoded packets at each device, from encoded packets while preserving the statistical properties of LT codes. With LTNC, the freshly recoded packets preserve the structure and properties of LT codes to maintain decodability using the low complexity decoding algorithm. Since LTNC are linear network codes, traditional optimizations (e.g., generations [44, 71]) and security schemes (e.g., homomorphic hashes and signatures [23, 48, 51, 106]) can be directly applied. This enables the use of LTNC in practical content dissemination systems such as Avalanche [43].

The rest of this chapter is organized as follows. Section 3.2 describe the context in which we propose LT network codes. Section 3.3 provides a high level overview of LTNC and describes in detail the algorithms involved in the recoding method. An evaluation of LTNC is presented in Section 3.4. Section 3.5 reviews the related work.

3.2 Context

Coding techniques have been used in push-based dissemination applications where content, divided into k *native packets* $\{x_i\}_{i=1}^{k}$ of size m, is broadcast from one or multiple sources to a set of devices connected by a network. With erasure correcting codes, the k native packets are combined *at the source* into $n > k$ *encoded packets*, and can be *recovered at the devices* from any set of $(1+\varepsilon)\cdot k$ encoded packets ($\varepsilon \geq 0$). Intermediary devices of the network taking part in the dissemination simply forward encoded packets to their neighbors.

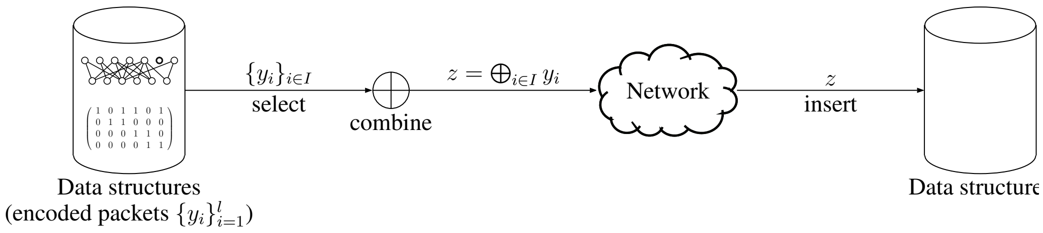


Figure 3.1: Global picture of (Linear) Network Coding.

Yet efficient, erasure correcting codes are suboptimal since only the sources can increase the packets diversity by generating distinct encoded packets (i.e., intermediary devices only forward the packets they receive). In network coding [2], intermediary devices are able to generate *fresh* encoded packets from the encoded packets they received, namely *recoding*, as illustrated in Figure 3.1. This results in a higher diversity of the encoded packets circulating in the network leading to increased performance as compared to erasure correcting codes: network coding allows the dissemination throughput to reach the network capacity.

As explained in Chapter 1, rateless codes are well suited for network coding as linearly combining encoded packets results in fresh encoded packets. Random linear codes for instance can easily be turned into random linear network codes (RLNC) by recoding encoded packets received into fresh ones using random linear combinations. In other words, the recoding operation is the same as the coding operation except that it operates on encoded packets instead of native packets. Yet, random linear network codes suffer a high decoding complexity ($\mathcal{O}(mk^2)$ for data and $\mathcal{O}(k^3)$ for control, where k is the code length and m the size of each packet).

Similarly, LT codes, other rateless codes which have been extensively described in Chapter 1, could be well suited for network coding and exhibit low complexity decoding ($\mathcal{O}(mk\log k)$ for data and $\mathcal{O}(k\log k)$ for control). However, building network codes from LT codes requires intermediary devices to be able to generate, with only partial information, encoded packets which degrees follow a specific distribution while keeping the variance of degrees of native packets low. Effectively, while this can easily be achieved at the source where all native packets are available, this is very challenging when a device has only some encoded packets available. In the following, we describe LTNC, network codes based on Luby Transform code.

3.3 LT Network Codes

In this section we present LTNC, low complexity network codes based on Luby Transform codes for push-based content dissemination applications where devices

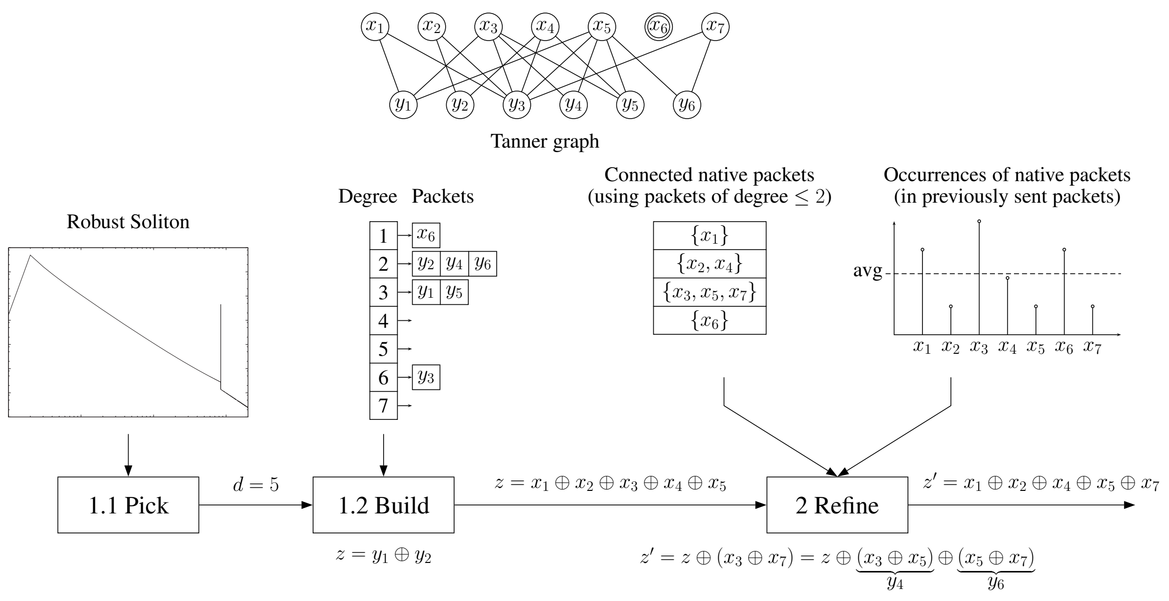

Tanner graph
Robust Soliton
Degree Packets
Connected native packets (using packets of degree ≤ 2)
$\{x_1\}$
$\{x_2, x_4\}$
$\{x_3, x_5, x_7\}$
$\{x_6\}$
Occurrences of native packets (in previously sent packets)
avg
1.1 Pick
$d = 5$
1.2 Build
$z = x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5$
2 Refine
$z' = x_1 \oplus x_2 \oplus x_4 \oplus x_5 \oplus x_7$
$z = y_1 \oplus y_2$
$z' = z \oplus (x_3 \oplus x_7) = z \oplus \underbrace{(x_3 \oplus x_5)}_{y_4} \oplus \underbrace{(x_5 \oplus x_7)}_{y_6}$

Figure 3.2: Overview of LTNC ($\mathbf{k} = \mathbf{7}$).

periodically send possibly encoded packets to their neighbors. As mentioned in Chapter 1 (Section 1.1.2, Page 1.1.2), low complexity decoding can be obtained using the belief propagation algorithm which efficiency highly relies on statistical properties of encoded packets available at the device. More specifically, the degree distributions of native and encoded packets must respectively match a Robust Soliton and a Dirac. Therefore, it is of the utmost importance to ensure, when recoding encoded packet into fresh ones, that the structure of LT codes is preserved. This problem is especially challenging in a network coding scenario where intermediary devices operate with a limited number of encoded packets available.

In a nutshell, our solution works as follows: when a device needs to generate a fresh encoded packet (i.e., recode), it *(i)* builds a packet of degree d, where d is drawn from a Robust Soliton distribution, using the encoded packets available; *(ii)* refines the obtained packet so that the variance of the distribution of degrees of native packets is reduced. The first step involves NP-Complete sub-problems and thus cannot be solved at a low computational cost. The performance of each step of the recoding method, and thus the overall performance of LTNC relies on efficient heuristics and complementary data structures allowing low complexity recoding with a good approximation of the structure of LT codes.

In the following, we first give the rationale behind LTNC and illustrate its functioning and the data structures used on a concrete example. Table 3.1 (page 46) summarizes the different data structures used by LTNC with their purpose. We then dive into the algorithmic details of the two aforementioned steps. Finally, we present various optimizations for LTNC including application-specific optimizations that rely on some features available on the application framework (e.g., feedback channel for Internet applications).

3.3.1 Overview of LTNC

Consider the example depicted in Figure 3.2, where a device p recodes a fresh encoded packet from previously received ones. The initial content is split into $k = 7$ native packets and device p stores 6 encoded packets $\{y_i\}_{i=1}^{6}$ and the native packet x_6.

First, p picks a random degree d drawn from the Robust Soliton distribution for the packet to be recoded (Step 1.1). The degree distribution is an input of the system, fixed in advance to its optimal value (i.e., Robust Soliton [66]) and known at each device. In the example of Figure 3.2, the picked value is $d = 5$.

The device p tries to build, by linearly combining previously received encoded packets and decoded native packets, a fresh encoded packet of degree $d = 5$ (step 1.2). To this end, p maintains an index that maps degrees to a list of encoded packets of each particular degree allowing fast lookup of encoded packets of a given degree. In the example of Figure 3.2, p picks two encoded packets y_1 and y_2 of respective degrees 2 and 3 and builds a fresh encoded packet $z = y_1 \oplus y_2$. In terms of native packets, $z = x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5$ and its degree is therefore $d(z) = 5$, that is the degree picked in the previous step. Steps 1.1 and 1.2 ensures that the degrees of fresh encoded packets sent by a device follow a Robust Soliton distribution as specified by LT codes.

Finally, p refines the encoded packet z built from the previous steps in order to decrease the variance of the distribution of degrees of natives packets in previously sent encoded packets. This step substitutes some native packets in the fresh encoded packet built in the previous steps (i.e., z here) with other ones that appeared less frequently in previously encoded packets, without jeopardizing the degree of z. This results in a fresh encoded packet z'. In LTNC, this is achieved with the help of encoded packets of degree 1 and 2. Effectively, if a native packet x appears in an encoded packet z and a native packet x' does not appear in z, then, adding the packet of degree 2 $x \oplus x'$ to z boils down to substituting x' to x in z (since $x \oplus x = 0$ and $z \oplus 0 = z$). Note that such an operation does not change the degree of the encoded packet. In the example of Figure 3.2, the native packet x_3 appears in more previously sent encoded packets than x_7 (this information is available from a dedicated data structure that gathers statistical information on previously sent encoded packets: the number of occurrences of each native packet) and appears in z while x_7 does not appear in z. x_3 is therefore replaced with x_7 by adding $x_3 \oplus x_7$ to z. This steps relies on the ability of the device to build $x_3 \oplus x_7$. To this end, each device maintains a specific data structure to determine which encoded packets of degree two it can build from the available ones. More specifically, a device maintains a partition of native packets where two native packets x and x' are in the same set if $x \oplus x'$ can be generated using only encoded packets of degree 2. In the example of Figure 3.2, x_3 and x_7 are in the same set since $x_3 \oplus x_5$ and $x_5 \oplus x_7$ are available (y_4 and y_6 in the Tanner graph).

It can be seen from this concrete example that the recoding method of LTNC involves several difficult algorithmic problems. For instance, how to determine if, for a given degree d, the device is able to build an encoded packet of degree d and how to select encoded packets to combine to reach that degree. The next section presents algorithmic solutions to the aforementioned problems and describes in detail the data structures and the associated maintenance techniques used by LTNC.

Table 3.1: Complementary data structures used by LTNC.

Data structure	**Purpose**
Encoded packets by degrees	find a set of encoded packets to build a fresh one of a given degree
Connected native packets	determine packets of degree 2 that can be built using only degree 1 and 2 encoded packets
Occurrences of native packets	determine substitutions of native packets that decrease the variance of degrees

3.3.2 Recoding LT encoded packets

In the following, we detail the different steps involved in the recoding method of LTNC.

Picking a degree

To match a Robust Soliton distribution of degrees for encoded packets sent, a device building a fresh encoded packet first picks a *target degree* d at random drawn from this specific distribution. However, the target degree may not be *reachable*, i.e., no packet of degree d can be built from the set of encoded and decoded packets available at the device. Assuming that a fresh encoded packet of degree d is built only from decoded native packets and encoded packets of degree lower than d (i.e., the building method does not leverage collisions, which is the case of LTNC as explained in the next paragraph), LTNC uses two heuristics to detect if a degree d is unreachable.

First, a degree d is unreachable if $\sum_{i=1}^{d} i \cdot n(i) < d$, where $n(i)$ is the number of encoded packets of degree i available at the device. For instance, the maximum reachable degree of a fresh encoded packet built from the set of encoded packet $\{x_1 \oplus x_2 \oplus x_3, x_1 \oplus x_3, x_2 \oplus x_5\}$ is $2 \times 2 + 3 = 7$.

Second, the maximum reachable degree is upper-bounded by the number of native packets that either are decoded or appear in at least one encoded packet of degree less than d. For instance a packet of degree 5 cannot be generated from the set of encoded packets $\{x_1 \oplus x_2 \oplus x_3, x_1 \oplus x_3, x_2 \oplus x_5\}$ since any linear combination of these encoded packets involve only 4 different native packets (i.e., x_1, x_2, x_3 and x_4).

If a picked degree is classified as unreachable (i.e., larger than one of the two upper bounds), a new one is picked and so on and so forth until the picked degree is accepted (i.e., not classified as unreachable). Note that this allows to discard immediately *some* unreachable degrees, however this does not guarantee that the picked degree is effectively reachable. For instance, none of the two aforementioned bounds would discard degree 3 for the set $\{x_1 \oplus x_2, x_3 \oplus x_4\}$ while this degree cannot be effectively reached. Similarly, a picked degree of 4 is not discarded for the set $\{x_1 \oplus x_2, x_2 \oplus x_3, x_4\}$. In our simulations, the first picked degree is accepted in 99.9% of the cases and the average number of retries (when the first degree is discarded) is 1.02.

Coping with a picked degree

This steps takes as input a picked degree d, a set $\mathcal{X}$ of decoded native packets and a set $\mathcal{Y}$ of encoded packets and builds a fresh encoded packet of degree d. Formally, the problem writes: given d, $\mathcal{X} = \{x_i\}_{i \in I}$ and $\mathcal{Y} = \{y_i\}_{i=1}^{k'}$, find $\mathcal{W} \subset \mathcal{X} \cup \mathcal{Y}$ so that $d(z) = d$, where $z = \bigoplus_{w \in \mathcal{W}} w$. The problem of finding a set of packets so that the sum of the degrees is exactly d is known as the *subset sum* problem which is NP-complete. The fact that the degree of the sum of two encoded packets may not be the sum of their respective degrees (e.g., the degree of $(x_1 \oplus x_2) \oplus (x_2 \oplus x_3)$ is 2 and not 4, this is called a *collision*) makes this problem even more difficult.

LTNC finds a sub-optimal solution in a greedy fashion, preventing collisions that decrease the degree of the fresh encoded packet being built. It examines the encoded packets ordered by decreasing degrees starting from d. A packet is added to the fresh encoded packet being built if the degree of the resulting encoded packet (i.e. the sum) *(i)* is increased and *(ii)* remains lower or equal to d . The algorithm uses a specific data structure, namely an index $\mathcal{S}$ of packets grouped by degrees (i.e., $\mathcal{S}[1] = \mathcal{X}$ and $\mathcal{S}[i]$ is the set of encoded packets of degree i in $\mathcal{Y}$ for $i > 1$). The degree of the resulting fresh encoded packet is lower than d. A pseudo-code version is given in Algorithm 1. In our simulations, the building step reaches the target degree 95% of the time and the average relative deviation to the target degree (i.e., target degree - obtained degree / target degree) is 0.2%.

In the example depicted in Figure 3.2, z has been obtained using Algorithm 1: the building algorithm starts with encoded packets of degree 3 since there is no packet of degree 4 or 5. It picks y_1 at random and adds it to z. y_5 is then examined and discarded as it would decrease the degree of z (i.e., $y_1 \oplus y_5$ is of degree 2). The algorithm then moves to encoded packet of degree 2 and picks y_2 at random. The encoded packet $z \oplus y_2 = y_1 \oplus y_2$ is of degree 5, y_2 is thus added to z and no more packets are further added. Effectively, as soon as $d(z) = d$, the condition $d(z) < d(z \oplus y) \leq d$ cannot be satisfied anymore.

Refining an encoded packet

This step *refines* the fresh encoded packet z obtained from the previous step by replacing some native packets with less frequent ones in order to decrease the variance of the degree distribution of native packets. This information is available from a

Algorithm 1 Building an encoded packet of a given degree

```
Input: $d$                                          ▷ Target degree
Output: $z$              ▷ Fresh encoded packet with a maximum degree of $d$

 1: $z \leftarrow 0$                             ▷ Fresh encoded packet being built
 2: $i \leftarrow d$
 3: $\mathcal{S}' \leftarrow \mathcal{S}[i]$
 4: while $d(z) < d$ and $i > 0$ do
 5:     if $\mathcal{S}' = \varnothing$ then          ▷ If there is no more packets of degree $i$
 6:         $i \leftarrow i - 1$                         ▷ Move to $\mathcal{S}[i-1]$
 7:         $\mathcal{S}' \leftarrow \mathcal{S}[i]$
 8:     else
 9:         $y \leftarrow$ pickAtRandom($\mathcal{S}'$)
10:         $\mathcal{S}' \leftarrow \mathcal{S}' \setminus \{y\}$
11:         if $d(z) < d(z \oplus y) \leq d$ then
12:             $z \leftarrow z \oplus y$
13:         end if
14:     end if
15: end while
```

specific data structure that stores, for each native packet, the number of occurrences in the previously sent encoded packets. The data structure is updated every time a fresh encoded packet is sent.

As explained above, a native packet x can be replaced with x' (denoted $x \sim x'$) if $x \oplus x'$ can be generated. In LTNC, refinement is achieved using only decoded native packets and encoded packets of degree 2: it is considered that $x \oplus x'$ can be generated if *(i)* x and x' are decoded or *(ii)* $x \oplus x'$ is available or *(iii)* there exists a third native packet x'' such that $x \sim x''$ and $x'' \sim x'$. By construction, the relation $\sim$ is an equivalence. Its equivalence classes correspond to the connected components in the graph where the vertices are the k native packets and there exists an edge between x and x' if $x \oplus x'$ is in the Tanner graph.

This information is available from a dedicated data structure *cc* that maps a native packet to an integer so that $x \sim x' \Leftrightarrow \mathrm{cc}(x) = \mathrm{cc}(x')$. The value $\mathrm{cc}(x)$ can be thought of as the index of the *leader* of the connected component. Initially, $\mathrm{cc}(x_i)$ is set to i for all $1 \leq i \leq k$. The data structure is then dynamically updated as follows: when a native packet x is decoded $\mathrm{cc}(x)$ is set to 0, when an encoded packet of degree 2, say $x \oplus x'$, is received (or obtained by belief propagation during the process of decoding) $\mathrm{cc}(x'')$ is set to $\mathrm{cc}(x)$ for all x'' so that $\mathrm{cc}(x'') = \mathrm{cc}(x')$. This enables LTNC to determine in $\mathcal{O}(1)$ if an encoded packet of degree 2 can be generated. Figure 3.3 returns to the example depicted in Figure 3.2 and gives a leader-based representation of the connected components of native packets (using encoded packets of degree 1 and 2). When the encoded packet $x_3 \oplus x_4$ is received, both $\mathrm{cc}(x_4)$ and $\mathrm{cc}(x_2)$ are updated to $\mathrm{cc}(3) = 5$. The algorithm for updating the leader-based representation is given as Algorithm 2. The algorithm also shows that since x and x' are equivalent, we can always choose to update the smallest number of values in *cc* by merging the smallest component into the largest one.

Algorithm 2 Updating the leader-based representation

```
Input: $x \oplus x'$                                ▷ Encoded packet of degree 2
Input: cc                                           ▷ Leader-based representation
Output: cc                                          ▷ Updated leader-based representation

1: if $\#\{x''|\mathrm{cc}(x'') = \mathrm{cc}(x')\} < \#\{x''|\mathrm{cc}(x'') = \mathrm{cc}(x)\}$ then   ▷ Merge the smallest into the largest
2:     for each $x'' \in \{x''|\mathrm{cc}(x'') = \mathrm{cc}(x')\}$ do
3:         $\mathrm{cc}(x'') \leftarrow \mathrm{cc}(x)$
4:     end for
5: else
6:     for each $x'' \in \{x''|\mathrm{cc}(x'') = \mathrm{cc}(x)\}$ do
7:         $\mathrm{cc}(x'') \leftarrow \mathrm{cc}(x')$
8:     end for
9: end if
```

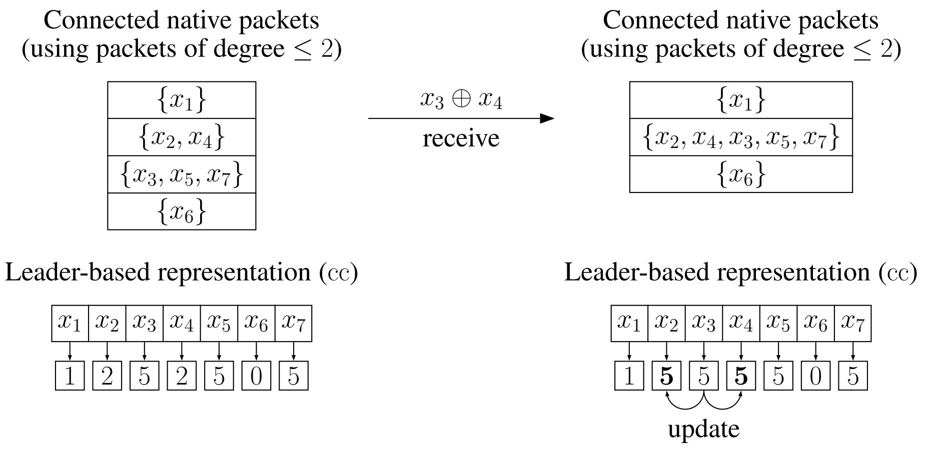

Connected native packets (using packets of degree ≤ 2)
$\{x_1\}$
$\{x_2, x_4\}$
$\{x_3, x_5, x_7\}$
$\{x_6\}$
$x_3 \oplus x_4$
receive
Connected native packets (using packets of degree ≤ 2)
$\{x_1\}$
$\{x_2, x_4, x_3, x_5, x_7\}$
$\{x_6\}$
Leader-based representation (cc)
x_1 x_2 x_3 x_4 x_5 x_6 x_7
1 2 5 2 5 0 5
Leader-based representation (cc)
x_1 x_2 x_3 x_4 x_5 x_6 x_7
1 5 5 5 5 0 5
update

Figure 3.3: Leader-based representation of the connected components of native packets.

Using these two data structures, LTNC replaces iteratively each native packet x in z with the less frequent native packet x' that verifies the following three properties: (1) $x \sim x'$; (2) x' is less frequent than x; (3) z does not contain x'. This results in a refined fresh encoded packet z' that gives the minimum (for a given fresh encoded packet z to refine and a set of encoded packets of degree 1 or 2) variance of occurrences of native packets. Algorithm 3 gives a pseudo-code version of the refinement step.

In the example depicted in Figure 3.2, z' has been obtained using Algorithm 3: x_1 cannot be replaced with any other native packet (i.e., it is the only native packet in its connected component); x_2 is less frequent than any other native packet; x_3 can be replaced with x_7 (since $x_3 \sim x_5$ and $x_5 \sim x_7$), x_7 is less frequent than x_3 and x_7 is the least frequent such native packet: x_3 is therefore replaced with x_7; x_4 and x_5 are not replaced since they are less frequent than x_3 (i.e., which is the only native packet not contained in z' at this stage).

Algorithm 3 Refining an encoded packet

```
Input: $z$                                                        ▷ Fresh encoded packet
Output: $z'$                                              ▷ Refined fresh encoded packet

1: $z' \leftarrow z$                                       ▷ Fresh encoded packet being built
2: for each $x \in z$ do
3:     $\mathcal{A} \leftarrow \{x''$ s.t. $x \sim x'$ and $x'' \notin z'$ and $x''$ is less frequent than $x$ $\}$
4:     if $\mathcal{A} \neq \varnothing$ then
5:         $x' \leftarrow \arg\min_{x'' \in \mathcal{A}} \text{frequency}(x'')$
6:         $z' \leftarrow z' \oplus (x \oplus x')$
7:     end if
8: end for
```

The efficiency of the refinement algorithm highly relies on the fact that, due to the Robust Soliton distribution used in LT codes, more than half of the encoded packets are of degree 1 or 2, thus resulting in a high refining power. In our simulation, the relative standard deviation (standard deviation /average) of the number of occurrences of native packets in encoded packets sent is 0.1%.

Moreover, the maintenance of the added data structure does not increase the overall complexity of the decoding. Indeed, the complexity of updating this data structure is the same as the decoding (i.e., $\mathcal{O}(k \ln k)$). Since we always merge the smallest component into the largest one, the worst case happens when, at each merge, the components have the same size. In this case, the tree representing the successive merges is balanced and its depth is $\log_2 k$. Half of the values ($k/2$) will be updated $\log_2 n$ times. Hence, the update complexity for k encoded packets is $\mathcal{O}(k \log_2 k)$.

3.3.3 Optimizations

With random linear codes, including LT codes, encoded packets have a low – but non-zero – probability of being non-innovative (i.e., packets that can be generated from other encoded packets already available at the device). In Random Linear Network Coding (RLNC), a partial Gaussian reduction step detecting non-innovative packets is performed when a fresh encoded packet received is inserted in the data structures (i.e., the code and data matrices). However, in LTNC, belief propagation does not provide immediate detection of non-innovative packets. This results in some redundancy in the data structures.

As both memory and CPU usage are related to the number of encoded packets stored at a device, redundant packets should be avoided. To this end LTNC includes a low-complexity redundancy mechanism to detect and remove non-innovative encoded packets upon reception or during the process of decoding. The detection mechanism can be adapted for systems where a feedback channel is available allowing the sender and the receiver to communicate and agree on the encoded packet to send. This results in an increased convergence speed and a decreased bandwidth usage since: *(i)* the encoded packets sent are more likely to be innovative for the receiver, and *(ii)* if the sender detects that it cannot generate an innovative packet for the receiver, no packet is sent.

Detecting redundancy

A packet is said redundant or non-innovative for a device if it can be generated from the encoded packets available at the device. Considering an encoded packet $z = \bigoplus_{i \in I} x_i$, z can be built without collision if there exist two sets J and J' so that: $J \cup J' = I$ and both $y = \bigoplus_{j \in J} x_j$ and $y' = \bigoplus_{j \in J'} x_j$ can be generated (or are available at the device). Using this recursive formalization under the non-collision assumption, a large proportion of non-innovative packets can be detected. However, the complexity increases exponentially with the degree of the encoded packet being checked, even when using dynamic programming. Interestingly enough, when using LT codes, most encoded packets circulating in the system have a low degree. Applying this redundancy detection mechanism to low degree encoded packets allows discarding a large proportion of non-innovative encoded packets at low cost. Moreover, high-degree packets are less likely to be non-innovative. Therefore, detecting non-innovative packets of high-degree is useless in most of the cases.

To reduce the complexity of the redundancy detection mechanism, in LTNC, it is applied only to encoded packets of degree less than or equal to 3 (that is almost two thirds of the encoded packets with Robust Soliton). Furthermore, we improve on the algorithm described in the previous paragraph by taking into account collisions for encoded packets of degree 2. Detection makes use of the connected components of native packets. An encoded packet of degree 1 is redundant if it is available at the device. An encoded packet $y = x \oplus x'$ of degree 2 is redundant if x and x' are in the same connected component (i.e., $\mathrm{cc}(x) = \mathrm{cc}(x')$). Algorithm 4 gives a pseudo-code version of the redundancy detection mechanism used in LTNC. Note that the redundancy detection mechanism can be applied on encoded packets stored which degree drops to 3 during the process of decoding. For instance, in the example depicted in Figure 3.2 the device stores an encoded packet $y_5 = x_3 \oplus x_4 \oplus x_5$. If the device somehow decodes x_4, x_4 will be propagated to y_5 which incurs a `xor` operation. This operation is useless, as it would give $x_3 \oplus x_5$ which can be generated from the other encoded packets. The redundancy mechanism of LTNC prevents such useless operations.

Determining if a packet of degree 1 or 2 is redundant can be done in $\mathcal{O}(1)$ operations. Assuming the use of a complementary structure allowing $\mathcal{O}(\log k)$ lookups of encoded packets of degree 3 (e.g., a binary search tree), the redundancy detection mechanism of LTNC is $\mathcal{O}(\log k)$. In our simulations, this mechanism decreases by 31% the number of redundant encoded packets inserted in the data structure upon reception.

Algorithm 4 Detecting redundant packets: isRedundant()

```
Input: $y$                                              ▷ Encoded packet of degree $\leq 3$
Output: $b$                  ▷ Returns true if $y$ can be generated and false otherwise

 1: if $d(y) = 1$ then                                  ▷ $y$ is a native packet $y = x$
 2:     $b \leftarrow$ isDecoded($x$)
 3: else if $d(y) = 2$ then                                         ▷ $y = x \oplus x'$
 4:     $b \leftarrow$ (cc($x$) = cc($x'$))
 5: else if $d(y) = 3$ then                                 ▷ $y = x \oplus x' \oplus x''$
 6:     $b \leftarrow$ (isRedundant($x$) $\wedge$ isRedundant($x' \oplus x''$)$\vee$
 7:            (isRedundant($x'$) $\wedge$ isRedundant($x \oplus x''$)$\vee$
 8:            (isRedundant($x''$) $\wedge$ isRedundant($x \oplus x'$)$\vee$
 9:            isAvailable($x \oplus x' \oplus x''$)
10: end if
```

Preventing redundancy

We now assume the existence of a feedback channel allowing the receiver to provide the sender with useful information helping it to increase the probability of sending an innovative encoded packet.

Consider in a first step a basic binary feedback channel that allows the receiver to abort the transfer if the encoded packet sent is detected as a redundant one. Assume for instance that an encoded packet is sent through a TCP connection and that the corresponding code vector precedes the data. Then, as soon as the code vector is received, the receiver can run the redundancy detection mechanism and close the connection if the packet is non-innovative. This prevents the sender from wasting bandwidth sending useless data.

Consider now, a fully operational feedback channel allowing the receiver to provide the sender with more complex information. A naive solution is to run the very same redundancy detection algorithm at the sender (assuming that the required information available at the receiver has been transfered using the feedback channel beforehand) and abort the transmission if the fresh generated packet is not innovative for the receiver. In this case the bandwidth is saved but the *session* is wasted as no packet is sent. A more sophisticated solution is to determine the intersection of what can be generated at the sender and what is innovative for the receiver. Again, this is a difficult problem in general, but simple and efficient solutions can be implemented for low degree encoded packets at a reasonable computational cost. In LTNC, a "smart" packet construction algorithm using information from the receiver is used only for packets of degree 1 and 2 in order to limit the amount of data exchanged: the leader-based representation cc_r is sent to the sender through the feedback channel. Note that similar information can be partially obtained or inferred in a wireless setting by snooping packets sent by close devices as in COPE [56]. For degree 1 or 2, the problem can be formalized as follows:

$$(d = 1)\text{: Find } x \text{ s.t. isAvailable}_s(x) \text{ and not(isAvailable}_r(x))$$

$$(d = 2)\text{: Find } x,\ x' \text{ s.t. } \mathrm{cc}_s(x) = \mathrm{cc}_s(x') \text{ and } \mathrm{cc}_r(x) \neq \mathrm{cc}_r(x'),$$

where cc_s (resp. cc_r) is the leader-based representation of the connected components of native packets at the sender (resp. the receiver).

The first case is straightforward. The second can be addressed by constructing iteratively a mapping σ between the connected components of native packets at the source and at the receiver. The native packets are processed in random order. If a connected component at the source overlaps with two components at the receiver (i.e., maps to two distinct components), an innovative packet can be generated. Algorithm 5 gives a pseudo-code version of the smart packet construction algorithm.

Algorithm 5 Constructing an innovative packet ($d = 2$)

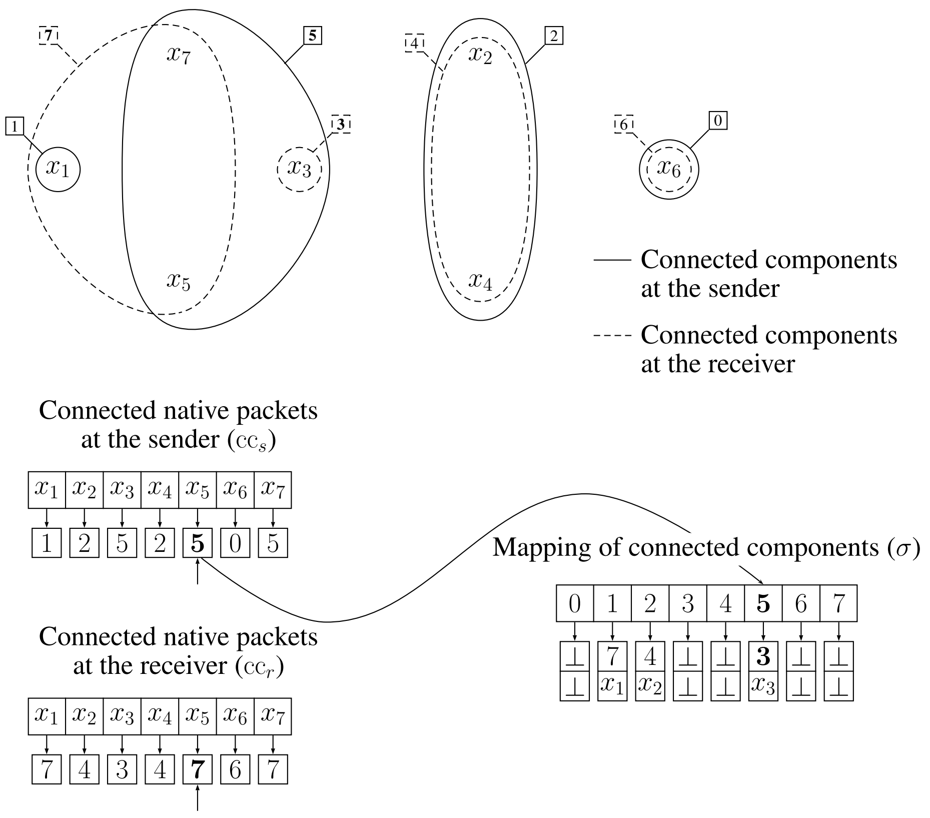
```
   $\sigma\{0, \ldots, k\} \mapsto (\perp, \perp)$          ▷ Mapping between connected components

1: for each $x_i \in \{x_i\}_{i=1}^{k}$ do
2:     $(j, x) \leftarrow \sigma[\mathrm{cc}_s(i)]$
3:     if $j = \perp$ then                    ▷ First time component $\mathrm{cc}_s(i)$ is visited
4:         $\sigma[\mathrm{cc}_s(i)] \leftarrow (\mathrm{cc}_r(i), x_i)$
5:     else if $j \neq \mathrm{cc}_s(i)$ then      ▷ Already visited with different mapping
6:         send($x \oplus x_i$)
7:     end if
8: end for
```

Consider the example depicted in Figure 3.4 and assume that native packets are processed by increasing indexes. When x_1 is processed, $\sigma[\mathrm{cc}_s(x_1)]$ (i.e., $\sigma[1]$) has not been initialized yet and is thus set to $(\mathrm{cc}_r(x_1), x_1)$ (i.e., $(7, x_1)$). Similar updates are performed for x_2 and x_3. When processing x_4, a mapping already exists and is consistent with $\mathrm{cc}_r(x_4)$ (i.e., the packet $x_2 \oplus x_4$ is not innovative). However, with x_5 the mapping is not consistent: an innovative packet (i.e., $x_3 \oplus x_5$) can therefore be generated.

3.4 Evaluation

In this section we present an evaluation of LTNC based on simulations. In order to evaluate LTNC, we compared it against two dissemination schemes: without coding and with random linear network coding (RLNC). Our simulations show that LTNC incurs only 20% more message emissions than RLNC while reducing the computational complexity by up to 99% at decoding. In addition, despite the fact that LTNC does not perform as well as RLNC with respect to latency, it still outperforms epidemic schemes that do not use network codes, thus preserving the benefits of using coding techniques.



Connected components at the sender
Connected components at the receiver
Connected native packets at the sender (cc_s)
Connected native packets at the receiver (cc_r)
Mapping of connected components (σ)

Figure 3.4: Sample execution of the "smart" packet construction algorithm: component 5 at the sender overlaps with components 3 and 7 at the receiver.

3.4.1 Experimental setup

We consider a network of N devices where a content is disseminated from a source to all of the N devices in an epidemic fashion. The content is divided into k native packets of size m. The code vectors of encoded packets, represented by bitmaps, are included in the headers of the packets. The source periodically injects encoded packets in the network. As soon as a device has received more than a given proportion of the packets, it starts periodically generating and pushing fresh encoded packets. The proportion of packets required to trigger recoding is controlled by a parameter of the system called *aggressiveness*. In our simulations, the aggressiveness is set so that the completion time is minimized (typically 1% for LTNC, note that in WC and RLNC, recoding can be done without delay). Packets are pushed to devices picked uniformly at random in the network, using an underlying peer sampling service (e.g., [52]). The set of devices to which a device pushes packets is renewed periodically in a gossip fashion. The underlying overlay is therefore *dynamic*. Communications are *unicast* and we assume the existence of a *binary feedback channel* allowing the receiver to abort the transfer of a native or encoded packet detected as non-innovative. Aborting a transfer is simply achieved by closing the TCP connection (assuming that code vectors are in the headers of the packets, the receiver is able to determine if the corresponding packet is redundant before the content is actually sent). In such an application, the size of the system N is generally *a few thousands* of devices, and a typical content is a file of 512 MB (e.g., a video) divided into $k = 2,048$ blocks of size $m = 256$ KB.

We compare LTNC against two reference schemes, namely Random Linear Network Coding (RLNC) and Without Coding (WC), that capture the trade-off between the performance with respect to content dissemination (i.e., average time to complete and communication cost) and the computational cost of the operations performed at the devices (i.e., recoding and decoding). In our simulations, we therefore implemented our proposed approach (LTNC) and the two reference schemes:

- **LT Network coding (LTNC):** In this scheme, the degree distributions of encoded and recoded packets and the distribution of native packets follow the distributions of LT codes. Linear combinations of native packets are performed over GF(2). The recoding and redundancy detection techniques used are those described in Sections 3.3.2 and 3.3.3. Decoding is performed using the belief propagation algorithm.

- **Without Coding (WC):** In this scheme, no coding is used. Therefore devices exchange only native packets and detecting a non-innovative packets boils down to checking if the packet has already been received. Devices buffer the innovative packets they receive up to a fixed number b. If the buffer is full, the oldest packet is discarded. Each received innovative packet is forwarded to f devices (unless the packet is removed from the buffer). At each gossip period *one* buffered packet (typically the one that has been sent the least number of times) is sent to *one* random device. It has been shown that f must be greater than $\lceil \ln N \rceil$ to ensure with high probability that all devices eventually receive all the native packets [39].

- **Random Linear Network Coding (RLNC):** In this scheme, devices generate fresh encoded packets by linearly combining, over GF(2), random combinations of previously received encoded packets. The number of encoded packets involved in the recoding operation is bounded by a given parameter, namely the *sparsity* of the codes, set to $\ln k + 20$. Limiting the number of encoded packets combined to generate a fresh encoded packet decreases the computational cost of the recoding operation without impacting the performance of content dissemination. This set of parameters is widely acknowledged as the optimal setting for linear network coding [68, 101]. Detecting non-innovative packets and decoding are performed using Gauss reduction.

3.4.2 Experimental results

We now compare the performance of the three dissemination schemes presented in the previous paragraph. Experimental results have been averaged over 25 Monte-Carlo simulations. We evaluate LTNC, RLNC and WC along three metrics: *(i)* convergence time, *(ii)* overhead, and *(iii)* computational complexities.

Convergence Figure 3.5 plots the proportion of devices that were able to decode the k native packets as a function of time. The network is composed of $N = 1,000$ devices and the file disseminated is composed of $k = 2,048$ packets of size $m = 256\,\text{KB}$. Results show that the convergence using LTNC is slightly slower than using RLNC. Yet, LTNC largely outperforms WC, showing the benefit of coding schemes in content dissemination. This is confirmed by the results depicted in Figure 3.6 where the average time to complete for several values of the code length k ranging from 512 to $4,096$ is plotted. Interestingly enough, the time overhead of LTNC with respect to RLNC decreases with the code length k.

Overhead Figure 3.7 depicts the communication overhead for several values of the code length k ranging from 512 to $4,096$. For $k = 2,048$, LTNC sends 20% more packets than necessary. The communication overhead decreases with k. Note that without coding and with RLNC the communication overhead is null as the redundancy detection mechanism can abort *all* transfers of non-innovative packets using respectively lookups and Gauss reductions.

Computational cost Figure 3.8 compares the computational complexities in terms of CPU cycles of LTNC and RLNC. The complexity of the operations performed on the control structures (e.g., Tanner graph for LTNC and code matrix for RNLC) and those on the data are plotted separately for encoding, recoding and decoding. This results have been obtained on an Intel Xeon 32bit at 2.33GHz with 1GB of RAM. The program has been compiled with gcc 4.4 with the optimization parameter set to `-O3`.

We observe that for $m = 256$ KB the cost of the operations performed on the control structure is negligible as compared to those performed on data. The costs of encoding are slightly lower in the case of LTNC thanks to the low degree of encoded packets. Due to the building and refining steps used by LTNC, its complexity at recoding is higher than for RLNC. However, since the average degree of encoded packet sent is lower for LTNC, the cost of recoding data is lower for LTNC. Note that in both cases, the recoding complexity on data scales well with the code length. For RLNC this is due to the use of sparse codes as described in the experimental setup. For $k = 2,048$, LTNC decreases the decoding complexity by more than 99%, thanks to belief propagation (made possible by the structure of our distributed LT network codes), which fully justifies the reasonable communication overhead for time constrained applications. In conclusion, LTNC advantageously trades communication overhead for computational complexity. More specifically, the increase (20%) of the

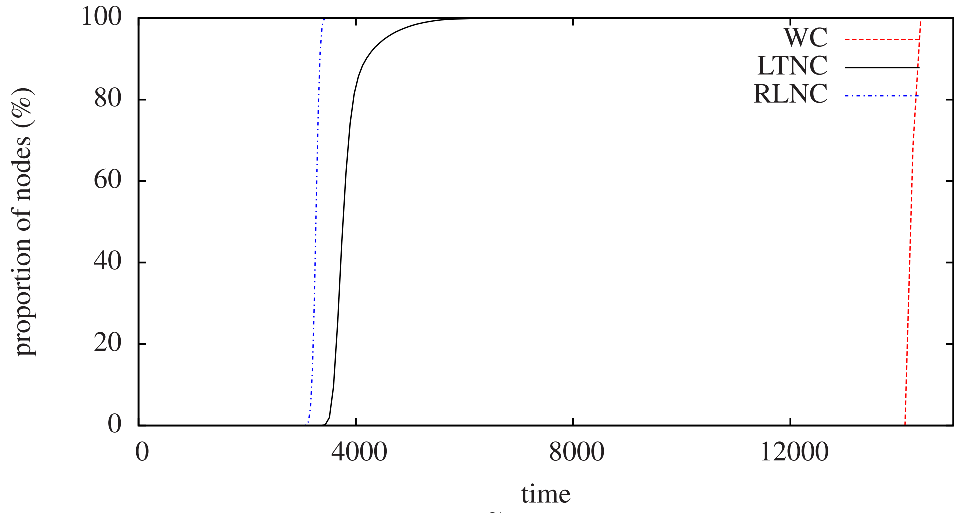


Figure 3.5: Convergence

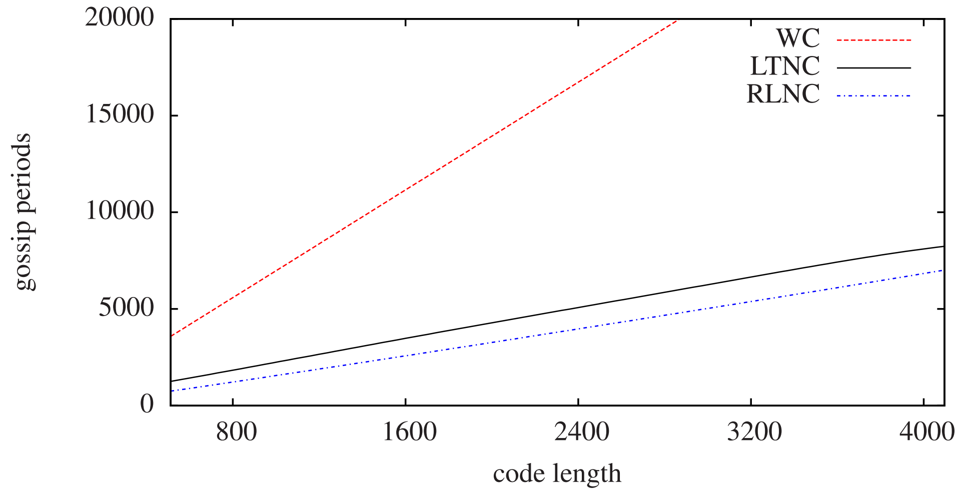


Figure 3.6: Average time to complete.

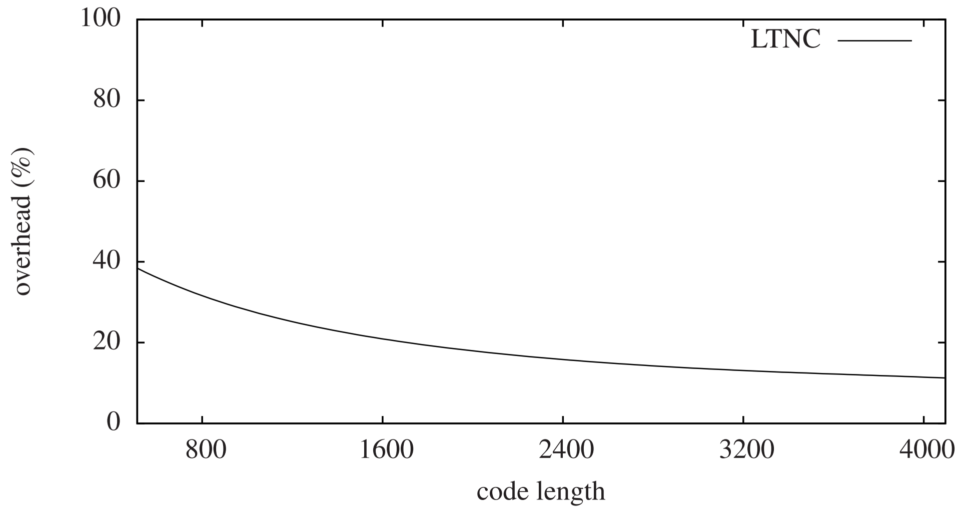
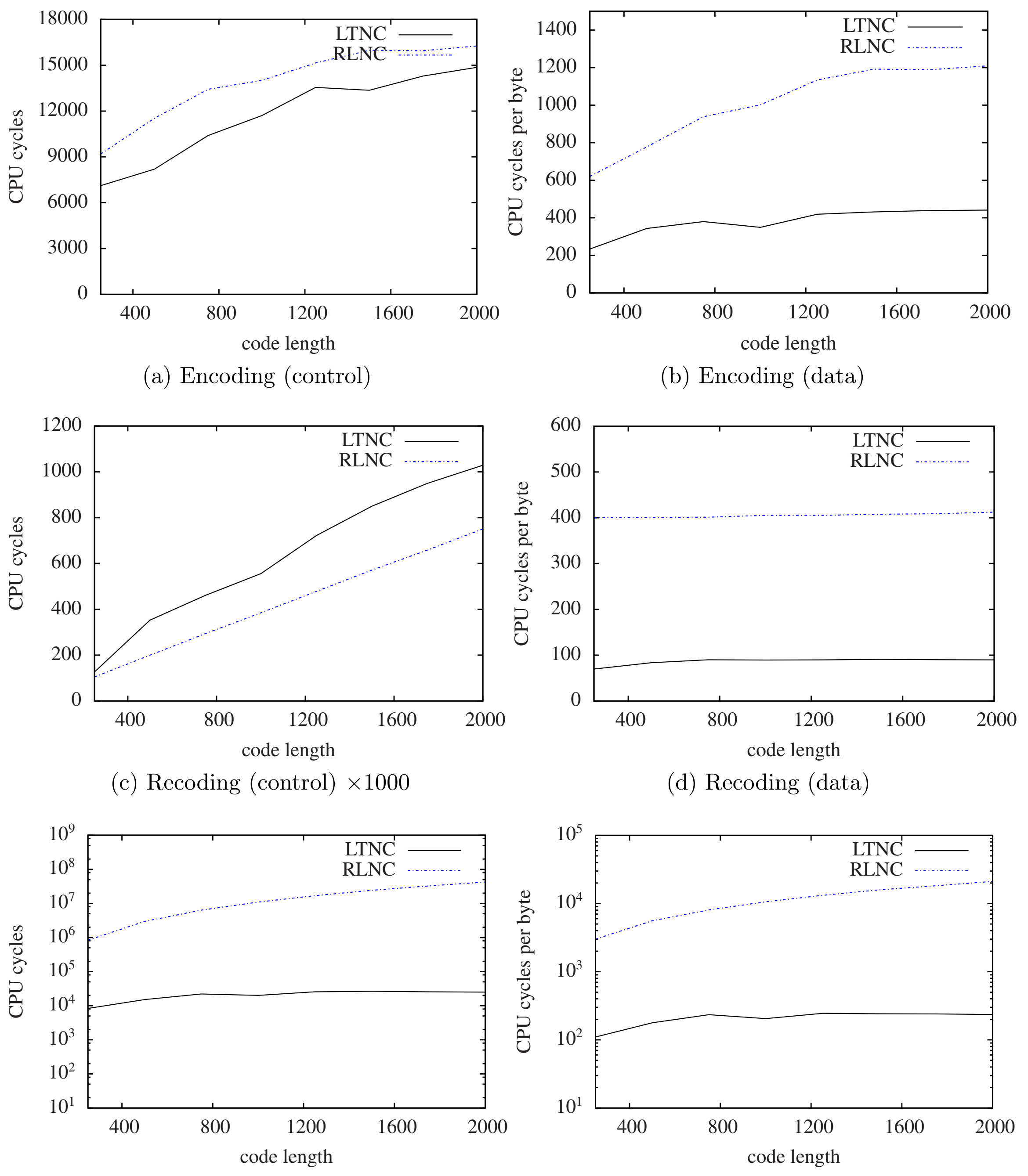


Figure 3.7: Overhead.



(a) Encoding (control) (b) Encoding (data)

(c) Recoding (control) ×1000 (d) Recoding (data)

(e) Decoding (control) (f) Decoding (data)

Figure 3.8: Computational cost of each operation (CPU cycles).

number of packets sent is largely compensated by a huge gain in CPU cycles at decoding (99%).

3.5 Related Work

To the best of our knowledge, LTNC is the first coding scheme to tackle the problem of generating encoded packets that match a given degree distribution using *encoded packets* which makes it the first LT network coding scheme.

Linear network coding is an efficient paradigm introduced in [2] that allows reaching max flow using path diversity in a communication network. However to achieve optimal performance it requires to determine the recoding matrices applied at each intermediary devices function of the global network topology which is difficult in a decentralized setting. Random linear network codes [49] (RLNC) alleviate the need for global coordination between intermediary devices by generating at each device random linear combinations of previously received encoded packets. RLNC achieves close-to-optimal performance and has been successfully applied to content dissemination in sensor networks [103] and file sharing systems, namely Avalanche [43]. However it has been widely criticized for the high computational cost of the decoding [68, 101]. Some optimizations to reduce the complexity of Gauss reduction – such as generations [71] – have been proposed but the complexity remains quadratic with the length of the codes inside generations.

LT Codes [66] and Raptor Codes [94] (LT codes built on precoded native packets) are erasure codes that rely on specific degree distributions of encoded packets to allow low-complexity decoding using belief propagation. In their initial versions, they do not provide a recoding method to use them as network codes. In [99] a network coding solution based on Raptor codes is proposed: sources devices send Raptor-encoded packet and intermediary devices generate fresh encoded packets using specific recoding matrix. However, this recoding technique does not preserve the degree distribution of Raptor codes. Therefore, the decoder must perform a high complexity Gauss reduction thus loosing the benefit of belief propagation.

Distributed LT Codes [81] distributes the construction of LT encoded packets on several *server devices*. The output of the server devices are then summed up by a front-end machine, namely the *relay device* and injected in the network. Intermediary devices then just forward encoded packets received: no network coding is involved. The main contribution of the paper is to derive an appropriate distribution of degrees for the packets generated at each server device such that the degree distribution of packets emitted by the relay device fits a Robust Soliton distribution.

In [46], stacked LT codes for dissemination trees are proposed: encoded packets at level i in the tree are LT codes built over encoded packets at level $i-1$ (i.e., considering level $i-1$ encoded packet as native packets). The problem of collecting, at a sink device, data initially stored at networked sensor devices is addressed in [54] using Growth codes: the degree of the encoded packets circulating in the network grows with time (initially native packets are sent) such that each encoded packet received enables to decode a new native packet. This solution addresses a different problem than LTNC and requires the devices to be loosely synchronized. Furthermore, the use of Growth codes was motivated by the fact that the authors want to maximize the number of native packets decoded in a many-to-one scenario while LTNC aims at maximizing the proportion of devices that can recover *all* the native packets in a one-to-many scenario. It is shown in [54] that Growth codes outperform LT codes in the first case but perform worse than LT codes in the latter.

Network codes have been widely used in distributed storage as well to ensure data persistence. LTCDS [4] (LT Codes Distributed Storage) replicates on all the devices – in an encoded form – k native packets, each of them being initially available at a single device. The native packets circulate in the network using random walks and each device uses them to build the encoded packet it stores: LT codes are built using only native packets. In the end, the degree distribution of the encoded packets distributed in the network follows a Robust Soliton distribution. Similar solutions have been proposed in [3] and [33].

In [27], the author derives the probability of an encoded packet to be innovative, function of the number of native packets decoded at the receiver. Deriving this distribution allows the sender to optimize the utility of the encoded packet sent by carefully choosing its degree. The information on the number of native packets decoded by a device is available from an Oracle. An extension of this work alleviating the need for an Oracle is proposed in [14]. It provides algorithmic tools to evaluate the similarity between the set of encoded packets available at the sender and the receiver. Similarity is estimated using a Bloom filter that summarize the content available at both devices with a small fixed number of bits. Information about similarity is then used to choose an optimal degree (i.e., that maximizes the probability of being innovative) for the encoded packet sent. However, the two aforementioned techniques address neither the problem of generating an encoded packet of a given degree from a set of encoded packets nor the problem of matching a given degree distribution of native packets.

3.6 Summary

In this chapter, we presented novel low complexity network codes (LTNC) based on LT codes, initially proposed in erasure correcting codes. LTNC provides an attractive alternative to random linear network codes for they greatly simplify the decoding complexity, trading the Gauss reduction for belief propagation. LTNC implements a set of algorithms enabling to recode from encoded packets. This is achieved by preserving on the fly and in a fully-decentralized manner the statistical properties of LT codes required to benefit from belief propagation decoding.

Experimental evaluations show that LTNC consistently outperforms an unencoded dissemination protocol with respect to delay, preserving the benefit of coding. Random linear network codes are optimal and not surprisingly LTNC introduces a small overhead in term of latency and communication. However, LTNC substantially reduces, up to 99%, the decoding complexity when compared to random linear network codes. This significantly broadens the spectrum of application settings for network codes, typically where devices have low capabilities.

Chapter 4

Network Codes for Storage

In this chapter, we present our advances in the use of codes in storage systems. Besides their uses in dissemination protocols, codes are also a very efficient way to deal with the unpredictability of device failures in distributed storage systems. Furthermore, Dimakis et al. *have recently shown [32] that network coding allows significant improvements (with respect to the cost of repairs) over erasure correcting codes. We build upon [32] and propose new codes that allow simultaneous coordinated repairs and that can self-adapt to the system state. The improvements in both repair costs and flexibility through the new adapative form of regenerating codes allow integration in more systems, in particular very dynamic one.*

This work is under submission.

4.1 Motivation

Over the last decade, the growth of digital information (photos, videos, scientific data) and the widespread access to networking technology has favored the development of large-scale distributed storage systems. The large amounts of data to store put stress upon both storage devices and network links thus requiring distributed storage systems that are both optimal with respect to storage and with respect to network communications. In the previous chapter, we studied network coding in dissemination as it allows very efficient yet simple protocols. We now consider how network coding can allow efficient storage systems with simple mechanisms.

Distributed storage systems are traditionally built by aggregating numerous physical devices to provide large and resilient storage [7, 28, 42, 84]. In such systems, which are prone to disk and network failures, redundancy is the natural solution to prevent permanent data losses. However, as failures occur, the level of redundancy decreases, potentially jeopardizing the ability to recover the original data. This requires the storage system to self-repair to go back to its healthy state (i.e., keep redundancy above a minimum level).

Repairing redundancy is of paramount importance for the design and implementation of distributed storage systems. A self-healing mechanism is usually composed of three phases. Firstly, the system must self-monitor to detect failures. Secondly, the system must trigger a repair on a set of spare devices. Finally, the system must regenerate the lost redundancy from the remaining one. In this chapter, we focus on this last phase. Redundancy in storage systems has been extensively implemented using erasure correcting codes [62, 86, 102] because they enable tolerance to failures at a low storage overhead. In this context, the repair used to induce a large communication overhead, as it required to download and decode the whole file.

Several approaches have been proposed to avoid decoding, thus reducing cost. Low costs repairs can be achieved by structuring the data stored either in hybrid systems [86] (combination of replication and erasure correcting codes), or using specific codes [36, 75]. Yet, these approaches are either not optimal in storage or not optimal in bandwidth. Moreover, they add complexity to the system as the repair strongly depends on which blocks are lost. Contrary to such approaches, network codes (i.e., regenerating codes [31, 32]) allow very efficient (optimal) yet simple (non-structured) distributed storage systems. In the aforementioned papers, the authors limit their regenerating codes to single failures and static systems.

In this chapter, we go beyond these works by considering simultaneous repairs in regenerating-like codes. We propose *coordinated regenerating codes* allowing devices to leverage simultaneous repairs: each of the t devices being repaired contacts d live (i.e., non-failed) devices and then coordinates with the $t-1$ others. Our contribution is threefold:

- As deliberately delaying repairs in erasure correcting codes leads to savings [7,29, 30], it is natural to wonder if the same additional savings can be expected when delaying repairs for regenerating codes. By defining *coordinated regenerating codes*, we prove that, when relying on regenerating-like codes (MSR or MBR) [32], deliberately delaying repairs (so that $t > 1$) cannot lead to further savings.

- Yet in practical systems, it might be difficult to detect every single failure and fix it before a second one occurs (i.e., ensure $t = 1$). We establish the optimal quantities of information to be transferred when t devices must be repaired simultaneously from d live devices. Our *coordinated regenerating codes* consistently outperform existing approaches.

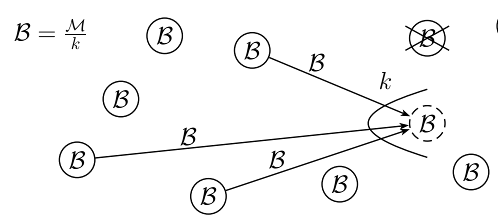
- In addition, most practical systems are highly dynamic. Therefore, assuming that t and d remain constant across repairs is unrealistic. To address this issue, we define *adaptive regenerating codes* achieving optimal repairs according to t and d. Under a constant system size $d + t$, their performance does not depend on t (i.e., the cost of a repair does not increase if the number of failed devices to repair increases).

Previous approaches are either known for not supporting simultaneous coordinated repairs [32] or known for assuming that repairing implies decoding which requires to download the whole file [7, 29, 30, 62, 102]. Hence, we define *coordinated regenerating codes* that fill the gap between the two aforementioned approaches by achieving simultaneous repairs without decoding (Figure 4.3). Since, for erasure correcting codes, simultaneous repairs reduce the communication overhead, we study the impact of deliberately delaying repairs with regenerating-like codes. Moreover, existing regenerating codes [32] assume a static setting: they require that d (the number of devices involved in the repairs) remains constant across repairs. Our *adaptive regenerating codes* consider a dynamic setting.

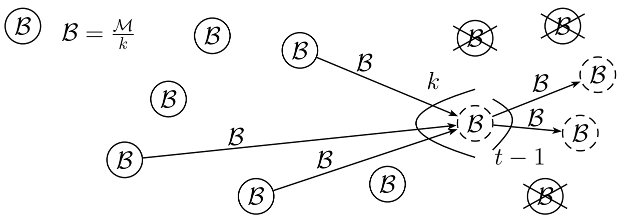
The chapter is organized as follows. Section 4.2 describes the background on codes. Section 4.3 presents our *coordinated regenerating codes.* In this section, we prove the optimality of our codes, we derive closed-form expressions for specific subsets of codes, and we show that deliberately delaying repairs does not reduce the costs further as long as each failure can be fixed before a second one occurs. In Section 4.4, we propose *adaptive regenerating codes* that self-adapt to the dynamic of the system and we prove their optimality.

4.2 Context

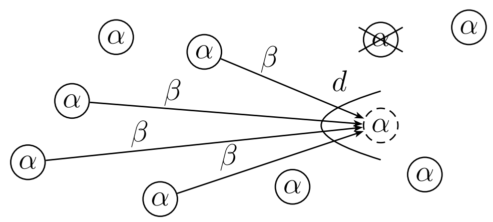
We consider a n device system storing a file of $\mathcal{M}$ bits split in k blocks of size $\mathcal{B} = \mathcal{M}/k$. To cope with device failures, blocks are stored with some redundancy so that a single failure cannot lead to permanent data losses. We consider code-based approaches to redundancy since they have been proved to be more efficient than replication with respect to both storage and repair costs [102]. We focus on self-healing systems (i.e., systems that automatically repair themselves upon failure). Distributed storage systems are required to be self-healing so as to ensure that the system does not gradually lose its ability to recover the initial file. Self-healing systems are equipped with a self-monitoring component that detects failed devices and triggers repairs [74] on new spare devices. To repair, the new spare devices regenerate the lost redundancy from data downloaded from live devices. The repair is constrained by the communication costs [8]. The main approaches to introducing redundancy in distributed storage systems have been described in Chapter 1 (page 20). In the rest of this section, we give more details about the various code-based approaches that serve as references for comparison (erasure correcting codes, lazy repairs, and regenerating codes). We also highlight the limitations of regenerating codes [32] so as to motivate this work.



(a) Erasure correcting codes (immediate repair)



(b) Erasure correcting codes (delayed repair)



(c) Regenerating codes

Figure 4.1: Repairing failures with codes. In a n devices network, failed devices are replaced by new ones. The new devices get a given amount of data from live devices to repair the redundancy. In our examples, $k = 3$, $d = 4$, $\mathcal{B} = 1$, $\alpha = 1$, and $\beta = 1/2$. Some example of gains for $k = 32$ are given in Table 4.1

4.2.1 Erasure correcting codes (immediate/eager repairs)

Erasure correcting codes have been widely used to provide redundancy in distributed storage systems [62, 86, 102]. Devices store n encoded blocks of size $\mathcal{B}$, which are computed from the k native (original) blocks. As erasure correcting codes are optimal with respect to recovery (i.e., they allow recovering the k native blocks from any subset of k encoded blocks), storing encoded blocks at $n = k + f$ devices is sufficient to tolerate f failures. This approach is efficient with respect to storage. Yet, repairing a lost encoded block is very expensive since the devices must fully decode the initial file. Hence, repairing a single lost block implies downloading k encoded blocks as shown on Figure 4.1a.

4.2.2 Erasure correcting codes (delayed/lazy repairs)

A first approach to limit the repair cost of erasure correcting codes is to delay repairs and factor downloading costs [7,29,30]. When a device has downloaded k blocks, it can produce as many new encoded blocks as wanted without any additional cost. Therefore, instead of performing a repair upon every single failure, repairs are deliberately delayed until t (threshold) failures are detected. This repair strategy is depicted on Figure 4.1b. One of the new devices downloads k blocks, regenerates t blocks and dispatches them to the $t - 1$ other spare new devices.

4.2.3 Network coding and regenerating codes

A second approach to increase the efficiency of repairs relies on network coding. Network coding differs from erasure correcting codes as it allows devices to generate new blocks with only partial knowledge (i.e., with less than $\mathcal{M}$ bits). Network coding was initially applied to multicast, for which it has been proved that linear codes achieve maxflow in a communication graph [2, 57, 60]. Network coding has latter been applied to distributed storage and data persistence [1,33,34,54,63]. A key contribution in this area are regenerating codes [31, 32] introduced by Dimakis *et al.* They infer the minimum amount of information to be transferred to repair lost redundancy.

The idea behind regenerating codes [31] is that, when compared to erasure correcting codes, more devices are contacted upon repair but much less data is downloaded from them thus offering low repair cost for each single failure. Similarly to erasure correcting codes, n encoded blocks of size α bits are computed from the k native blocks. Each device stores $\alpha \approx \frac{\mathcal{M}}{k}$ bits. During a repair, the new device contacts $d > k$ other devices to get $\beta \ll \mathcal{M}/k$ bits from each and stores α bits as shown on Figure 4.1c.

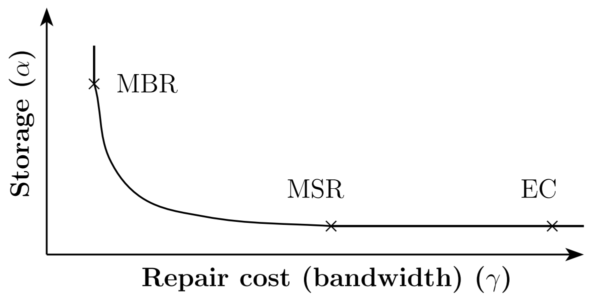


Figure 4.2: Regenerating codes (MSR or MBR) offer improved performances when compared to erasure correcting codes (EC)

Regenerating codes achieve an optimal trade-off between the storage α and the repair cost $\gamma = d\beta$. The graph on Figure 4.2 depicts the performance of the optimal regenerating codes. Points (α, γ) above the curve correspond to correct (but non-optimal) regenerating codes. Two specific regenerating codes are interesting: MSR (Minimum Storage Regenerating codes) offer optimal repair cost $\gamma = \frac{\mathcal{M}}{k}\frac{d}{d-k+1}$ for a minimum storage cost $\alpha = \frac{\mathcal{M}}{k}$, and MBR (Minimum Bandwidth Regenerating codes) offer optimal storage cost $\alpha = \frac{\mathcal{M}}{k}\frac{2}{2d-k+1}$ for a minimum repair cost $\gamma = \frac{\mathcal{M}}{k}\frac{2d}{2d-k+1}$. Regenerating codes can be implemented using linear codes [57, 60] be they random [37, 49] (i.e., random linear network codes), or deterministic [35, 82, 91, 97, 104]. Similarly to regenerating codes, our codes can be implemented using random linear network codes.

Table 4.1 gives some examples of storage cost α and repair cost γ for the codes we describe including the coordinated codes we propose. These costs depend on the number k of devices needed to recover the file, the number d of contacted live devices, and the number t of devices being repaired simultaneously. Regenerating codes by Dimakis *et al.* [32] represent a clear improvement over erasure correcting codes, and our coordinated regenerating codes allow reducing further the costs.

4.2.4 Design rationale

Regenerating codes transfer the minimal quantity of information needed to repair one device storing α bits. Dimakis *et al.* assume fully independent repairs: simultaneous failures are fixed independently. The cost of repairs increases linearly in t.

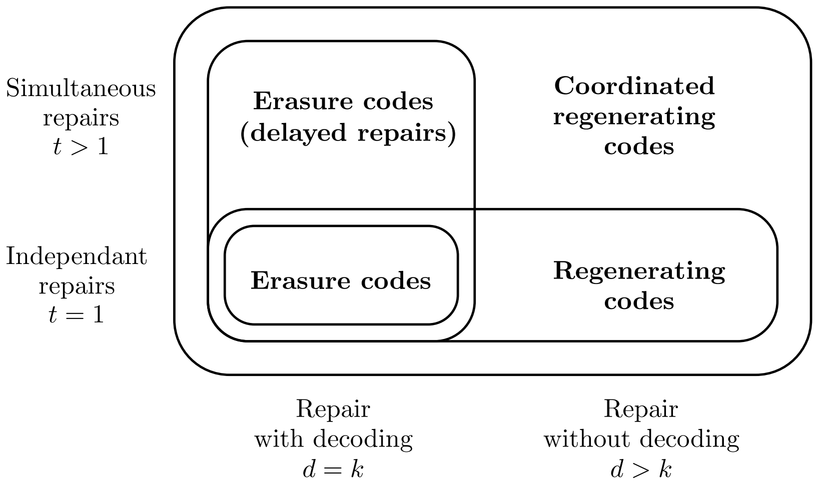

Simultaneous repairs $t > 1$
Independant repairs $t = 1$
Erasure codes (delayed repairs)
Coordinated regenerating codes
Erasure codes
Regenerating codes
Repair with decoding $d = k$
Repair without decoding $d > k$

Figure 4.3: Our coordinated regenerating codes encompass all existing codes. Moreover, they allow the repair of multiple devices at once without decoding.

In this work, we investigate coordinately repairing simultaneous failures in an attempt to reduce the cost, along the lines of delayed erasure codes. Contrary to regenerating codes that repair only using data from live devices, our coordinated regenerating codes also allow the use of data from other devices being repaired. By coordinating the repair, we show that it is possible to repair multiple failures at once with an average cost of $\tilde{\gamma} < \gamma$. As depicted on Figure 4.3, our new codes encompass

Table 4.1: Some examples of repairs of codes for a file of 32 MB

	k	d	t	α	γ
Erasure codes	32	*NA*	*NA*	1 MB	32 MB
Erasure codes (delayed repair)	32	*NA*	4	1 MB	8.8 MB
Dimakis *et al.*'s MSR	32	36	*NA*	1 MB	7.2 MB
Dimakis *et al.*'s MBR	32	36	*NA*	1.8 MB	1.8 MB
Our MSCR (cf. Sec. 4.3.4)	32	36	4	1 MB	4.9 MB
Our MBCR (cf. Sec. 4.3.4)	32	36	4	1.7 MB	1.7 MB

both erasure correcting codes ($d = k$) and regenerating codes ($t = 1$). In the next section, we detail our coordinated regenerating codes supporting coordinated repairs.

4.3 Coordinated Regenerating Codes

We consider a situation where t devices fail and repairs are performed simultaneously. We assume that an underlying monitoring service triggers the repair and contacts all involved devices (i.e., the t spare devices that join the system to replace failed ones). Directly applying erasure correcting codes delayed repairs (Fig. 4.1b) to regenerating codes (Fig. 4.1c) is not appropriate. In short, the fact that all the data goes through the device that regenerates blocks for others induce more network transfers than needed: as repairing does not require decoding, gathering all the information at a single device is not necessary.

4.3.1 Repair algorithm

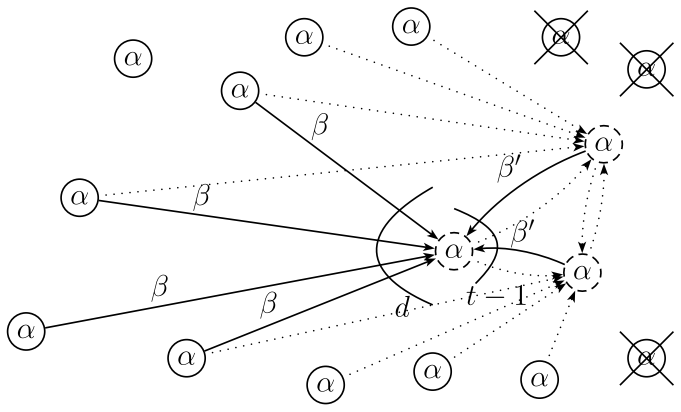


Figure 4.4: Repairing failures in coordinated regenerating codes. In a network of n devices storing α bits, when t devices have failed, t new devices collect β bits from d live devices. They coordinate by exchanging $(t-1)\beta'$ bits with other new devices and store α bits.

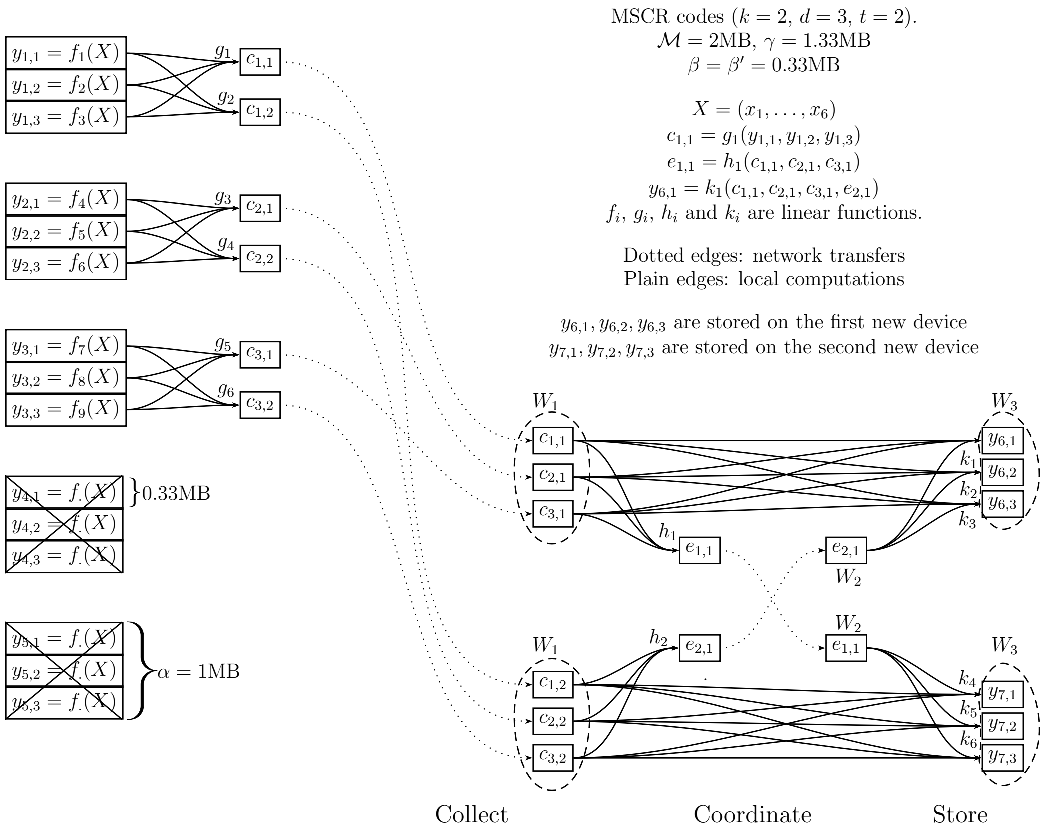
We introduce new *coordinated regenerating codes* allowing devices to repair simultaneously at the optimal cost (with respect to network communication). The repair process is illustrated on Figures 4.4 and 4.6 showing the amounts of information transfered, and on Figure 4.5 showing the computations and exchanges of sub-blocks of data. A device being repaired performs the three following tasks, jointly with all other devices being repaired:

1. **Collect.** The device downloads a set of sub-blocks (size β) from each of d live devices. The union of the sets is stored as W_1.

2. **Coordinate.** Then, the device uploads a set of sub-blocks (size β') to each of the $t-1$ other devices being repaired. These sets are computed from W_1. During this step, sub-blocks received from the $t-1$ other devices being repaired are stored as W_2. The data exchanged during this step can be considered as a digest of what each has received during the collecting step.

3. **Store.** Finally, the device stores a set W_3 of sub-blocks (size α) computed from $W_1 \cup W_2$. W_1 and W_2 can be erased upon completion.

Interestingly, thanks to the explicit coordination step involving all devices, this approach evenly balances the load on all devices. Hence, coordinated regenerating codes avoid the bottleneck existing in erasure correcting codes delayed repairs (i.e., the device gathering all the information) (cf. Fig. 4.1b).



MSCR codes ($k=2$, $d=3$, $t=2$).
$\mathcal{M}=2$MB, $\gamma=1.33$MB
$\beta=\beta'=0.33$MB
$X=(x_1,\dots,x_6)$
$c_{1,1}=g_1(y_{1,1},y_{1,2},y_{1,3})$
$e_{1,1}=h_1(c_{1,1},c_{2,1},c_{3,1})$
$y_{6,1}=k_1(c_{1,1},c_{2,1},c_{3,1},e_{2,1})$
f_i, g_i, h_i and k_i are linear functions.
Dotted edges: network transfers
Plain edges: local computations
$y_{6,1},y_{6,2},y_{6,3}$ are stored on the first new device
$y_{7,1},y_{7,2},y_{7,3}$ are stored on the second new device
$y_{1,1}=f_1(X)$
$y_{1,2}=f_2(X)$
$y_{1,3}=f_3(X)$
$y_{2,1}=f_4(X)$
$y_{2,2}=f_5(X)$
$y_{2,3}=f_6(X)$
$y_{3,1}=f_7(X)$
$y_{3,2}=f_8(X)$
$y_{3,3}=f_9(X)$
$y_{4,1}=f_.(X)$
$y_{4,2}=f_.(X)$
$y_{4,3}=f_.(X)$
0.33MB
$y_{5,1}=f_.(X)$
$y_{5,2}=f_.(X)$
$y_{5,3}=f_.(X)$
$\alpha=1$MB
g_1 g_2 g_3 g_4 g_5 g_6
$c_{1,1}$ $c_{1,2}$ $c_{2,1}$ $c_{2,2}$ $c_{3,1}$ $c_{3,2}$
W_1 W_2 W_3
h_1 h_2
$e_{1,1}$ $e_{2,1}$
k_1 k_2 k_3 k_4 k_5 k_6
$y_{6,1}$ $y_{6,2}$ $y_{6,3}$
$y_{7,1}$ $y_{7,2}$ $y_{7,3}$
Collect
Coordinate
Store

Figure 4.5: Coordinated regenerating codes based on linear codes. The system store a file X and is compound of 5 devices. Device i stores 3 sub-blocks $\{y_{i,1}, y_{i,3}, y_{i,3}\}$. Devices 4 and 5 fail and are replaced by devices 6 and 7.

In the rest of this section, we present our main results: we investigate the optimal tradeoffs between storage and repairs costs (i.e., optimal values for α (data to store), β and β' (data to transfer)). As we consider the problem from an information theory point of view, we can ignore the nature of the information and only consider the

Table 4.2: Notation used in Section 4.3

k	Constant (Integer)	Number of devices needed to recover
t	Constant (Integer)	Number of devices being repaired
d	Constant (Integer)	Number of live devices ($d \geq k$)
α	Variable (Real)	Quantity stored
β	Variable (Real)	Quantity transferred (collect)
β'	Variable (Real)	Quantity transferred (coordinate)
γ	Expression (Real)	Quantity transferred over the network

amount of information that must be exchanged to repair the redundancy. Our results define the fundamental tradeoffs that can be achieved (i.e., lower bounds on amounts of information to be transfered to repair).

Our proof is inspired from the proof by Dimakis *et al.* found in [32]. We represent the system as an information flow graph. For we add a coordination step, our graph differs from the one proposed in [32]. However, Lemmas 2 and 3 are identical to the ones proposed in [32] as they still apply to our information flow graph. When compared to [32], we allow the coordination of multiple repairs while they assume fully independent repairs.

We determine the optimal codes (i.e., we minimize the storage and the repair cost under some constraints obtained by studying information flow graphs). We give the expressions for optimal values of α (storage at each node), γ (repair cost), β (data transferred during collecting step) and β' (data transferred during coordinating step) as a function of d, k and t (parameters of the system). We also examine the influence of deliberately delaying repairs (i.e., increasing t while decreasing d). Our notations are summarized in Table 4.2.

4.3.2 Information flow graphs

Our study is based on information flow graphs similar to the ones defined in [32]. An information flow graph is a representation of a distributed storage system that describes how the information about the file stored is communicated through the network. The information flow graph $\mathcal{G}$ is a directed acyclic graph composed of a source S, intermediary nodes $x_{\text{in}}^{i,j}$, $x_{\text{coor}}^{i,j}$ and $x_{\text{out}}^{i,j}$, and data collectors DC_i (data collectors try to contact k devices to decode and recover the file). The capacities of the edges (α, β, β') correspond to the amount of information that can be stored or transferred.

In our approach, a real device $x^{i,j}$ is represented in the graph by 3 nodes ($x_{\text{in}}^{i,j}$, $x_{\text{coor}}^{i,j}$ and $x_{\text{out}}^{i,j}$) corresponding to its successive states. The graph of a repair of t devices is shown on Figure 4.6 (assume t divides k.). First, devices perform a collecting step represented by edges $x_{\text{out}}^{k,j} \rightarrow x_{\text{in}}^{i,j}$ ($k < i$) of capacity β (d such edges). Second, devices undergo a coordinating step represented by edges $x_{\text{in}}^{i,j} \rightarrow x_{\text{coor}}^{i,j'}$ of capacity β' for $j \neq j'$ ($t - 1$ such edges). Devices keep everything they obtained during the first step justifying the infinite capacities of edges $x_{\text{in}}^{i,j} \to x_{\text{coor}}^{i,j}$. Third, they store α using edges $x_{\text{coor}}^{i,j} \to x_{\text{out}}^{i,j}$. Figure 4.7 gives other examples of information flow graphs.

The graph $\mathcal{G}$ evolves as repairs are performed. When a repair is performed, a set of nodes is added to the graph and the nodes corresponding to failed devices become inactive (i.e., subsequently added intermediary nodes or data collectors cannot be connected to these nodes).

The rest of the chapter relies on the concept of minimum cuts in information flow graphs. A cut $\mathcal{C}$ between S and DC$_i$ is a subset of edges of $\mathcal{G}$ such that there is no path from S to DC$_i$ that does not have at least one edge in $\mathcal{C}$. The minimum cut is the cut that has the smallest sum of edge capacities.

4.3.3 Achievable codes

We define two important properties on codes:

Correctness A code $(n, k, d, t, \alpha, \gamma)$ is correct iff, for any system corresponding to an information flow graph which contains corresponding repairs, a data collector can recover the file by connecting to any k devices.

Optimality A code $(n, k, d, t, \alpha, \gamma)$ is optimal iff it is correct and any code $(n, k, d, t, \bar{\alpha}, \bar{\gamma})$ with $(\bar{\alpha}, \bar{\gamma}) < (\alpha, \gamma)$ is not correct.

The following theorem is an important result of our work.

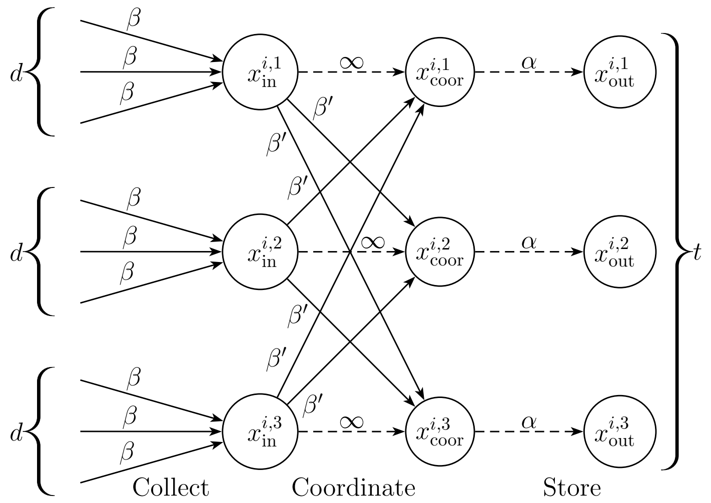


Figure 4.6: Information flow graph of a repair of $t = 3$ devices. The internal nodes of the graph represent intermediary steps in the repair. First, each device collects β from d live devices. Second, devices coordinate by exchanging β' with each other. Third, they store α. Plain edges correspond to network communication and dashed edges correspond to local communication.

Theorem 1. *A coordinated regenerating code* $(n, k, d, t, \alpha, \gamma)$ *is correct iff it is possible to find* β *and* β' *such that the Constraints* (4.2) *are satisfied. A code minimizing Equation* (4.1)*, along Constraints* (4.2) *is optimal. If the code is correct, linear network codes suffice to build one instance. The repair cost of this code is* γ *as defined by Equation* (4.1)*.*

$$\gamma = d\beta + (t-1)\beta' \tag{4.1}$$

$$\sum_{i=0}^{g-1} u_i \min\{\alpha, (d - \sum_{j=0}^{i-1} u_j)\beta + (t-u_i)\beta'\} \geq \mathcal{M} \tag{4.2}$$

$$\textit{where } k = \sum_{i=0}^{g-1} u_i \textit{ with } 1 \leq u_i \leq t$$

These constraints, proved in the rest of this subsection, mean that the sum of the amounts of information that can be downloaded from each of the k devices contacted by a data collector must be greater than the file size (amount of information needed to recover the original file). To this end, we consider, for a given coordinated regenerating code $(n, k, d, t, \alpha, \gamma)$, all (infinite) corresponding information flow graphs and evaluate the flow of information that can go from the source to any data collector in such graphs. The vector $\mathbf{u} = (u_i)_{0 \leq i < g}$ represents a possible recovery scenario (i.e., the number u_i of devices contacted in each repair group of size t during the recovery). Since the recovery must be possible for any scenario, we consider all possible $\mathbf{u}$. We show that Equation (4.2) must be satisfied to allow decoding at anytime (i.e., as long as the aforementioned constraints are satisfied, no data is lost).

Lemma 2. *For any information flow graph* $\mathcal{G}$*, no data collector* DC *can recover the initial file if the minimum cut in* $\mathcal{G}$ *between* S *and* DC *is smaller than the initial file size* $\mathcal{M}$*.*

Proof. Similarly to the proof in [32], since each link in the information flow graph can be used at most once, and since source to data collector capacity is less than the file size $\mathcal{M}$, the recovery of the file is impossible. □

Lemma 3. *For any finite information flow graph* $\mathcal{G}$*, if the minimum of the min-cuts separating the source and each data collector is larger than or equal to the file size* $\mathcal{M}$*, then there exists a linear network code such that all data collectors can recover the file. Furthermore, randomized network coding allows all collectors to recover the file with high probability.*

Proof. Similarly to the proof in [32], since the reconstruction problem reduces to multicasting on all possible data collectors, the result follows from the results in network coding theory which are briefly discussed in Chapter 1 and in Section 4.2. □

Lemma 4. *For any information flow graph $\mathcal{G}$ compounded of initial devices that obtain α bits directly from the source* S *and of additional devices that join the graph in groups of t devices obtaining β from d existing devices and β' from each of the other $t-1$ joining devices, any data collector* DC *that connects to a subset of k out-nodes (g groups of u_i out-nodes) of $\mathcal{G}$ satisfies:*

$$\mathrm{mincut}(\mathrm{S},\mathrm{DC}) \geq \sum_{i=0}^{g-1} u_i \min\{\alpha, (d - \sum_{j=0}^{i-1} u_j)\beta + (t-u_i)\beta'\} \tag{4.3}$$

where g is the number of different groups contacted, u_i is the size of each group, and $k = \sum_{i=0}^{g-1} u_i$ with $1 \leq u_i \leq t$.

Proof. We show that Equation (4.3) must be satisfied for any graph $\mathcal{G}$ (see examples in Figure 4.7) formed by adding devices according to the repair process described above. Consider a data collector DC that connects to a k-subset of "out-nodes", corresponding to a set of devices I, say $\{x_{\text{out}}^{i,j} : (i,j) \in I\}$. We show that any $\mathrm{S}-\mathrm{DC}$ cut in $\mathcal{G}$ has a capacity that satisfies Equation (4.3).

As all incoming edges of DC have infinite capacity, we only examine cuts $(U, \bar{U})$ with $\mathrm{S} \in U$ and $\{x_{\text{out}}^{i,j} : (i,j) \in I\} \subset \bar{U}$.

Let $\mathcal{C}$ denote the edges in the cut ($i.e$, the set of edges going from U to $\bar{U}$). Every directed acyclic graph has a topological sorting [6], which is an ordering of its vertices such that the existence of an edge $x \to y$ implies $x < y$. In the rest of the analysis, we group nodes that were repaired simultaneously. Since nodes are sorted, nodes considered at one step cannot depend on nodes considered at the following steps.

First group. Let J be a set of indexes such that $\{x_{\text{out}}^{0,j} : j \in J\}$ are the topologically first output nodes in $\bar{U}$ corresponding to a first (same) repair. The set contains $\#\{x_{\text{out}}^{0,j} : j \in J\} = u_0$ nodes. Consider a subset $M \subset J$ of size m such that $\{x_{\text{in}}^{0,j} : j \in M\} \subset U$ and $\{x_{\text{in}}^{0,j} : j \in J - M\} \subset \bar{U}$. m can take any value between 0 and u_0.

First, consider the m nodes $\{x_{\text{in}}^{0,j} : j \in M\}$. For each node, $x_{\text{in}}^{0,j} \in U$. We consider two cases.

- If $x_{\text{coor}}^{0,j} \in U$, then $x_{\text{coor}}^{0,j} \to x_{\text{out}}^{0,j} \in \mathcal{C}$. The contribution to the cut is α.
- If $x_{\text{coor}}^{0,j} \in \bar{U}$, then $x_{\text{in}}^{0,j} \to x_{\text{coor}}^{0,j} \in \mathcal{C}$. The contribution to the cut is ∞.

Second, consider the $u_0 - m$ other nodes $\{x_{\text{in}}^{0,j} : j \in J - M\}$. For each node, the contribution comes from multiple sources.

- The cut contains d edges carrying β: since $x_{\text{out}}^{0,j}$ are the topologically first output nodes in $\bar{U}$, edges come from output nodes in U.
- The cut contains $t - u_0 + m$ edges carrying β' thanks to the coordination step. The node $x_{\text{coor}}^{0,j}$ has t incoming edges $x_{\text{in}}^{0,k} \to x_{\text{coor}}^{0,j}$. However, since $\#(\{x_{\text{in}}^{0,k}\} \cap \bar{U}) = u_0 - m$, the cut contains only $t - (u_0 - m)$ such edges.

Therefore, the total contribution of these nodes is

$$c_0(m) \geq m \min(\alpha, \infty) + (u_0 - m)(d\beta + (t - u_0 + m)\beta')$$

Since the function c_0 is concave on the interval $[0 : u_0]$, the contribution can be bounded thanks to Jensen's inequality.

$$c_0(m) \geq u_0 \min\{\alpha, d\beta + (t - u_0)\beta'\}$$

Second group. Let $\{x_{\text{out}}^{1,j} : j \in J\}$ be the topologically second output nodes in $\bar{U}$ corresponding to a second (same) repair. We follow a similar reasoning.

First, consider the m nodes $\{x_{\text{in}}^{1,j} : j \in M\} \subset U$. Similarly to the above, the contribution of each node is $\min(\alpha, \infty)$.

Second, consider the $u_0 - m$ nodes $\{x_{\text{in}}^{1,j} : j \in J - M\} \subset \bar{U}$. For each node, the contribution comes from multiple sources.

- The cut contains at least $d - u_0$ edges carrying β: since $x_{\text{out}}^{1,j}$ are the topologically second output nodes in $\bar{U}$, at most u_0 edges come from output nodes in U, and at least $d - u_0$ other edges come from output nodes in $\bar{U}$.
- Similarly to the above, the cut contains $t - u_1 + m$ edges carrying β' thanks to the coordination step.

Therefore, the total contribution of these nodes is

$$c_1(m) \geq u_1 \min\{\alpha, (d - u_0)\beta + (t - u_1)\beta'\}$$

***i*-th group.** Following the same reasoning, we find that the i-th group of nodes $(i = 0, \ldots, g-1)$ in the sorted set $\bar{U}$ contributes

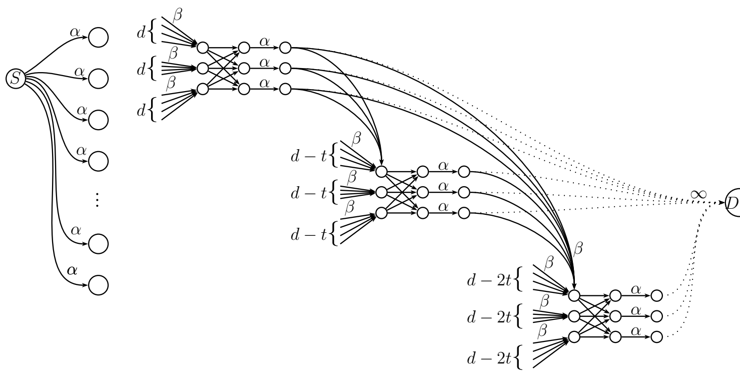
$$c_i(m) \geq u_i \min\{\alpha, (d - \sum_{j=0}^{i-1} u_j)\beta + (t - u_i)\beta'\}$$

Summing these contributions leads to Equation (4.3). □

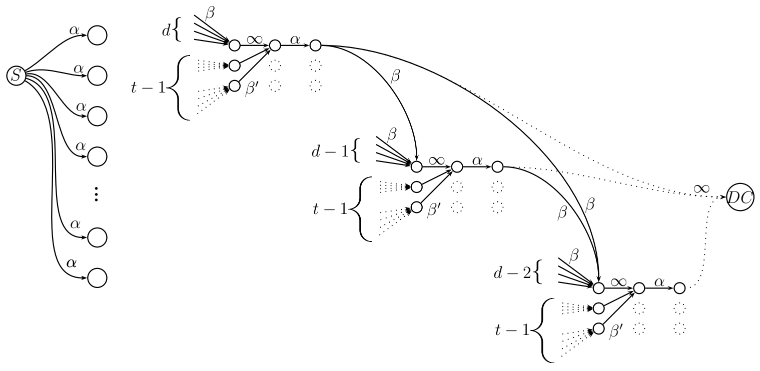
Proof of Theorem 1. From Lemmas 2 and 3, we require $\text{mincut}(\mathrm{S}, \mathrm{DC}) \geq \mathcal{M}$ and, from Lemma 4, we know $\text{mincut}(\mathrm{S}, \mathrm{DC})$ satisfies Equation (4.3). By combining both, we know that a code satisfying the following equation is correct.

$$\sum_{i=0}^{g-1} u_i \min\{\alpha, (d - \sum_{j=0}^{i-1} u_j)\beta + (t - u_i)\beta'\} \geq \mathcal{M}$$

$$\text{where } k = \sum_{i=0}^{g-1} u_i \text{ with } 1 \leq u_i \leq t$$



(a) $\mathcal{G}_1^\star$



(b) $\mathcal{G}_2^\star$

Figure 4.7: Information flow graphs for which bounds in Equations (4.2) *are matched with equality for some* u.

We now prove the optimality of such constraints by contradiction. Let us assume that the constraints are not optimal. The minimal value that satisfies all the aforementioned constraint is (α, β, β'). It means that there exists some $(\bar{\alpha}, \bar{\beta}, \bar{\beta}') < (\alpha, \beta, \beta')$ which do not satisfy one of the constraints but for which the code is valid.

Furthermore, for all u such that $\sum_{i=0}^{g-1} u_i = k$ (i.e., for each of the constraints), there exist a graph $\mathcal{G}$ such that the corresponding constraint is matched with equality. Figure 4.7 gives two such examples when $\forall i, u_i = t$ (Fig. 4.7a) and $\forall i, u_i = 1$ (Fig. 4.7b). More generally, such graphs are constructed in the following way:

- The data collector contacts a set S of k devices and get all information from them.
- The contacted devices repaired simultaneously are grouped in subset S_i of size u_i such that $S = \bigcup_{i=0}^{g-1} S_i$.
- Each device $x \in S_i$ gets β information from all devices in $\bigcup_{j=0}^{i-1} S_j$, β' from $u_i - 1$ devices taking part to the reconstruction, β from $d - \sum_{j=0}^{i-1} u_j$ devices not in S, and β' from $t - u_i$ devices not taking part to the reconstruction.

If $(\bar{\alpha}, \bar{\beta}, \bar{\beta}') < (\alpha, \beta, \beta')$ (i.e., some constraints are not matched), then for the corresponding graphs $\mathcal{G}$, the repair is invalid. Hence, there cannot exist a repair with $(\bar{\alpha}, \bar{\beta}, \bar{\beta}') < (\alpha, \beta, \beta')$. A code minimizing (α, β, β') under Constraints (4.2) is optimal. □

4.3.4 Optimal tradeoffs

Determining the optimal tradeoffs comes down to minimizing storage cost α and repair cost γ, as defined by Equation (4.1), under constraints of Equation (4.2). k, d and t are constants and α, β and β' are the parameters to be optimized. In this subsection, we provide optimal tradeoffs (α, γ) between storage cost and repair cost (bandwidth).

MSCR codes

Minimum Storage Coordinated Regenerating Codes correspond to optimal codes that provide the lowest possible storage cost α while minimizing the repair cost γ. Figure 4.8 compares MSCR codes to both Dimakis *et al.*'s MSR [32] and erasure correcting codes with delayed repairs (ECC). Note that for $d = k$, our codes share the same repair cost as erasure correcting codes with delayed repair. Yet, in this case, our codes still have the advantage that they balance the load evenly thus avoiding bottlenecks.

$$\alpha = \frac{\mathcal{M}}{k} \qquad \beta = \frac{\mathcal{M}}{k} \frac{1}{d-k+t} \qquad \beta' = \frac{\mathcal{M}}{k} \frac{1}{d-k+t}$$

We determine the values for MSCR codes in two step. We study two particular cuts and find the minimal values required to ensure that the quantity of information going through the cuts is at least equal to the file size (thus proving the optimality of the solution if it is correct). We then prove that these quantities are sufficient for all possible cuts.

Proof of MSCR (Optimality). Let us consider two particular successions of repairs leading to the graphs shown on Figure 4.7. The repairs corresponding to such graphs are described in the Proof of Theorem 1.

We minimize α first. It is clear that $\alpha = \frac{\mathcal{M}}{k}$ is minimal since $\alpha < \frac{\mathcal{M}}{k}$ makes impossible to reconstruct a file of size $\mathcal{M}$ using only k blocks. Hence, what is important is now that every element of the sum is at least equal to $\frac{\mathcal{M}}{k}$.

$$\forall i \in 0 \ldots g-1, (d - \sum_{j=0}^{i-1} u_i)\beta + (t - u_i)\beta' \geq \frac{\mathcal{M}}{k}$$

When $\forall i, u_i = t$ (Fig. 4.7a), it is required that

$$\forall i \in 0 \ldots k/t-1, (d - \sum_{j=0}^{i-1} t)\beta \geq \frac{\mathcal{M}}{k}$$

which is equivalent to

$$\beta \geq \frac{\mathcal{M}}{k}\frac{1}{d-k+t}$$

When $\forall i, u_i = 1$ (Fig. 4.7b), it is required that

$$\forall i \in 0 \ldots k-1, (d - \sum_{j=0}^{i-1} 1)\beta + (t-1)\beta' \geq \frac{\mathcal{M}}{k}$$

which is equivalent to

$$\beta' \geq \frac{1}{(t-1)}(\frac{\mathcal{M}}{k} - \beta(d-k+1))$$

If we consider the smallest possible value $\beta' = \frac{1}{(t-1)}(\frac{\mathcal{M}}{k} - \beta(d-k+1))$, the associated repair cost is $\gamma = \frac{\mathcal{M}}{k} + (k-1)\beta$. The repair cost hence grows linearly with β. Hence, we should minimize β. The minimum value for β is $\beta = \frac{\mathcal{M}}{k}\frac{2}{2d-k+t}$.

Since lower values would lead to incorrect repairs for the graphs depicted on Figure 4.7, these values are optimal. □

Proof of MSCR (Correctness). It consists in proving that

$$\sum_{i=0}^{g-1} u_i \min\{\alpha, (d - \sum_{j=0}^{i-1} u_j)\beta + (t-u_i)\beta'\} \geq \mathcal{M}$$

is always verified when α, β and β' take the aforementioned values. The rest of the proof is similar to the Proof of Theorem 7 (Correctness) given in the next section. □

MBCR codes

Minimum Bandwidth Coordinated Regenerating Codes correspond to optimal codes that provide the lowest possible repair cost (bandwidth consumption) γ while minimizing the storage cost α. Figure 4.9 compares MBCR codes to both Dimakis *et al.*'s MBR [32] and erasure correcting codes with delayed repairs (ECC). Note that similarly to MBR [32], there is no *expansion* since every bit received is stored (i.e., $\alpha = \gamma$).

$$\alpha = \frac{\mathcal{M}}{k}\frac{2d+t-1}{2d-k+t}$$

$$\beta = \frac{\mathcal{M}}{k}\frac{2}{2d-k+t} \qquad \beta' = \frac{\mathcal{M}}{k}\frac{1}{2d-k+t}$$

We determine the values for MBCR codes in two step. We study two particular cuts and find the minimal values required to ensure that the quantity of information going through the cuts is at least equal to the file size (thus proving the optimality of the solution if it is correct). We then prove that these quantities are sufficient for all possible cuts.

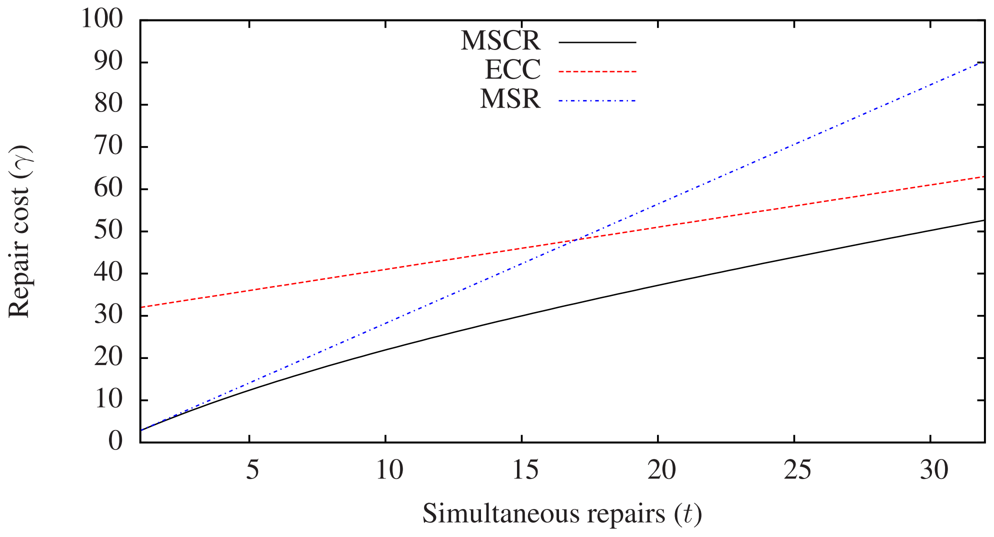


Figure 4.8: Total repair cost $t\gamma$ for $d = 48$ and $k = 32$. MSCR codes permanently outperform both erasure correcting codes and regenerating codes

Proof of MBCR (Optimality). Let us consider two particular successions of repairs leading to the graphs shown on Figure 4.7. The repairs corresponding to such graphs are described in the Proof of Theorem 1. As we want to minimize γ before α, we assume $\alpha \geq d\beta + (t-1)\beta'$ (i.e., the stored amount is always larger than the downloaded amount).

When $\forall i, u_i = t$ (Fig. 4.7a), it is required that

$$\sum_{i=0}^{k/t-1} t\left((d - \sum_{j=0}^{i-1} t)\beta\right) \geq \mathcal{M}$$

which is equivalent to

$$\beta \geq \frac{\mathcal{M}}{k} \frac{2}{2d - k + t}$$

When $\forall i, u_i = 1$ (Fig. 4.7b), it is required that

$$\sum_{i=0}^{k-1} \left((d - \sum_{j=0}^{i-1} 1)\beta + (t-1)\beta'\right) \geq \mathcal{M}$$

which is equivalent to

$$\beta' \geq \frac{1}{t-1}\left(\frac{\mathcal{M}}{k} - \beta\frac{2d - k + 1}{2}\right)$$

If we consider the smallest possible value $\beta' = \frac{1}{t-1}\left(\frac{\mathcal{M}}{k} - \beta\frac{2d-k+1}{2}\right)$, the associated repair cost is $\gamma = \frac{M}{k} + \frac{k-1}{2}\beta$. The repair cost hence grows linearly with β. Hence, we should minimize β. The minimum value for β is $\beta = \frac{\mathcal{M}}{k}\frac{2}{2d-k+t}$.

Since lower values would lead to incorrect repairs for the graphs depicted on Figure 4.7, these values are optimal. □

Proof of MBCR (Correctness). We have proved that the aforementionned values are required for two particular cuts. We now prove that such values ensure that enough information flow through every cut and, hence, is such code correct. According to Theorem 1, the following condition is sufficient for the code to be correct. We show that the constraint is satisfied when α, β and β' take the values defined for MBCR codes.

$$\sum_{i=0}^{g-1}\left(u_i \min\{(d-\sum_{j=0}^{i-1}u_j)\beta + (t-u_i)\beta', \alpha\}\right) \geq \mathcal{M}$$

since α (the stored part) is always larger than or equal to the transmitted data,

$$\sum_{i=0}^{g-1}u_i\left((d-\sum_{j=0}^{i-1}u_j)\beta + (t-u_i)\beta'\right) \geq \mathcal{M}$$

replacing α, β and β' by their values,

$$\sum_{i=0}^{g-1}u_i\left((d-\sum_{j=0}^{i-1}u_j)2 + (t-u_i)\right) \geq k(2d-k+t)$$

which is equivalent to,

$$(2d+t)\sum_{i=0}^{g-1}u_i - 2\sum_{i=0}^{g-1}u_i\sum_{j=0}^{i-1}u_j - \sum_{i=0}^{g-1}u_i^2 \geq k(2d-k+t)$$

$$(2d+t)\sum_{i=0}^{g-1}u_i - \left(\sum_{i=0}^{g-1}u_i\right)^2 \geq k(2d-k+t)$$

as $k = \sum_{i=0}^{g-1}u_i$, it simplifies to

$$(2d+t)k - k^2 \geq k(2d-k+t)$$

which is always true. Hence, minimum bandwidth regenerating codes are correct. □

General CR codes

The general case of Coordinated Regenerating Codes corresponds to all possible trade-offs in between MSCR and MBCR. Valid points (α, β, β') can be determined by performing a numerical optimization of the objective function. Figure 4.10 shows the optimal tradeoffs (α, γ): coordinated regenerating codes $(t > 1)$ can go beyond the optimal tradeoffs for independent repairs $(t = 1)$ defined by regenerating codes by Dimakis *et al.* [32].

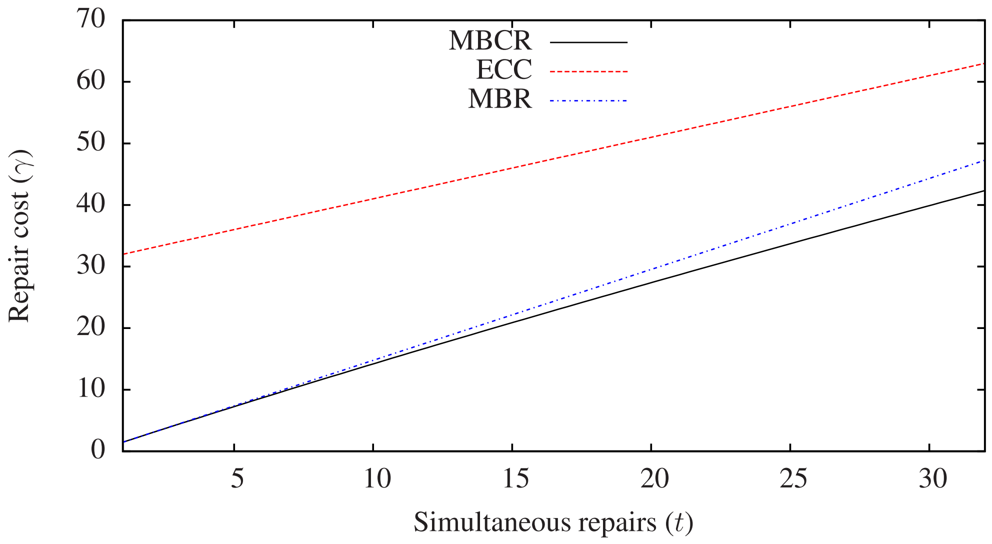


Figure 4.9: Total repair cost $t\gamma$ for $d = 48$ and $k = 32$. MBCR codes permanently outperform both erasure correcting codes and regenerating codes

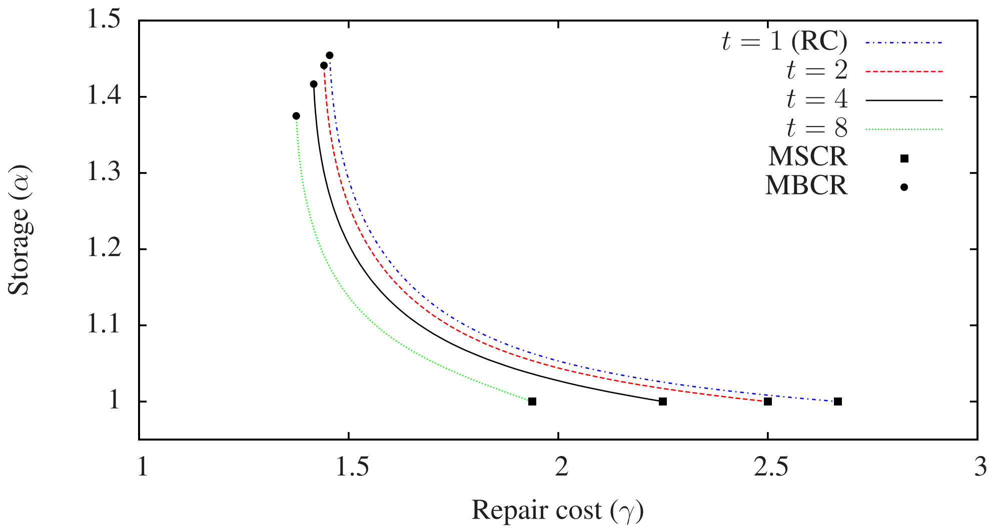


Figure 4.10: Optimal tradeoffs between storage and repair costs for $k = 16$ and $d = 24$. Regenerating codes (RC) [32] are depicted as $t = 1$. For each t, both MSCR and MBCR are shown. Costs are normalized by $\mathcal{M}/k$.

4.3.5 Optimal threshold

As previously explained, in regenerating codes, the higher number of devices being contacted d, the higher the savings on the repair cost γ. Moreover, when repairs are delayed, higher values for the number of devices being repaired t lead to higher savings on the repair cost γ. If we consider a system of constant size $n = d + t$, these two objectives are contradictory: the longer the delay, the lower the number of live devices d. An interesting question is what is the optimal threshold t for triggering repairs considering $d+t$ to be constant (i.e., is it useful to deliberately delay repairs?). This question is addressed hereafter by studying how MBCR codes and MSCR codes behave as t changes in a system of constant size.

Theorem 5. *If we consider a system of size $n = d + t$, for MBCR codes, the optimal value is $t = 1$ while for MSCR codes any value $t \in \{1 \dots n-k\}$ is optimal.*

Proof. Let us consider the repair cost assuming that $n = d + t$ is constant. For MBCR codes, the cost $\gamma = \frac{\mathcal{M}}{k}\frac{2n-t-1}{2n-k-t}$ increases when t increases. The optimal value of t for MBCR codes is the lowest possible value (i.e., $t = 1$). For MSCR codes, the cost $\gamma = \frac{\mathcal{M}}{k}\frac{n-1}{n-k}$ does not depend on t. The repair cost of MSCR remains constant, and t can be set to any value as there is no optimum. Neither MSCR nor MBCR allow additional gains when deliberately delaying repairs (i.e., deliberately setting $t > 1$). □

Corollary 6. *If we consider a system of size $n = d + t$ where t can be freely chosen (i.e., the value of t is not constrained by the system) both MSR and MBR regenerating codes [32] are optimal. Hence deliberately delaying repairs to force high values for t does not bring additional savings.*

However, if several failures are detected simultaneously, coordinated regenerating codes remain more efficient as they leverage simultaneous failures by coordinating during repairs.

4.4 Adaptive Regenerating Codes

In the previous section, we presented coordinated regenerating codes that assume t and d to remain constant across repairs. This is similar to regenerating codes [32], in which d remains constant across repairs. Yet, in real systems, it is not realistic to assume that the failure rate remains constant over time, and to assume that every single failure can be repaired before a second one occurs.

In the particular case of Minimum Storage ($\alpha = \frac{\mathcal{M}}{k}$), such strong assumptions are not needed. Indeed, when minimizing the sum in Equation (4.2), we can minimize the different elements of the sum, which correspond to repairs, independently. Therefore, repairs are independent. We propose to adapt the quantities to transfer β and β' to the system state which is defined by the number t of devices being repaired and the number d of live devices.

Adaptive regenerating codes simplify the design of a system based on regenerating codes. Indeed, when designing such a system, at least two parameters must be fixed: the number k of devices needed to recover the file and the number d of devices needed to repair a file. The higher the d is, the lower the repair cost is. Hence, d should be chosen as high as possible. Yet, if less than d devices are alive, regenerating codes cannot perform the repair, and the only proven but costly way to repair is to decode (gathering the whole file). Therefore, d should be fixed to reasonably low value (sub-optimal) so that there are always at least d live devices. On the contrary, adaptive regenerating codes self-adapt by choosing, at each repair, the highest possible d (the lowest repair cost γ is achieved for the highest d). Hence, designing a system using adaptive regenerating codes implies setting only the right value for the parameter k thus leading to simpler designs.

4.4.1 Our approach

Theorem 7. *Adaptive regenerating codes (k, Γ) are both correct and optimal. Γ is a function $(t, d) \rightarrow (\beta_{t,d}, \beta'_{t,d})$ that maps a particular repair setting to the amounts of information to be transferred during a repair.*

$$\beta_{t,d} = \frac{\mathcal{M}}{k}\frac{1}{d-k+t} \qquad\qquad \beta'_{t,d} = \frac{\mathcal{M}}{k}\frac{1}{d-k+t} \tag{4.4}$$

In this subsection, we prove they are correct and optimal.

Lemma 8. *For any information flow graph $\mathcal{G}$ compounded of initial devices that obtain α bits directly from the source* S *and of additional devices that join the graph in groups of t_i devices obtaining β_{t_i,d_i} from d_i existing devices and β'_{t_i,d_i} from each of the other $t_i - 1$ joining devices, any data collector* DC *that connects to a subset of k out-nodes (g groups of u_i out-nodes) of $\mathcal{G}$ has a* $\mathrm{mincut}(\mathrm{S}, \mathrm{DC})$ *greater or equal to:*

$$\sum_{i=0}^{g-1} u_i \min\{(d_i - \sum_{j=0}^{i-1} u_j)\beta_{t_i,d_i} + (t_i - u_i)\beta'_{t_i,d_i}, \frac{\mathcal{M}}{k}\}$$

where g is the number of different groups contacted, u_i is the size of each group, and $k = \sum_{i=0}^{g-1} u_i$ with $1 \leq u_i \leq t_i$.

Proof. The proof is similar to the proof of Lemma 4. □

Proof of Theorem 7 (Correctness). Using Lemmas 2, 3 and 8, we can define the following sufficient condition for the code to be correct. The constraint is satisfied when β and β' take the values defined in Equations (4.4).

$$\sum_{i=0}^{g-1} u_i \min\{(d_i - \sum_{j=0}^{i-1} u_j)\beta_{t_i,d_i} + (t_i - u_i)\beta'_{t_i,d_i}, \frac{\mathcal{M}}{k}\} \geq \mathcal{M}$$

Since each element of the sum is at most $u_i\frac{\mathcal{M}}{k}$, each element of the sum must satisfy the following constraint.

$$\forall i < g, \min\{(d_i - \sum_{j=0}^{i-1} u_j)\beta_{t_i,d_i} + (t_i - u_i)\beta'_{t_i,d_i}, \frac{\mathcal{M}}{k}\} \geq \frac{\mathcal{M}}{k}$$

which simplifies to

$$\forall i < g,\ (d_i - \sum_{j=0}^{i-1} u_j)\beta_{t_i,d_i} + (t_i - u_i)\beta'_{t_i,d_i} \geq \frac{\mathcal{M}}{k}$$

Applying formulas of Equations (4.4),

$$\forall i < g,\ \frac{1}{d_i - k + t_i}(d_i - \sum_{j=0}^{i-1} u_j + (t_i - u_i)) \geq 1$$

which is satisfied if $\forall i \leq g-1,\ \sum_{j=0}^{i} u_j \leq k$, which is true since $\sum_{j=0}^{g-1} u_j = k$ and $u_j > 0$. Therefore, adaptive regenerating codes are correct. □

Proof of Theorem 7 (Optimality). We prove by contradiction that the adaptive regenerating codes are optimal. Let us assume that there exists a correct code $(k, \bar{\Gamma})$ such that $\bar{\Gamma} < \Gamma$ (i.e., for some (t,d), $\bar{\Gamma}(t,d) < \Gamma(t,d)$). This is equivalent to (k, Γ) not being optimal.

Consider a set of failures such that all repairs are performed by groups of t devices downloading data from d devices. Consider the corresponding information flow graph. Assuming repairs are performed with a correct code $(k, \bar{\Gamma})$, the information flow graph also corresponds to a correct code $(d+t, k, t, d, \alpha, \bar{\beta}_{t,d}, \bar{\beta}'_{t,d})$.

Moreover, according to the previous section, these failures can be repaired optimally using the MSCR code $(d+t, k, t, d, \alpha, \beta_{t,d}, \beta'_{t,d})$. Therefore, there is a contradiction since the code $(d+t, k, t, d, \alpha, \bar{\beta}_{t,d}, \bar{\beta}'_{t,d})$ cannot be correct if the code $(d+t, k, t, d, \alpha, \beta_{t,d}, \beta'_{t,d})$ is optimal. A correct code $(k, \bar{\Gamma})$ cannot exist, and the adaptive regenerating code (k, Γ) defined in this section is optimal. □

Building on results from coordinated regenerating codes (especially MSCR), we have defined adaptive regenerating codes and proved that they are both correct and optimal. These codes are of particular interest for dynamic systems where failures may occur randomly and simultaneously.

4.4.2 Strawman approach

For the purpose of comparison, we define a strawman approach to the problem of adaptive repair. This approach would be to build, from MSR codes defined by Dimakis *et al.* in [32], and using a similar approach, adaptive regenerating codes without delayed repairs (k, Γ') where Γ' is a function $d \to \beta_d$. The t repairs needed are performed independently.

$$\beta_d = \frac{\mathcal{M}}{k} \frac{1}{d-k+1} \tag{4.5}$$

Proof. These codes are correct. A possible proof relies on observing that they are a particular sub-case of our approach described above ($t = 1$). □

4.4.3 Performance

We can compare our approach to the strawman approach in the particular case where $d + t = n$. The average cost per repair of our codes remains constant $\gamma = \frac{\mathcal{M}}{k}\frac{n-1}{n-k}$. In the strawman approach, which requires repairs to be performed independently, the average repair cost $\gamma' = \frac{\mathcal{M}}{k}\frac{n-t}{n-t-k+1}$ increases with t. Therefore, the performance of our adaptive regenerating codes does not degrade as the number of failures increases, as opposed to the simple construction upon Dimakis *et al.* 's codes. This is also shown on Figure 4.11.

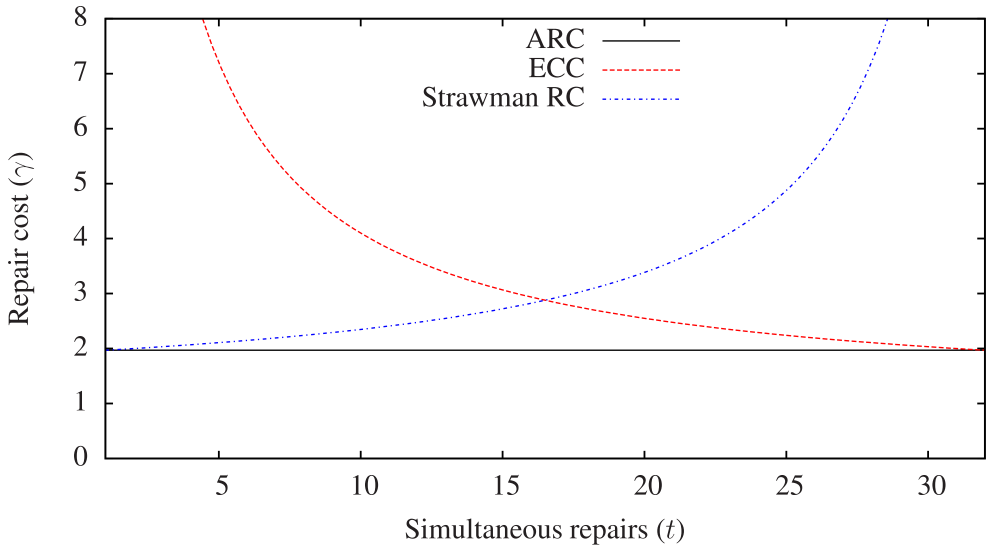


Figure 4.11: Average repair cost γ for $n = 64$ and $k = 32$. Adaptive Regenerating Codes (ARC) permanently outperform both erasure correcting codes (ECC) and the strawman approach based upon regenerating codes

4.4.4 Implementation

Our approach also has significant advantages over the strawman approach with respect to the implementation. The implementations are similar in principle to the one described in Subsection 4.3.1 and Figure 4.5. The only difference is that the values d and t may differ from one repair to the other. Each device stores sub-blocks of data and combines them to send the appropriate quantities of information. To be able to send $\beta = \frac{2}{3}\frac{\mathcal{M}}{k}$, each device must store $z = 3$ sub-blocks. To be able to send $\beta = \frac{1}{3}\frac{\mathcal{M}}{k}$ or $\beta = \frac{1}{4}\frac{\mathcal{M}}{k}$, each device must store $z = \text{lcm}(\{3, 4\}) = 12$ sub-blocks. Hence, the length[1] of any random linear code used to implement such a system is $l = zk$ where z is the

[1] In previous chapters, the code length was k. In this chapter, k is used for the number of devices needed to recover : we wanted to stay coherent with previously published work [32]. Hence, the few occurrences of the code length in this chapter are designed as l.

number of sub-blocks stored by each device. We now consider a system of constant size $n = d + t$ and compare both implementations.

The implementation of the strawman approach implies that to support $d \in \{k \dots n-1\}$, each device must be able to send all quantities $\beta \in \{\frac{1}{1}\frac{\mathcal{M}}{k} \dots \frac{1}{n-k}\frac{\mathcal{M}}{k}\}$. Hence, $z = \mathrm{lcm}(\{1 \dots n-k\})$. It can be shown that $2^{n-k} \leq z \leq 3^{n-k}$. Hence, the length of the codes required to implement such codes grows exponentially with $n-k$.

The implementation of our approach implies that to support $d \in \{k \dots n-1\}$, each device must be able to send quantities $\beta = \frac{1}{n-k}\frac{\mathcal{M}}{k}$ and $\beta' = \frac{1}{n-k}\frac{\mathcal{M}}{k}$. Hence, $z = n-k$. Hence, the length of the codes required to implement such codes grows linearly with $n-k$.

Even though simple adaptive regenerating codes can be built from Dimakis *et al.*'s MSR codes, our adaptive regenerating codes have two clear advantages. First, their repair cost is always lower than any other approach (i.e., optimal) and is constant on systems of constant size. Second, an optimal implementation of our adaptive regenerating codes requires much shorter codes (i.e. smaller l). This is very important since l has a direct impact on the complexity of all operations (encoding, recoding, and decoding).

4.5 Related Work

Let us examine how the new regenerating codes we define compare to the various approaches already mentionned in Chapter 1 (Section 1.2.2). As already explained, our coordinated and adaptive regenerating codes, which support performing optimally multiple simultaneous repairs, outperform both regenerating codes [32] and erasure correcting codes with lazy repairs [7, 29, 30]. In Chapter 1, we also mentioned a third approach aiming at reducing the repair costs by contacting less devices while still downloading the same amount from each device [36, 75]. Such approach requires structuring the code so that encoded packets can be recovered without decoding by combining a few other encoded packets. Even though this approach offer a reasonably low repair cost, it is neither optimal in storage nor optimal in repair cost. Moreover, these studies provide storage and repair costs that apply only to the specific codes proposed. Applying a methodology similar to ours (i.e., reasoning about information without considering the underlying code structure) could allow determining the fundamental tradeoffs between storage and repairs costs that can be achieved using such constructions.

Regenerating codes [32] and coordinated regenerating codes presented in this chapter assume a symmetric role for all devices (i.e., they all transfer the same amounts of information). Since network connections between every device may not be equivalent, it is interesting to adapt the repair strategy to take into account the underlying network topology. A first study [59] has focused on structuring the repair as a tree instead of a star. Indeed, with regenerating codes, repairs are performed by having the repaired device download directly from live devices. In [59], a first live device can receive information from a second live device so as to forward this information to the repaired device. This is of interest in systems where devices are connected in a mesh network and cannot contact all other devices directly. In such case, this can avoid a potential bottleneck link between a live device and the repaired device. However, in our study, we assumed the most frequent case where all devices are connected to the Internet/to a regular network thus allowing direct connection from one device to all other devices. Another study [91] has focused on downloading unequal amounts of information from other devices during repairs. They define the total amount of information that must be downloaded depending on the maximum amount of information that can be downloaded from each device. They show that the lowest repair cost is offered when all devices download the same amount of data (i.e., regular regenerating codes). This last study can also be applied to our codes and would show that allowing unequal downloads (i.e., non symmetric system) would also increase the global repair cost.

Independently from our result, the work [50] addresses a subset of the problem we consider. They notice that regenerating codes can only repair single failures and come up with a solution that can handle multiple failures. They naturally define a similar repair method (i.e., they add a coordination step to the information flow graph). Yet, their solution is much more limited than ours as they only study the Minimum Storage case (MSR). Not only, we also study the Minimum Bandwidth (MBR) point, but this cannot be covered by their model since they assume all transfers are equal (i.e., $\beta = \beta'$). Finally, we also determine numerically the general case (i.e., points between Minimum Storage (MSR) and Minimum Bandwidth (MBR) points). Their paper is also restrictive with respect to system they consider as, they assume a system of constant size where all devices are involved (i.e., $n = d + t$). Finally, we do build upon our result to define an Adaptive form of Regenerating Codes that is more flexible to use in practical systems while they do not consider such constructions. Hence, the previously published paper [50], which is yet another proof of the importance of the considered problem, covers only a subset of our results even if it shares both the problem and some tools used (an adaptation of Information Flow Graphs from Dimakis et al. [31, 32]) .

4.6 Summary

In this chapter, we have proposed novel and *optimal coordinated regenerating codes* by considering simultaneous repairs in regenerating codes. This work has both theoretical and practical impacts. From a theoretical standpoint, we proved that deliberately delaying repairs cannot provide further gain in term of communication costs. In practical systems, however, where several failures are detected simultaneously, our coordinated regenerating codes outperform regenerating codes [32] and are optimal. We also proposed *adaptive regenerating codes* that allow adapting the repair strategy to the current state of the system so that it always performs repairs optimally. For both codes, we provided and proved the minimum amounts of information that need to be transfered during the repair thus defining the fundamental tradeoffs between storage and repair costs for coordinately repairing multiple failures. Finally, our coordinated regenerating codes can also be viewed as a global class of codes that encompass erasure correcting codes with delayed repairs ($d = k$), regenerating codes ($t = 1$), erasure correcting codes ($t = 1$ and $d = k$), and new codes ($t > 1$ and $d > k$) that combine, previously incompatible, existing approaches of regenerating codes and delayed repairs (cf. Figure 4.3).

Chapter 5

Conclusions and Perspectives

5.1 Context

Distributed systems are built upon unreliable and hardly predictable devices. Codes, be they network or channel codes, have been widely used to hide the apparent randomness of the underlying devices. Even though codes exhibit appealing performances with respect to transmission or storage, they are often left aside and replaced by less efficient but simpler mechanisms such as multiple transmissions and replication. The main arguments against codes are that their benefits are often overtaken by side costs introduced. In the case of dissemination, random linear network codes implies that the decoding must be performed using a complex algorithm. In the case of storage, the criticisms focus on the repair cost that is high when using erasure correcting codes.

5.2 Contributions

In this thesis, we considered various approaches for making codes more practical by limiting the side costs of codes. In a first part, we considered the use of low complexity codes in dissemination systems. In a second part, we considered the use of network codes to reduce the repair cost (with respect to communication) in distributed storage systems.

5.2.1 Codes in Dissemination

In the first part of this thesis, we considered the uses of codes in dissemination protocols. We focused on limiting the decoding costs associated with the use of codes.

We proposed a new protocol to best leverage fountain codes and a new kind of network codes relying on a low complexity decoding algorithm.

FoG, a biased protocol for dissemination using Fountain Codes

During a push-based epidemic dissemination, devices forward what they receive to other devices. In a first phase, the forwarded data tends to be always useful (i.e., it is not redundancy) as devices have not received the forwarded data. Yet, in a second phase, peers have received most data and the forwarded data tends to be useless since it is likely to have been already received. The overhead generated by the end of the dissemination of a packet is necessary to ensure that all peers receive the packet w.h.p.

In FoG, we propose to stop prematurely dissemination to avoid the overhead, and to add additional disseminations of redundancy data so as to ensure that all peers receive all data. FoG is designed to leverage fountain codes. Fountain codes are interesting because they can be implemented efficiently (Raptor codes or LT codes). Yet, in fountain codes, a device can encode only when having the whole information (i.e., it has received and decoded everything). Hence, a protocol relying on fountain codes must take into account the difference between peers that can encode and the others that can only foward. Since a peer is much more useful when encoding, in FoG, we introduce a bias toward some peers to ensure a non-uniform progression of peers. The peers that finish earlier can then help the other peers more efficiently by encoding instead of forwarding. Compared to a protocol that does not take into account the various roles, we achieve 30% faster dissemination while reducing the overhead by 50%.

LT Network Codes

In FoG, we considered only fountain codes for they can be implemented with low complexity encoding and decoding. However, the need to distinguish between devices that can encode and devices that can only forward implies that the dissemination protocol is rather complicated. Hence, in LT network codes, we consider a simple push-based epidemic dissemination protocol combined with a network coding method that use a low complexity decoding algorithm (belief propagation).

Our main contribution in LT network codes is to extend LT codes with a recoding method. Our recoding method preserves statistical properties of codes to allow decoding with belief propagation. We compare our approach, which can be decoded with a low complexity ($\mathcal{O}(mk \log k)$) to random linear network coding, which can be decoded using Gauss reduction ($\mathcal{O}(mk^2)$). As a result, the CPU cost for decoding is greatly reduced, by 99%. However, to allow such a reduction in complexity, LT network codes trade the communication efficiency (20% communication overhead) for reduced CPU costs.

5.2.2 Codes in Storage

In the last part of this thesis, we studied how network codes can be used in storage systems. The aim is to reduce the repair cost in terms of network communications. Network codes have allowed significant gains over regular erasure correcting codes with respect to the repair cost. By applying recent results of information theory to storage systems, Dimakis *et al.* [31, 32] define regenerating codes that achieve optimal tradeoffs between storage and repair costs. Yet, the regenerating codes defined are rather limited since they only support single failures and very static systems. We extend these codes to support multiple failures (Coordinated Regenerating Codes) and adaptive repairs (Adaptive Regenerating Codes).

Coordinated Regenerating Codes

Regenerating codes define the optimal (with respect to both storage cost and repair cost) way to repair single failures in distributed storage. However, if multiple repairs must be performed, they have to be performed independently. We define coordinated regenerating codes, having a lower repair cost than regenerating codes. We provide closed form expressions for the minimum storage coordinated regenerating codes and for the minimum bandwidth coordinated regenerating codes.

The ability to repair multiple failures optimally with coordinated regenerating codes makes lazy (i.e., delayed) repair schemes possible. In the context of erasure correcting codes, lazy repairs have allowed factoring some of the repair costs thus reducing such costs. It is natural to study the effect of lazy repairs on coordinated regenerating codes. We study a system of constant size and we show that, in the minimum storage case, performing lazy repairs has no effect on the repair cost. In the minimum bandwidth case, the efficiency of the repair is decreased when performing lazy repairs. Hence, we can conclude that regenerating codes [31, 32] are optimal even if we are able to deliberately delay repairs.

Adaptive Regenerating Codes

Moreover, regenerating codes [31, 32] assume that all successive repairs are performed under similar conditions (i.e., the number of live devices contacted and the number of devices being repaired remain constant across repairs during the whole lifetime of the system). We define adaptive regenerating codes that support changing the number of devices being repaired and the number of live devices contacted. Indeed, in the particular case of minimum storage coordinated regenerating codes, it can be shown that the assumptions about the static state of the system can be relaxed.

The resulting adaptive regenerating codes are interesting for implementing optimal schemes in practical systems. Adaptive regenerating codes can support very dynamic systems. The number of parameters that need to be defined is reduced: only k and n need to be fixed. Moreover, their repair cost is constant and lower than the one of an adaptive form of erasure correcting codes, or of a simple adaptive form of regenerating codes based upon regular regenerating codes [31,32]. Finally, adaptive regenerating codes allow simpler coding schemes as the size of blocks transfered remains constant and as they naturally balance the load evenly.

5.3 Perspectives

Our current results suggest various perspectives. First, low complexity codes specially designed for epidemic pull-based dissemination systems might offer interesting performances. Such codes could reuse some algorithms and data structures defined in Chapter 3. Second, network codes offer huge opportunities for reducing repair costs in distributed systems as explained in Chapter 4. Yet, the implementation aspect of coordinated or adaptive regenerating codes has not been considered and only randomized codes suffering a high complexity exist. Moreover, as our study provides bounds on the repair costs, it might be interesting to use these bounds on codes to define system-wide bounds (usable storage, mean time to failure…) on the service that can be offered given some resources (storage and bandwidth). In the rest of this section, we present in more details these perspectives.

Low density network coding for pull systems

Pull-based epidemic dissemination systems can request content (through a feedback channel). However, when using coding, in particular network coding, determining which block or which combination of blocks to request may become complicated. It implies determining an encoded packet that the sender can generate and that the receiver finds innovative. In the case of RLNC, this comes down to determining the orthogonal complement of the subspace corresponding to encoded packets of the receiver and then computing the projection of the subspace of the sender onto this orthogonal complement. Since it is a complex operation, network coding traditionally assume that a correctly recoded packet will be useful with high probability, thus not using the feedback channel and offering only probabilistic guarantees.

In Chapter 3 describing LT network codes, we defined efficient structures and algorithms for dealing with packets of degree 2, which represent almost half of transmitted packets in LT codes. The aforementioned data structure is used for two purposes. First, it allows to compute the exhaustive list of encoded packets of degree 2 that can be generated. These encoded packets are then used to perform the substitutions. Second, it also allows to determine (in constant time) whether a packet is useful or useless.

We believe that these two uses could be combined so as to build a system merging an efficient use of the feedback channel and network coding. While using efficiently the feedback channel can allow null overhead and efficient disseminations, using network coding increase the diversity of packets thus avoiding pathological case such as the appearance of rare blocks that can be downloaded from only very few peers. Building a system able to use both a simple network coding scheme limited to packets of degree 1 and 2, and a feedback channel may increase the performance of pull systems. This is of interest since the corresponding recoding and decoding algorithms have a low complexity thus offering an efficient hybrid scheme when a system can use a full feedback channel and a network coding scheme.

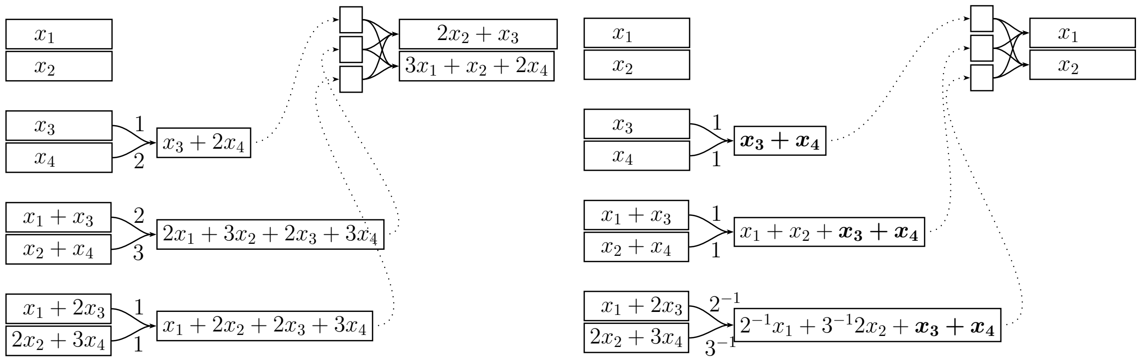
In Section 3.3.3 (Page 52), we give an overview of how the defined structure allows determining at a low complexity which combinations of packets are likely to be useful. As explained in Chapter 3, the structure describes the various connected components of native packets: two native packets x_i and x_j are in a same connected component if and only if $x_i \oplus x_j$ can be computed. Hence, if x_i and x_j are in a same connected component, receiving $x_i \oplus x_j$ is useless and $x_i \oplus x_j$ can be sent. The basic idea is, therefore, to use these structures to find out combinations of packets that are in the same connected component at the sender side and in different connected components at the receiver side. This allows using the feedback channel efficiently : only useful packets are sent (null overhead) and the complexity of a belief propagation algorithm for decoding such network codes is $\mathcal{O}(k)$.

Deterministic adaptive/coordinated regenerating codes

Our study of regenerating codes has focused on the theoretical framework for performing multiple coordinated repairs in distributed storage. A straightforward implementation of such codes relies on random linear network codes (cf. Figure 4.5 on page 70). Coordinated regenerating codes can therefore be used as they are for self-healing distributed storage. Yet, the seminal paper on regenerating codes [31] has been followed by studies of more constrained repairs (deterministic repairs known as exact repairs by opposition with functional repairs studied in Chapter 4). In short, in functional repairs (Figure 5.1a), the regenerated redundancy is not necessarily the same as the lost one while, in exact repairs (Figure 5.1b), the regenerated redundancy is exactly the same as the lost one. A recent survey [35] details the various works on exact repairs. Since our coordinated and adaptive regenerating codes only consider the problem of functional repair (i.e., they leave open the problem of coordinated or adaptive exact repairs), an interesting extension of this work would be to study deterministic coordinated or adaptive regenerating codes performing exact repairs. Indeed, codes with exact repairs would have appealing properties (simpler integrity checking, lower decoding complexity) when compared to coordinated and adaptive regenerating codes based upon random linear network codes.

Tradeoffs between storage and bandwidth at the system scale

Adaptive regenerating codes simplify the design of a storage system as only two parameters need to be chosen (n and k) and all repairs can be performed optimally. Yet, as initially observed by Dimakis *et al.* [32], there are interactions between the system and the parameters of the code. For example, let us consider a code of length k



(a) Functional (random) repair (b) Exact (deterministic) repair

Figure 5.1: Comparison of two kinds of repair on a simple regenerating code ($n = 4, k = 2, d = 3, \alpha = \mathcal{M}/2, \beta = \mathcal{M}/4$)

in a system of size n where each device fails and gets repaired with rate λ. We want to determine the value n that minimizes the repair cost. If we only consider the equation $\gamma = \frac{n-1}{n-k}$ defining the repair cost for the code, we find that increasing n decreases the repair cost. Yet, when increasing n, the rate of failures $n\lambda$ increases. Hence, as shown on Figure 5.2, increasing the system size beyond a certain value is useless and only consumes more space without providing any improvement with respect to the global repair cost. Indeed, there is an optimal value of $n_{\text{opt}} = \frac{k}{k-\sqrt{k^2-k}}$ which minimizes the global repair cost $n\lambda\gamma$ and such that increasing n above n_{opt} does not provide more gain at the system level even if the cost of each individual repair decreases. However, the optimal value of k is not known and it would be interesting to also optimize the mean time to failure instead of only optimizing the global repair cost.

Hence, using our optimal and adaptive regenerating codes, it would be interesting to determine fundamental tradeoffs between the global storage cost and the global repair cost. Ignoring metadata size (i.e, information describing which linear combinations of sub-blocks are exchanged), we can express the repair bandwidth of a system storing $\mathcal{M}$ bytes on devices having a failure rate λ. The best performance is achieved for large values of k. Hence, with an infinite k (infinite MTTF), the repair bandwidth is $\frac{\mathcal{M}\lambda}{\omega^{-1}(1-\omega^{-1})}$ where ω is the normalized storage cost (i.e., storing 1 GB costs ω GB). Let us consider an hypothetical case where devices only suffer permanent failures, where k is infinite and, where the headers do not add costs. Figure 5.3 gives hints about the requirements of bandwidth for running a distributed storage system with each device failing at the rate of $\lambda = 1$ failure per day. As illustrated by this example, such study, leveraging the fact that repair costs are constant with our adaptive regenerating codes, allows dimensioning a distributed storage system, and determining the amount of data that each device can store given some caps on bandwidth and available storage.

A more realistic study should consider temporary failures (thus avoiding some repairs), the additional costs associated with headers (even though they can be

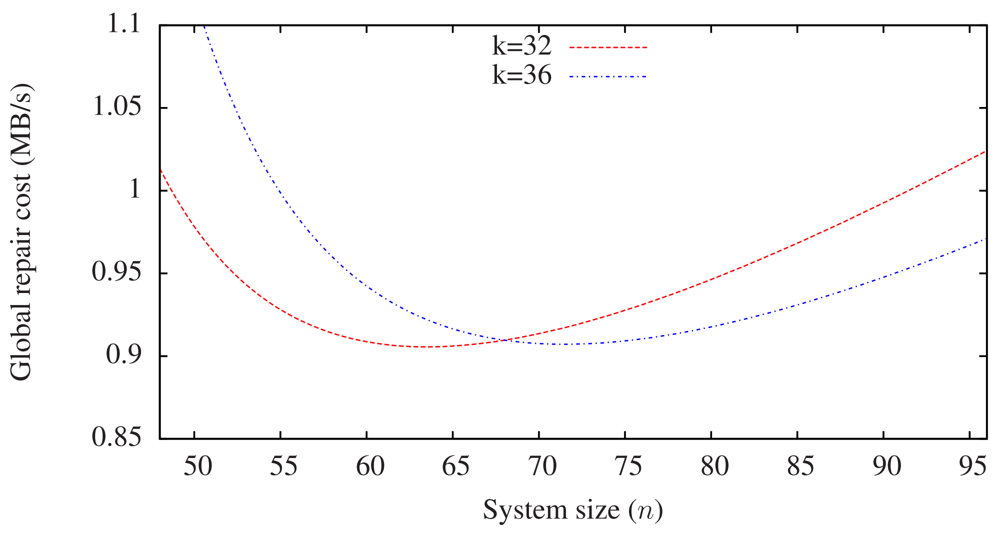


Figure 5.2: The system size n that provides the lower repair cost for some k is $n = \frac{k}{k-\sqrt{k^2-k}}$, which is $n \approx 2k$. The graph plots the storage of a file of size 20 GB and the failure rate of individual devices is $\lambda = 1$ per day.

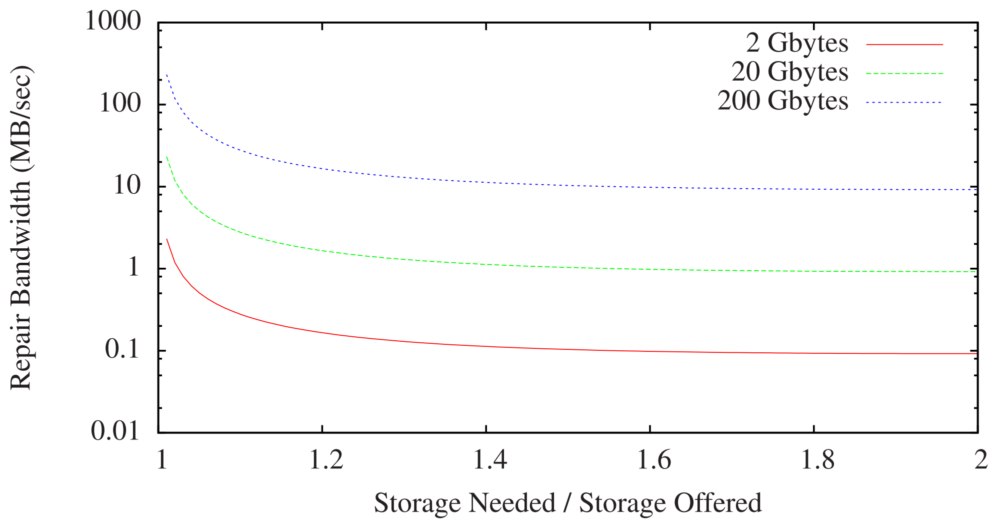


Figure 5.3: The graph plots the tradeoffs between bandwidth and repair cost for various file size. These tradeoffs show that if each user can afford to dedicate 40 GB of disk space and 0.9 MB/s , in a system with frequent failure ($\lambda = 1$ per day per device), the system can provide each user with a reliable storage of 20 GB.

neglected for large enough files), and finite values for k. Such a study would provide the system designer with the corresponding values n and k that offer the best performance given some caps on storage and bandwidth. Our adaptive regenerating codes are interesting for such a study as they are optimal, and as their performance does not depend on t (the number of simultaneous repairs), thus reducing the number of parameters to consider.

Low complexity adaptive/coordinated regenerating codes

As described in Figure 4.5 (page 70) and in Section 4.4.4 (page 85), implementing coordinated or adaptive regenerating codes requires splitting the file in k blocks which are then sub-divided in z sub-blocks. This allows devices to combine their z sub-blocks to send quantities of information that are multiple of $\frac{\mathcal{M}}{k}\frac{1}{z}$. Hence, the implementation of such regenerating codes requires linear network codes of length $l = zk$. The implementation of adaptive regenerating codes minimizing the global repair cost, as described in the previous sub-section ($\beta = \beta' = \frac{\mathcal{M}}{k}\frac{1}{n-k}$), hence requires a random linear network code of length $l = (n-k)k \approx k^2$ (as explained previously, the best performance is achieved when $n \approx 2k$). The corresponding size of data is $m \approx \frac{\mathcal{M}}{k^2}$.

The high code length l and the small block size m put a high stress upon the encoding, recoding, and decoding algorithms. Indeed, performing a Gauss reduction on such codes has a complexity of $\mathcal{O}(k^6)$ for the control (inverting matrix) and $\mathcal{O}(mk^4)$=$\mathcal{O}(Mk^2)$ for the data. Moreover, sending a sub-block of size $\frac{\mathcal{M}}{k}\frac{1}{z}$ implies combining (thus reading from disk) all z sub-blocks stored. It would be interesting to read only a subset of blocks thus avoiding the disks from being bottlenecks during repairs. Overall, reducing the encoding, recoding and decoding costs implies using sparse matrices. Yet, it is not clear whether regenerating codes can accommodate such sparse matrices. Hence, it would be interesting to study if there exist fundamental limits on the acceptable sparsity of recoding matrices. It would also be interesting to provide implementations of regenerating codes that can be decoded by low complexity algorithms such as belief propagation (similarly to what we did for dissemination in LTNC).

This thesis has addressed several problems and made contributions in coding for distributed systems. We also presented some interesting research directions towards more practical codes. In a first part of the thesis, we explored the use of low complexity fountain codes for dissemination systems and we proposed new low complexity network codes offering an interesting tradeoff between decoding costs and communication efficiency. In a second part, we focused on the benefits of using network codes in storage systems as they allow to achieve optimal tradeoffs between storage and repair costs. Finally, we proposed various perspectives around codes and distributed systems. The most interesting of these perspectives is probably the study of low complexity adaptive regenerating codes as they would allow significant gain in distributed storage systems.

Bibliography

[1] Szymon Acedański, Supratim Deb, Muriel Médard, and Ralf Koetter. How good is random linear coding based distributed networked storage? In *NetCod*, 2005.

[2] Rudolf Ahlswede, Ning Cai, Shuo-Yen Li, and Raymond W. Yeung. Network Information Flow. *IEEE Transactions On Information Theory*, 46(4):1204–1216, Jul. 2000.

[3] S.A. Aly, Zhenning Kong, and E. Soljanin. Raptor Codes Based Distributed Storage Algorithms for Wireless Sensor Networks. In *ISIT*, pages 2051–2055, 2008.

[4] Salah A. Aly, Zhenning Kong, and Emina Soljanin. Fountain Codes Based Distributed Storage Algorithms for Large-Scale Wireless Sensor Networks. In *IPSN*, pages 171–182, 2008.

[5] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable Data Possession at Untrusted Stores. In *CCS*, 2007.

[6] Jørgen Bang-Jensen and Gregory Gutin. *Digraphs: Theory, Algorithms and Applications.* Springer-Verlag, 2001.

[7] Ranjita Bhagwan, Kiran Tati, Yu-Chung Cheng, Stefan Savage, and Geoffrey M. Voelker. Total Recall: System Support for Automated Availability Management. In *NSDI*, pages 337–350, 2004.

[8] Charles Blake and Rodrigo Rodrigues. High Availability, Scalable Storage, Dynamic Peer Networks: Pick Two. In *HotOS*, pages 1–6, 2003.

[9] Johannes Blöemer, Malik Kalfane, Marek Karpinski, Richard Karp, Michael Luby, and David Zuckerman. An XOR-Based Erasure-Resilient Coding Scheme. Technical Report TR-95-048, ICSI, 1995.

[10] William J. Bolosky, John R. Douceur, David Ely, and Marvin Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. In *SIGMETRICS*, 2000.

[11] Thomas Bonald, Laurent Massoulié, Fabien Mathieu, Diego Perino, and Andrew Twigg. Epidemic Live Streaming: Optimal Performance Trade-Offs. In *SIGMETRICS*, pages 325–336, 2008.

[12] John Byers, Jeffrey Considine, Michael Mitzenmacher, and Stanislav Rost. Informed Content Delivery Across Adaptive Overlay Networks. *IEEE/ACM Transactions on Networking*, 12:767–780, 2004.

[13] John Byers, Michael Luby, and Michael Mitzenmacher. A Digital Fountain Approach to Asynchronous Reliable Multicast. *IEEE JSAC, Special Issue on Network Support for Multicast Communication*, 20(8):1528–1540, 2002.

[14] John W. Byers, Jeffrey Considine, Michael Mitzenmacher, and Stanislav Rost. Informed Content Delivery Across Adaptive Overlay Networks. *IEEE/ACM Transactions on Networking*, 12(5):767–780, 2004.

[15] Niklas Carlsson and Derek L. Eager. Peer-assisted On Demand Streaming of Stored Media using BitTorrent-like Protocols. In *IFIP/Networking*, 2007.

[16] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. SplitStream: High-Bandwidth Multicast in Cooperative Environments. In *SOSP*, 2003.

[17] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE JSAC, Special issue on Network Support for Multicast Communications*, 20:1489–1499, 2002.

[18] Mary-Luc Champel, Kévin Huguenin, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. Lt network codes: Low complexity network codes (short paper). In *ACM CoNEXT'09 (Student Workshop): 5th ACM International Conference on emerging Networking EXperiments and Technologies*, 2009.

[19] Mary-Luc Champel, Kévin Huguenin, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. Lt network codes. In *IEEE ICDCS'2010: 30th International Conference on Distributed Computing Systems*, 2010.

[20] Mary-Luc Champel, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. Fog : fighting the achilles' heel of gossip protocols with fountain codes. In *SSS'2009: 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, 2009.

[21] Mary-Luc Champel, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. Phosphite : Guaranteeing out-of-order download in p2p video on demand. In *P2P'2009: 9th IEEE International Conference on Peer-to-Peer Computing*, 2009.

[22] Mary-Luc Champel, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. Phosphite : Incitation à la collaboration pour la vidéo la demande en p2p. In *Algotel'2009: 11eme Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*, 2009.

[23] D. Charles, K. Jian, and K. Lauter. Signature for Network Coding. Technical Report MSR-TR-2005-159, Microsoft Research, 2005.

[24] Yung Ryn Choe, Derek L. Schuff, Jagadeesh M. Dyaberi, and Vijay S. Pai. Improving VoD Server Efficiency with BitTorrent. In *MM*, 2007.

[25] Byung-Gon Chun, Frank Dabek, Andreas Haeberlen, Emil Sit, Hakim Weatherspoon, Frans Kaashoek, John Kubiatowicz, and Robert Morris. Efficient Replica Maintenance for Distributed Storage Systems. In *NSDI*, pages 45–58, 2006.

[26] Bram Cohen. Incentives build robustness in BitTorrent. In *P2PEcon*, June 2003.

[27] Jeffrey Considine. Generating Good Degree Distributions for Sparse Parity Check Codes using Oracles. Technical Report BUCS-TR-2001-019, CS Department, Boston University, 2001.

[28] Frank Dabek, Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area Cooperative Storage with CFS. In *SOSP*, pages 202–215, 2001.

[29] Olivier Dalle, Frédéric Giroire, Julian Monteiro, and Stéphane Pérennes. Analysis of Failure Correlation Impact on Peer-to-Peer Storage Systems. In *P2P*, pages 184–193, 2009.

[30] Anwitaman Datta and Karl Aberer. Internet-scale storage systems under churn – A Study of steady-state using Markov models. In *P2P*, pages 133–144, 2006.

[31] Alexandros G. Dimakis, P. Brighten Godfrey, Yunnan Wu, Martin O. Wainwright, and Kannan Ramchandran. Network Coding for Distributed Storage Systems. In *INFOCOM*, pages 2000–2008, 2007.

[32] Alexandros G. Dimakis, P. Brighten Godfrey, Yunnan Wu, Martin O. Wainwright, and Kannan Ramchandran. Network Coding for Distributed Storage Systems. *IEEE Transactions On Information Theory*, 56:4539–4551, 2010.

[33] Alexandros G. Dimakis, Vinod Prabhakaran, and Kannan Ramchandran. Ubiquitous Access to Distributed Data in Large-Scale Sensor Networks Through Decentralized Erasure Codes. In *IPSN*, 2005.

[34] Alexandros G. Dimakis, Vinod Prabhakaran, and Kannan Ramchandran. Decentralized erasure codes for distributed networked storage. In *Joint special issue, IEEE/ACM Transactions on Networking and IEEE Transactions on Information Theory*, 2006.

[35] Alexandros G. Dimakis, Kannan Ramchandran, Yunnan Wu, and Changho Suh. A Survey on Network Codes for Distributed Storage. *CoRR*, abs/1004.4438, 2010.

[36] Alessandro Duminuco and Ernst Biersack. Hierarchical Codes: How to Make Erasure Codes Attractive for Peer-to-Peer Systems. In *P2P*, 2008.

[37] Alessandro Duminuco and Ernst Biersack. A Pratical Study of Regenerating Codes for Peer-to-Peer Backup Systems. In *ICDCS*, 2009.

[38] P. Th. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec. Lightweight Probabilistic Broadcast. *ACM Transactions on Computer Systems*, 21(4):341–374, 2003.

[39] Patrick T. Eugster, Rachid Guerraoui, Anne-Marie Kermarrec, and Laurent Massoulieacute;. Epidemic Information Dissemination in Distributed Systems. *Computer*, 37(5):60–67, 2004.

[40] Pascal Felber, Anne-Marie Kermarrec, Lorenzo Leonini, Etienne Rivière, and Spyros Voulgaris. Pulp: Un protocole épidémique hybride. In *Algotel*, 2009.

[41] Robert G. Gallager. Low Density Parity Check Codes. *Transactions of the IRE Professional Group on Information Theory*, IT-8:21–28, 1962.

[42] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System. In *SOSP*, pages 29–43, 2003.

[43] Christos Gkantsidis, John Miller, and Pablo Rodriguez. Anatomy of a P2P Content Distribution System with Network Coding. In *IPTPS*, pages 1–6, 2006.

[44] Christos Gkantsidis and Pablo Rodriguez. Network Coding for Large Scale Content Distribution. In *INFOCOM*, pages 2235–2245, 2005.

[45] Saikat Guha, Neil Daswani, and Ravi Jain. An Experimental Study of the Skype Peer-to-Peer VoIP System. In *IPTPS*, 2006.

[46] Ramakrishna Gummadi and Ramavarapu Sreenivas. Relaying a Fountain Code Accross Multiple Nodes. In *ITW*, pages 483–484, 2008.

[47] Richard W. Hamming. Error Detecting and Error Correcting Codes. *The Bell System Technical Journal*, 29, 1950.

[48] Tracey Ho, Ben Leong, Ralf Koetter, Muriel Médard, Michelle Effros, Michelle Effros, and David R. Karger. Byzantine Modification Detection in Multicast Networks using Randomized Network Coding. In *ISIT*, page 143, 2003.

[49] Tracey Ho, Muriel Médard, Ralf Koetter, David Karger, Michelle Effros, Jun Shi, and Ben Leong. A Random Linear Network Coding Approach to Multicast. *IEEE Transaction on Information Theory*, 52(10):4413–4430, October 2006.

[50] Yuchong Hu, Yinlong Xu, Xiaozhao Wang, Cheng Zhan, and Pei Li. Cooperative Recovery of Distributed Storage Systems from Multiple Losses with Network Coding. *IEEE Journal on Selected Areas in Communications*, 28:268–276, 2010.

[51] S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, and M. Médard. Resilient Network Coding in the Presence of Byzantine Adversaries. In *INFOCOM*, pages 616–624, 2007.

[52] Márk Jelasity, Spyros Voulgaris, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. Gossip-based peer sampling. *ACM Transactions on Computer Systems*, 25:8, 2007.

[53] Márk Jelasity, Alberto Montresor, Gian Paolo Jesi, and Spyros Voulgaris. The PeerSim Simulator. http://peersim.sourceforge.net/.

[54] Abhhinav Kamra, Vishal Misra, Jon Feldman, and Dan Rubenstein. Growth Codes: Maximizing Sensor Network Data Persistence. In *SIGCOMM*, pages 255–266, 2006.

[55] Richard Karp, Christian Schindelhauer, Scott Shenker, and Berthold Vöcking. Randomized Rumor Spreading. In *FOCS*, pages 566–574, 2000.

[56] Sachin Katti, Hariharan Rahul, Wenjun Hu, Dina Katabi, Muriel Médard, and Jon Crowcroft. XORs in the Air: Practical Wireless Network Coding. In *SIGCOMM*, 2006.

[57] Ralf Koetter and Muriel Médard. An algebraic approach to network coding. *IEEE/ACM Transactions on Networking*, 11:782–795, 2003.

[58] Harry C. Li, Allen Clement, Edmund L. Wong, Jeff Napper, Indrajit Roy, Lorenzo Alvisi, and Michael Dahlin. BAR Gossip. In *OSDI*, 2006.

[59] Jun Li, Shuang Yang, Xin Wang, and Baochun Li. Tree-structured Data Regeneration in Distributed Storage Systems with Regenerating Codes. In *INFOCOM*, 2010.

[60] Shuo-Yen Robert Li, Raymond W. Yeung, and Ning Cai. Linear Network Coding. *IEEE Transactions On Information Theory*, 49:371–381, 2003.

[61] Luisa Lima, Muriel Médard, and J. Barros. Random Linear Network Coding: A Free Cipher. In *ISIT*, 2007.

[62] W. K. Lin, D. M. Chiu, and Y. B. Lee. Erasure Code Replication Revisited. In *P2P*, 2004.

[63] Yunfeng Lin, Baochun Li, and Ben Liang. Differentiated Data Persistence with Priority Random Linear Codes. In *ICDCS*, 2007.

[64] Nathan Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21:193–201, 1992.

[65] Thomas Locher, Stefan Schmid, and Roger Wattenhofer. Rescuing Tit-for-Tat With Source Coding. In *P2P*, 2007.

[66] Michael Luby. LT Codes. In *FOCS*, 2002.

[67] Michael G. Luby, Michael Mitzenmacher, M. Amin Shokrollahi, Daniel A. Spielman, and Volker Stemann. Practical Loss-Resilient Codes. In *STOC*, 1997.

[68] Guanjun Ma, Yinlong Xu, Minghong Lin, and Ying Xuan. A Content Distribution System Based on Sparse Network Coding. In *NetCod*, 2007.

[69] David J.C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2002.

[70] Laurent Massoulié, A. Twigg, Christos Gkantsidis, and P. Rodriguez. Randomized decentralised broadcasting algorithms. In *INFOCOM*, 2007.

[71] Petar Maymounkov, Nicholas J. A. Harvey, and Desmond S. Lun. Methods for Efficient Network Coding. In *Allerton*, 2006.

[72] Robert J. McEliece. A Public-Key Cryptosystem Based on Algebraic Coding Theory. *JPL Deep Space Network Progress Report*, 42–44:90–97, 1978.

[73] Michael Mitzenmacher. Digital Fountains: A Survey and Look Forward. In *ITW*, 2004.

[74] Ramsés Morales and Idranil Gupta. AVMON: Optimal and Scalable Discovery of Consistant Availability Monitoring Overlays for Distributed Systems. *IEEE Transaction on Parallel and Distributed Systems*, 20:446–459, 2009.

[75] Frederique Oggier and Anwitaman Datta. Self-repairing Homomorphic Codes for Distributed Storage Systems. *CoRR*, abs/1008.0064, 2010.

[76] Nouha Oualha, Melek Önen, and Yves Roudier. A Security Protocol for Self-Organizing Data Storage. In *IFIPSEC*, 2008.

[77] Khandoker Parvez, Carey Williamson, Anirban Mahanti, and Niklas Carlsson. Analysis of BitTorrent-like Protocols for On-Demand Stored Media Streaming. In *SIGMETRICS*, 2008.

[78] David A. Patterson, Garth Gibson, and Randy H. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *SIGMOD International Conference on Management of Data*, 1988.

[79] Fabio Picconi and Laurent Massoulié. Is there a future for mesh-based live-video streaming ? In *P2P*, 2008.

[80] James S. Plank. Erasure codes for storage applications. Tutorial Slides, presented at *FAST-2005: 4th Usenix Conference on File and Storage Technologies*, `http://www.cs.utk.edu/~plank/plank/papers/FAST-2005.html`, 2005.

[81] Srinath Puducheri, Jorg Kliewer, and Thomas E. Fuja. Distributed LT Codes. In *ISIT*, 2006.

[82] K. V. Rashmi, Nihar B. Shah, P. Vijay Kumar, and Kannan Ramchandran. Explicit Construction of Optimal Exact Regenerating Codes for Distributed Storage. In *Allerton Conference on Control, Computing, and Communication*, 2009.

[83] Irving S. Reed and Gustave Solomon. Polynomial Codes over Certain Finite Fields. *Journal of the Society for Industrial and Applied Mathematics (SIAM)*, 8:300–304, 1960.

[84] Sean Rhea, Patrick Eaton, Dennis Geels, Hakim Weatherspoon, Ben Zhao, and John Kubiatowicz. Pond: the OceanStore Prototype. In *FAST*, 2003.

[85] Luigi Rizzo. Effective erasure codes for reliable computer communication protocols. *ACM Computer Communication Review*, 1997.

[86] Rodrigo Rodrigues and Barbara Liskov. High Availability in DHTs: Erasure Coding vs. Replication. In *IPTPS*, 2005.

[87] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Middleware*, 2001.

[88] Sujay Sanghavi, Bruce Hajek, and Laurent Massoulié. Gossiping with Multiple Messages. In *INFOCOM*, 2007.

[89] Bianca Schroeder and Garth Gibson. A Large-scale Study of Failures in High-performance-computing Systems. In *DSN*, 2006.

[90] Thomas Schwarz and S.J. Ethan L. Miller. Store, Forget, and Check: Using Algebraic Signatures to Check Remotely Administered Storage. In *ICDCS*, 2006.

[91] Nihar B. Shah, K.V. Rashmi, P. Vijay Kumar, and Kannan Ramchandran. Explicit Codes Minimizing Repair Bandwidth for Distributed Storage. In *ITW*, 2010.

[92] Claude E. Shannon. A Mathematical Theory of Communication. *The Bell System Technical Journal*, 27:379–423, 1948.

[93] Claude E. Shannon. Communication Theory of Secrecy Systems. *Bell System Technical Journal*, 28:656–715, 1949.

[94] Amin Shokrollahi. Raptor Codes. *IEEE/ACM Transactions on Networking*, 14(6):2551–2567, Jun. 2006.

[95] R. C. Singleton. Maximum distance q-nary codes. *IEEE Transaction on Information Theory*, 10:116–118, 1964.

[96] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *SIGCOMM*, pages 149–160, 2001.

[97] Changho Suh and Kannan Ramchandran. Exact Regeneration Codes for Distributed Storage Repair Using Interference Alignment. In *ISIT*, 2010.

[98] R. Michael Tanner. A Recursive Approach to Low Complexity Codes. *IEEE Transactions on Information Theory*, 27(5):533–547, 1981.

[99] Nikolaos Thomos and Pascal Frossard. Raptor Network Video Coding. In *MV*, 2007.

[100] Gilbert S. Vernam. Cipher Printing Telegraph Systems For Secret Wire and Radio Telegraphic Communications. *Journal of the IEEE*, 55:109–115, 1926.

[101] Mea Wang and Baochun Li. How Practical Is Network Coding? In *IWQoS*, 2006.

[102] Hakim Weatherspoon and John Kubiatowicz. Erasure Coding Vs. Replication: A Quantitative Comparison. In *IPTPS*, 2002.

[103] Jörg Widmer and Jean-Yves Le Boudec. Network Coding for Efficient Communication in Extreme Networks. In *WDTN*, 2005.

[104] Yunnan Wu and Alexandros G. Dimakis. Reducing repair traffic for erasure coding-based storage via interference alignement. In *ISIT*, 2009.

[105] Raymond W. Yeung and Ning Cai. Secure Network Coding. In *ISIT*, 2002.

[106] Zhen Yu, Yawen Wei, Bhuvaneswari Ramkumar, and Yong Guan. An Efficient Signature-based Scheme for Securing Network Coding against Pollution Attacks. In *INFOCOM*, 2008.

[107] Xinyan Zhang, Jiangchuan Liu, Bo Li, and Tak-Shing Peter Yum. CoolStreaming/DONet: A Data-Driven Overlay Network for Efficient Live Media Streaming. In *INFOCOM*, 2005.

Résumé

Au sein de cette thèse, nous étudions l'utilisation de codes dans les systèmes de diffusion distribuée et dans les systèmes de stockage auto-réparant. La diffusion vise à copier sur tous les composants d'un système une donnée initialement disponible auprès une seule source. Les systèmes de stockage distribués autorisent un stockage fiable (persistent malgré les défaillances) réparti sur plusieurs composants de stockage. Cependant, les différents composants d'un système distribués sont non-fiables et difficilement prévisibles à cause des défaillances possibles et des algorithmes locaux et aléatoires.

Les codes, qui transforment un ensemble de k paquets de données natives en un ensemble de n paquets de données encodées de telle sorte que n'importe quel sous ensemble de k paquets encodés permet de retrouver les paquets natifs, ont été largement utilisés afin de construire des systèmes fiables (transmission ou stockage) à partir de composants non fiables. A ce titre, les codes sont très utiles dans les systèmes distribués et surpassent les approches sans codes (réplication par exemple) grâce à: *(i)* une diversité plus élevée des paquets diffusés, réduisant ainsi les surcoûts de communication, *(ii)* une redondance de stockage plus efficace offrant ainsi une meilleure tolérance aux défaillances pour un coût de stockage plus faible. Cependant, dans la pratique, même si les codes sont attirants du fait des grandes améliorations en termes de diffusion et de stockage, ils sont souvent délaissés notamment à cause de leurs coûts de décodage élevés (diffusion) et de leurs coûts de réparation (les coûts réseaux associés à large-création d'un bloc perdu suite à une défaillance) plus élevés (stockage auto-réparant).

Nous étudions la possibilité de réduire les coûts annexes (coûts de décodage et coûts de réparation) afin de rendre les codes plus attrayants. Afin de réduire les coûts de décodages dans les systèmes de diffusion, nous nous intéressons aux codes fontaines basse complexité. Ainsi, nous proposons d'adapter un protocole de diffusion épidémique afin de profiter au plus des codes fontaines basse complexité. Par ailleurs, nous construisons de nouveaux codes réseaux basse complexité en étendant des codes fontaines existants (LT codes). Dans le contexte du stockage auto-réparant, nous étudions l'utilisation de codes réseaux afin de fournir une solution offrant des compromis optimaux entre coûts de stockage et coûts de réparation. Plus précisément, nous étendons les codes régénérants afin de supporter, de façon optimale, des réparations multiples et des systèmes dynamiques. Enfin, nous concluons cette thèse en proposant diverses perspectives concernant les codes réseaux basse complexité et les codes régénérants.

Abstract

In this thesis, we consider the use of codes in both distributed dissemination systems and self-healing distributed storage systems. Dissemination aims at making some data initially available at a single source available to all other devices of a distributed system. Distributed storage systems aggregate many storage devices and allow storing data safely (i.e., persistently in spite of failures). Yet, the underlying components of such distributed systems are unreliable and hardly predictable due to possible failures and local randomized algorithms.

Codes, expanding k native data packets/blocks into n encoded data packets such that any subset of k encoded data packets allows recovering the native data packet, have been widely used to build reliable systems (transmission, storage) out of unreliable and randomly behaving components. Consequently, codes are very useful in distributed systems and outperform non code-based approaches (replication for example). Codes are interesting for *(i)* they increase the diversity of disseminated packets thus reducing the communication overhead, *(ii)* they implement a more efficient redundancy thus offering a better tolerance to failures with lower storage costs. Yet, in practice, even if codes are appealing for they greatly reduce dissemination or storage costs, they are often left aside for various reasons, including because of increased side costs such as the decoding complexity in dissemination systems, and the repair cost (i.e., the network communication cost associated with the regeneration of a lost block that was stored at a failed device) in self-healing distributed storage.

Overall, we consider the reduction of side costs (decoding and repair costs) as a way to make codes more appealing. To reduce the decoding costs in dissemination systems, we consider two approaches based on low complexity fountain codes. First, we propose to adapt a push-based epidemic dissemination protocol (i.e., gossip) to best leverage low complexity fountain codes. Second, we build new low complexity network codes by extending low complexity fountain codes (LT codes). In the context of self-healing storage, we consider the use of network codes as a way to achieve optimal tradeoffs between storage and repair costs. More specifically, we extend regenerating codes to support, optimally, multiple repairs and dynamic systems. Finally, we conclude our thesis by presenting various perspectives around low complexity network codes and regenerating codes.

N° d'ordre : XXXXXXXX