Un protocole de fiabilité basé sur un code à effacement "on-the-fly"

Pierre Ugo Tournoux

THÈSE

de

Pierre Ugo Tournoux

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par *l'Institut supérieur de l'aéronautique et de l'espace*

Discipline: *Informatique et Télécommunications*

Un protocole de fiabilité basé sur un code à effacement "on-the-fly"

Jury

M. Serge Fdida,	Rapporteur
M. Jean-Jacques Pansiot,	Rapporteur
M. Patrick Gélard,	Examinateur
M. Patrick Sénac,	Examinateur
M. Emmanuel Lochin,	Co-encadrant de thèse
M. Michel Diaz,	Directeur de thèse

Remerciements

Je tiens à remercier tout les membres de ce jury : M. Serge Fdida de l'Université Pierre et Maris Curie, et M. Jean Jacques Pansiot qui ont accepté d'être rapporteurs de ma thèse, pour l'intérêt qu'ils ont porté à mes travaux ainsi que pour leurs corrections qui ont grandement contribué à améliorer ce manuscrit. M. Patrick Gélard, directeur de Recherche au CNES de Toulouse et M. Patrick Sénac, professeur à l'ISAE et directeur du département mathématiques et informatique pour avoir eu l'amabilité de bien vouloir participer à ce jury.

Je tiens à remercier M. Michel Diaz pour son encadrement, son soutien et sa confiance par rapport aux différentes problématiques traitées durant mon doctorat. Je n'ai pas eu à me soucier des contraintes matérielles ce qui m'a permis de me focaliser pleinement sur mes travaux de recherche ainsi que sur les prérequis du métier d'enseignant-chercheur. Je tiens également à remercier M. Emmanuel Lochin pour la qualité de son encadrement, son enthousiasme et son soutien. En sus d'avoir eu les réponses à mes questions, d'avoir été libre de mes choix tout en étant dirigé lorsque le besoin s'en faisait sentir, j'ai toujours été écouté et conseillé lorsque les inévitables questions sur l'après doctorat surgissaient. Je souhaite à tout étudiant d'évoluer dans un tel environnement durant ses années de thèses.

Un grand merci à l'ensemble du groupe OLC du LAAS-CNRS et notamment à son directeur M. François Vernadat que j'ai fréquemment sollicité pour les diverses formalités administratives. Johan, Ion, Nicolas, Guillaume merci pour votre accueil et la décoration du bureau A43.

De la même manière je tiens à remercier l'ensemble du département DMIA-MARS de l'ISAE et en particulier à son directeur, M. Patrick Sénac pour avoir facilité ma venue à Toulouse et pour avoir tissé et maintenu cette collaboration entre le l'ISAE et le LAAS. Merci à M. Fabrice Frances et M. Tanguy Perennou pour leur sens de la critique constructive ainsi qu'à M. Yves Caumel pour son goût du partage de la connaissance et ses après midi « probabilité et statistiques. » Aux nombreux doctorants ISAE[1], merci d'avoir animé le long couloir du batiment E, que ce soit au travers de tlmvpsp, du miroir magique ou des discussions scientifiques.

Je tiens à remercier Pascal Anelli, Emmanuel Lochin et Prométhée Spathis qui, à l'Université de la Réunion, m'ont donné goût à la recherche et me suivent encore aujourd'hui. Mon passage au LIP6 et à Thales/TAI m'a apporté énormément et je tiens à remercier M. Marcelo Dias De Amorim, M. Vania Conan et M. Jérémie Leguay pour leur encadrement exceptionnel. Je tiens également à les remercier (ainsi que M. Farid Benbadis et M. John Withbeck) pour m'avoir accordé une partie significative de leur temps lors de nos collaborations que j'espère encore nombreuses.

Je tiens à remercier M. Jérôme Lacan pour avoir introduit le mécanisme de codage qui est l'objet de cette thèse, pour avoir élargi mon domaine de compétence et pour m'avoir accordé toute son attention et son soutien malgré la pléthore d'étudiants qu'il encadre

1. Ahn Dung, Alexandre, Ali, Amine, Chi, Dino, Guillaume, Guo Dong, Hugo, Juan, Lei, Remi, Thomas, Tuan et Victor.

par ailleurs. Les longues séances de *brain storming* avec le duo M. Lacan et M. Lochin m'ont beaucoup apporté et j'espère en partager de nombreuses autres. A vous deux, merci, ainsi qu'à M. Vincent Roca pour y avoir également contribué.

En dépit du fait que désormais je me porterai naïvement volontaire chaque fois qu'on demandera un docteur, la thèse a été pour moi une aventure humaine et intellectuelle qui m'a mené bien loin de ma contrée natale et par laquelle j'ai fait de nombreuses rencontres.

Je tiens à remercier mes parents qui m'ont toujours apporté leur soutien et leur encouragement. Je tiens à remercier mon grand père, Pierre Tournoux, qui en plus de faire gonfler mon *h-index* m'a accueilli chaleureusement lors de mon arrivée à Paris et m'a fait l'immense plaisir d'assister à cette soutenance de thèse.

Je tiens à remercier les amis de tous bords et souhaite notamment tous mes voeux de réussites au très créatif futur docteur Ismail Salhi.

Le grand écart entre deux hémisphères requière beaucoup de souplesse. J'ai donc une pensée très spéciale pour Charmila qui a su prévenir les déchirures musculaires durant ces trois ans. Merci pour ton énergie, ton soutien et ta compréhension. Cette thèse est également la tienne, pas uniquement pour l'éprouvant weekend de correction.

.

Résumé

Le monde du protocole de transport à fortement évolué depuis la création de l'internet. On retrouve désormais une grande diversité de liens avec des caractéristiques hétérogènes en termes de débit, taux de pertes et délais. Plus récemment, le caractère ubiquitaire des périphériques sans fil a permis d'envisager un nouveau mode de transfert prenant en compte la mobilité des utilisateurs pour propager l'information dans le réseau. Ce paradigme de communication rompt définitivement avec les hypothèses de conceptions des protocoles de transport classique. Les applications qui utilisent l'internet ont également évolué. Les réseaux *best effort* sont maintenant utilisés pour transporter des flux à contrainte de délai tels que la voix sur IP et la vidéo conférence. Cependant, malgré ces changements importants, le principe de fiabilité utilisé n'a guère évolué et se base toujours sur des retransmissions.

C'est dans ce contexte que s'inscrit cette thèse qui vise à fournir une brique de fiabilité novatrice pour le support de tout type d'application sur les réseaux *best effort* et les *challenged neworks* qui font de plus en plus partie du paysage de l'internet. A cette fin, nous proposons un mécanisme de codage dénommé Tetrys. Ce code est sans rendement et ses symboles de redondance sont générés à la volée. Il permet à la fois une fiabilité totale et un délai de reconstruction quasi-optimal.

Après une étude détaillée de ses propriétés, nous illustrons la généricité de ce mécanisme. Nous verrons que ses caractéristiques mènent à des contributions aussi bien sur le transport de flux de vidéo-conférence que sur le support du streaming dans les DTN ou encore la fiabilisation des *handovers*.

De manière plus prospective, cette thèse reconsidère les hypothèses initiales du contrôle de congestion. Tetrys est utilisé comme code optimal dans le cas de réseaux dit "anarchiques" (i.e. caractérisés par une absence totale de contrôle de congestion). Nous montrons que le concept de réseau anarchique est une alternative viable au contrôle de congestion et qu'il le remplace même avantageusement.

.

Abstract

Transport protocol area has significantly evolved this last decade. Indeed, several TCP variants or minor enhancements have been proposed at the IETF. These multiple proposals can be explained by the numerous kind of access links in terms of bandwidth, loss rate and delay available today which allowed, for instance, the emergence of challenged networks. Furthermore, the Internet usage has also evolved. Although this network does not provide any guarantee, multimedia applications are pervasive today and Internet is used to carry delay constrained flows generated by Voice over IP or video-conferencing applications. Despite of this evolution, the reliability layer of transport protocols did not evolved and remains based on ARQ schemes.

In this context, this thesis explores a novel erasure code concept named *Tetrys* which can be used as a generic reliability mechanism able to perform over both best effort and challenged networks. After a thorough study which both highlights the benefits brought by Tetrys and dissects its major characteristics (rate-less code, full-reliability capable, nearly optimal decoding delay), we illustrate its versatile behavior with several contributions in various domains. In particular, we present possible use of Tetrys by applying it to video-conferencing applications, Delay Tolerant Mobile Networks streaming applications and handovers management.

As a prospective work, this thesis tackles existing issues from anarchic networks (*i.e.* networks characterized by an absence of congestion control). In this context, we propose a Tetrys-based optimal transmission mechanism for such networks and show that Tetrys is able to complete *congestion-less* protocol stack and allows the deployment of anarchic networks as an alternative to congestion controlled networks in terms of efficiency and fairness.

Table des matières

Chapitre 1

Introduction

1.1 Contexte

Le protocole de transport majoritairement utilisé dans l'Internet reste TCP. Malgré une évolution de quelques mécanismes intrinsèques à la couche transport tels que l'adjonction du principe de contrôle de congestion suite au *congestion collapse* observé dans les années 1980 et de nouvelles propositions pour gérer la congestion ou les flots de données, le principe de base n'a guère évolué et la récupération d'une perte s'effectue toujours grâce à une retransmission. Pourtant le réseau a changé : de quelques hôtes et routeurs qui interconnectaient les différent sites, nous sommes passés à des centaines de milliers de routeurs qui permettent l'interconnexion de millions d'hôtes. Les liens utilisés pour connecter les différents routeurs et hôtes ont également changé. Leurs caractéristiques (en termes de débit et qualité de service) et leurs types (sans-fil ou filaire) diffèrent des liens sur lesquels les hypothèses initiales de TCP ont été fondées. En effet, nous sommes passés des $50kb/s$ initiaux de l'Arpanet à des liens permettant le transfert de plusieurs gigabits par seconde. L'évolution, à une échelle différente, est également remarquable en ce qui concerne les hôtes d'extrémités où les utilisateurs observent maintenant des connexions de plusieurs dizaines de mb/s. Ces liens d'accès présentent également une diversité de délai assez forte, allant de la milliseconde à quelques centaines de millisecondes en ce qui concerne les communications longue distance ou satellite. Il semble donc évident, et la littérature prolifique sur les évolutions de TCP en est un témoin, que l'algorithme du contrôle de congestion doit s'adapter à cette évolution de débits, délais et qualité de lien afin d'utiliser de manière plus efficace cette capacité maintenant disponible. Cependant, l'évolution de TCP possède des limites. Nous verrons donc dans cette thèse qu'il n'est peut être pas improbable qu'une possible évolution serait de revoir le concept

de partage équitable chère à l'internet et que le contrôle de congestion pourrait être un frein à l'évolution du réseau.

Une autre évolution significative concerne l'usage que nous faisons de l'Internet. L'application initiale était le transfert de type "lettre", avec pour seule contrainte la nécessité que le message soit reçu sans erreur et dans son intégralité à la fin de la connexion. Dans ce contexte, il était suffisant d'effectuer une simple retransmission d'un paquet envoyé si ce dernier n'avait pas été accusé au bout d'un certain temps. Cependant, bien que l'Internet *best-effort* ne semblait pas conçu pour l'utilisation d'applications temps-réel, force est de constater qu'aujourd'hui, la vidéo-conférence, la VoIP, les jeux en ligne et les applications interactives (jeux en ligne, commande à distance, ordre boursiers, ...) dominent l'Internet.

Ces applications sont caractérisées par un délai limite tolérable et bien souvent par une dégradation des performances lorsqu'elles observent une augmentation du délai. Le temps de transfert de bout en bout joue un rôle prépondérant quant à la qualité perçue par ces applications, et il est clair que lorsque le délai dépasse celui requis par l'application celle ci ne peut pas utiliser le réseau. Lorsque l'application tolère le délai de bout en bout, le problème de la tolérance aux pertes se pose et avec lui celui du temps requis pour les corriger. En effet, si des applications telles que la video-conférence et la video tolèrent des pertes, il demeure que : *1)* ces pertes sont préjudiciables pour l'utilisateur en termes de qualité perçue *2)* il existe de nombreuses applications (commande à distance, ordres boursier, télé-médecine, ...) qui nécessitent la réception de l'intégralité du flux de donnée initialement produit par l'application. La retransmission de paquets perdus n'est pas toujours compatible avec une application dont le délai d'utilisation des données par la couche applicative est pourtant supérieur au délai de bout en bout.

Les protocoles de transport qui corrigent les pertes par des retransmissions (par exemple TCP) présentent donc un frein a l'utilisation des applications temps réel sur les réseaux de communication.

Dans un autre contexte, le caractère ubiquitaire des périphériques sans fil a permis d'envisager un nouveau mode de transfert utilisant la mobilité et les contacts potentiels entre utilisateurs pour propager l'information dans le réseau. Ce paradigme de communication rompt définitivement avec les hypothèses de conception des réseaux sur lesquels opèrent les mécanismes de transport existant. Dans ces réseaux tolérants aux délai et aux déconnexions, les mécanismes de routage sont fondamentalement différents et l'utilisation de mécanismes de fiabilité des protocoles de transport classique est délicate à cause d'un large délai de bout de bout et d'un taux de perte à la fois élevé et difficilement prévisible qui conduit à un nombre de transmissions substantiel.

Les codes à effacement qui permettent la correction de certaines pertes sans intervention de l'émetteur sont utilisés depuis peu au niveau application et transport, notamment afin d'améliorer la qualité des applications à contrainte de délai. Cependant, ces codes sont aussi sensibles au taux d'erreurs du lien qu'à leur répartition. De plus, leur résistance à ces deux paramètres se fait au coût d'un délai supplémentaire ou d'une surcharge à l'émission. Enfin, il n'existe à l'heure actuelle aucun code qui permette de garantir la reconstruction de l'intégralité des pertes sans avoir recours à la source, et les applications temps réel qui requièrent une fiabilité totale ne sont donc pas supportées par ces codes.

C'est dans ce contexte que s'inscrit cette thèse qui vise à fournir un mécanisme de codage cherchant à combler le fossé qui demeure entre fiabilité totale et application temps réel ainsi qu'à étudier les possibilités offertes aux applications et aux protocoles de transport.

1.2 Contributions

La contribution de cette thèse se découpe en deux parties. Tout d'abord celle concernant l'étude des propriétés intrinsèques de Tetrys, un code à effacement à fenêtre d'encodage élastique. Ensuite nous verrons comment il peut être appliqué à divers domaines et évaluerons le gain apporté. Nous détaillons ci-dessous le contenu de ces deux contributions.

1.2.1 Étude des propriétés et implémentation de Tetrys

Cette partie s'attache a démontrer différentes propriétés du code Tetrys qui trouveront une application en seconde partie de ce manuscrit. Contrairement aux codes à effacement actuels, le code proposé offre une nouvelle propriété qui est de reconstruire l'ensemble des paquets sans avoir à faire intervenir la source, et donc, en un temps indépendant du délai aller retour entre la source et la (ou les) destination(s). De plus, ce code est sans rendement et ses symboles de redondance sont générés à la volée. Nous verrons que bien qu'il ne soit pas fortement MDS au sens propre du terme, le surcout est négligeable. Cela veut dire que son délai de reconstruction est quasi-optimal et ce bien qu'il opère sur des corps fini de petite taille (GF(4)). Nous apportons également une étude théorique sur les performances du code avec pour métrique le délai requis pour le décodage ainsi que la surcharge induite par l'utilisation du code. Nous comparons la robustesse de la configuration de Tetrys à celle des codes en bloc ainsi que le délai de reconstruction.

Enfin, deux implémentations de ce code sont proposées. Une est intégrée au simulateur ns-2 tandis que l'autre est directement utilisable sur les environnements BSD et Linux et peut s'interfacer entre les couches 3/4 ou au niveau application.

1.2.2 Contributions à la fiabilité de bout en bout

Les propriétés énoncées en première partie de ce document nous ont permis d'utiliser Tetrys comme mécanisme de fiabilisation dans des domaines ne pouvant pas se satisfaire de simples retransmissions :

1. Le premier concerne le transport de trafic multimédia à forte contrainte de délai. En sus de l'évaluation du gain apporté aux applications de vidéo conférence, nous proposons une heuristique pour la configuration du code afin qu'il réponde aux besoins de ce type d'application tout en minimisant le coût induit en termes de débit. Les décodeurs vidéo ne permettent pas tirer pleinement profit des caractéristiques de Tetrys. Nous proposons donc une modification des codecs qui permet de prendre en compte les trames décodées tardivement ce qui améliore sensiblement les performances de la transmission video ;
2. Le second traite de la fiabilisation de flots sur réseau DTN. Contrairement aux mécanismes existants, l'utilisation de Tetrys permet le support des applications dites de *streaming*[1] dans les DTN. Nous évaluons leurs performances dans ce contexte et montrons que Tetrys est un candidat parfait pour la fiabilisation de flots de *streaming* ;
3. Le dernier se situe au sein des protocoles de management de la mobilité. En particulier, nous illustrons que Tetrys, avec une configuration adaptée, minimise l'impact des pertes qui surviennent durant un *handover* vertical. L'évaluation démontre que contrairement aux autres propositions basées sur les codes à effacement, Tetrys permet à TCP de conserver un débit optimal même lorsque la qualité du lien sans fil se dégrade fortement. Nous montrerons également que grâce à Tetrys, les protocoles de transport classiques peuvent tirer profit de la disponibilité de plusieurs réseaux de manière efficace et transparente.

1.2.3 Tetrys pour le déploiement des réseaux anarchiques

De manière plus prospective, cette thèse aborde également les réseaux dits "anarchiques" qui sont caractérisés par une absence totale d'utilisation du contrôle de congestion de bout en bout. De récents travaux théoriques ont montré la faisabilité du concept en se basant sur l'existence théorique d'un code parfait. Cependant, l'existence d'un tel code, qui est une condition nécessaire à la mise en oeuvre de ces réseaux, n'a jamais été énoncé. Nous montrons que Tetrys est un code possible sur ces réseaux anarchiques et qu'il permet d'amener la brique de fiabilité nécessaire à leur déploiement. Au meilleur de

1. Par application de *streaming*, nous faisons référence aux applications qui produisent et consomment les données dans l'ordre et en continue. Ces applications n'ont pas nécessairement de contrainte de délai.

notre connaissance, c'est la première fois qu'un réseaux anarchique est évalué de façon pragmatique (avec une pile protocole et des topologies réalistes) en termes d'efficacité [2] et d'équité.

1.3 Organisation

En plus des chapitres d'introduction et de conclusion, ce manuscrit est composé de six chapitres et trois parties.

Dans le chapitre 2, nous passons en revue les principaux mécanismes de fiabilité de niveau transport. Le principe des codes à effacement y est présenté et permet d'appréhender le mécanisme Tetrys.

La partie I est consacrée à Tetrys. Dans un premier temps, le mécanisme de codage est détaillé. Ensuite, nous présentons une modélisation de Tetrys à partir de laquelle nous évaluerons le temps de décodage et sa complexité. La comparaison avec les codes bloc et H-ARQ est également abordée. Finalement, nous proposons une heuristique pour la configuration de Tetrys.

La deuxième partie II est consacrée aux applications de Tetrys. Au chapitre 4, nous étudierons le gain apporté par Tetrys à l'application de vidéo conférence. Nous verrons qu'une adaptation des codecs permettrait d'améliorer encore les résultats. Le chapitre 5, étudie la fiabilisation des applications de *streaming* dans les réseaux tolérant au délai. Enfin, dans le chapitre 6, nous montrons que l'utilisation de Tetrys permet d'améliorer sensiblement les performances des protocoles de transport durant un *handover* vertical.

La partie III est consacrée à un travail prospectif. Nous y utiliserons Tetrys comme mécanisme de transmission optimal afin d'illustrer les propriétés des réseaux anarchiques.

2. L'efficacité correspond au rapport entre le débit utile pour les applications et le débit maximum qui peut être écoulé entre les noeuds du réseau

Chapitre 2

Etat de l'art

De toutes les strates du modèle OSI, la quatrième couche, dite de transport, est une de celle qui est en charge du plus grand nombre de tâches. En effet, étant le premier protocole qui opère de bout en bout, son rôle est à la fois d'identifier le canal logique de communication [1, 2] ; d'assurer le contrôle de flux ; d'assurer via le contrôle de congestion que les ressources du réseau soient équitablement partagées entre les utilisateurs [3] ; plus récemment, de gérer la diversité des chemins disponibles [4] ; et finalement, d'assurer que les données émises par l'application soient correctement reçues par le destinataire. Ce chapitre, et la thèse plus généralement, se focalise sur cette seule mission de fiabilité de bout en bout [1] des protocoles de transport et applications, passant en revue les principaux mécanismes existants ainsi que leurs limitations. Dans la suite du document, par *mécanisme* ou *protocole* de transport, nous ferons implicitement référence à leur fonction de fiabilité.

2.1 Applications, réseaux et protocoles de transport

Dans la première partie de ce chapitre nous rappelons les liens existants entre les réseaux, les applications et les protocoles de transport, avant d'introduire les principaux mécanismes de fiabilité.

2.1.1 Contraintes des applications :

Les applications ont différents besoins en termes de qualité de service. Ces besoins s'expriment et se quantifient en termes de bande passante, de délai et de fiabilité. Nous

1. Une partie de la communauté considère d'ailleurs que chacune de ces fonctions devraient être traitées séparément [5].

considérons qu'une application dispose toujours de la bande passante minimale requise pour son bon fonctionnement. A l'inverse, nous faisons l'hypothèse qu'un mécanisme de transport doit satisfaire les besoins en termes de délai et de fiabilité tout en minimisant l'utilisation de la bande passante.

Du point de vue d'un protocole de transport, les principales classes d'applications sont les suivantes :

- fiabilité totale sans contrainte de délai : l'e-mail nécessite que l'intégralité du message soit reconstruit mais ne possède pas de contrainte temporelle forte. Les paquets peuvent être reçus dans un ordre quelconque, ils ne sont utilisés qu'une fois que le message est intégralement reçu ;
- fiabilité partielle et faible contrainte de délai : les applications de diffusion de contenu (*e.g. streaming video*) requièrent que tout ou une partie des messages soient reçus avant une borne de délai qui peut être ajustée selon le contexte. La contrainte de délai provient du fait que l'application consomme les données au fur et à mesure qu'elle les reçoit ;
- fiabilité partielle et forte contrainte de délai : les délais requis par les applications de la Voix sur IP (400 *ms* [6]) et la vidéo conférence (100 *ms* [7]) sont plus contraignants. Ces applications tolèrent qu'une partie des informations soit perdue [8] ;
- fiabilité totale et forte contrainte de délai : comme toutes les applications de commande à distance, la télé-chirurgie [9, 10], combine les deux contraintes précédentes [2]. et nécessite donc une fiabilité totale avec une forte contrainte sur le délai de réception des messages par l'application.

2.1.2 Contraintes imposées par le réseau

Les protocoles de transport évoluent dans des environnements hétérogènes dans lesquels ils sont soumis à trois principales contraintes :

- la perte de paquets et la distribution des pertes ;
- le délai de bout en bout et ses variations ;
- l'ordre d'arrivée des paquets.

2.1.2.1 Origine des pertes et délai

Le délai de bout en bout varie selon le contexte. Il peut atteindre moins d'une milliseconde sur un réseau local, varier entre 1ms et plusieurs centaines de millisecondes

2. Il serait regrettable qu'une partie des mouvements du chirurgien ne soit pas retranscrits ou qu'il le soit trop tardivement

dans l'internet [11] et enfin atteindre plusieurs dizaines de minutes sur les liens inter-satellitaire ou planétaire [12].

Des paquets peuvent être perdus à plusieurs niveaux. Les pertes peuvent être liées à la congestion ; dans ce cas, un paquet arrivant sur une interface dont la file d'attente est pleine sera jeté et donc perdu. Dans l'internet, le taux de perte relatif à la congestion est assez faible et n'excède pas 1 ou 2% dans la majorité des cas [13, 14].

Les paquets peuvent être perdus au niveau des routeurs par manque d'information (*i.e.* si ils ne peuvent choisir la bonne interface de sortie ou parce que le TTL des paquets a expiré. C'est souvent le cas dans les réseaux Ad-Hoc mobiles (MANET) [15] où les routes changent fréquemment et dans l'Internet lorsque les routeurs sont mal configurés [16] [3].

Finalement, au niveau physique ou liaison, les pertes sont dues à des interférences ou autres bruits sur le canal qui mènent à des changements de valeur. Si ces erreurs ne peuvent être corrigées au niveau liaison, la trame est détruite et le paquet qu'elle contient est perdu. Les pertes de paquet induites par la couche liaison sont généralement rares sur les mediums d'accès couramment rencontrées. Cependant, lorsqu'il y a une densité d'utilisateur assez forte, ou que ces derniers sont fortement mobiles, les pertes au niveau physique augmentent de manière sensible et peuvent atteindre jusqu'à 50% [17, 18].

Les réseaux tolérants aux délais [19] (*Delay Tolerant Network*) présentent un contexte bien plus contraignant que nous introduisons succinctement avant d'y revenir plus en détail au Chapitre 5 dans le cadre d'une solution de fiabilité pour les applications de *streaming*.

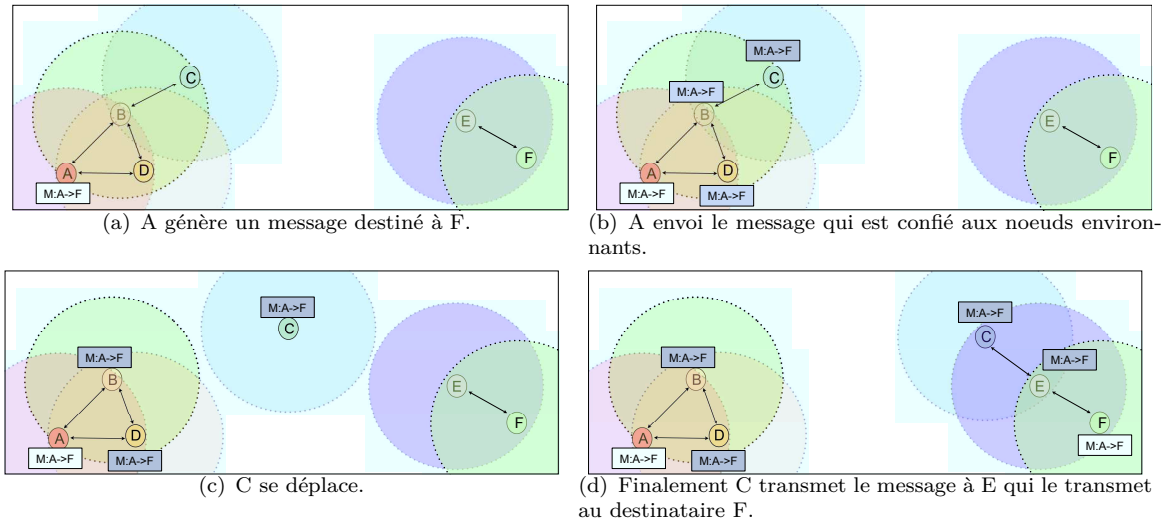
2.1.2.2 Digression sur les réseaux mobiles tolérants aux délais (DTNM)

Le caractère ubiquitaire des périphériques sans fil a récemment permis d'envisager un nouveau mode de transfert utilisant la mobilité et les contacts potentiels entre utilisateurs pour propager l'information dans le réseau. Ces réseaux DTN mobiles sont donc composés de noeuds sans-fil mobiles capables d'échanger des données lorsqu'ils sont à portée de transmission.

Comme le montre la figure 2.1, à un instant donné, la topologie induite par ces noeuds peut ne pas être connexe et la communication entre composantes connexes du réseau n'est rendue possible que par la mobilité des utilisateurs.

La plupart des contextes DTNM étudiés sont ceux de la transmission de messages unitaire appelés *bundles* entre personne d'une même conférence [20], d'un même campus [21]

3. Les pertes liées au routage impliquent parfois trop de paquets consécutifs perdus (la connexion est coupée trop longtemps) pour que les mécanismes de fiabilité niveau transport puissent les corriger.



(a) A génère un message destiné à F.

(b) A envoi le message qui est confié aux noeuds environnants.

(c) C se déplace.

(d) Finalement C transmet le message à E qui le transmet au destinataire F.

FIGURE 2.1: A un instant donné la topologie d'un DTN n'est pas forcement connexe. Cependant, la mobilité des utilisateurs (en l'occurence C) permet la communication et rend le réseau connexe si l'on considère une échelle de temps adéquate.

ou d'une même ville [22]. Le vecteur d'information (l'entité mobile) peut être des piétons, des voitures ou des bus et les délais varient de quelques minutes à plusieurs heures. Nous nous focaliserons sur des réseaux ou la densité ainsi que la mobilité des utilisateurs est beaucoup plus forte et en particulier sur le cas RollerNet [23], (un sous-ensemble des travaux portant sur les DTN ont été effectués en parallèle au sujet principal de cette thèse). Dans ce réseau formé de plusieurs milliers de participant, le délai moyen varie entre quelques secondes et plusieurs minutes. Quand bien même le réseau est connexe, les changements de topologies induits par la mobilité des utilisateurs sont tels que les protocoles de routage Ad-hoc classiques [24] ne convergent pas [25]. A cause de ces perturbations, les approches de routage que l'on trouve communément dans les réseaux ad-hoc ne peuvent être transposées sur ces réseaux et donc de nouvelles solutions sont nécessaires.

Parmi les protocoles de routage proposés pour ces réseaux, le routage épidemique [26] consiste à diffuser l'information de proche en proche jusqu'à ce qu'elle soit présente sur l'ensemble des noeuds du réseaux et atteigne donc le ou les destinataires. D'autres approches limitent le nombre de réplications possibles pour un bundle donné [27] ou conditionnent la réplication en fonction de la probabilité qu'a un noeud de rencontrer le destinataire [28].

La caractéristique commune à ces protocoles et aux réseaux DTNM en général est que le nombre de messages présents dans le réseau génère un taux de perte important. Lorsque les protocoles limitent la réplication, la probabilité qu'un *bundle* ne parvienne pas à

son destinataire est également importante. De plus, les messages suivant des chemins aléatoires sont souvent reçus dans le désordre et pour finir, le délai n'est pas prévisible.

Les conditions rencontrées par un mécanisme de fiabilité de niveau transport dans ce qui est naturellement appelé *challenged networks* sont donc particulièrement délicates. Au Chapitre 5, nous reviendrons sur les mécanismes de fiabilité existants et sur les problèmes liés au support des applications de *streaming*.

2.1.3 Mécanismes à utiliser en fonction du contexte

Lorsque qu'il n'y a pas de contrainte de délai, les pertes peuvent être récupérées par des mécanismes dit d'ARQ qui détecte la perte et en informe l'émetteur qui retransmet alors les paquets perdus. Ainsi la récupération de la perte dure au minimum un RTT. Les applications à contraintes de délai (*e.g. streaming*, VoIP et vidéo-conférence) peuvent donc également utiliser cette classe de mécanisme de fiabilité lorsque le RTT est faible devant le délai imparti par l'application (*e.g.* RTT 20 *ms* et 100 *ms* de délai tolérable pour l'application). Par contre, si le RTT augmente et atteint 2/3 du délai tolérable, les pertes ne peuvent plus être corrigées par de tels mécanismes et nécessitent l'emploi de mécanismes de correction d'erreurs proactifs. Ces mécanismes sont également incontournables lorsque la voie retour n'existe pas ou qu'elle n'est pas techniquement envisageable comme ça peut être le cas pour la diffusion ou le multicast avec un grand nombre de récepteurs.

Dans les sections suivantes, nous passons en revue les principaux mécanismes de récupération d'erreurs proactifs (section. 2.2) ou réactifs (basés sur les retransmission) (section.2.3) avant d'étudier en section 2.5 les avantages et inconvénients de l'utilisation de chacun et de positionner notre étude.

2.2 Mécanismes pro-actifs : codes à effacement

Le mécanisme de récupération d'erreurs proactif le plus simple consiste à répéter les messages plusieurs fois. Au minimum, ces derniers doublent le débit du flux initial ce qui les rend peu efficaces et inutilisables en pratique. Les codes à effacement permettent de reconstruire les paquets perdus avec beaucoup moins de paquets redondants et sont donc la solution retenue lorsque les retransmissions sont impossibles.

Nous savons grâce aux travaux de Shannon [29] qu'il est possible avec un codage adéquat, de transmettre des informations avec un taux d'erreurs arbitrairement faible sur un canal de communication tout en s'approchant de sa capacité. Sur ce type de canal, un

symbole X en entrée peut être altéré et produire un symbole Y en sortie. Les codes correcteurs d'erreurs permettent de récupérer les erreurs qui surviennent sur un canal de communication. Dans [30], Elias présente le modèle du canal à effacement sur lequel les symboles transmis sont soit reçus, soit perdus. Nous avons vu que par les propriétés des différents mécanismes de détection d'erreurs des couches liaisons et transports, les paquets reçus ont peu de chance d'être altérés et qu'ils sont soit correctement reçus, soit jetés. Ainsi, nous considérons que du point de vue des mécanismes de fiabilité de bout en bout, le canal de communication est à effacement et les codes utilisés seront également appelés à effacement.

Afin d'illustrer le fonctionnement général d'un code à effacement, nous présentons un code Reed-Solomon sur matrice de Vandermonde avec au préalable l'introduction des définitions qui nous seront utiles tout au long du document :

Corps fini : Un corps fini (ou corps de Galois) est un ensemble d'éléments sur lequel est défini l'addition, la soustraction, la multiplication et la division. Pour chaque nombre premier p et entier m il existe (à un isomorphisme près) un corps fini à $q = p^m$ éléments noté $GF(q)$ où $\mathbb{F}_q$.

Elément générateur : Un élément α est un élément générateur si l'ensemble des éléments du corps fini $GF(q)$ peut être généré à partir de ce dernier.
$GF(q) = \{0, 1, \alpha, \alpha^2, ..., \alpha^n - 2\}$.

Symbole : Un symbole est un élément d'un corps fini $GF(q)$.

Code en bloc de longueur n et dimension k : Un code en bloc linéaire $\mathcal{C}$ de longueur n et de dimension k est un sous espace vectoriel de dimension k de l'espace vectoriel $\mathbb{F}_q^n$, q désignant le cardinal du corps fini. Nous le noterons *code en bloc* (n, k) ou un *code* $FEC(n, k)$. Un élément de $\mathbb{F}_q^n$ est un mot de code, lequel est formé de n symboles.

Taux de redondance et taux de codage : Le rendement du code ou taux de codage correspond au nombre à k/n. Le taux de redondance noté $R = (n - k)/n$.

Matrice génératrice : Une matrice génératrice d'un code ou matrice d'encodage, est l'application linéaire du sous espace vectoriel des symboles sources $\mathbb{F}_q^k$, dans $\mathbb{F}_q^n$.

Soit X, un vecteur de symboles sources appartenant à $\mathbb{F}_q^k$ et $G \in \mathcal{M}_{k,n}$, la matrice d'encodage, X est encodé par la multiplication à gauche de G :

$$Y = X \cdot G \tag{2.1}$$

et Y est un mot de code de n symboles.

En pratique il est utile que les k premiers symboles de Y correspondent au vecteur X. Cela permet de lire les symboles en clair même si il n'a pas été possible de récupérer les symboles sources perdus.

Code systématique : Un code est dit systématique si les symboles sources sont reproduits en clair dans le mot de code correspondant. Dans ce cas :

$$G = [I_k|C] \tag{2.2}$$

avec I_k la matrice carré identité de taille k et C une matrice de k lignes et $n-k$ colonnes.

On peut déjà entrevoir comment le codage des symboles sources est appliqué aux paquets. Dans les codes à effacement utilisés dans les couches hautes, un symbole n'est pas un élément de $GF(q)$ mais un paquet. Ainsi le vecteur de symbole source correspond à $X = [P_1, P_2, P_3...P_k]$ avec P_i le $i - eme$ paquet du bloc à encoder. De la même manière, si le code est systématique, le mot de code $Y = XG$ est tel que $Y = [P_1, P_2, P_3...P_k, R_1....R_{n-k}]$ avec R_i sont les paquets codés dits de redondance. Pour appliquer le code, les paquets sont simplement divisés en symboles [4], par exemple de un octet pour $GF(255)$. Les paquets de t octets sont donc des tableaux de t symboles et le code génère le i-éme symbole des paquets du vecteur Y en utilisant le vecteur source $P_1[i], P_2[i], P_3[i]...P_k[i]$.

Ces opérations sont effectuées à la suite mais dans le cas d'implémentation matérielle, elles peuvent être effectuées en parallèle.

Code MDS : Un code est dit MDS si la réception d'au moins k paquets parmi n générés par le code permet de reconstruire les k paquets sources initiaux. On parlera également de code parfait ou idéal.

2.2.1 Un code en bloc MDS : Reed-Solomon sur matrice de Vandermonde

Les codes Reed-Solomon sont des codes correcteurs parfaits qui ont été introduits en 1960 et rendus célèbres par leur utilisation dans les disques compacts et la communication avec les sondes Voyager. Dans le cas des codes correcteurs d'erreurs sur canal de communication, il est couplé à l'algorithme de Berlekamp-Massey [31] lequel n'est cependant pas approprié au canal à effacement de paquets et on préféra d'autres constructions basées sur des matrices génératrices de Cauchy ou Vandermonde. C'est sur ces dernières

4. En pratique on considère que l'ensemble des paquets d'un bloc ont la même taille. Si ce n'est pas le cas, à l'encodage les paquets de taille inférieure au paquet le plus grand sont comblés par des 0, lesquels seront supprimés après le décodage.

qu'en 1997, Luigi Rizzo a basé son implémentation de codes à effacement [32], laquelle est aujourd'hui standardisée [33] et sert de référence pour l'évaluation et la comparaison des codes à effacement.

L'intérêt des matrices de Vandermonde est que chacune des ses sous matrices est inversible. Cela permet, quelles que soient les lignes et colonnes considérées (i.e. quels que soient les paquets reçus parmi les n envoyés) de trouver l'application inverse à celle de l'encodage. Une matrice de Vandermonde $V_{k,n}$ de taille (k, n) dont les éléments sont dans $GF(q)$ avec $q \geq 2^{n-1}$ se construit de la façon suivante à l'aide de α un élément générateur de $GF(q)$:

$$V = \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \alpha & \alpha^2 & \alpha^3 & \cdots & \alpha^{(n-1)} \\ 1 & \alpha^2 & \alpha^4 & \alpha^6 & \cdots & \alpha^{2(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha^{(k-1)} & \alpha^{(k-1)2} & \alpha^{(k-1)3} & \cdots & \alpha^{(k-1)(n-1)} \end{pmatrix} \tag{2.3}$$

Afin que la matrice génératrice V soit systématique, une élimination de Gauss peut faire apparaitre [32] la matrice identité. Notons G la matrice génératrice systématique (n, k) ainsi obtenue, l'encodage se fait par multiplication du vecteur source $X_{k,1}$ par la matrice G :

$$Y = G \cdot X \tag{2.4}$$

Les k premières lignes du vecteur $Y_{k,1}$ correspondent au vecteur X et les $n-k$ dernières correspondent alors aux symboles redondants. Si le vecteur Y' reçu par le destinataire comprend des effacements sur des paquets source, alors ces derniers doivent être corrigés, ce qui est possible s'il y a eu moins de $n-k$ effacements.

Soit $\mathbb{E}$ la liste des positions de symboles effacés dans le vecteur Y'. Si l'on supprime de G l'ensemble des lignes L_i tel que $i \in \mathbb{E}$, on obtient une matrice G' carrée de taille k avec :

$$Y' = G' \cdot X \tag{2.5}$$

Le vecteur X peut alors être récupéré par

$$X = G'^{-1} \cdot Y' \tag{2.6}$$

ce qui est possible car l'ensemble des sous matrice est inversible (propriété des matrices de Vandermonde précédemment énoncée).

Il existe cependant un inconvénient à l'utilisation de ces codes. La longueur du code qui peut être généré est borné par la taille du corps fini sur lequel il est construit et ne peut donc dépasser $2^p - 1$ sur $GF(2^p)$. Ainsi, pour augmenter la longueur du code il faut augmenter la taille du corps fini. Or les multiplications sur les corps finis $GF(2^p)$ nécessitent au minimum $p/2$ XOR et le coût associé peut devenir prohibitif. De plus la complexité en temps de calcul de ces codes est en $O(nlog^2n)$ [5] ce qui motive l'utilisation d'autres types de codes en bloc, dont l'encodage et le décodage sont d'une complexité moindre.

2.2.2 Codes LDPC, sans rendement et Raptor

Les codes suivants sont également largement utilisés ne sont pas MDS et nécessitent plus de k symboles pour permettre la reconstruction des pertes.

Codes LDPC : Les codes LDPC pour "Low Density Parity Check" sont des codes en bloc qui peuvent être systématiques ou non. Comme leur nom l'indique, ils ont la particularité d'avoir une matrice de parité de faible densité, autrement dit, contenant le plus d'entrées nulles possibles. Cette caractéristique leur confère une complexité linéaire à l'encodage et au décodage.

Codes sans rendement : Les codes sans rendement ne sont pas régis par les paramètres n et k et peuvent générer un nombre potentiellement infini de symboles de redondance. Ces codes visent notamment les applications de diffusion de contenu fiable de type *carroussel*. Dans ce contexte, plusieurs utilisateurs s'abonnent à un flux à des instants différents afin de recevoir une même information. L'avantage de ces codes est que l'instant où les paquets peuvent être générés en continue et l'instant d'abonnement n'influent pas sur le nombre de paquets à recevoir avant de pouvoir récupérer les symboles sources. Les codes LT (Luby Transform) proposés par M.Luby [36] sont des codes sans rendement dont les symboles sont générés en choisissant au hasard d symboles sources. La complexité et la capacité de correction de ces codes sont guidées par le degré d de symboles de redondance (préconisé à $log(k)$ [36]) et la complexité d'encodage est donc $O(log(k))$ par symbole soit $O(n \cdot log(k))$ par mot de code. Le décodage est du même ordre et se fait selon les mêmes algorithmes que pour les codes LDPC.

Codes Raptor : Les codes Raptor (Rapid Tornado) sont une amélioration des codes LT dans la mesure où ils sont également sans rendement mais systématiques. Ils sont présentés comme ayant une complexité linéaire et ils se comportent bien mieux pour des mots de codes de petite taille (≈ 1000). Ces propriétés sont assurées par la concaténation

5. Récemment, il a été montré dans [34] que la complexité pouvait être réduite à $O(plog^2p)$ et à $O(nlogn)$ [35] si le corps fini est $GF(q)$ avec q un nombre de Fermat premier.

d'un pré-code basé sur un code LDPC et un code de Hamming. Leur but est d'améliorer l'efficacité des décodeurs et de réduire leur complexité. Ce code est actuellement le plus plébiscité car il est aussi efficace en termes de capacité de correction que de complexité de décodage.

La caractéristique commune de ces codes est qu'ils ont une capacité de correction proche de celle des codes MDS pour une complexité bien moindre (linéaire contre $nlog^2n$). En pratique, cela n'est vrai que pour des valeurs de k (longueur du mot de code) très grandes car la capacité de décodage décroit assez rapidement lorsque k est inférieur à 1000. De plus, ils sont donc inutilisables pour les applications temps réel à moins que le produit du débit généré par l'application et de la borne de délai tolérable permettent l'utilisation de mots de code de taille suffisante. Nous y reviendrons dans la section 2.5.

2.2.3 Codes convolutifs pour le canal à effacement

Dans les deux précédents types de codes, le message est soit considéré par blocs de longueur fixe pour former des mots de code (codes bloc), soit considéré dans son intégralité avec des symboles de redondance calculés sur l'ensemble du message. Dans le cas des codes convolutifs (ou convolutionels), les mots de code peuvent être considérés comme de taille infinie car ces derniers sont traités à la volée et influent sur la sortie du code durant plusieurs pas d'exécutions. Initialement proposés en 1955 par Elias [37], les codes convolutifs tels que les turbo codes [38] sont largement utilisés pour corriger les erreurs des canaux de communications satellites et de téléphonie mobile.

Plus récemment, le concept de codes convolutifs a été adapté aux codes LDPC par Felstrom [39] et Sridharan [40]. Ce dernier a montré dans sa thèse que ces codes sont plus robustes que les codes blocs LDPC équivalents. Comme pour les codes LDPC bloc, la longueur du code considérée demeure conséquente. De plus, le décodage se faisant dans le désordre, ils ne sont pas non plus adaptés aux contraintes temps réel.

Martinian [41] a également adapté le principe des codes convolutifs aux codes pour canal à effacement avec une matrice génératrice non creuse. Par rapport aux codes en bloc MDS, ces codes sont présentés comme plus robustes aux pertes en rafales et nécessitent moins de redondance. De plus, le délai de décodage est en moyenne bien moindre que celui des codes en bloc.

En 2001, Smarandache a montré dans sa thèse qu'il existe des codes MDS convolutifs au taux de codage k/n et propose une construction à partir d'un code Reed-Solomon très large. Comme nous l'avons vu, l'inconvénient de cette construction est que le corps fini requis est très grand, rendant inutilisable ce type de code en pratique. Dans [42],

il propose la construction d'un code fortement MDS (strongly MDS) à savoir MDS et qui reconstruit les pertes le plus tôt possible. La construction de ce code se fait via les matrices de Toeplitz triangulaires, *super-régulieres* [43]. La propriété de ces matrices est que le déterminant de chacune des sous matrices qui n'est pas trivialement nul est non nul (*i.e.* la matrice de décodage peut donc toujours être inversée). L'inconvénient est qu'en pratique, il n'existe aucun moyen de construire de telles matrices automatiquement. De plus pour une matrice d'ordre n donnée, l'existence de ces matrices, n'est pas assurée si la taille du corps fini ne dépasse pas $GF(2^{n-2})$ ce qui en pratique, revient à utiliser des tailles de corps prohibitifs.

2.3 Mécanismes réactifs : ARQ

Lorsqu'il y a une voie retour, le moyen qui semble le plus évident afin d'assurer une certaine fiabilité est de retransmettre les parties du message identifiées comme perdues. Ces techniques dites *ARQ (Automatic Repeat reQuest)* [44] sont utilisées aussi bien au niveau liaison de données qu'au niveau transport et se décomposent en deux étapes distinctes : la détection de la perte et sa retransmission.

Les pertes peuvent être détectées par l'émetteur de deux manières. La première est explicite et le récepteur envoie des messages NAK (pour Negative Acquittement) qui indique à l'émetteur que le message reçu contient des erreurs. Lorsque même l'entête du message n'a pu être transmise au récepteur, cette seule technique ne le permet pas et donc doit être combinée à des horloges de retransmission. En effet, à l'envoi de chaque message, l'émetteur arme une horloge qui à son expiration considérera comme perdu le paquet et provoquera une retransmission. L'utilisation d'horloge suppose que chaque message soit acquitté. Dans le cas de SAW (Stop And Wait Arq), les messages sont explicitement acquittés puisqu'il n'est transmis qu'un message à la fois. Cette méthode est clairement inefficace [44, 45] lorsque le délai de propagation du message est significatif et elle n'est donc en pratique utilisée qu'au niveau liaison sur des liens à faible délai. Au niveau transport, l'usage d'une fenêtre glissante est préférable. L'acquittement des plus vieux messages (encore non acquittés) permettant de décaler cette fenêtre et de valider l'envoi de nouveaux messages. Dans ce cas, chaque message reçu ne génère pas obligatoirement d'acquittement et ces derniers seront de type cumulatifs afin de limiter l'impact de leur pertes et éviter des retransmissions abusives [46].

Les différents schémas de retransmission sont d'un moindre intérêt étant donné qu'ils n'ont, dans la plupart des cas, qu'un faible impact sur le délai de récupération. Notons tout de même le mécanisme *Go-Back-N*, initialement choisit par TCP, et particulièrement inefficace. Comme son nom l'indique, lorsqu'un paquet est perdu, ce n'est

pas uniquement le paquet perdu qui retransmit mais l'ensemble de la fenêtre d'émission. L'efficacité est ainsi réduite mais sur les liens de faible débit et long délai, cela influence également le délai des messages suivants puisque le lien est inutilement occupé par des messages déjà reçus. Il est donc préférable que seul les paquets perdus soit retransmis, ce qui est rendu possible par l'utilisation d'accusé sélectifs *SACK* [47] qui indiquent exactement quels paquets ont été reçus.

Le récepteur utilise les numéros de séquence des paquets pour déterminer ceux qui ont été perdus. Or nous avons vu que les paquets peuvent être désequencés par le réseau et un paquet peut donc être considéré comme perdu et faire l'objet d'un acquittement négatif alors qu'il est simplement retardé. Afin de palier au problème d'efficacité qui en découle, le récepteur doit attendre un certain temps pour s'assurer que le paquet est bien perdu. Si c'est à l'émetteur qu'incombe la détection des pertes via des horloges de retransmission, ces dernières ne peuvent correspondre directement au RTT puisque qu'elles expireraient inutilement lors des variations du RTT. En pratique, ces horloges sont donc largement supérieures au RTT (e.g. $4 \cdot RTT$ pour TCP [48]).

ARQ idéal : Quelle que soit la variante d'ARQ utilisée, le délai de récupération [6] d'une perte est donc de $\frac{3 \cdot RTT}{2}$ auxquels s'ajoute le délai pour éviter les fausses détections de perte. Dans la suite de ce document, nous considérerons uniquement une version idéale des mécanismes ARQ avec une fenêtre d'émission infinie [7] qui ne retransmet que les paquets perdus et dont le temps de récupération d'une perte n'est que de $\frac{3 \cdot RTT}{2}$ (si la retransmission n'est pas elle même perdue).

2.4 Mécanismes hybrides : H-ARQ

Dès 1960, Wozencraft et Horstein ont proposé l'idée de combiner les codes correcteurs d'erreurs aux mécanismes de retransmissions [49]. Ces mécanismes dits hybrides ont par la suite été définis selon deux types : les H-ARQ de type I ou de type II. Le type I consiste à envoyer un bloc (n, k) selon un code correcteur systématique quelconque [8]. Si le bloc ne peut être décodé il est intégralement retransmis. Pour les H-ARQ de type II, aussi appelés IRH-ARQ (*incremental redundancy*) le code correcteur génère un bloc (n, k) mais seule la partie systématique et éventuellement quelques bits de parité sont émis. Si le bloc ne peut être décodé, l'émetteur envoie d'autres bits de parité et ainsi de suite jusqu'à ce que le bloc soit décodé ou qu'il n'y ait plus suffisamment de symboles de parité. Dans

6. Délai entre l'instant auquel un paquet perdu est initialement émis et l'instant auquel il est effectivement reconstruit et reçu par le destinataire.

7. L'application considérée génère un flux CBR ou VBR de débit inférieur à la capacité du lien.

8. Le plus souvent ce sont des turbo codes mais des codes LDPC ou Raptors sont également utilisés [50, 51].

ce cas, le bloc entier est retransmis (et éventuellement régénéré à partir des symboles sources) [44, 50, 52]. Les H-ARQ de type II sont bien entendu plus performant que ceux de type I ou que l'ARQ de base. Il sont intensivement utilisés dans les technologies sans fil telle que la 3GPP et Wimax mais moins répandus dans les couches hautes. On retrouve cependant des propositions d'utilisation de H-ARQ au niveau transport [53] ainsi qu'au niveau application (vidéo [54]).

2.5 Conclusion

Si le RTT est négligeable, les applications à contraintes de délais peuvent utiliser l'ARQ. Le cas échéant, elles peuvent utiliser des codes à effacement MDS étant donné que ces applications tolèrent souvent un taux de perte résiduel. Lorsque le débit (d *pkt/s*) est important devant la contrainte de délai D_{max} de l'application ($d \cdot D_{max} \geq 1000$), elle peut utiliser des codes LDPC (ou sans rendement), puisque la taille de bloc autorisée permet une probabilité de décodage élevée pour une complexité bien moindre.

Néanmoins, configurer ces codes de manière optimale, c'est à dire en protégeant l'application tout en minimisant la bande passante consommée, est en pratique une tâche délicate. En effet, les paramètres (n, k) constituent un compromis entre la robustesse du code (le taux de perte résiduel après décodage), le délai de décodage (en moyenne $k/2$) et son rendement. Augmenter la taille du bloc permet d'augmenter la capacité de correction sans augmenter le taux de redondance. La capacité de correction est maximisée pour $k \approx D_{max}/d$. Si l'application est de type VBR ou si le réseau a un RTT variable, la taille du bloc peut être trop grande et les paquets reçus après D_{max} seront considérés comme perdus. A l'inverse, la taille du bloc peut être trop petite et ne pas protéger au mieux l'application. Ainsi pour garder un codage optimal, il faut en permanence adapter ces derniers aux conditions du réseau.

Au chapitre suivant, nous proposons Tetrys, un code dont le seul paramètre est le taux de redondance et dont le comportement est similaire aux codes en bloc de configuration optimale tout en ne nécessitant pas de configuration dynamique. Contrairement aux codes en bloc dont le délai moyen de reconstruction est $k/2$ quel que soit le taux de perte, Tetrys permet de reconstruire les paquets manquant dès que possible sans devoir attendre la fin d'un bloc. De plus il permet une fiabilité totale et comble donc le fossé existant entre application temps réel et fiabilité totale.

Première partie

Tetrys

Chapitre 3

Le mécanisme de codage Tetrys

Contents

Le principe de codage que nous exposons ici est totalement différent des codes à effacement précédemment présentés, que ce soit les codes MDS, LDPC, sans rendement ou H-ARQ. Il est issu de deux travaux indépendants sur les codes à effacement et le *network coding*. Ils ont convergé pour produire un mécanisme de codage à la volée intégrant des informations retournées par le récepteur dans le processus de codage. Ces travaux sont les suivants :

1. Dans le contexte d'un transfert à plusieurs receveurs, Sundararajan [55] propose un mécanisme de codage qui utilise la voie retour en vue de diminuer la complexité de

l'encodage sans influer sur l'efficacité du transfert. Ce mécanisme leur permet de réduire la taille de leur fenêtre d'encodage et par effet de bord le délai de décodage. Leur étude se focalise sur le gain en terme de complexité et n'adresse pas le délai de décodage induit par les pertes observées sur les canaux de transmissions des différents récepteurs. La contribution majeure de leurs travaux est le concept d'acquittement partiel *ack-when-seen* qui, plutôt que d'acquitter les paquets chaque fois qu'ils sont décodés, acquitte les paquets de redondances lorsqu'ils apportent de l'information (*i.e.* lorsque la combinaison linéaire reçue assure le décodage futur d'un paquet de données). Notre proposition reprend ce concept d'acquittement partiel.

2. Dans le contexte de la communication unicast avec de larges délais, Lacan et Lochin ont proposé dans [56] un mécanisme de codage à la volée intégrant la voie retour. Le principe de base est l'ajout de paquets de redondance qui sont en fait la combinaison linéaire de l'ensemble des paquets envoyés mais pas acquittés. Le but était de permettre une fiabilité totale dans le contexte de Deep Space Networks (DSN) où le délai prévient l'utilisation des mécanismes ARQ standards. Notre proposition est directement dérivée de ce mécanisme.

Contrairement aux mécanismes de fiabilité présentés dans le chapitre précèdent, ces mécanismes de codage permettent de combler le fossé qui existe entre fiabilité totale et délai de décodage indépendant du RTT. Dans ce chapitre nous étudions en détail la caractéristique principale du mécanisme, à savoir, le délai de décodage des paquets perdus. Nous montrons qu'il peut être adapté aux besoins des applications et qu'il est indépendant du RTT.

Ce chapitre est structuré comme suit. La section 3.1 introduit puis détaille le fonctionnement de Tetrys combiné aux acquittements partiels. La section 3.1.3 propose un modèle du mécanisme et dérive ses caractéristiques principales que sont le délai de décodage et sa complexité. La section 3.1.7.1 étudie un autre aspect de la complexité qui est lié à la taille du corps fini requis par Tetrys. La section 3.2.1 (resp. 3.3) compare Tetrys, les codes en bloc et l'H-ARQ du point de vue de la robustesse de la configuration ainsi que du délai. Finalement la section 3.4 présente un mécanisme de configuration de Tetrys de manière à satisfaire les contraintes des applications quel que soit l'état du réseau.

P_i	Le i^{me} paquet source envoyé
$R_{(i..j)}$	Un paquet de redondance correspondant à une combinaison linéaire des paquets source allant de i à j : $R_{(i..j)} = \sum_{k=i}^{j} \alpha_k^{(i,j)} P_k$
k	Le nombre de paquets source entre la transmission de deux paquets de redondance
n	Le nombre total de paquets source et redondance pour chaque groupe de k paquets est égal à n (afin de garder la même notation que FEC) et est toujours égal à $k+1$ pour Tetrys
R	Le taux de redondance : $R = (n-k)/n = 1-(k/n)$ où (k/n) est le taux d'encodage
Δ_R	La différence entre le taux de redondance et le taux de perte : $\Delta_R = R - p$
p	Le taux de perte (PLR) courant
b	La taille moyenne des rafales de perte dans le cas d'un modèle de canal Gilbert Elliot. Cette valeur est égale à 1 pour un canal type Bernoulli. Ainsi ce paramètre permet de définir le canal utilisé
L_i	Le i^{me} paquet perdu
F_{sack}	La fréquence d'émission des acquittements
BS	La fenêtre d'encodage élastique composée de paquets source non encore acquittés
BR	Le buffer de réception où les paquets décodés ou reçus sont conservés jusqu'à ce qu'ils ne soient plus inclus dans la combinaison linéaire générant un paquet de redondance

TABLE 3.1: Notations utilisées.

3.1 Présentation du mécanisme

3.1.1 *Tetrys en bref*

Avant de plonger dans un descriptif plus exhaustif du mécanisme, considérons un simple exemple. L'émetteur Tetrys utilise une fenêtre d'encodage élastique noté BS qui inclut tous les paquets sources envoyés mais non acquittés. Soit P_i le paquet source de numéro de séquence i. Chaque k paquets sources donnés, l'émetteur envoit un seul paquet de redondance $R_{(i..j)}$ qui est une combinaison linéaire (avec des coéfficients aléatoires) de tous les paquets présents dans BS. Le récepteur est supposé acquitter périodiquement les paquets qu'il a reçus ou décodés. Chaque fois que l'émetteur reçoit un acquittement, il supprime les paquets acquittés de la fenêtre BS. Le récepteur peut décoder les paquets perdus dès que le rang du système linéaire, correspondant aux paquets de redondances disponibles, est supérieur ou égal au nombre de paquets perdus. En d'autres termes, dans la plupart des cas, le décodage est possible aussitôt que le nombre de paquets perdus est plus petit ou égal au nombre de paquets de redondances reçus.

Ceci implique que :

1. Tetrys est tolérant aux rafales de pertes de paquets sources, de paquets de redondance et des acquittements tant que le taux de perte (PLR) n'excède pas le taux de redondance ;

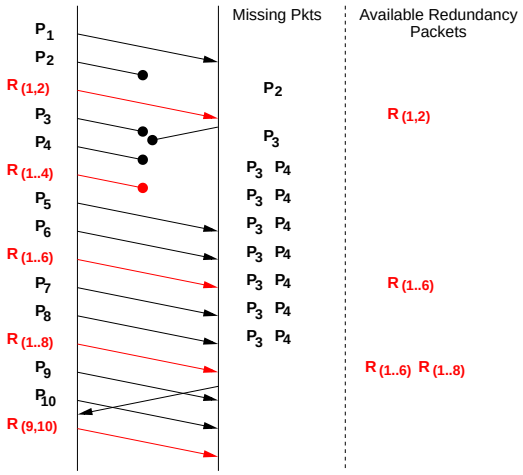


FIGURE 3.1: Exemple simple de transfert unidirectionnel avec acquittements pour un taux d'encodage de 2/3

2. Un paquet perdu est reconstruit dans un temps indépendant du RTT ce qui est très important pour les applications temps réel.

Supposons que la source ait envoyé les six paquets suivants (P_1, P_2, $R_{(1..2)}$, P_3, P_4, $R_{(1..4)}$). Par exemple, si P_2 est perdu, les paquets P_1 et $R_{(1..2)}$ permettent de récupérer P_2. Par contre, si P_1 et P_2 sont les seuls paquets perdus de la séquence, il faut attendre la réception du paquet de redondance supplémentaire (i.e. $R(1..4)$) afin de lui soustraire les paquets reçus P_3 et P_4 en effectuant $R'_{(1..4)} = R_{(1..4)} - \alpha_3^{(1..4)}.P_3 - \alpha_4^{(1..4)}.P_4$ pour obtenir $R'_{(1..4)}$ comme deuxième paquet de redondance nécessaire au décodage.

On obtient donc : $(R_{(1..2)}, R'_{(1..4)})^T = G.(P_1, P_2)^T$ où G est la matrice 2×2 :

$$G = \begin{pmatrix} \alpha_1^{(1..2)} & \alpha_2^{(1..2)} \\ \alpha_1^{(1..4)} & \alpha_2^{(1..4)} \end{pmatrix} \tag{3.1}$$

Ainsi, la reconstruction de (P_1, P_2) à partir de $(R_{(1..2)}, R'_{(1..4)})$ n'est possible que si G est inversible car si G^{-1} existe, il en résulte $(P_1, P_2) = G^{-1}.(R_{(1..2)}, R'_{(1..4)})$. La probabilité d'inversion de la matrice dépend du choix initial des coefficients de codage α. Ces coefficients sont choisis aléatoirement dans le corps fini $\mathbb{F}_{2^q}$ qui assure une probabilité d'inversion de cette matrice (et donc de décodage) très proche de 1 si l'on considère des tailles de corps suffisantes [57] [1].

La figure 3.1 illustre le fonctionnement de Tetrys avec des acquittements. Le taux de redondance est de 1/3 avec $n = 3$ et $k = 2$ et c'est donc un paquet de redondance qui est construit et envoyé tous les 2 paquets de données. Dans ce nouvel exemple, le paquet P_2 est perdu tandis que le paquet de redondance $R_{(1,2)}$ qui est correctement reçu

1. $\mathbb{F}_4$ i.e. des coefficients codés sur deux bits semblent êtres suffisant. L'impact de la taille du corps fini est discuté section *3.1.7.2*

permet de reconstruire P_2. Le paquet accusant la réception du paquet P_1 est perdu ce qui implique que P_1 continuera a être inclus dans les prochains paquets de redondance sans conséquence sur le devenir du transfert. Les paquets suivants, P_3, P_4 et $R_{(1..4)}$ sont perdus, contrairement au mécanisme H-ARQ. Aucun d'entre eux ne nécessite d'être retransmis puisqu'ils seront reconstruits grâce à la réception des paquets allant de P_5 à $R_{(1..8)}$. En supposant que l'application source soit à débit constant, qu'elle émette un paquet toutes les $10ms$ et que le délai de transit soit symétrique (égal à $100ms$, *ie.* un RTT de $200ms$) [2], le délai observé entre la génération de P_3 et sa réception effective serait de $160ms$ contre un minimum $320ms$ pour H-ARQ. Comme pour la matrice (3.1) le récepteur peut reconstruire P_3, P_4 en soustrayant dans un premier temps les paquets sources de manière à obtenir $(R'_{(1..6)}, R'_{(1..8)})$. L'étape suivante consiste alors en l'inversion puis la multiplication par $(R'_{(1..6)}, R'_{(1..8)})$ de la matrice 2×2. Enfin, cet exemple illustre bien que la voie retour n'est utilisée que pour diminuer la complexité d'encodage au niveau de la source. En effet, à la réception du second acquittement (le premier ayant été perdu), la source recommence son processus de codage à partir du paquet #9, ce qui permet à la fois de diminuer la complexité du codage et de s'assurer que les précédents paquets sources ont bien été reçus.

3.1.2 Principe général du mécanisme

3.1.2.1 Encodage

Un paquet de redondance est envoyé chaque k paquets source.

On notera par P_x le $x$ème paquet de données envoyé et P_1 le premier paquet. La taille en nombre d'octets Sz des paquets de données envoyés étant fixe, les données incluses dans P_x sont celles allant de $Sz*(x-1)$ à $(Sz*x-1)$ en supposant que la numérotation des octets commence à 0. $R_{(i..j)}$ représente le paquet de redondance qui contient une combinaison linéaire de tous les paquets source P_k avec k allant de i à j présents dans la fenêtre de l'émetteur BS. Chaque paquet de redondance est construit comme suit :

$$R_{(i..j)} = \sum_{k=i}^{j} \alpha_k^{(i,j)}.P_k$$

où les coefficients $\alpha_k^{(i,j)}$ appartiennent au corps fini $\mathbb{F}_q$ et la multiplication avec P_x est réalisée en considérant que chaque paquet est un vecteur d'élément de $\mathbb{F}_q$ [32]. De façon pratique, plutôt que de transmettre tous les coefficients avec le paquet de redondance

2. A noter que la figure 3.1 présente un diagramme de séquence symbolique sans axe des temps à l'échelle.

associé (ce qui potentiellement introduit un surcoût de transmission), nous utilisons un générateur de nombre aléatoire [58]) et transmettons seulement la graine (*seed*) utilisée.

La valeur R est directement liée au taux d'encodage et est égale [3] à $\frac{k}{k+1}$. Idéalement, ce paramètre devrait être ajusté de manière dynamique en fonction des conditions du réseau. Dans un souci de simplicité, nous choisirons un taux d'encodage fixé.

3.1.2.2 Décodage

Le décodage (i.e. la reconstruction des paquets manquants) consiste en la résolution d'un système d'équations linéaires du côté du récepteur. Les paquets sources disponibles (reçus ou décodés) sont stockés par le récepteur tant qu'ils sont utilisés par la source pour construire le prochain paquet de redondance $R_{(i..j)}$. Les paquets de redondance sont également stockés tant qu'ils peuvent être utilisés pour reconstruire un paquet manquant. Plus précisément, lorsqu'un nouveau paquet de redondance arrive, tous les paquets sources disponibles P_i .. P_j qui font partie de sa construction sont alors soustraits de $R_{(i..j)}$. On obtient alors $R_{(L_1..L_l)}$, où $(L_1..L_l) \in (P_i..P_j)$ est un sous-ensemble de paquets de la combinaison linéaire manquante.

Supposons que l paquets source $(L_1..L_l)$ sont manquants et que l paquets de redondance sont reçus et stockés dans BR. Soit R^i le i^{eme} paquet de l'ensemble des l paquets de redondance (par souci de lisibilité, cette notation ne fait pas mention des paquets source utilisés par la combinaison linéaire). On obtient :

$$(R^1, .., R^l)^T = G \cdot (L_1, .., L_l)^T$$

avec :

$$G = \begin{pmatrix} \alpha_{L_1}^{R^1} & .. & \alpha_{L_l}^{R^1} \\ . & .. & . \\ . & .. & . \\ \alpha_{L_1}^{R^l} & .. & \alpha_{L_l}^{R^l} \end{pmatrix} \tag{3.2}$$

où $\alpha_{L_j}^{R^i}$ est le coefficient utilisé pour encoder le j^{th} paquet perdu dans R^i. Si G est inversible, les paquets perdus $(L_1..L_l)$ sont reconstruits avec :

$$(L_1, .., L_l)^T = G^{-1} \cdot (R^1, .., R^l)^T$$

Une fois que le décodage est réalisé, tous les l paquets de redondance peuvent être supprimés de BR. Si la matrice G est singulière, le paquet de redondance qui possède

3. Étant donné que $n = k + 1$.

des coefficients qui sont linéairement dépendants est rejeté et le récepteur doit attendre un nouveau paquet de redondance pour retenter l'inversion.

Une solution possible pour améliorer la probabilité que cette matrice soit inversible est d'utiliser des matrices super-régulières [59]. Cependant la nature dynamique de Tetrys rend cette solution complexe à utiliser. Enfin, il est remarquable que l'utilisation de coefficients aléatoires permet à G d'avoir une forte probabilité d'être inversible si le corps fini est choisit suffisamment grand [57].

3.1.2.3 Paquet *seen*

Un paquet perdu est considéré comme "vu" (*seen packet* [60]) par un récepteur lorsqu'il a reçu une combinaison linéaire qui inclut ce paquet manquant (i.e. ce paquet manquant faisait donc partie de BS au moment ou le paquet de redondance a été créé). Même si un paquet *seen* ne peut pas être décodé immédiatement, le paquet de redondance reçu contient assez d'information pour permettre la future reconstruction d'au moins un d'entre eux (à condition bien sur que la matrice de décodage soit inversible). Ceci explique pourquoi un paquet *seen* acquitte le paquet source perdu de plus faible numéro de séquence comme si il avait été effectivement reçu.

3.1.2.4 Acquittements

Un récepteur envoie périodiquement des acquittements. Chaque acquittement contient une liste (sous la forme d'un vecteur SACK [47]) des paquets *seen* ou effectivement reçus ou décodés. Dès qu'il reçoit un acquittement, l'émetteur retire les paquets acquittés de sa fenêtre d'encodage BS. Ainsi, ces paquets sources ne seront plus inclus dans la combinaison linéaire générant le prochain paquet de redondance [60]. Ceci permet de réduire la complexité de calcul de l'encodage et du décodage.

Nous fixons une fréquence d'acquittement F_{SACK} en fonction du RTT courant :

$$F_{SACK} = \frac{1}{s \times RTT}$$

où s prend des valeurs type allant de 0.25 à 2. Bien que le choix de F_{SACK} n'ait pas d'impact sur la fiabilité du mécanisme, il y a un compromis à faire entre l'augmentation de F_{SACK} qui réduit la complexité de calcul de l'encodage et du décodage, le coût de la transmission et celui de la génération des acquittements.

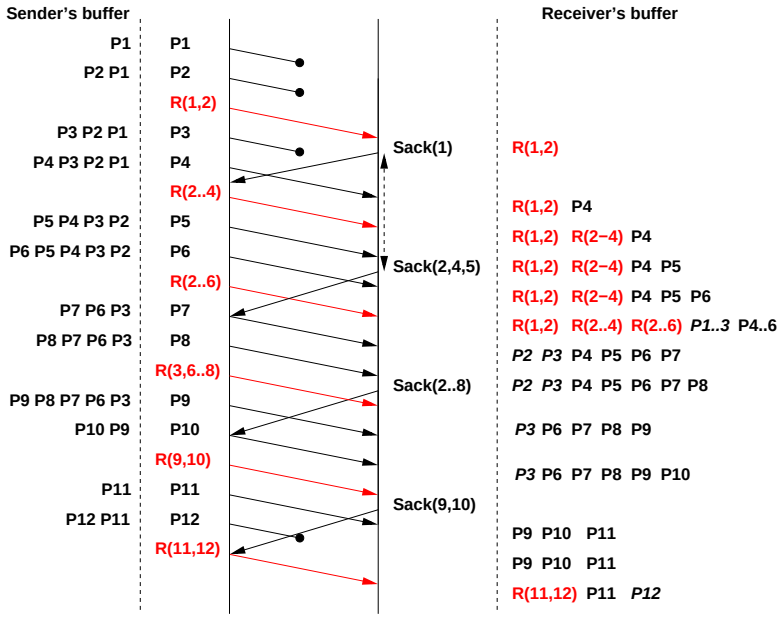


FIGURE 3.2: Un exemple plus complet avec des acquittements selectifs et des paquets *seen* ($k = 2$). Les paquets reconstruits sont en italique.

3.1.2.5 Un exemple complet

La figure 3.2 détaille un exemple plus complet dans lequel le récepteur renvoie des acquittements à une fréquence fixée à F_{sack}. Nous voyons que l'émetteur transmet les paquets P_1, P_2 et $R_{(1,2)}$. Puisque le paquet de redondance $R_{(1,2)}$ se trouve être le seul arrivé, le récepteur considère que P_1 et P_2 sont perdus ou en retard. Le récepteur acquitte P_1 puisque $R_{(1,2)}$ contient une combinaison linéaire de P_1 et est donc considéré comme un paquet *seen*. Comme déjà expliqué plus haut, chaque fois qu'un paquet de redondance est reçu, le récepteur peut acquitter un paquet source inclus dans la combinaison linéaire. L'émetteur poursuit sa transmission avec P_3 et P_4 avant de recevoir juste après l'acquittement pour P_1. Il va donc composer un nouveau paquet de redondance $R_{(2..4)}$ sans y inclure P_1 puisque acquitté. Le récepteur reçoit ensuite P_4 et $R_{(2..4)}$, signifiant que l'acquittement a bien été reçu par l'émetteur. Il génère alors un nouveau vecteur SACK qui acquitte P_2, P_4, P_5. Cependant, il ne peut toujours pas reconstruire P_1, P_2, P_3 puisqu'il ne possède pas encore assez de paquets de redondance. En conséquence, il stocke $R_{(1,2)}$ et $R_{(2..4)}$ pour une utilisation future. Puisqu'il n'y a plus de perte après cela, dès réception du troisième paquet le récepteur peut reconstruire les paquets manquants. Les paquets source inclus dans la combinaison linéaire sont soustraits ce qui donne $R_{(1,2)}, R'_{(2..4)}, R'_{(2..6)}$ tel que :

$$(R_{(1,2)}, R'_{(2..4)}, R'_{(2..6)})^T = G \cdot (P_1, P_2, P_3)^T$$

avec :

$$G = \begin{pmatrix} \alpha_{P_1}^{R_{(1,2)}} & \alpha_{P_2}^{R_{(1,2)}} & 0 \\ 0 & \alpha_{P_2}^{R_{(2..4)}} & \alpha_{P_3}^{R_{(2..4)}} \\ 0 & \alpha_{P_2}^{R_{(2..6)}} & \alpha_{P_3}^{R_{(2..6)}} \end{pmatrix} \tag{3.3}$$

où $\alpha_{P_z}^{R_{(i..j)}}$ est le coefficient utilisé pour l'encodage P_z du paquet de redondance $R_{(i..j)}$.

En supposant que G est inversible, on obtient G^{-1} grâce au pivot de Gauss-Jordan et P_1, P_2, P_3 sont donnés par :

$$(P_1, P_2, P_3)^T = G^{-1} \cdot (R'_{(1,2)}, R'_{(2..4)}, R'_{(2..6)})^T$$

Ces paquets peuvent être considérés comme décodés. Cependant, avant de les supprimer de BR, le récepteur doit attendre la réception de $R_{(3,6..8)}$ afin d'être certain que l'émetteur ne les utilisera plus pour construire un autre paquet de redondance.

Cet exemple permet d'illustrer plusieurs métriques importantes qui seront étudiées par la suite telles que : le temps de décodage, la taille des buffers émetteur et récepteur, le nombre d'opérations nécessaires au codage et décodage.

3.1.3 Modéle analytique

Nous proposons dans cette section un modèle Markovien de Tetrys qui sera utilisé pour l'étude du délai de décodage, de la taille de la mémoire requise et de la complexité en temps de l'algorithme.

Considérons dans un premier temps que les pertes suivent une loi de Bernoulli de paramètre p. Sous cette hypothèse, nous définissons la chaine de Markov $\{Y_n, n > 0\}$, qui représente la différence entre le nombre de paquets perdus et le nombre de paquets de redondance reçus à l'instant de réception n de chaque paquet de redondance. Les paquets perdus sont décodés lorsque $Y_j = 0$. Lorsque $\{Y_n, n > 0\}$ doit être actualisé, un bloc de $k + 1$ a potentiellement pu être reçu. Soit $X_{i,j}$ avec $i > 0$ et $0 \leqslant j \leqslant k$ la variable aléatoire caractérisant l'événement de perte du $j$éme paquet au sein du $i$éme bloc. Sur un canal à effacement type Bernoulli on a : $P[X_{i,j} = 1] = p$ lorsque le paquet est effacé et $P[X_{i,j} = 0] = 1 - p$ lorsqu'il est reçu. Les variables $X_{i,j}$, ou $0 \leqslant j \leqslant k - 1$, correspondent aux paquets de données et les variables $X_{i,k}$ correspondent aux paquets de redondance.

Enfin nous définissons la variable aléatoire X_i, avec $i > 0$, qui indique le degré de liberté apporté par le bloc (un paquet perdu apporte une variable donc un degré de liberté, tandis qu'un paquet de redondance reçu apporte une équation).

$$X_i = \sum_{j=0}^{k} X_{i,j} - 1 \tag{3.4}$$

ou encore $X_i = \sum_{j=0}^{k-1} X_{i,j} + (X_{i,k} - 1)$.

La figure .3.3 illustre le fonctionnement de X_i et la manière dont les paquets de données et de redondance y contribuent.

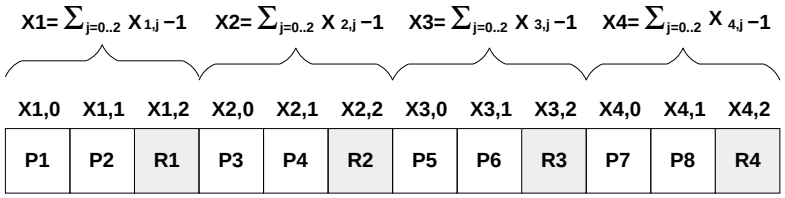


FIGURE 3.3: Illustration de la variable aléatoire X_i du degré de liberté apporté par le bloc i.

Etant donné que X_i est une somme de variable de Bernoulli : $P(X_i = u-1) = \binom{k+1}{u} p^u (1-p)^{k+1-u}$ avec $u = 0, \ldots, k+1$. De plus $\{Y_n, n \geqslant 0\}$ peut s'écrire de la façon suivante :

$$Y_n = \begin{cases} Y_{n-1} + X_n & \text{si } Y_{n-1} + X_n \geqslant 0 \\ 0 & \text{sinon} \end{cases}$$

ce qui nous permet de définir $a_{i,j} := P(Y_n = j | Y_{n-1} = i)$ la probabilité de transition de Y_{n-1} à Y_n .

Soit A la matrice $(a_{i,j})_{i,j \geqslant 0}$ et $a_{i,j}^{(n)}$ les entrées de A^n.

L'analyse de cette chaine montre que si le taux de redondance $R = 1/(k+1)$ est strictement supérieur à p, tous les paquets perdus sont récupérés en un temps fini. En effet, cette chaine est irréductible et $R > p$ puisque l'espérance de X_i notée $E(X_i)$ est négative. Enfin il peut être prouvé que l'état 0 est récurrent positif.

A l'inverse, si $R = p$, la chaine devient récurrente nulle ce qui implique que les paquets perdus seront décodés dans un délai moyen potentiellement infini. Etant donné les limitations de mémoire, ce cas n'est pas souhaitable en pratique. De surcroit, si $R < p$, les états deviennent transitoires et les paquets ne sont alors jamais décodés.

Nous nous focalisons sur le cas ou $R > p$. Etant donné que la chaine est irréductible et que 0 est un état récurrent positif, la distribution stationnaire existe pour les $P(Y_j = i)$

avec $i, j \geqslant 0$ et peut être obtenue par :

$$P(Y_j = i) = \lim_{n \to \infty} a_{j,i}^{(n)}$$

pour tout $i, j \geqslant 0$.

3.1.4 Délai de décodage

Afin d'étudier le délai de décodage, nous devons au préalable avoir la distribution du temps d'atteinte de l'état 0. Ce temps de première atteinte T_i est défini comme suit :

$$T_i = \{\min t \text{ tel que } Y_t = 0 | Y_0 = i\}$$

Obtenir la distribution du hitting time[4] n'est pas trivial. Nous utiliserons donc la méthode d'Aldous via la transformée en Z [61], lequel procède de la façon suivante :

Soit

$$G_i(z) = \sum_{t \geqslant 0} a_{i,0}^{(t)} z^t$$

et

$$F_i(z) = \sum_{t \geqslant 0} P(T_i = t) z^t$$

la fonction de densité de probabilité (p. g. f.) de T_i. D'après [61, chap. 2, lemma 25], on a :

$$F_i(z) = G_i(z)/G_0(z)$$

La distribution de T_i peut être déduite de $F_i(z)$ via l'évaluation de :

$$P(T_i = t) = \frac{1}{t!} \frac{d^t F_i(z))}{dz^t}|_{z=0}$$

Sachant que la chaine Y_t à la granularité d'un bloc (t correspond à la réception du dernier paquet du t-ème bloc), il faut l'affiner à l'échelle d'un paquet.

Considérons que le j-ème paquet du bloc i (c.f. figure 3.3) est perdu. Son délai de décodage noté D_j est de la forme $k - j + z(k + 1)$ puisque le décodage ne peut se faire qu'à l'instant de réception du paquet de redondance du z-ème bloc suivant. Il correspond au temps de premier retour de Y en 0 sachant qu'à la fin du bloc i il y a $y + u$ degrés de liberté. Or la probabilité qu'il y ait $y + u$ degrés de liberté sachant qu'au bloc $i - 1$

4. Temps à partir duquel l'on revient à l'état initial à partir d'un état i.

il n'y en avait que y correspond à [5] :

$$P(Y_i = y + u | Y_{i-1} = y) = \binom{k}{u} p^u (1-p)^{k-u}$$

avec $u = 0, \ldots, k$.

Il s'ensuit que :

$$P(D_j = k - j + z(k+1)) = \sum_{y \geqslant 0} \sum_{u=0}^{k} P(D_j = k - j + z(k+1), Y_{i-1} = y, Y_i = y + u)$$

Etant donné que les pertes sont indépendantes :

$$\begin{aligned} P(D_j = k - j + z(k+1)) = \textstyle\sum_{y \geqslant 0} \sum_{u=0}^{k} \cdot & P(D_j = k - j + z(k+1) | Y_i = y + u) \\ & P(Y_i = y + u | Y_{i-1} = y) P(Y_{i-1} = y) \end{aligned}$$

Sachant qu'au bloc i il y a $y+u$ degrés de liberté, la probabilité que le délai de décodage soit de z blocs peut s'exprimer à l'aide de la distribution T_{y+u} du temps de premier retour à 0 de Y :

$$P(D_j = k - j + z(k+1)) = \sum_{y \geqslant 0} \sum_{u=0}^{k} P(T_{y+u} = z) P(Y_i = y + u | Y_{i-1} = y) P(Y_{i-1} = y)$$

Sachant que le j-ème paquet est perdu, la valeur de la v.a. X est supérieure ou égale à 0 et donc :

$$P(Y_i = y + u | Y_{i-1} = y) = \binom{k}{u} p^u (1-p)^{k-u}$$

Ce qui nous permet d'écrire :

$$P(D_j = k - j + z(k+1)) = \sum_{y \geqslant 0} \sum_{u=0}^{k} P(T_{y+u} = z) \binom{k}{u} p^u (1-p)^{k-u} P(Y_{i-1} = y)$$

3.1.5 Taille (Z) des matrices à inverser

Le décodage de Tetrys comprend deux étapes. La première consiste à soustraire les paquets sources reçus des paquets de redondance puis, lorsque $Y_i = 0$, à construire et inverser la matrice G. Les algorithmes d'inversion de matrices sont de complexité cubique en fonction du nombre de pertes. Contrairement aux autres codes à effacement, la taille

5. On rappel que si le j-ème paquet est perdu, la valeur de la v.a. X est supérieure ou égale à 0.

de cette matrice, que nous noterons Z, n'est pas bornée et l'étude de la complexité du décodage passe donc par l'étude du nombre de pertes à corriger entre chaque passage à $Y_i = 0$.

Le temps (en nombre de paquets émis) qui s'écoule entre chaque passage entre $Y_i = 0$ est nommé temps de récurrence. Pour obtenir la taille de la matrice à inverser nous utiliserons le fait qu'elle correspond au nombre de paquets de redondance reçus pendant un temps de récurrence.

Soit F la variable aléatoire qui correspond à la position j du premier paquet perdu dans son bloc i (e.g. avant qu'il ne soit perdu $Y_{i-1} = 0$). $P(F = j) = p(1-p)^j/(1-(1-p)^k)$, avec $j = 0, \ldots, k-1$. Sachant que la première perte à lieu à la position j du i-éme bloc, la probabilité qu'il y ait u pertes dans le bloc i est $P(Y_i = u|F = j) = \binom{k-j}{u} p^u (1-p)^{k-j-u}$.

Le temps de récurrence, noté U, est de la forme $k - j + z(k+1)$, où z correspond au nombre de blocs reçus pendant un temps de récurrence (le bloc i n'est pas pris en compte).

On a donc :

$$\begin{aligned} P(U = k-j+z(k+1)) &= \textstyle\sum_{u=0}^{k} P(U = k-j+z(k+1), Y_i = u|F = j)P(F = j) \\ &= \textstyle\sum_{u=0}^{k} P(D_j = k-j+z(k+1)|Y_i = u)P(Y_i = u|F = j)P(F = j) \\ &= \textstyle\sum_{y\geqslant 0} \sum_{u=0}^{k} P(T_u = z)P(Y_i = u|F = j)P(F = j) \\ &= \textstyle\sum_{y\geqslant 0} \sum_{u=0}^{k} P(T_u = z)\binom{k-j}{u} p^u (1-p)^{k-j-u} p(1-p)^j/(1-(1-p)^k) \end{aligned}$$

On sait que durant un temps de récurrence $U = k - j + z(k+1)$, $z+1$ paquets de redondance sont envoyés et il y a donc entre 1 et $k+1$ paquets de redondance reçus.

Sachant que le paquet de redondance du dernier bloc à été reçu et que les pertes sont indépendantes, la probabilité d'avoir reçu i paquets de redondance sachant U est :

$$P(Z = i|U = k-j+z(k+1)) = \binom{z}{i} p^{z-i}(1-p)^i$$

Finalement :

$$P(Z = i) = \textstyle\sum_{z\geqslant i} \sum_{j=0}^{k-1} P(Z = i|U = k-j+z(k+1))P(U = k-j+z(k+1))$$

3.1.6 Taille de buffers

De la même manière que pour la taille des matrices, Tetrys implique que la taille des buffers n'est pas bornée, aussi bien chez l'émetteur que chez le récepteur. Nous évaluons ici la taille de ces différents buffers.

3.1.6.1 Coté émetteur

Notons BS_t le nombre de paquets stockés par la source à l'instant t (le nombre de paquets émis mais pas acquittés). Cette valeur se décompose en trois parties respectivement relatives au RTT, à la fréquence d'envoi des blocs SACK ainsi qu'au nombre de pertes. Le RTT correspond au temps aller-retour en termes de nombre de paquets envoyés (nous considérons un trafic type CBR [6]).

Les acquittements (SACK) contiennent des informations relatives à des paquets émis un RTT plutôt. Durant cette période, l'émetteur doit donc stocker RTT paquets.

Notons S_1 le temps (en nombre de paquet émis) écoulé depuis la réception du dernier SACK. Les SACK sont émis par le destinataire avec une fréquence $s.RTT$ et perdus avec une probabilité p. L'espérance de S_1 est donc :

$$E(S_1) = s.RTT(1/2 + 1/(1-p))$$

On suppose pour cela que les t soient uniformément répartis au sein de l'intervalle $s.RTT$ (d'où le facteur $1/2$) et $1/(1-p)$ à l'espérance de la période d'arrivée des SACKs.

Finalement, à l'instant où ont été générés les paquets relatifs au dernier SACK ($t - S_1 - RTT$), des paquets sources n'étaient pas acquittés puisqu'ils étaient perdus. Par l'utilisation du mécanisme *ack-when-seen* (lequel est inclus dans les SACK), chaque paquet de redondance reçu permet d'acquitter un paquet perdu. L'espérance du nombre de paquets considérés comme perdus à l'instant $(t - S_1 - RTT)$ correspond à l'espérance du nombre du degré de liberté $E(Y_n)$ 3.1.4.

Le nombre moyen de paquets contenus dans BS est donc :

$$E(BS_t) = RTT(1 + s/2 + s/(1-p)) + E(Y_n)$$

Y_n étant négligeable dans le contexte des applications temps réel, on peut observer que $E(BS_t)$ évolue linéairement en fonction du RTT.

6. Constant Bit Rate

3.1.6.2 Coté récepteur

Le récepteur à deux buffers : BRS qui contient les paquets sources (reçus ou décodés) potentiellement utiles pour les prochains décodages ; BRR pour les paquets de redondance, lequel est vidé à chaque décodage. Le nombre de paquets contenus dans le buffer source (resp. redondance) à l'instant t est noté BRS_t (resp. BRR_t).

Pour rappel, lorsque qu'un paquet source est reçu, il est acquitté par les blocs SACK suivants. Quand l'émetteur reçoit le premier SACK accusant un paquet, il est supprimé du buffer source BS et n'est plus inclus dans les paquets de redondance qui suivent. Quand le récepteur reçoit un paquet de redondance, il supprime de BRS tous les paquets qui ne sont plus encodés dans la redondance.

La figure 3.4 montre que les paquets sources sont inclus dans dans BRS durant $S_2 + S_3 + RTT$.

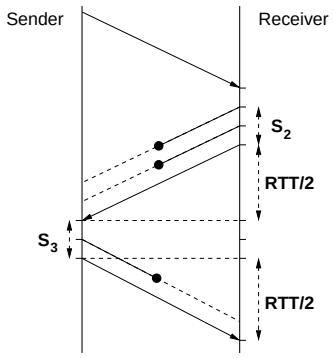


FIGURE 3.4: Impact des acquittement sur la taille de buffer

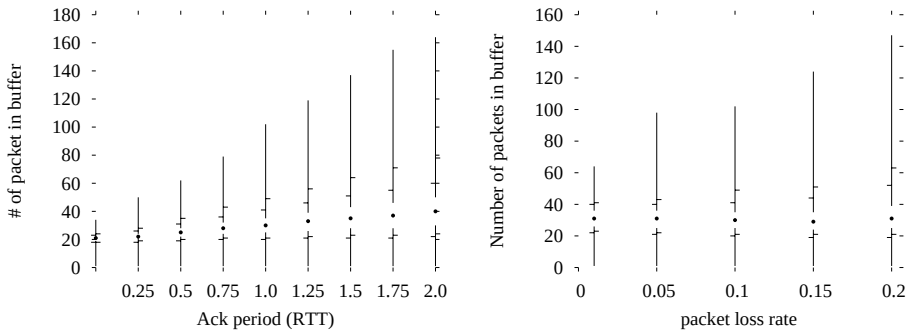
Il est clair que S_2 correspond à S_1. S_3 peut être obtenu de la même façon à l'exception que les paquets de redondance sont envoyés tous les k paquets source et perdus avec une probabilité p :

$$E(S_3) = k(1/2 + 1/(1-p))$$

Le temps moyen de résidence d'un paquet dans BRS est donc :

$$E(S_2 + S_3 + RTT) = RTT + (k + s.RTT)(1/2 + 1/(1-p))$$

Pour obtenir $E(BRS_t)$, il faut simplement considérer la probabilité que certains d'entre eux soient perdus :



(a) Nombre de paquets contenus dans BRR et BRS en fonction de la fréquence des acquittements PLR=0.1

(b) Nombre de paquets contenus dans BRS en fonction du PLR avec un acquittement par RTT

FIGURE 3.5: Minimum, maximum et (5, 10, 25, 50, 75, 90, 95) percentiles des tailles de buffer requises pour décoder avec un taux de redondance de 1/4

$$\begin{aligned} E(BRS_t) &= (1-p)E(RTT + S_2 + S_3) \\ &= (1-p)(RTT + (k + s.RTT)(1/2 + 1/(1-p))) \\ &= (1-p)RTT + (k + s.RTT)((1-p)/2 + 1) \end{aligned}$$

Pour estimer BRR_t, nous étudions la probabilité qu'il soit vide (c.a.d. $P(Y_n = 0)$), puis Z, la taille de la matrice à inverser qui va correspondre au nombre de paquets de redondance dans BRR_t.

Les pertes étant uniformément distribuées, l'occupation moyenne de BRR durant une récurrence $U = k - j + z(k+1)$ 3.1.4 correspond à $(1-p)z/2$. Il en découle que la taille moyenne correspond à :

$$E(BRR_t) = \frac{\sum_{z>0}(1-p)z \sum_{j=0}^{k-1}(k-j+z(k+1))P(U=k-j+z(k+1))}{2.P(Y_n=0)}$$

Étant donné que lorsque R tend vers p, la probabilité $P(Y_n = 0)$ tend vers 0 la taille de buffer requise par le récepteur tend vers l'infinie nécessite donc la mise en place d'un contrôle de flux.

3.1.7 Evaluation des tailles de buffer par simulation

Nous évaluons maintenant les tailles de buffer requises par notre implémentation. Dans la section suivante, ces résultats nous permettrons d'avoir une idée de la complexité du mécanisme.

Seul les buffers du récepteur sont étudiés étant donné qu'ils sont plus grand que ceux de l'émetteur. Les expérimentations sont faites avec des pertes uniformes[7], un RTT de 200ms, un taux de redondance de 3/4 et un débit de 100 paquets sources par seconde. Les deux paramètres qui peuvent affecter les tailles de buffer sont le rapport entre le taux de perte et le taux de redondance ainsi que la fréquence des acquittements. La figure 3.5(a) présente l'impact de la fréquence des acquittements lorsque le taux de perte est de 10%. De manière à illustrer les variations inhérentes au temps de récurrence, les valeurs minimum, maximum et les (5, 10, 25, 50, 75, 90, 95) percentiles des tailles de buffers sont affichées. Les valeurs utilisées pour calculer la distribution sont relevées à chaque réception d'un paquet de redondance.

On peut voir sur la figure 3.5(a) que la moyenne des tailles de buffer évolue bien linéairement en fonction de la fréquence des acquittements et confirme l'expression de BRS.

L'autre paramètre à étudier est le taux de perte et ce, de par le fait que le délai de décodage augmente lorsqu'il se rapproche du taux de redondance. La figure 3.5(b) présente les résultats pour un taux de perte variant de 1% à 20% avec un acquittement par RTT. On remarque que les (5, 10, 25, 75, 90, 95) percentiles restent très proches du 50-éme percentile lequel ne varie d'ailleurs que très peu en fonction du taux de perte.

3.1.7.1 Complexité en temps

En termes de complexité en temps, on utilisera comme unité de base le temps requis pour l'addition d'un paquet de données à un paquet de redondance. Du coté émetteur, la complexité pour l'encodage d'un paquet de redondance sera donc en $O(BS_t)$ avec BS_t le nombre de paquets de données présents dans la fenêtre d'encodage. Cette quantité qui est au minimum égale au produit du RTT et du débit d'émission est directement liée à la fréquence d'envoi d'acquittements ainsi qu'au taux de perte sur la voie retour. Du coté récepteur, la complexité va dépendre du nombre de pertes L à reconstruire. Ce nombre de paquets étant égal au nombre de paquets de redondance impliqués dans l'opération de décodage, il faut compter de l'ordre de $L * BS_t$ opérations pour l'étape de soustraction. Quant à l'inversion et le produit matriciel, ils sont de l'ordre de L^2 opérations. Un décodage complet (pour les L paquets) est donc en $O(L * BS_t + L^2)$.

7. Les résultats sont du même ordre de grandeur avec des pertes en rafales.

3.1.7.2 Impact de la taille du corps fini

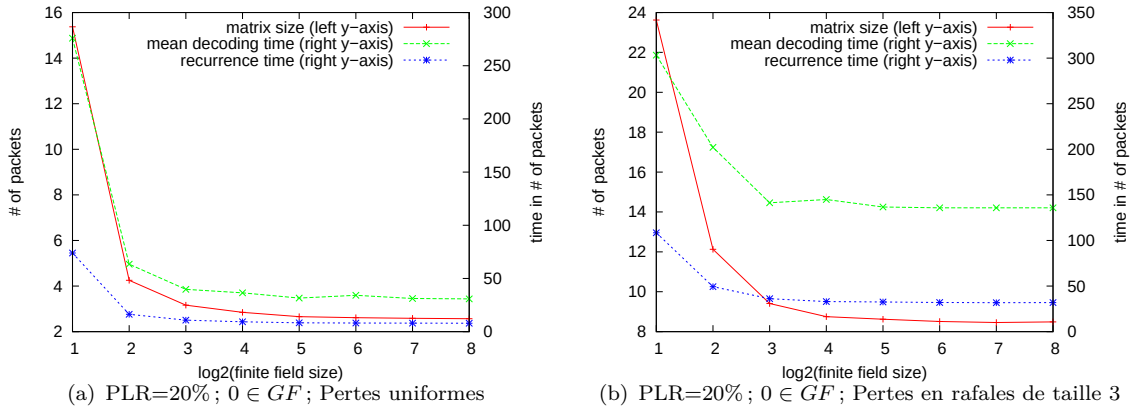
Il a été souligné dans la section 3.1 que le décodage d'une matrice est possible lorsque le nombre de paquets de redondance reçus est égal au nombre de pertes à corriger sous condition que la matrice soit inversible (ce qui n'est évidemment pas toujours le cas). Lorsque la matrice n'est pas inversible, cela signifie qu'il existe une ou plusieurs colonnes (paquets de redondance) qui sont des combinaisons linéaires des précédentes. L'algorithme de Gauss-Jordan permet d'identifier le paquet de redondance à l'origine de cette dépendance linéaire et ce dernier est supprimé. Ceci nécessitant donc d'attendre un nouveau paquet de redondance afin de retenter l'opération d'inversion. Le fait de devoir attendre un ou plusieurs paquets de redondance nuit au temps de décodage et donc aux performances du mécanisme. Augmenter la probabilité que la matrice soit inversible revient à augmenter le nombre d'éléments différents qui peuvent être choisis lors du tirage des coefficients et donc la taille du corps fini. Or la complexité du code est directement liée à la taille du corps fini puisque d'après [62], la multiplication peut être implémentée par $m^2/2$ opérations de XOR avec m le nombre de bits utilisés pour coder le coefficient.

Nous étudions donc l'impact du corps fini sur les performances du mécanisme qui se caractérisent par le temps de récurrence, le temps de décodage et la taille de la matrice. La figure. 3.6 montre l'évolution de ces métriques en fonction de GF pour un taux de perte de 20% et un taux de redondance de 25%. La sous figure de gauche correspond aux pertes uniformes tandis que celle de droite correspond à des pertes en rafales de taille moyenne 3.

On peut voir que les performances obtenues par les deux plus petites tailles de corps fini ($\mathbb{F}_2$ and $\mathbb{F}_4$) sont significativement moins bonnes que pour les autres tailles supérieures. De plus, on peut voir que la distribution des pertes n'a pas d'impact sur ces observations et ce, quelque soit la taille moyenne des pertes considérées (et du taux de pertes). Les performances sont stables à partir de $\mathbb{F}_8$.

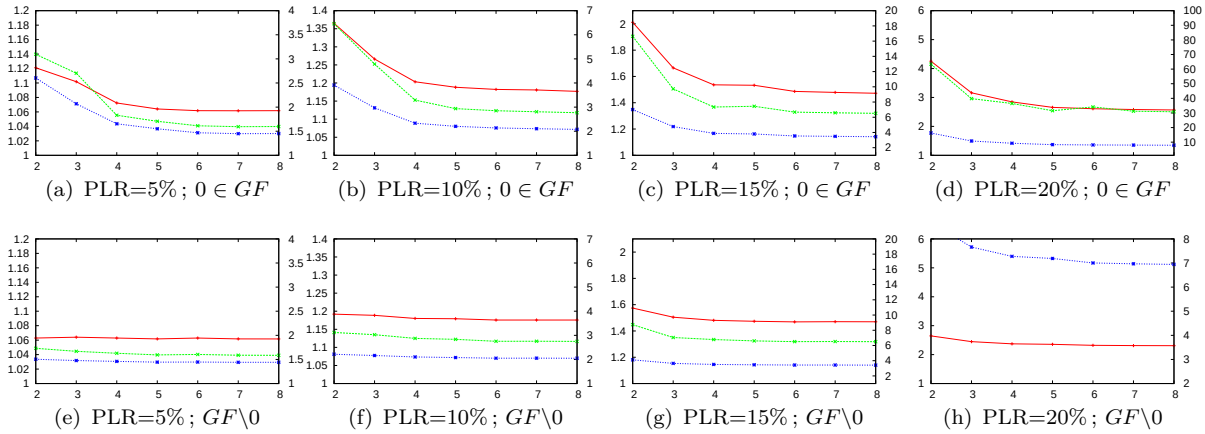
Bien que l'utilisation du corps binaire soit attractif du fait qu'il n'effectue qu'un simple XOR, l'impact sur la probabilité d'inversion est tel qu'il n'est pas avantageux de l'utiliser pour Tetrys.

Impact de 0 comme coefficient : jusqu'à maintenant, les coefficients étaient choisis aléatoirement dans $GF(2^x) = \{0, 1, 2, ..., 2^x - 1\}$. A priori la logique voudrait que l'on choisisse les coefficients parmi un grand ensemble car plus forte sera la diversité et la probabilité d'inverser la matrice. Il se trouve cependant que se priver de l'élément nul permet d'améliorer sensiblement les performances. En effet, pour $GF(4)$ un paquet à une chance sur 4 d'être encodé avec 0 comme coefficient dans un paquet de redondance.



(a) PLR=20% ; $0 \in GF$; Pertes uniformes (b) PLR=20% ; $0 \in GF$; Pertes en rafales de taille 3

FIGURE 3.6: Impact de la taille du corps fini sur la taille de la matrice à inverser, le temps de récurrence et le délai de décodage moyen.



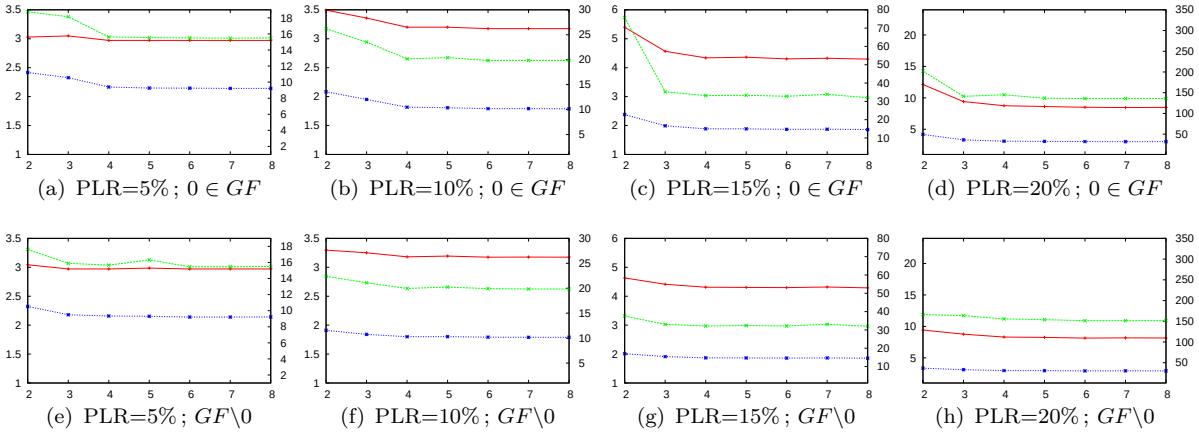
(a) PLR=5% ; $0 \in GF$ (b) PLR=10% ; $0 \in GF$ (c) PLR=15% ; $0 \in GF$ (d) PLR=20% ; $0 \in GF$

(e) PLR=5% ; $GF\backslash 0$ (f) PLR=10% ; $GF\backslash 0$ (g) PLR=15% ; $GF\backslash 0$ (h) PLR=20% ; $GF\backslash 0$

FIGURE 3.7: Impact du coefficient zéro et de la taille du corps fini sur la taille de la matrice à inverser, le temps de récurrence et le délai de décodage moyen. Les pertes sont uniformes

Cela signifie que si il n'y a qu'un seul paquet à décoder, il a une chance sur 4 qu'il ne puisse pas l'être. Cette remarque reste vraie pour toutes les matrices dont le dernier paquet aurait été encodé avec un 0. La figure 3.7 compare les résultats obtenus lorsque les coefficients sont choisis dans GF (ligne du haut) ou dans $GF\backslash 0$(ligne du bas). Les pertes sont uniformes et varient 5% à 20% pour un taux de redondance de 25%. Pour des raisons de meilleure lisibilité, les légendes ne sont pas reproduites sur chacune des sous-figures mais restent les même que la figure 3.6.

On peut voir sur la figure 3.7 Tetrys sur $GF(2)$ ou $GF(4)$ est bien plus efficace lorsque zéro ne fait pas partie des coefficients. Pour des taux de perte de 5% et 10%, il n'y a même aucune différence avec les autres tailles de corps. Cela s'explique par le fait que



(a) PLR=5% ; $0 \in GF$ (b) PLR=10% ; $0 \in GF$ (c) PLR=15% ; $0 \in GF$ (d) PLR=20% ; $0 \in GF$

(e) PLR=5% ; $GF \backslash 0$ (f) PLR=10% ; $GF \backslash 0$ (g) PLR=15% ; $GF \backslash 0$ (h) PLR=20% ; $GF \backslash 0$

FIGURE 3.8: Impact du coefficient zéro et de la taille du corps fini sur la taille de la matrice à inverser, le temps de récurrence et le délai de décodage moyen. Les pertes sont en rafales de taille 3.

les matrices à inverser sont souvent de taille 1 ou 2 ce qui réduit la probabilité d'avoir une matrice non inversible.

La figure 3.8 présente les même résultats pour des pertes en rafales de taille 3 en moyenne. On observe également que Tetrys sur $GF(4)$ ou $GF(8)$ est plus performant lorsque zéro ne fait pas partie des coefficients.

En conclusion, nous pouvons dire que Tetrys permet de générer des symboles de redondance à la volée (Rateless) et que ces derniers sont quasi MDS pour des tailles corps supérieures ou égales à $GF(4)$. Cela permet une implémentation efficace (2 XOR par multiplication).

3.2 Codes en bloc et robustesse de la configuration

Nous avons souligné le fait qu'un des avantages de Tetrys est qu'il permet une fiabilité totale avec un délai de récupération des paquets indépendants du RTT. Cependant, les applications temps réel ont des besoins différents étant donné qu'elles supportent un certain nombre de pertes tout en nécessitant que les paquets arrivent avant un délai. Dans ce contexte, cette section discute des aspect de configuration de Tetrys et des FEC en bloc. En particulier nous verrons que la configuration de Tetrys (contrôlé par un unique paramètre) demeure bien plus robuste aux variations des conditions du réseaux.

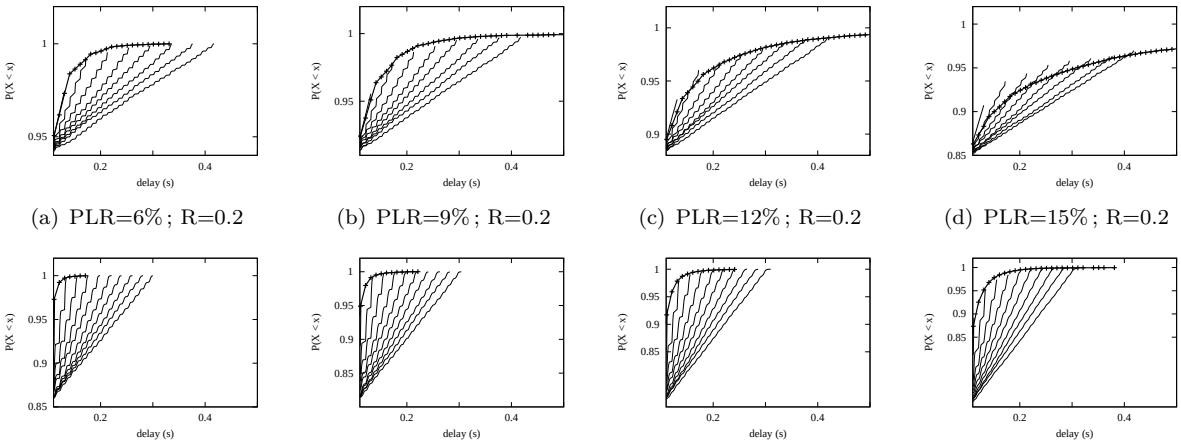
Le code FEC considéré dans cette évaluation est un Reed-Solomon systématique [32] qui envoie l'ensemble des paquets de données dès que l'application (un CBR) les lui

transmet. On considère également que les $n - k$ paquets de redondance sont générés et envoyés instantanément après l'émission du dernier paquet de donnée du bloc et que la bande passante est suffisamment large pour absorber cette rafale de paquets sans les perdre, ou ajouter du délai lié à la file d'attente d'accès au lien. Les paquets perdus sont récupérés et transmis à l'application dès que k paquets du bloc ont été reçus et le délai ainsi induit dépend du paramètre k.

Dans des conditions réelles, le taux de perte n'est pas constant et au cours du temps, la taille du bloc aussi bien que le taux de redondance doivent être adaptés. Cette adaptation se fait sur la base des informations retournées par le ou les récepteurs. Les paramètres choisis à un instant donné ont donc un RTT de retard par rapport aux conditions pour lesquelles ils sont optimaux. Il en résulte qu'en pratique, l'utilisation de FEC n'est pas optimale.

A l'opposé, les paquets de redondance de Tetrys sont répartis le plus régulièrement possible parmis les paquets de donnée et sont exploitables indépendamment les uns des autres, ce qui permet de ne pas avoir à attendre la fin d'un bloc de taille fixé et minimise le délai.

3.2.1 Evaluation du temps de décodage



(a) PLR=6% ; R=0.2 (b) PLR=9% ; R=0.2 (c) PLR=12% ; R=0.2 (d) PLR=15% ; R=0.2

(e) PLR=15% ; R=0.5 (f) PLR=20% ; R=0.5 (g) PLR=25% ; R=0.5 (h) PLR=30% ; R=0.5

FIGURE 3.9: Distribution cumulative du délai de livraison de paquets à l'application pour Tetrys (courbe en gras) et FEC (courbe en escalier) pour differents taux de perte et un RTT de $200ms$. Le taux de redondance R est soit de 0.2 (première ligne) soit de 0.5 (seconde ligne). Pour FEC, la taille des bloc est k=$\{4\,;\,8\,;\,12\,;\,16\,;\,20\,;\,24\,;\,28\,;\,32\}$ (resp. k=$\{2\,;\,4\,;\,6\,;\,8\,;\,10\,;\,12\,;\,14\,;\,16\,;\,18\,;\,20\}$) pour la première ligne (resp. la seconde).

Nous illustrons les avantages précédemment cités, via l'étude de la distribution de délai de Tetrys et FEC pour différents taux de perte et taux de redondance.

La figure 3.9 montre les résultats pour $R = 0.20$ (première ligne) ou $R = 0.5$ (seconde ligne) avec un taux de perte qui augmente pour chaque sous figure en partant de la gauche pour finalement approcher le taux de redondance. Pour chaque sous figure, la distribution est tracée pour les différentes tailles de bloc (la valeur de n) ce qui nous permet d'identifier aisément la taille de bloc optimale et d'y comparer Tetrys.

Ainsi, il peut être observé que :
- dans l'ensemble des mesures effectuées, Tetrys permet une fiabilité totale puisque la distribution atteint 1.0, ce qui n'est pas le cas de FEC (en particulier pour les petites tailles de bloc).
- la probabilité que Tetrys transmette un paquet à l'application avant un délai donné est supérieure à celle de la plupart des FEC considérés (la courbe de Tetrys est au dessus de celle des FEC). De plus, il n'y a que lorsque FEC ne corrige pas toutes les pertes que son délai de récupération est inférieur à celui de Tetrys (comme on peut l'observer sur la figure. 3.9(d)).

Ces résultats restent valides pour la plupart des taux de perte (e.g. figures. 3.9(a), 3.9(b), 3.9(c), pour lesquels le PLR varie de 6% à 12%). Par contre, lorsque le taux de perte se rapproche du taux de redondance (e.g. figure. 3.9(d), avec $PLR = 15\%$ et $R = 20\%$), comme l'indique le modèle de Tetrys, le temps de récurrence augmente et le délai de reconstruction dépasse celui de FEC qui rappelons le, ne reconstruit pas l'ensemble des paquets. Il apparait clairement sur la figure. 3.9(d) que le taux de redondance de Tetrys doit malgré tout être adapté en fonction du taux de perte si l'on veut que les paquets soient décodés avant un certains délai. Dans la section 3.4, nous proposons un mécanisme qui permet à Tetrys d'adapter ses paramètres de codage de manière à satisfaire les besoins exprimés par l'application tout en minimisant la bande passante consommée par le flux.

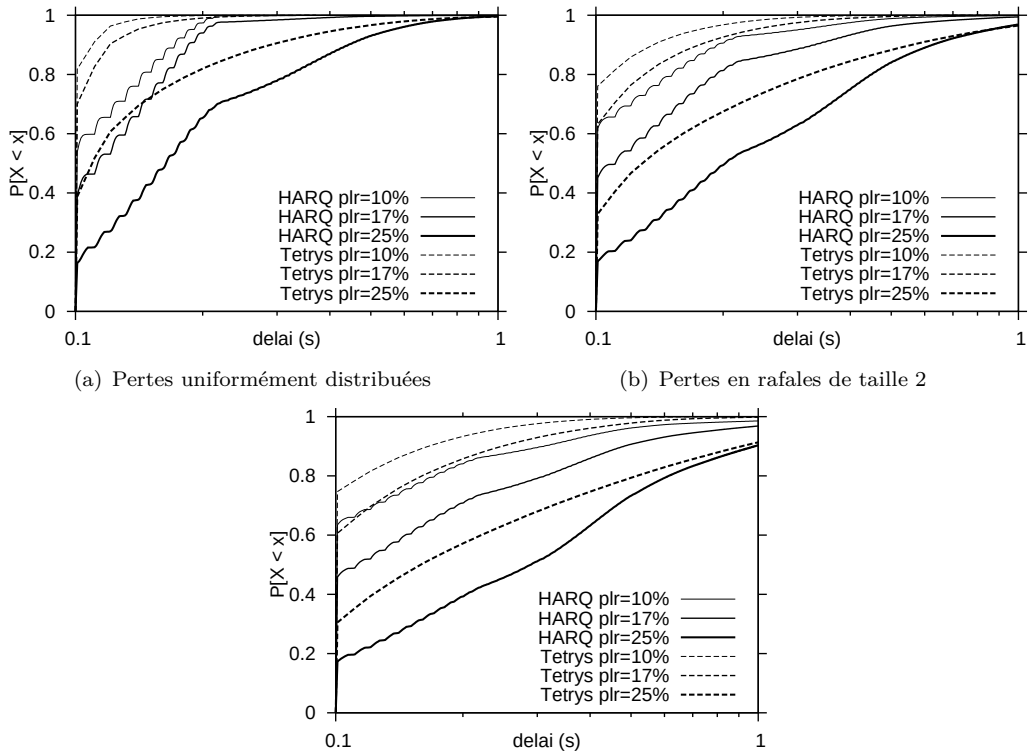
A taux de redondance égale et pour un compromis délai de décodage/robustesse donné, Tetrys est aussi performant que la FEC de paramètre optimal [8]. Cela lui permet de garder une configuration optimale quelle que soit la distribution des pertes dans le réseau. Son taux de redondance ne nécessite une adaptation que lorsque le taux de perte se rapproche du taux de redondance courant. Ce problème est adressé dans la section 3.4.

3.3 Comparaison à H-ARQ type II

Dans cette section, nous comparons Tetrys à un autre mécanisme qui fait intervenir les codes à effacement et permet la fiabilité totale. Nous utilisons l'H-ARQ de type II

8. Il existe une configuration optimale de FEC qui décode la même proportion de paquets que Tetrys avant le délai cible

décrit dans la section 2.4. Chaque fois qu'un bloc ne peut être reconstruit, le récepteur envoie un acquittement signifiant le nombre de paquet manquant. Les nombre de symboles redondants retransmis correspond alors au nombre de paquets rapportés comme manquant [63]. Comme pour le code en bloc considéré dans la section précédente, les paquets de redondance sont générés et envoyés instantanément après l'émission du dernier paquet du bloc et les retransmissions de l'ARQ ne retardent pas l'émission des autres paquets. Les paquets perdus sont instantanément détectés et un acquittement négatif est immédiatement émis.



(a) Pertes uniformément distribuées

(b) Pertes en rafales de taille 2

(c) Pertes en rafales de taille 3

FIGURE 3.10: CDF du délai de livraison des paquets en ordre pour Tetrys et H-ARQ avec un taux de redondance de 1/3, un RTT de $200ms$ et un débit de 10 paquets par seconde

La métrique utilisée est le délai entre l'instant d'émission du paquet par la source et l'instant où il est transmis par le mécanisme de fiabilité (dans l'ordre) à la couche supérieure. Les figures 3.10(a), 3.10(b) et 3.10(c) présentent la fonction de répartition (CDF) des délais de chaque paquet avec une échelle logarithmique pour l'axe des abscisses. Ce choix de représentation se justifie par le fait que les applications sont différemment impactées par les variations de délai et une simple moyenne n'aurait pas été suffisamment descriptive. Les paramètres de simulation utilisés sont les suivants : un taux de redondance de 1/3 ; chaque simulation dure 1000 secondes correspondant à une réception de 10^5 paquets ; les pertes sont respectivement distribuées de manière uniforme et par rafale de

moyenne 2 et 3 pour les figures 3.10(a), 3.10(b) et 3.10(c). Pour chacune de ces figures, la distribution des délais est présentée pour H-ARQ et Tetrys et pour des taux de perte de 10%, 17% et 25%. On constate qu'à délai égal, la probabilité d'être inférieur à ce dernier est toujours significativement supérieure pour Tetrys. Cela reste vrai lorsque l'on compare Tetrys avec 17% de perte et H-ARQ avec 10% de perte. A titre d'exemple, avec un taux de perte de 17% et des pertes uniformément distribuées, la probabilité d'obtenir un délai inférieur à $150ms$ est de 67% pour H-ARQ contre 94% pour Tetrys.

Lorsque le taux de perte se rapproche du taux de redondance, le temps de récurrence est tel que les délais induits par Tetrys seront supérieurs à ceux d'H-ARQ. Il faut néanmoins rappeler que le nombre de paquets generés par H-ARQ est supérieure à celle de Tetrys de part la nécessité de générer des acquittements et des retransmissions. Si Tetrys s'autorise la génération de ces paquets additionnels, il peut facilement être montré que le délai de Tetrys est toujours inférieur ou égale à celui d'HARQ.

3.4 Choix des paramètres de codage

Les applications "temps réel" (VoIP, vidéo-conférence...) requièrent que tous, ou alors une certaine proportion des paquets, arrivent à la dite application avant un délai maximum D_{max} pour être utilisable et fournir à l'utilisateur la qualité de service espérée. Le problème à résoudre est donc le suivant :

(Problème) : Etant donné D_{max} le délai en dessous duquel une proportion $P_{min}\%$ des paquets doit arriver pour que l'application ait la qualité requise par l'utilisateur, nous cherchons les paramètres du mécanisme de fiabilité Tetrys qui permettent de satisfaire ces contraintes tout en minimisant la charge induite sur le réseau (i.e. envoyer le moins de redondance possible).

Les performances du mécanisme Tetrys (et donc celles perçues par l'application) sont elles même sujettes aux paramètres environnants dont les plus importants sont :
- d, le délai de bout en bout.
- T la période d'émission des paquets source (le débit correspond à $1/T$).
- p, le taux de perte des paquets dans le réseau (plr).
- b, la taille moyenne des rafales de pertes dans le réseau. Par commodité, $b = 1$ signifie un canal de Bernoulli, sinon il est de type Gilbert Eliot.
- R, le taux de redondance qui sera égale à $1/n$ avec $n-1$ le nombre de paquets source envoyés entre la construction et l'envoi de deux paquets de redondances.

Plus concrètement, le problème revient donc à trouver une fonction qui retourne R_{min} le taux de redondance minimal qui satisfasse les contraintes de l'application. Cette fonction

requière au préalable de connaître la distribution des délais (de chaque paquet reçu) noté θ à partir des paramètres d, p, b, T et R.

$$\theta(t)_{(d,p,b,T,R)} \tag{3.5}$$

Soit $\Psi_{\theta(t)_(d,p,b,T,R)}(D_{max}, P_{min})$ la fonction qui teste la capacité de Tetrys à satisfaire la requête (D_{max}, P_{min}) de l'application étant donné les paramètres (d,p,b,T,R). Ψ retourne VRAI si la probabilité qu'un paquet arrive avant D_{max} est supérieure à P_{min} et FAUX sinon.

R_{min} est donc la plus petite valeur de R pour laquelle $\Psi_{\theta(t)_(d,p,b,T,R)}(D_{max}, P_{min})$ est vrai et peut s'écrire :

$$R_{min} = min(R|\Psi_{\theta(t)_(d,p,b,T,R)}(D_{max}, P_{min})) \tag{3.6}$$

Par souci de simplicité, les arguments de Ψ seront implicites et Ψ se réfère à :

$$\Psi_{\theta(t)_(d,p,b,T,R)}(D_{max}, P_{min})$$

La suite de ce chapitre consiste à trouver une expression de $\theta(t)_{(d,p,b,T,R)}$.

3.4.1 Heuristique pour la distribution du délai

Dans la section 3.1.3 nous avons donné une expression de la distribution du délai à partir d'une chaine de Markov. Cette méthode est particulièrement coûteuse en termes de temps de calcul et ne peut donc pas être utilisée dans un protocole qui requiert de s'adapter en quelques milli-secondes. De plus le modèle fait l'hypothèse de pertes uniformément distribuées. Si il est possible de modéliser le comportement de Tetrys avec des pertes en rafales (Gilbert-Eliot), la complexité n'en sera que plus importante. Nous ne pouvons malheureusement donc pas nous appuyer sur ce modèle pour obtenir $\theta(t)_{(d,p,b,T,R)}$ en un temps raisonnable.

Nous proposons donc d'inférer $\theta(t)$ par simulation. Pour ce faire nous avons fait varier les paramètres suivant :

1. le taux de perte de 1% à 50% ;
2. le taux de redondance de avec $n = k + 1$ variant de 2 à 10
3. des pertes uniformes ($b = 1$) ou en rafales de taille moyenne 2 et 3.

Les simulations durent chacune 10^5 paquets et ont été faites pour un FTT noté $D_{ref} = 100ms$ et un temps inter-paquets de $T_{ref} = 10ms$. Ce sont les paramètres de référence à partir desquels il faudra traduire les débits et délais.

Distribution de réference : On cherche une distribution de référence qui corresponde au délai de Tetrys. Il serait pratique que la distribution exponentielle convienne. Cependant lorsque le taux de perte est proche du taux de redondance, la distribution du hitting time a une queue lourde ce qui n'est pas capturé par la distribution exponentielle. De la même manière les distributions à queues lourdes telles que la Pareto ne capturent pas le fonctionnement de Tetrys pour de grande valeur de Δ_R. La log-normal et la Weibull permettent de capturer aussi bien les distributions à décroissance exponentielle que les distributions à queue lourde. Les distributions de Weibull obtenues à partir des paramètres inférés [9] sont les plus proches de la distribution empirique de Tetrys [10]. C'est donc via celle ci que nous exprimons $\theta(t)_{(d,p,b,T,R)}$.

$$P[X < x] = 1 - e^{-(x/\lambda)^{\kappa}}$$

Le paramètre λ est dédié à la forme et κ à l'échelle. Lorsque $\lambda = 1$, la distribution de Weibull correspond à l'exponentielle. Lorsqu'il est inférieur à 1 elle tend vers des distributions à queue lourde.

3.4.1.1 Estimation des paramètres λ et κ :

Paramètre de forme (λ) **:** Pour un canal à effacement donné (e.g. Bernoulli ou Gilbert Elliot), la distribution du délai est influencée par n $(k+1)$ et p $(\Delta_R = \frac{1}{n} - p)$). Pour chaque taille de bloc n et chaque canal à effacement (b), la distribution de λ évolue de manière quasi linéaire en fonction de Δ_R comme on peut le voir sur la figure 3.11. Les coefficients de cette fonction affine sont stockés dans le tableau 3.2.

Paramètre d'échelle (κ) **:** De la même manière, κ est uniquement influencé par n et la distribution des pertes. Il peut être approché par :

$$\lambda(\Delta_R) = \frac{a}{{\Delta_R}^c} \tag{3.7}$$

9. Les paramètres λ et κ ont été obtenus par un ajustement des moindres carrés.
10. Pour générer les eps des figures de ce chapitre il faut utiliser les script *generateToPlot_fit_R.sh* et *generateCorrelation.sh*

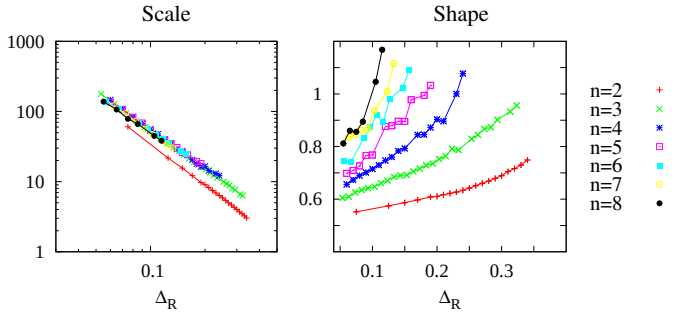


FIGURE 3.11: Evolution du paramètre de forme (λ) et d'échelle (κ) en fonction of Δ_R

N-1	1	2	3	4	5	6	7
a_{ber}	0.72	1.25	2.0	2.65	3.44	3.866	5.6
b_{be}	0.473	0.51	0.512	0.525	0.53	0.55	0.46
a_{b2}	0.48	1.31	1.92	2.15	3.69	5.15	4
b_{b2}	0.57	0.6	0.61	0.62	0.56	0.48	0.67
a_{b3}	0.62	1.8	2.8	4	4.54	5.5	5.4
b_{b3}	0.65	0.61	0.57	0.53	0.6	0.62	0.72

TABLE 3.2: Valeurs des coefficients des fonctions linéaires permettant de générer les valeurs de κ.

N-1	1	2	3	4	5	6	7
a_{ber}	0.83	0.35					
b_{ber}	1.815	2					
a_{b2}	4.2	7.15	9.9	10.48	5.6	2.7	6.3
b_{b2}	1.14	1.35	1.3	1.3	1.65	1.94	1.57
a_{b3}	11.8	11.4	18.2	9.3	7.1	19.1	36
b_{b3}	1.04	1.44	1.3	1.6	1.7	1.28	1.05

TABLE 3.3: Valeurs des coefficients des fonctions linéaires permettant de générer les valeurs de λ.

avec a et c des coefficient spécifiques à la répartition des pertes b et à la taille du bloc n stockés dans le tableau 3.2.

Finalement $\theta(t)_{(d,p,b,T,R)}$ peut s'écrire :

$$1 - e^{-(\frac{t}{\lambda(n,p,b)})^{\kappa(n,p,b)}} \tag{3.8}$$

avec :

- $\lambda(n,p,b) = \frac{a_{b,n}}{(\frac{1}{n}-p)^{c_{b,n}}}$,
- $\kappa(n,p,b) = a_{b,n} * (\frac{1}{n} - p) + c_{b,n}$,
- b le canal $\in ber, b2, b3$, $a_{b,n}$ et $c_{b,n}$ les valeurs appropriées dans 3.2 et 3.3.

En intégrant les paramètres de référence T_{ref} et le délai, Ψ s'écrit :

$$\Psi = \begin{cases} VRAI & \text{si } 1 - e^{-(\frac{(D_{max}-d)\cdot\frac{T_{ref}}{T}}{\lambda(n,p,b)})^{\kappa(n,p,b)}} > \frac{Pkt_{min}+p-1.0}{p} \\ FAUX & \text{sinon} \end{cases}$$

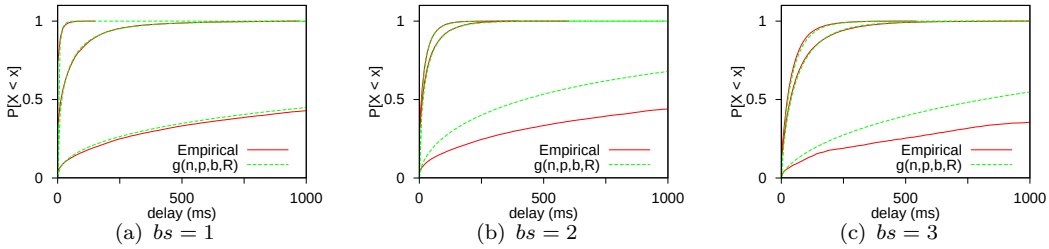
3.4.2 Précision de $\theta(t)_{(d,p,b,T,R)}$

La figure 3.15 montre la distribution des délais de décodage de Tetrys pour différents taux de pertes, uniformes ou en rafales. Sur chacune des sous figures, la courbe rouge correspond à la distribution empirique, obtenue par simulation et la courbe verte correspond à $\theta(t)$. Sur chacune des sous figures de 3.12 les distributions sont tracées pour des taux de pertes de 20%, 40% et 49% pour un taux de redondance de 50%. De la même manière, les sous figures de 3.13 et 3.14 contiennent chacune les distributions pour des taux de pertes de respectivement {2%, 10%, 18%} et {2%,6%,12%}. Conformément aux attentes plus Δ_R diminue, moins la fonction $\theta(t)_{(d,p,b,T,R)}$ est précise. Pour des valeurs de Δ_R de l'ordre de 1%, l'estimation est tantôt optimiste tantôt pessimiste et ne peut pas être utilisée de manière fiable. Ce biais disparait dès lors que $\Delta_R > 4\%$ et on peut constater que pour la plage de valeurs correspondant aux contraintes des applications temps réelles, l'écart entre $\theta(t)$ et la distribution empirique est très faible.

3.4.3 Précision de $min(R|\Psi_{\theta(t)_(d,p,b,T,R)}(D_{max}, Pkt_{min}))$

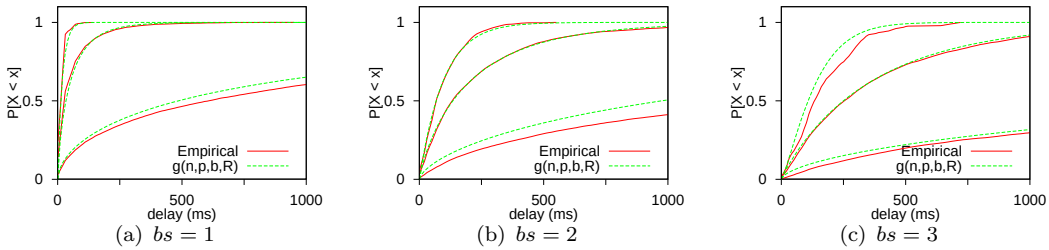
Après une description des paramètres de simulation, cette section montre que $min(R|\Psi_{\theta(t))}$ vérifie bien ses deux contraintes qui sont les suivantes : assurer que les besoins formulés par l'application sont satisfaits et d'autre par que le taux de redondance est minimisé.

Paramètres de simulation : La courbe verte des figures 3.16(b), 3.16(c) et 3.16(a) montre le 95éme percentile du délai obtenu par $min(R|\Psi_{\theta(t))}$ en fonction du taux de perte pour des pertes uniformes et en rafales de taille moyenne 2 et 3. La courbe rouge correspond à une version sous optimale de $min(R|\Psi_{\theta(t)})$ qui retourne $R < R_{min}$ avec $R_{min} = min(R|\Psi_{\theta(t)})$ et $R = 1/(1+1/R_{min})$. Elle peut donc être vue comme la meilleure alternative à $min(R|\Psi_{\theta(t)})$. Le délai maximum toléré par l'application est de $300ms$, le FTT est de 100ms, la proportion de paquet à décoder avant D_{max} est de 95% et le débit est de 100 paquets par secondes. Nous avons implanté Tetrys et $min(R|\Psi_{\theta(t)})$ dans



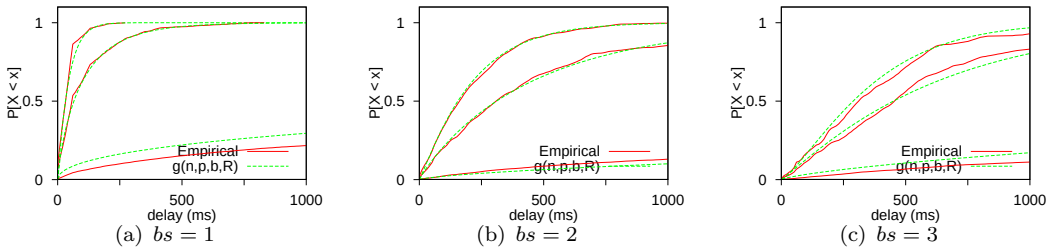
(a) $bs = 1$ (b) $bs = 2$ (c) $bs = 3$

FIGURE 3.12: PLR={0.2, 0.4, 0.49} et $R = 0.5$



(a) $bs = 1$ (b) $bs = 2$ (c) $bs = 3$

FIGURE 3.13: PLR={0.02, 0.1, 0.18} et $R = 0.2$



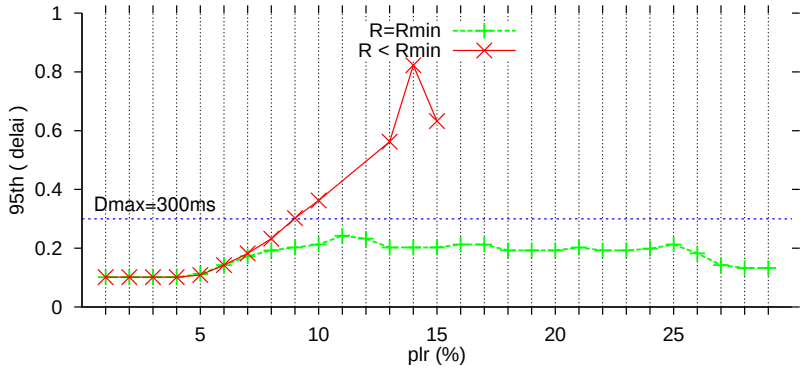
(a) $bs = 1$ (b) $bs = 2$ (c) $bs = 3$

FIGURE 3.14: PLR={0.02, 0.06, 0.12} et $R = 0.125$

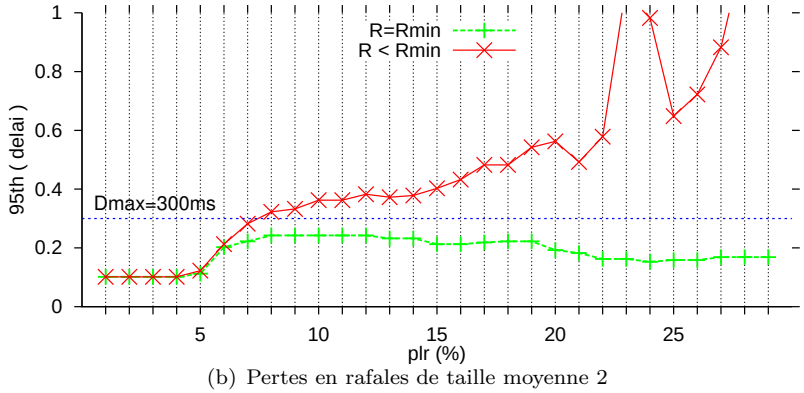
FIGURE 3.15: Distribution empirique et estimée par $g_{(p,b,n)}(x)$ du délai de livraison des paquets à l'application.

l'environnement NS2 et les pertes sont appliquées sur le lien par un module développé pour simuler un canal avec des pertes de type uniforme ou Gilbert Eliot.

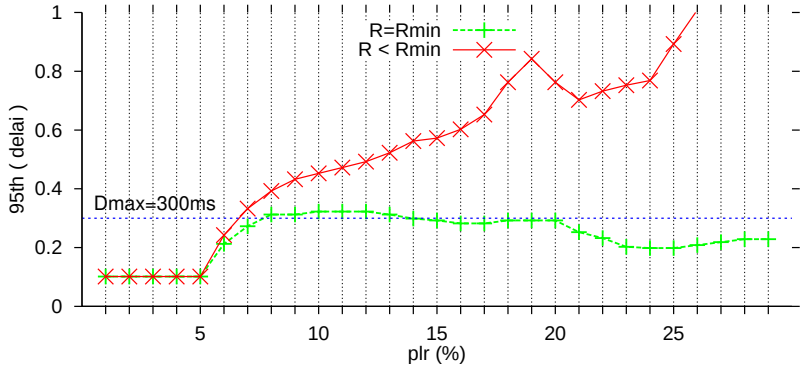
Respect de D_{max} : On peut observer que le 95éme percentile reste quasiment toujours sous la borne des 300ms tolérée par l'application. Lorsque le taux de perte est inférieur à 5% il correspond au FTT puisque qu'au moins 95% des paquets sont reçus et ne rencontrent pas d'autre délai que celui de transmission. On constate par contre que pour des taux de pertes allant de 8 à 10% et des rafales de taille 3 (figure 3.16(c)), le délai est légèrement au dessus des $300ms$ et atteint un maximum de $322ms$. Cet imprécision est en fait due à la mesure du taux de perte par l'implémentation de Tetrys sous NS. En effet ce dernier est mesuré en temps réel et s'il n'y a pas de perte pendant un grand laps de temps comme ça peut être le cas avec des pertes en rafales, le taux de perte mesuré descend en dessous de sa valeur réelle et Tetrys peut donc choisir des



(a) Pertes uniformes



(b) Pertes en rafales de taille moyenne 2



(c) Pertes en rafales de taille moyenne 3

FIGURE 3.16: PLR=$\{0.02, 0.06, 0.12\}$ et $R = 0.125$

valeurs inadéquates par rapport à la rafale de pertes suivante. Nous n'avons pas défini de borne pour la durée d'échantillonnage du taux de perte car les valeurs adéquates dépendent de la vitesse de variation du taux de perte dans le réseau.

Respect de $R_{min} = min(R|\Psi_{\theta(t)})$: On peut observer qu'à l'exception des taux de perte de 6, 7 et 8%, $min(R|\Psi_{\theta(t)})$ choisit la plus grande valeur de N (la plus petite valeur de R) tel que le 95éme percentile du délai est inférieur à D_{max}. La courbe rouge (sous

optimale [11] est en effet bien au delà des $300ms$ tolérées par l'application. Elle minimise donc bien la charge induite sur le réseau tout en assurant le respect des contraintes de l'application.

3.5 Conclusion

Dans ce chapitre, nous venons de présenter Tetrys, un mécanisme dont la fenêtre d'encodage a pour caractéristique principale d'être élastique. En effet, cette dernière inclut l'ensemble des paquets émis mais pas accusés par le destinataire. Cette particularité implique que Tetrys permet une fiabilité totale sans retransmission. Le principal enseignement que l'on peut retenir de cette étude est que Tetrys possède les mêmes propriétés de délai qu'un code convolutionnel fortement MDS alors qu'il ne nécessite qu'un corps fini de petite taille, qu'il est sans rendement et permet une fiabilité totale sans retransmission. Nous avons proposé un modèle analytique de Tetrys à partir duquel a été dérivé le délai de décodage et la complexité du mécanisme en espace et en temps de calcul. Comme pour tout code convolutionnel fortement MDS, il en résulte que le délai de décodage n'est fonction que de la distribution des pertes et de l'écart entre le taux de redondance et le taux de pertes. La complexité de Tetrys à l'encodage est une fonction linéaire de la taille de la fenêtre d'encodage.

Dans la partie suivante, nous allons illustrer les différentes propriétés de Tetrys énoncées dans ce chapitre au travers de son utilisation en tant que mécanisme de fiabilisation pour les réseaux de communication.

11. Le taux de redondance de la courbe rouge correspond à $R = \frac{1}{N_{min}+1}$) en supposant que $R_{min} = \frac{1}{N_{min}}$

Deuxième partie

Applications

Chapitre 4

Application de Tetrys aux applications temps réel

Contents

4.1 Application aux applications à contrainte de délai : la video conférence

En guise d'exemple d'application temps-réel ne requérant qu'une fiabilité partielle, nous avons choisi la vidéo conférence. Du point de vue de la transmission sur le réseau, la vidéo possède des caractéristiques qui rendent sa protection plus délicate comparé à d'autres trafics tel la VoIP. Premièrement, le délai de bout en bout ne doit pas dépasser les 100ms afin de préserver l'interactivité de la conférence [7]. Les images sont transmises à intervalles réguliers mais sont de trois types différents I, P et B. Les trames I, images Intra-codées qui comme leur nom l'indique sont des images entières et donc de taille conséquente. Ensuite viennent les trames P, images de prédiction qui ne contiennent que les modifications depuis la précédente trame I ou P et sont donc significativement plus légères. Les trames B sont à priori encore plus légères puisqu'elle sont codées à partir

des trames I et P précédentes et suivantes [1]. La première caractéristique majeure de ces différents types de trames est le fait que le débit généré par leur émission est variable (VBR). La seconde est que la perte d'une trame I n'a pas le même impact que celle d'une trame B ou P. En effet, la perte d'une trame I rend inutilisable l'ensemble des trames B ou P suivantes alors que la perte d'une trame P peut induire une erreur qui se propagera jusqu'à la prochaine trame I.

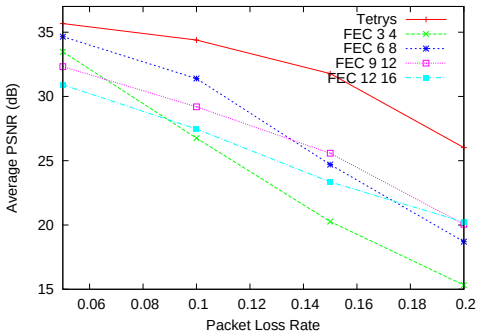
Ces caractéristiques ont un impact majeur sur les mécanismes que l'on peut utiliser pour les protéger. L'utilisation de FEC est par exemple limité. En effet, la taille des blocs ne doit être ni trop longue car l'interactivité pourrait en pâtir, ni trop courte car la proportion de pertes non corrigées détériorerait la qualité de la vidéo. Configurer FEC de manière à utiliser à la fois la taille de bloc et le taux de redondance optimaux peut être particulièrement délicat. A l'inverse, nous avons vu sur la figure 3.9 que Tetrys permet un compromis similaire à celui de la configuration FEC optimale pour un taux de redondance donné. Dans le cas d'une source à débit variable, le manque de tolérance des paramètres de FEC complique sa configuration et la rend même sous optimal. Cela s'explique par le fait que reconstruire un nombre donné de pertes revient à attendre le même nombre de paquets de redondance. Deux paquets de redondance étant séparés par k paquets de données, lorsque le débit d'émission instantané des paquets augmente, cela réduit considérablement la reconstruction et donc la probabilité qu'un paquet soit disponible avant son instant de lecture. A l'inverse, pour FEC cela réduit également le délai mais la probabilité de reconstruire un paquet perdu reste la même et il ne bénéficie donc pas des variations de débit.

4.1.0.1 Paramètres de simulation

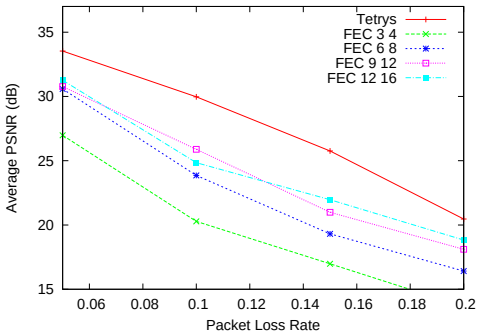
Nous comparons Tetrys à des codes FEC idéaux avec différentes tailles de blocs et un taux de codage constant, en l'occurrence (k,n)= { (3,4), (6,8), (9,12), (12,16) }. Comme pour la VoIP, nous répétons les mesures avec des pertes uniformément distribuées ainsi qu'avec des pertes en rafale de taille moyenne 2 et 3 suivant un modèle de Gilbert Elliot. Nous avons utilisé la dernière recommandation en date en matière de codec vidéo à savoir H.264 avec l'implantation JM 15.1 H.264/AVC [64]. La séquence video utilisée est celle de Foreman au format CIF (352x288) avec 15 trames par secondes dont une trame I et 14 trames P. Les trames B ne sont pas utilisées. Le débit moyen ainsi produit par l'encodeur vidéo est de $384kbps$ et les paquets sont de 500 octets chacun. Nous fixons à $200ms$ le délai maximum tolérable de bout en bout. Tous les paquets arrivant après cette limite sont jetés. La séquence dure 10 secondes que nous répétons 20 fois afin d'obtenir

1. En pratique, de par les contraintes de délai, on ne peut pas utiliser les trames de type B pour la video conférence.

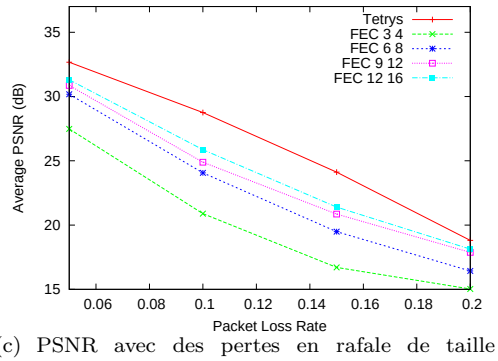
des résultats représentatifs ce qui fait un total de 3000 trames pour 200 secondes. Cette configuration correspond aux recommandations faites par Stephan Wenger dans [65]. Pour évaluer la qualité de la vidéo reçue, nous avons utilisé le framework Evalvid [66] dans lequel la métrique utilisée est le PSNR (Peak Signal to Noise Ratio).



(a) PSNR avec pertes uniformes



(b) PSNR avec des pertes en rafale de taille moyenne 2 (Gilbert Elliot)



(c) PSNR avec des pertes en rafale de taille moyenne 3 (Gilbert Elliot)

FIGURE 4.1: PSNR obtenu par Tetrys et FEC sous différents types de pertes.

4.1.0.2 Résultats

Sans surprise, à taux de codage équivalent, la qualité de la video est meilleure lorsqu'elle est protégée avec Tetrys que lorsque qu'elle est protégée avec FEC. Dans le cas des pertes uniformément distribuées, la figure 4.1 présente un gain de 7.19dB pour un taux de perte de 15% obtenu par Tetrys comparé au FEC optimal (FEC 6 8). De plus, la chute du PSNR n'est que de 4bB pour Tetrys lorsque le taux de perte passe de 5% à 15%. Il reste ainsi au dessus de 30dB ce qui correspond à une image de bonne qualité. La diminution de la qualité est donc douce pour Tetrys car sa protection peut être considérée comme inégale et profitant aux trames de type I tandis que les trames de type P sont les premières à ne pas être reconstruites de par leur plus faible débit et au temps de reconstruction requis. La partie supérieure de la figure 4.2 montre pour 10 secondes consécutives la proportion de trames I et P ainsi que leur débit sur une période

de $100ms$. La partie inférieure de la figure 4.2 montre le PSNR pour chaque trame avec FEC et Tetrys. Le débit des trames I est bien plus important que celui des trames P et Tetrys tire bien profit de cette différence de débit. En effet, les trames I sont quasiment toujours reconstruites (9 sur 10). FEC a la même probabilité de perdre un bloc issu d'une trame I ou P. Ainsi, on observe que seules 5 trames sont reconstruites. Le gain obtenu par Tetrys est donc essentiellement lié à sa protection efficace des trames I.

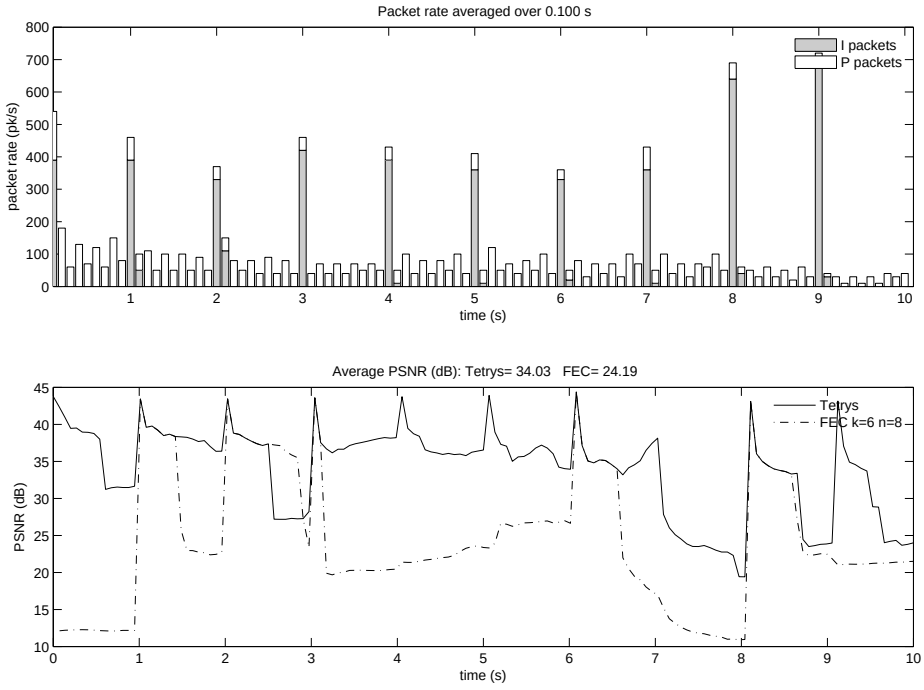


FIGURE 4.2: Débit et PSNR instantané d'une vidéo *live* avec un PLR de 15% sur un canal de Bernoulli.

Pour des pertes en rafale de taille, les figures 4.1(b) et 4.1(c) montrent la même tendance bien que les gains soient moins significatifs lorsque la taille des rafales augmente (respectivement $3.78dB$ et $2.7dB$ pour des rafales de taille 2 et 3). Cela s'explique notamment par le fait que la correction des pertes en rafale requière plus de redondance que pour des pertes uniformes. À la vue de leur PSNR, aussi bien FEC que Tetrys sont sous dimensionnés.

4.2 Vers une adaptation des codecs

Bien que l'application de vidéo ne tire pas profit de la caractéristique principale de Tetrys qui est de reconstruire l'ensemble des paquets perdus, la qualité de la vidéo est bien meilleure qu'avec FEC. En effet, le codec H.264/AVC ne prend pas en compte les paquets qui arrivent après leur instant de lecture initiale et ces derniers sont simplement jetés. Si FEC est dimensionné pour qu'il n'y ait pas de paquet arrivant après son instant de lecture, ce n'est pas le cas de Tetrys. Ainsi, des trames qui n'ont pu être reconstruites à leur instant de lecture peuvent être reconstruites et disponibles pour les trames suivantes. Les trames P étant codées à partir des trames I et P précédentes, les réintégrer dès qu'elles sont disponibles permettrait d'améliorer nettement la qualité des trames suivantes.

Exemple : considérons les paramètres de la section 4.1 précédente. Un groupe d'image ou GOP (Group Of Pictures) est envoyé chaque seconde. Il est composé d'une trame I, suivie de 14 trames P espacées de 1/15s. Les GOP sont indépendants et les trames d'un GOP sont notées $I_0, P_1, P_2, ..., P_{14}$. Supposons que des paquets issus d'une trame I_0 n'aient pas pu être reconstruits avant les 100ms de délai imparti. Le bruit généré par ces paquets manquants va se propager sur chacune des 14 trames P suivantes et avoir un impact sensible sur le PSNR. A l'inverse, si les paquets manquants de la trame I_0 sont reconstruits au moment où la trame P_i doit être jouée, P_i pourra être générée comme s'il n'y avait eu aucune erreur sur les trames précédentes (la reconstruction de I_0 au temps i implique que les paquets manquants des autres trames P_j avec $i < j$ sont également reconstruits). Le bruit n'aura affecté que $i - 1$ images au lieu de quinze dans le cas normal.

Il peut être intéressant de considérer deux versions, une qui ne réintègre que les trames I, et l'autre qui réintègre à la fois les trames I et P . Cela se justifie par le fait que la dépendance entre les trames P varie selon les codecs. Pour certains, une erreur sur une trame gèle l'image jusqu'à la fin du GOP tandis que pour d'autres, les trames P ne dépendent que de la trame I. D'autre part, les mécanismes d'atténuation d'erreur (*error concealement*) varient et ne sont pas toujours implementés. Dans certains cas, une trame pourra être partiellement reconstruite et affichée même si une partie de ses paquets sont manquants.

L'autre raison est liée à la complexité induite par le fait de recalculer toutes les trames jouées depuis la plus vieille trame du GOP qui contient une erreur. Les trames I ayant le plus d'importance, il peut être possible de se limiter à la réintégration de ces dernières.

Cette complexité est malgré tout bornée car **chaque trame n'est traitée au pire que 2 fois**. En effet, le décodage de Tetrys est tel que lorsque qu'une perte est reconstruite, plus aucune perte n'est à déplorer. Il en résulte que si une trame $T_i \in \{I, P\}$ avec $i < k$ est en erreur puis décodée à l'instant de réception de T_k, toutes les trames en erreur T_j avec $i < j < k$ sont également décodées à l'instant T_k. Pour prendre en compte T_i il faut que le décodeur reprenne toute la séquence $T_{i...k-1}$ dans laquelle l'ensemble des autres trames T_j sont également décodées sans erreur. Chaque trame est donc jouée au plus deux fois à l'exception de la dernière trame P qui n'est jouée qu'une et une seule fois.

Cette borne est fortement pessimiste et en réalité la probabilité de décoder une trame deux fois dépend du rapport entre le taux de redondance et le taux de perte.

4.2.1 Évaluation

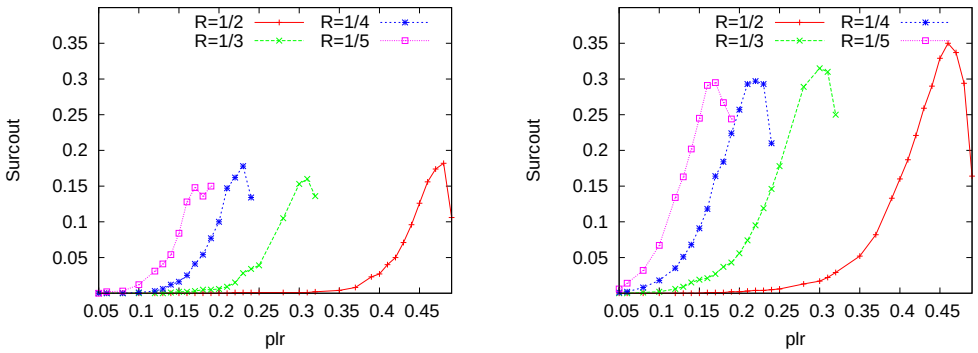
Afin de mieux appréhender le gain potentiel et les inconvénients liés à cette approche, nous proposons de simuler l'impact sur un codec vidéo selon trois cas de figures :

- Codec normal : l'ensemble des trames ne sont traitées qu'une seule fois. On considère qu'une trame contient une erreur si elle est incomplète ou si elle dépend d'une trame qui était incomplète à son instant de lecture ;
- Décodage tardif des trames I : seule la reconstruction d'une trame I_i jouée en erreur implique de re-calculer les trames P déjà jouées et dépendantes de I_i. Une image contient une erreur liée à une trame I si à son instant de lecture, la trame I dont elle dépend n'a toujours pas été décodée par Tetrys. A l'instant où une trame I est décodée par Tetrys, l'ensemble des autres trames sont également décodées. Ainsi la trame suivant le décodage d'une trame I est jouée sans erreur à condition qu'elle soit elle même complète ;
- Décodage tardif des trames I et P : une image contient une erreur si à l'instant où elle est jouée, une des trames dont elle dépend n'a pas été totalement décodée par Tetrys. Lorsqu'une trame T_i qui a été jouée avec une erreur est décodée, toutes les trames jouées depuis T_i sont recalculées par le codec afin que l'ensemble des trames de références des trames suivantes ne contiennent pas d'erreur.

Nous utilisons des paramètres similaires à ceux de la section précédente et nous faisons en plus les hypothèses suivantes. Les trames I et P sont de tailles constantes ($nbPkt(I) = 60$ et $nbPkt(P) = 6$) et nous n'utilisons pas les trames B. Il y a 15 trames par GOP et un GOP par seconde. Le délai inter-trame est noté $T_{inter} = 1/15$ de seconde et le délai inter-paquets lors de la transmission d'une trame est de $T_{inter-pkt} = T_{inter}/nbPkt(I)$. Une trame T_i est envoyée à l'instant $t_i = t_{GOP} + T_{inter} \cdot i$ avec t_{GOP} l'instant auquel est générée la première trame du GOP courant. Une trame T_i est jouée à l'instant $t_i + D_{max}$.

Les paquets de chaque trame sont envoyés à l'encodeur de la source Tetrys puis décodés par le récepteur Tetrys et passés à l'application vidéo.

4.2.1.1 Surcoût pour le décodeur



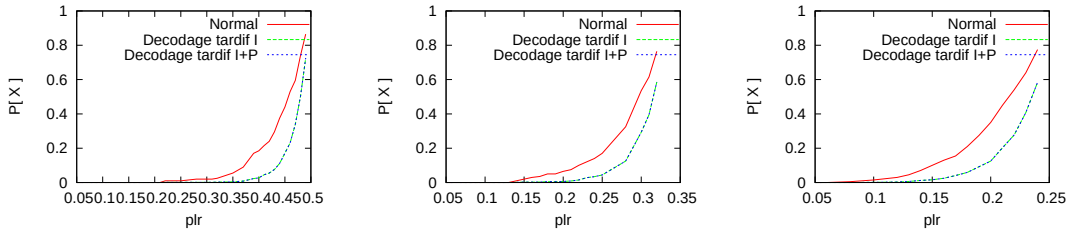
(a) Trames I (b) Trames I et P

FIGURE 4.3: Probabilité qu'une trame soit traitée deux fois.

Nous définissons le surcoût pour le décodeur comme la probabilité qu'une trame soit traitée deux fois. Les figures 4.3 montrent le surcoût selon que le décodeur réintégre uniquement les trames I ou bien les trames I et P. Pour les seules trames I, le surcoût ne dépasse pas les 20% et se limite même à 10% pour des taux de pertes et taux de codage conformes aux contraintes de délai de la vidéo conférence. Pour les trames I et P, le surcoût est plus de deux fois supérieur à celui du décodage tardif des seules trames I et peut monter dans le pire des cas jusqu'à 35%. Le surcoût est néanmoins assez faible et reste éloigné des 100% même quand le PLR est proche de R. Cela s'explique par la distribution du temps de récurrence de Tetrys. Une trame qui n'a pas été reconstruite en un laps de temps très court, à peu de chance d'être reconstruite avant la fin du GOP. Ainsi, une trame est soit rapidement corrigée et ne génère pas un surcoût important, soit elle n'est pas reconstruite et elle n'implique aucun coût supplémentaire. La distribution de Tetrys a également l'avantage de réduire le nombre de trames à réinterpréter en un temps très court ($T_{inter}/2$) lorsqu'une trame est décodée.

4.2.1.2 Gains

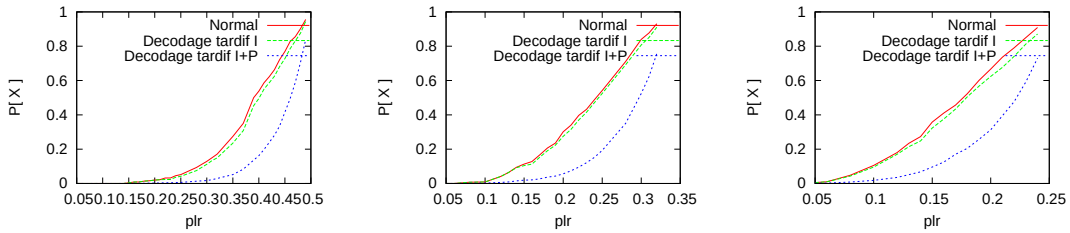
La figure 4.4 montre la probabilité qu'une trame soit construite et affichée à partir d'une trame I partiellement décodée en fonction du taux de perte. On peut voir que pour $R = 1/4$ et PLR=15% (resp. 20%), la probabilité est de 1.6% (resp. 12%) pour les mécanismes de reconstruction tardive tandis qu'elle est de 10% (resp. 35%) pour les codecs normaux. Bien que ces résultats soient à tempérer avec les mécanismes d'*error concealment* des différent codecs, les gains sont d'un ordre de grandeur significatif.



(a) R=1/2 (b) R=1/3 (c) R=1/4

FIGURE 4.4: Probabilité qu'une trame soit construite et affichée à partir d'une trame I partiellement décodée.

La figure 4.5 montre, en fonction du taux de perte, la probabilité qu'une trame affichée soit différente de l'image d'origine. Il est clair que le fait de ne réinterpréter que les trames I ne permet pas d'améliorer sensiblement cette métrique. Par contre on peut voir que pour $R = 1/4$ et PLR=15% (resp. 20%), la probabilité est de 5% (resp. 22%) pour le décodage tardif des trames I et P tandis qu'elle est de 35% (resp. 66%) pour les codecs normaux.

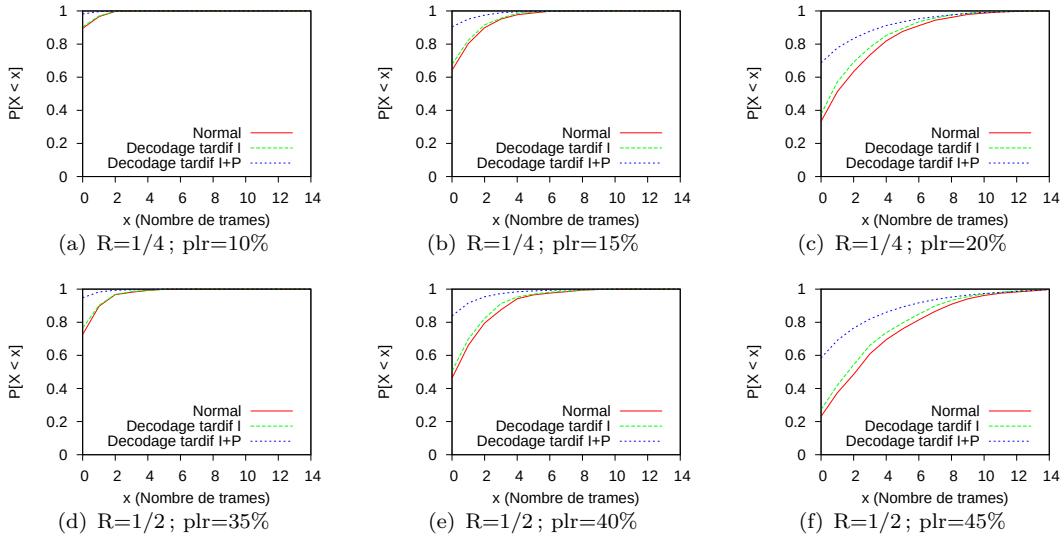


(a) R=1/2 (b) R=1/3 (c) R=1/4

FIGURE 4.5: Probabilité qu'une trame affichée contienne une erreur.

Pour les mécanismes d'*error concealment*, le nombre d'erreurs accumulées a une importance considérable. Nous proposons donc d'étudier la distribution du nombre de trames contenant des erreurs utilisées par le décodeur pour construire et afficher une trame donnée.

La figure 4.6 montre que seule la prise en compte des trames I et P permet de réduire significativement le nombre de trames contenant des erreurs utilisées dans la construction et l'affichage d'une image. Comme on peut le voir sur les figures 4.6(c) et 4.6(f), lorsque le taux de perte augmente, la distribution du nombre de trames incomplètes tend a être à queue lourde. En sus d'avoir un impact sur la performance des mécanismes d'*error concealment*, cela signifie qu'il y a potentiellement un grand nombre d'images à recalculer lors du décodage d'une trame par Tetrys. Dans ce cas la (*e.g.* $x > 4$), les hôtes dont la



(a) R=1/4 ; plr=10% (b) R=1/4 ; plr=15% (c) R=1/4 ; plr=20%

(d) R=1/2 ; plr=35% (e) R=1/2 ; plr=40% (f) R=1/2 ; plr=45%

FIGURE 4.6: Distribution cumulative de la probabilité qu'une trame soit construite et affichée à partir de x trames partiellement décodées.

capacité de calcul est limitée peuvent choisir de ne pas recalculer les trames et d'attendre le prochain GOP.

4.2.2 Travaux similaires

A notre meilleure connaissance, les décodeurs vidéo ne prennent pas en compte les erreurs corrigées après l'instant de lecture initiale de la trame et il n'existe pas de propositions dans ce sens [2]. Cela tient du fait que lorsque des paquets ne sont pas arrivés à l'instant de leur lecture, c'est que ces derniers ont été perdus, les fortes variations de délai étant assez rares. Ainsi, la plupart des solutions qui adressent le problème de dépendance entre les différents paquets d'un flux utilisent plusieurs MDC (Multiple Description Coding) avec niveaux d'encodage dans la vidéo [67]. La vidéo est alors découpée en plusieurs flux décodables indépendamment. Si les pertes sont espacées dans le temps, la probabilité de pouvoir décoder une partie du flux est donc bien plus élevée. Ces propositions sont souvent couplées à une utilisation de chemins différents car c'est la seule façon de réduire efficacement l'impact des pertes lorsqu'elles sont groupées [68]. Chacun des flux décodables est alors envoyé via différentes routes sur l'internet ou différentes interfaces. En pratique ces solutions sont peu commodes car il est quasi impossible de faire du routage à la source sur internet et les protocoles de transport actuels ne gèrent qu'une seule interface réseau.

2. Hormis pour les entêtes d'un GOP par exemple.

D'autre solutions comme celle de Wee et Al. [69] adaptent l'instant d'émission des trames en fonction de leur dépendance respective. Cette solution est présentée dans le cas du *streaming* sur les réseaux 3GPP où le délai peut être fortement variable.

4.3 Conclusion

Nous avons effectué une étude préliminaire sur le gain apporté par le couplage de Tetrys à un décodeur qui accepte les corrections tardives et reprend les calculs des images à partir de la première trame corrigée. La probabilité d'afficher une trame sans erreur est améliorée de plusieurs ordres de grandeur par rapport à un décodeur classique tel que H.264/AVC. De même, si l'on ne considère que le décodage tardif des trames I, le gain sur la probabilité d'afficher une image calculée avec une trame I complète est significativement amélioré. Ces résultats, bien qu'encourageant, sont à compléter avec des expérimentations sur des codecs réels. Le seul inconvénient est à priori le surcoût de complexité généré. En pratique il est borné et ne dépasse pas 10% et 30% selon que seules les trames I ou I et P tardivement décodées sont considérées par le décodeur. La solution proposée laisse ainsi au décodeur le choix d'adapter la qualité de la video à la complexité qu'il peut tolérer.

Chapitre 5

Streaming fiable pour les DTNs

Contents

Nous abordons dans cette partie le problème du transport de flots de données continues de manière fiable et efficace en termes de délai de transfert au-dessus de réseau de noeuds mobiles avec une transmission de type store-and-forward [19]. Nous proposons d'évaluer le mécanisme Tetrys pour réaliser ce type de transfert sur un réseau DTN et montrons que ses caractéristiques (détaillées au Chapitre 3) permettent d'offrir un service fiable pour la diffusion de données des flux de type streaming.

5.1 Introduction

La problématique du routage a été largement traitée par la communauté DTN (voir la section 2.1.2.2 du chapitre 2) avec pour objectifs (parfois disjoints) de réduire le délai, l'énergie consommée, la charge dans le réseau, le taux de perte (...). Les protocoles et algorithmes résultant de ces études ne se préoccupent que d'un sous-ensemble de ces

métriques et ne portent que peu d'attention aux applications et à leurs besoins. Pour ce qui est de la problématique du transport et de la fiabilité, deux classes de protocoles sont candidats au transport sur réseaux DTN. La première n'utilise pas de chemin de retour. Elle propose d'améliorer le taux de transfert et/ou le délai de transfert des *bundles* DTN [1] à l'aide de mécanismes tels que la réplication, les codes à effacement et le multi-chemin [70–72]. La seconde considère qu'un chemin retour est envisageable. En particulier, Harras et al. étudient différentes stratégies d'acquittement dans [73] et y associent des mécanismes de fiabilité notamment basés sur de l'ARQ. Dans un autre contexte, celui du *Deep-Space Networking* (DSN), les protocoles de transport tels Saratoga [74] ou LTP-T [12] utilisent l'ARQ et la protection inégale afin de réduire la proportion de *bundles* à retransmettre.

Comparé à Tetrys, ces algorithmes et protocoles [2] sont conçus pour transporter des unités *(bundle)* avec un service partiellement fiable. De plus, il est important de noter que pour utiliser correctement ces protocoles de transport, l'intégralité des données doit être disponible au début de la transmission afin de générer les blocs encodés qui seront diffusés lors de la prochaine opportunité de contact [70, 71]. Cependant et comme montré dans [70], ces algorithmes nécessitent une configuration complexe étroitement liée aux caractéristiques du réseau et n'offrent aucune garantie stricte en termes de délai ou de fiabilité. En effet, aucune de ces solutions n'aborde le transport des flots de données générés à la volée (en *live*) au dessus d'un réseau DTN.

Enfin si l'on cherche à maintenir un taux de perte en-dessous du seuil maximum toléré par l'application, la complexité de configuration de ces codes s'accroit de façon conséquente voire devient impossible.

Nous adressons ici ce que l'on appellera les flots d'applications *streaming-like* [3] produisant des données de façon continue en fonction du temps et qui nécessitent d'être consommées en-séquence (les données doivent être ordonnées) à la réception. Nous montrerons que Tetrys offre une solution possible à ce type de trafic sur réseaux DTNs par sa capacité à reconstruire les données sans compromis sur le délai et tout en restant indépendant du routage utilisé et de la répartition des pertes qu'il induit.

1. Le mot *bundle* possède deux définitions principales suivant le contexte. Dans notre cas, elle correspond à l'unité de message véhiculé sur ces réseaux.
2. Sauf LTP/LTP-T qui a été conçu pour un contexte DSN.
3. Nous insistons sur le fait que cette définition n'est pas à associer à celle inhérente aux flots des applications temps-réel et multimédia. Cette dernière n'ayant aucune relation avec celle exposée ici.

5.2 Challenges relatifs au support d'applications *streaming-like* dans les DTNs

Dans un DTN, le routage, les liens, les capacités de mémorisation et la mobilité des noeuds agissent sur les performances de transfert des *bundles* en termes de délai, de délivrance ordonnée et de perte. Chaque *bundle* émis n'emprunte pas forcément le même chemin de par la nature fortement mobile des noeuds. Ainsi, bien que le délai moyen puisse être relativement stable, l'écart type du délai de chaque bundle pour un flot donné peut être très grand. Dans nos expérimentations, cela se traduit par un taux de déséquencement (*reordering ratio* [75]) relativement élevé, souvent au dessus des 50% avec des pics à 90% alors que dans les réseaux "classiques" ce taux ne dépasse généralement pas les 5% [76, 77]. La structure du réseau en termes de connexité, densité de contact et centralité des noeuds évolue également en fonction du temps. Dans ce cas, le délai moyen peut fortement varier.

Un *bundle* reste dans le réseau jusqu'à expiration de son Time To Live. Ceci implique que le taux de perte observé sur un réseau DTN peut être dû à des expirations de TTL. De plus, la quantité de données potentiellement transférable durant un contact est limitée par la durée de ce contact. Dans le cas de congestion, ceci peut ralentir la dissémination des *bundles* dans le réseau et accroitre le nombre d'expirations de TTL. De manière évidente la congestion résulte également en des débordements de buffers avec une probabilité de rejeter toutes les copies d'un *bundle* et donc augmenter également le taux de perte.

Toutes ces caractéristiques intrinsèques aux DTNs empêchent l'utilisation d'applications nécessitant un service fiable ou ordonné. Lorsque le taux de déséquencement est faible ou que la gigue du réseau reste stable, les mécanismes de retransmissions sont des solutions possibles. Dans le cadre des Deep Space Networks (DSN), c'est le choix retenu par le protocol LTP-T protocol [12]. Dans ce contexte, il n'y a ni déséquencement, ni variation du RTT. Ainsi, le nombre de fausses détections de pertes est faible et il est alors possible d'attendre une période de temps raisonnable afin de prendre une décision de retransmission. Cependant dans le cas des DTNs, la combinaison de valeurs de gigue et de déséquencement importantes empêche l'utilisation de mécanismes ARQ. Leur utilisation engendrerait d'ailleurs un fort taux de retransmission injustifiées (*spurious retransmission*).

En ce qui concerne la réception ordonnée des *bundles*, le récepteur a besoin d'initialiser une horloge permettant d'attendre les messages manquants avant de les transmettre à la couche applicative. Il en résulte à la fois un plus haut délai de délivrance des données

à l'application et des pertes injustifiées puisque cette période d'attente peut conduire l'application à considérer ces paquets retardés comme perdus.

La conception d'un mécanisme permettant de déterminer si un paquet est perdu ou déséquencé dans une borne de temps raisonnable est alors délicat et les solutions possibles, si elles existent, seraient alors dépendantes du contexte d'utilisation (à la fois du point de vue du type de mobilité que de l'application). Tetrys se dispense de l'utilisation d'ARQ tout en permettant un service fiable sans retransmission des données manquantes. Les *bundles* délayés ou perdus sont récupérés de façon ordonnée. Ceci accélère considérablement la transmission des données à la couche supérieure et permet de contourner l'utilisation d'un algorithme permettant de décider si un paquet manquant est perdu ou en retard.

5.3 Evaluation sur un réseau DTN

Le mécanisme de Tetrys utilisé reste inchangé et conforme à la description donnée au Chapitre 3. Les données source sont ségmentées en *bundles* qui seront définis ci-après. Nous allons maintenant évaluer les performances de Tetrys en termes de taux de transfert à l'application ainsi qu'en termes de délai comparé aux mécanismes de fiabilisation classiques.

5.3.1 Mécanismes étudiés

Nous comparons Tetrys aux mécanismes suivant :

a) Stratégie sans codage : dans ce cas les messages sont envoyés sans modification et transportés via le mécanisme de routage sous-jacent ;

b) Mécanisme de code à effacement (FEC(n, k)) : nous considérons ici un code en bloc classique similaire à celui présenté au Chaptire 2. Les paramètres n et k influencent le codage comme suit : après l'émission de k *bundles* de données B_1,..,B_k, FEC(n, k) ajoute $(n-k)$ bundles de redondance (F_1..F_{n-k}) qui sont entrelacés avec les bundles de données du prochain bloc suivant la séquence d'émission : B_k, F_1, B_{k+1}, F_2.. B_n,F_{n-k},B_{n+1},B_{n+2},(...). Nous supposons un code parfait (MDS), ce qui signifie que la reconstruction d'un bundle perdu pour un bloc donné est possible aussitôt que k bundles (B ou F) sont reçus. Concernant la capacité de correction, on note que les codes MDS sont optimaux et donc meilleurs que les codes LT étudiés dans [71](lesquels requièrent $(1 + \epsilon) \cdot k$ bundles (avec $\epsilon > 0$)). Pour rappel, les principaux avantages des codes Fountain sont leur faible complexité et le fait qu'ils soient sans rendement. Ces deux paramètres ne sont pas critiques

dans notre contexte (le débit transférable sur le réseau est négligeable par rapport au délai de décodage). Pour FEC(n, k) et Tetrys, on note R le taux de redondance tel que $R = \frac{n-k}{n}$.

c) *Automatic Repeat ReQuest (ARQ)* :

Puisque les codes FEC ne fournissent pas un service 100% fiable, il aurait été intéressant d'étudier un protocol basé sur ARQ. Suivant ce que nous avons dit en section 5.2, il est évident que cette classe de mécanisme n'est pas applicable dans notre contexte. En supposant que le problème relatif à l'identification de la perte soit résolu, le délai de récupération des bundles perdus serait augmenté par au moins un RTT. La même remarque reste vraie pour Hybrid-ARQ où le principe de base est de coder les données avec FEC et de retransmettre des paquets source ou de redondance (ARQ) afin de reconstruire des blocs ayant $n - k$ pertes. Nous ne présenterons pas ici des expérimentations avec des mécanismes basés sur ARQ et HARQ que nous ne considérons pas pertinents pour notre évaluation.

De façon pratique, les récepteurs des codes FEC (ou des stratégies non-codées) doivent implémenter une horloge afin de déterminer si un bundle est perdu ou retardé. Comme déjà dit précédemment, ceci résulte en une augmentation du délai. Dans notre évaluation, les codes FEC n'accusent pas ce surcoût de délai car on considère qu'ils bénéficient d'un Oracle qui permet de déterminer à l'avance si un bundle est perdu. Cependant, il est à noter que Tetrys ne bénéficie pas de cet Oracle puisqu'il reconstruit tous les paquets. Afin de laisser au lecteur la possibilité d'apprécier ce lui même le compromis entre délai et fiabilité, nous utilisons à la fois les métriques de délai de livraison ordonnée et de taux de perte (i.e. ratio entre le nombre de bundles reçus et envoyés) observées par l'application.

5.3.2 Configuration du réseau

FIGURE 5.1: Le jeu de données RollerNet est issu de la randonnée Roller de Paris.

Afin d'évaluer tous ces mécanismes, nous rejouons les traces de connexion issues de l'expérience RollerNet [23], laquelle a consisté à mesurer les opportunités de contact entre les participants (≈ 1000) de la randonnée Roller de Paris. Comme on peut le voir sur la figure 5.1, le réseau ainsi formé est particulièrement dense et connexe la plupart du temps. Cependant, les connections et déconnections trop fréquentes empêchent l'utilisation de protocoles MANET [23].

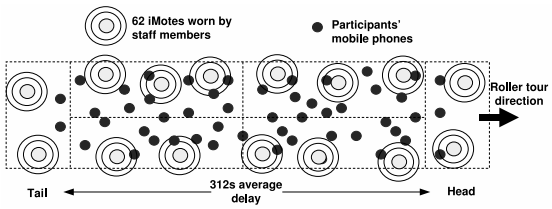


FIGURE 5.2: Le réseau RollerNet présente une structure linéaire.

On considère un flot de données qui génère 1/2 bundle par seconde entre les deux extrémités du cortège qui possède une structure linéaire comme on peut le voir sur la figure 5.2 (*i.e.* des noeuds spécifiques en tête et en queue sont identifiés). Nous utilisons un routage Spray and Wait (SW) [27] à 16 copies qui dissémine des bundles parmi les 62 iMotes déployés. Concernant le chemin de retour, nous utilisons le mécanisme de réception active *active receipt* [73] via un routage épidémique. D'autre mécanismes peuvent être utilisés, y compris via une transmission des acquittements au travers d'une architecture tierce (réseau GSM ou 3G [73]). Le TTL des bundles est infini et ces derniers sont générés durant 5000 secondes. Les messages bundles peuvent être reçus jusqu'à la fin de l'expérimentation à $t = 7000$ secondes.

La capacité des liens est limitée par la vitesse du Bluetooth ($\approx$ 700 KB/s) et chaque buffer a une taille de 300 MB. Afin de générer un trafic de fond, 50 paires sont choisies aléatoirement pour générer 1/2 bundle par seconde. Chaque bundle (data source ou de redondance) a pour taille 200 KB. Nous avons mesuré un taux de déséquencement de 95% avec une dispersion (*average extent*) de 348 bundles et un taux de délivrance des paquets (*delivery ratio*) de 82% (voir [75] section 4.2 pour ces métriques).

5.3.3 Résultats

La figure 5.3 montre la distribution du délai de livraison (ordonné) de Tetrys, FEC$(600, 300)$, FEC$(1000, 500)$ et de la stratégie sans codage. Nous pouvons voir que sous les mêmes conditions, la probabilité qu'un bundle soit délivré à l'application avant un délai donné est significativement supérieure avec Tetrys qu'avec FEC. Par exemple, un bundle est délivré avant 1000 secondes avec une probabilité de 0.48 pour Tetrys tandis que nous

obtenons seulement 0.16 pour FEC$(1000, 500)$. Il est intéressant de noter qu'en plus de récupérer les pertes, les codes à effacement permettent de réduire le délai résultant du déséquencement du réseau.

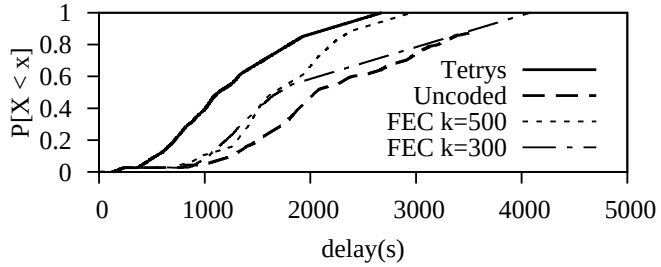


FIGURE 5.3: Distribution du délai de livraison (dans l'ordre) à l'application avec $R = 0.5$

La première ligne du tableau 5.1 montre que pour un taux de redondance de 0.5, Tetrys a un délai de délivrance moyen de 1251 secondes ce qui est significativement plus bas que le délai minimum moyen obtenu par FEC (1430 secondes avec $k = 100$). Nous pouvons également noter que FEC atteint un service 100% fiable quand $k \geqslant 300$. Cependant, comparé à Tetrys, les mécanismes basés sur FEC n'atteignent pas des délais et des taux de délivrance pour les mêmes valeurs. Ceci résulte en un étroit compromis entre délai et fiabilité. En moyenne, Tetrys permet au contraire d'obtenir un moindre délai de délivrance sans compromettre la fiabilité.

	Uncoded	Tetrys	FEC $k = 10$	FEC $k = 50$	FEC $k = 100$
$R = 1/2$	2172 - 0.82	1251 - 1.00	1870 - 0.89	1486 - 0.92	1430 - 0.92
$R = 1/3$	2172 - 0.82	2145 - 1.00	1829 - 0.87	1700 - 0.88	1564 - 0.89
$R = 1/4$	2172 - 0.82	2246 - 1.00	1912 - 0.85	1660 - 0.85	1775 - 0.85
	Uncoded	Tetrys	FEC $k = 200$	FEC $k = 300$	FEC $k = 500$
$R = 1/2$	2172 - 0.82	1251 - 1.00	1600 - 0.96	2035 - 1.00	1718 - 1.00
$R = 1/3$	2172 - 0.82	2145 - 1.00	1586 - 0.88	1689 - 0.92	1779 - 0.90
$R = 1/4$	2172 - 0.82	2246 - 1.00	1573 - 0.87	1662 - 0.88	2024 - 0.85

TABLE 5.1: Délai moyen de réception ordonnée (secondes) - taux de transfert en fonction du taux de redondance (Nous rappellons que le délai obtenu par les codes FEC bénéficie de l'Oracle présenté en section 5.3.1).

Cette même table montre également que si le taux de redondance est réduit à $R = 1/3$ ou $R = 1/4$, FEC n'est pas capable de reconstruire tous les bundles perdus, même avec de larges blocs. Dans ce cas, nous ne pouvons pas comparer à la fois le délai de délivrance ordonné de Tetrys et de FEC, étant donné que ce dernier ne reconstruit pas tous les bundles perdus et possède l'avantage de l'Oracle. Par exemple, pour $R = 1/4$, si on considère 85% des bundles reçus, le délai moyen de Tetrys n'est plus que 1786 secondes (contre 2246 secondes). Ceci correspond au couple (délai, fiabilité) atteint par FEC pour $k =$10, 50, 100, 500.

Bien que nous apportons une attention particulière à la délivrance ordonnée des bundles, Tetrys reste un très bon candidat si nous supprimons cette contrainte. Avec $R = 1/2$, le délai moyen est de 1125 secondes pour Tetrys, 1295 secondes pour FEC avec $k = 300$ et 1243 secondes pour FEC avec $k = 500$. Enfin, si on considère que FEC reconstruit tous les bundles perdus, le délai de Tetrys sera le plus souvent inférieur ou égal au délai obtenu par FEC.

5.4 Discussion

5.4.1 Choix des paramètres

La tableau. 5.2 montre la variation du délai moyen de récupération des pertes en fonction du taux de perte [4] et du taux de redondance. Pour chaque bundle récupéré, ce délai correspond au temps écoulé entre la détection de sa perte (grâce aux numéros de séquence) et sa reconstruction. Le tableau indique $\emptyset$ dans le cas où Tetrys n'a pas réussi à décoder tous les bundles perdus. Ce cas correspond bien évidemment à un taux de perte supérieur au taux de redondance.

Concernant le délai de décodage, nous observons que pour un taux de pertes donné, plus grand est le taux de redondance, plus petit est le délai de décodage. Par exemple, pour un PLR de 18%, Tetrys atteint un délai moyen de décodage de 252 secondes pour un taux de redondance de 0.5. Ce délai vaut respectivement 928 et 1395 secondes pour un taux de redondance de 1/3 et 1/4. Pour un taux de redondance fixé, le temps de décodage s'accroit donc conjointement avec le taux de perte comme montré pour $R = 1/2$.

Suivant ces résultats, il est évident que l'estimation d'un taux de perte moyen est nécessaire pour déterminer les performances du système. Comme ce dernier dépend du routage et du contexte de mobilité, cette information peut-être fournie par les couches plus basses grâce à une métrique *cross-layer*. Ces informations peuvent aussi être fournies par les messages de la voie retour afin de d'affiner dynamiquement la valeur du taux de redondance.

Dans une prochaine étude, nous chercherons l'obtention d'une dérivation analytique de la valeur du délai de décodage en fonction de métriques réseaux telles le taux de perte moyen et le taux de déséquencement.

4. Nous faisons varier la taille du buffer d'un noeud relai de 140MB à 310MB afin d'obtenir differents taux de perte.

Taux de perte de bundles	$R = 1/2$	$R = 1/3$	$R = 1/4$
15.0 %	217 sec	792 sec	1209 sec
18.0 %	252 sec	928 sec	1395 sec
19.3 %	446 sec	1064 sec	$\emptyset$
22.2 %	419 sec	1117 sec	$\emptyset$
24.6 %	625 sec	1376 sec	$\emptyset$
35.8 %	854 sec	$\emptyset$	$\emptyset$
40.0 %	937 sec	$\emptyset$	$\emptyset$
47.5 %	1380 sec	$\emptyset$	$\emptyset$

TABLE 5.2: Délai moyen de récupération des pertes en fonction du PLR et du taux de redondance.

5.4.2 Évaluation de la complexité

a) Complexité en termes de calcul : la complexité en temps de calcul (voir Chap 3) par bundle au niveau de l'émetteur et du récepteur est de $O(\frac{RTT \cdot \lambda}{k})$ avec (n, k) et les paramètres de codage, λ le taux d'émission de l'application source en bundles par seconde et RTT le délai moyen entre l'émission d'un bundle et la réception de son acquittement correspondant. À titre de comparaison, la complexité (simplifiée) par bundle du mécanisme FEC est de $O(k)$. Selon [78] avec une fenêtre d'encodage de 520 bundles, nous aurions pu atteindre un débit maximum de $210KB/s$ sur un PIII 933 Mhz, soit deux fois le débit dont nous avons fait l'hypothèse. En fonction du compromis entre la complexité et le coût d'un mécanisme d'acquittement, nous pouvons adapter ce dernier (*active/passive/bridged*) [73] puisque plus petit est le délai de retour, moindre est la complexité.

b) Complexity en termes de mémoire : la complexité de Tetrys concernant les buffers est également de l'ordre de $O(RTT * \lambda)$ puisque les bundles source restent dans le noeud source jusqu'à ce qu'ils soient acquittés. Au niveau du récepteur, les bundles source restent dans le buffer tant qu'ils participent à l'encodage des bundles de redondance. De son côté FEC a besoin d'un nombre fixe de k bundles à la fois du côté émetteur et récepteur.

Dans le scénario étudié, la complexité de Tetrys reste du même ordre de grandeur que les autres mécanismes de codage aussi bien en termes de calcul que d'espace.

5.5 Conclusion

Les caractéristiques des DTN se sont pas du tout adapté aux applications *streaming-like*, limitant ainsi leur intérêt. Nous avons montré le décodage ordonné, la fiabilité totale et la

facilité de configuration de Tetrys rend possible le support de ces applications et améliore nettement les performances comparé à une solution basée sur des codes en blocs.

Chapitre 6

Fiabiliser les *handovers* verticaux

Contents

Avec la prolifération des nouvelles technologies d'accès sans-fil, les utilisateurs nomades peuvent maintenant accéder à l'Internet via différents réseaux d'accès. Les caractéristiques de ces réseaux sont très différentes. En effet, les réseaux locaux sans-fil (WLAN) permettent un accès haut débit avec une latence de l'ordre de dix millisecondes tandis que les réseaux métropolitain sans-fil (WWAN) permettent une couverture plus large au prix d'un accès beaucoup moins rapide et d'une latence beaucoup plus élevée. Pour des raisons de coûts ou de performances, les utilisateurs de nouveaux périphériques mobiles type *smartphone*, qui prédominent maintenant dans notre quotidien, basculent dès que possible vers un point d'accès WiFi et ce, bien que la connexion cellulaire soit toujours disponible. Aussi, la possibilité de basculer de façon transparente entre ces différentes technologies permet à l'utilisateur de maximiser son débit de connexion et à l'opérateur de libérer les ressources du WWAN en maximisant l'utilisation du WLAN.

Il existe des solutions de management complètes permettant de basculer entre différentes technologies d'accès. Des protocoles tels que Mobile IP [79], peuvent être utilisés afin d'assurer la continuité de la connexion sans-fil. Paradoxalement, lors d'un *handover*

vertical [1] le trafic est souvent interrompu ou perturbé à cause des déficiences inhérentes aux protocoles de gestion de mobilité [80]. Bien que d'autres systèmes de gestion de la mobilité (tel l'architecture Safetynet [81]) permettent de mitiger le nombre de paquets perdus via une procédure de *bicasting* [2], les perturbations caractérisant ces *handovers* provoquent inévitablement des pertes de paquets ou des déclenchements de RTO. Ceci est tout particulièrement le cas des *upward vertical handovers* (handovers depuis un WLAN vers un WWAN) qui sont réalisés lorsque la dégradation du rapport signal/bruit du WLAN devient trop importante pour continuer à assurer une connectivité correcte.

Il existe deux grandes classes de solutions adressant le problème d'un point de vue transport. La première cherche à améliorer la tolérance de TCP vis à vis des *handovers* [82, 83] tandis que la seconde propose l'utilisation de la propriété multi-chemin (*multipath*) de SCTP [84] afin de bénéficier de la multi-connectivité [85, 86] offerte par les nouveaux périphériques mobiles.

Dans ce chapitre, nous utilisons à nouveau les propriétés du protocole Tetrys qui peut s'insérer tout naturellement au sein de ces systèmes de gestion de mobilité tels Fast Handovers Mobile IPv6 ou Safetynet, sans opérer de modifications des hôtes d'extrémité. En d'autres termes, l'architecture proposée ci-après est complètement indépendante du protocole de transport déployé de bout-en-bout. Enfin, bien que nous nous focalisions sur la fiabilisation de la procédure de *handover*, nous verrons que ces travaux sont généralisables dans le cadre d'une utilisation multi-chemin pure.

6.1 Pourquoi un n-ième système de management de la mobilité ?

Adapter TCP au contexte du *handover* nécessite malheureusement une refonte de la pile protocolaire Internet. De plus, la définition des nouveaux paramètres à prendre en compte (nouvelles valeurs du TCP *timeout* (RTO) et de la fenêtre de congestion *cwnd* adaptée au nouveau lien) nécessite soit de mesurer précisement le délai de la connexion, soit d'avoir une certaine connaissance *a-priori* des caractéristiques des liens. Mis à part un déploiement de SCTP limité aux systèmes de type Unix ou GNU/Linux, le principal frein des approches basées sur ce type de protocole *multipath* est le besoin de modifier les interfaces d'extrémités (API) entre l'application et la couche transport afin de l'utiliser

1. Un *handover* qualifie le basculement d'un réseau d'accès à un autre. Il est dit *horizontal* lorsqu'il n'y a pas de déconnexion ou entre deux réseaux de même technologie et *vertical* lorsqu'il y a déconnexion ou entre deux réseaux de technologies d'accès différentes.

2. Procédure durant laquelle le flot est dupliqué sur les deux réseaux (l'ancien et le nouveau) dans lequel se trouve un noeud en situation de *handover*.

(les applications devant utiliser la `socket` SCTP). Ceci rend *de facto* le déploiement de ces propositions lente et délicates.

Lorsque la procédure de *handover* est réalisée en utilisant une méthode de *bicasting*, le flot de données TCP est dupliqué sur les deux réseaux (ancien et nouveau). Ainsi, même lorsque des codes FEC sont employés, l'utilisation de ces mécanisme réduit de manière drastique (lorsqu'il ne la réduit pas de moitié) la somme de la bande passante utile disponible sur les interfaces.

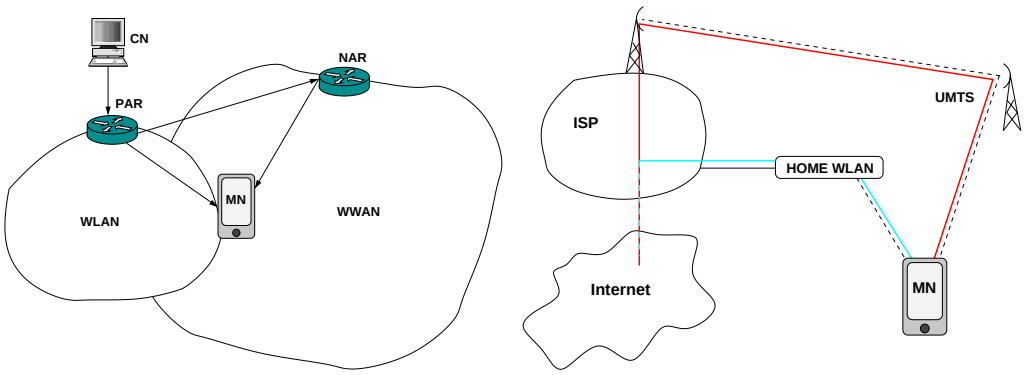
Enfin pour conclure, à l'exception des systèmes utilisant le *bicasting*, aucune de ces propositions ne semble être résistante aux forts taux de perte liés à la diminution de la qualité du signal et à la procédure de *handover*. À notre meilleure connaissance, il n'existe à ce jour pas de solution permettant de bénéficier à la fois de la robustesse et de l'agrégation de bande passante fournie par les capacités de *multihoming* des nouveaux périphériques sans-fil avec une application utilisant la pile TCP/IP standard (non modifiée). Ceci motive donc la proposition que nous faisons ici.

6.2 Notre proposition

L'idée que nous développons ici est de fiabiliser la partie du chemin qui peut être affectée par un fort taux de perte durant le *handover* grâce à Tetrys. Cette fiabilisation peut se faire de bout en bout si le noeud de communiquant [3] (CN) est conscient des différentes adresses du noeud mobile (MN). Sinon, comme le suggère la figure 6.2, dans le cas où le *multihoming* est obtenu par une solution de mobilité de niveau IP, le codage/décodage peut également se réaliser entre le routeur d'accès (AR) et le MN. Les données échangées entre le MN et l'AR ne sont alors pas protégées par Tetrys.

La figure 6.1 illustre deux cas d'utilisation de notre solution de fiabilité. D'une part, comme le décrit la figure 6.1(a), Tetrys peut être utilisé que lors du handover. On pourra distinguer le cadre les *hard vertical handover* (les deux interfaces ne sont pas utilisées simultanément) ou bien les *soft vertical handover* (les deux interfaces sont utilisées de manière concurrente durant un certain laps de temps). D'autre part (*c.f.* figure 6.1(b)), les interfaces peuvent être utilisées en parallèle de manière à palier aux faiblesses du WLAN ou à agréger les bandes passantes disponibles de façon transparente aux protocoles de transport.

3. Le terme utilisé dans la littérature est *Corresponding Node*, en l'occurrence celui qui communique avec le noeud mobile.



(a) *Handover* classique avec utilisation des deux interfaces

(b) Utilisation opportuniste des deux interfaces pour agréger la bande passante disponible

FIGURE 6.1: Deux illustrations du multi-chemin.

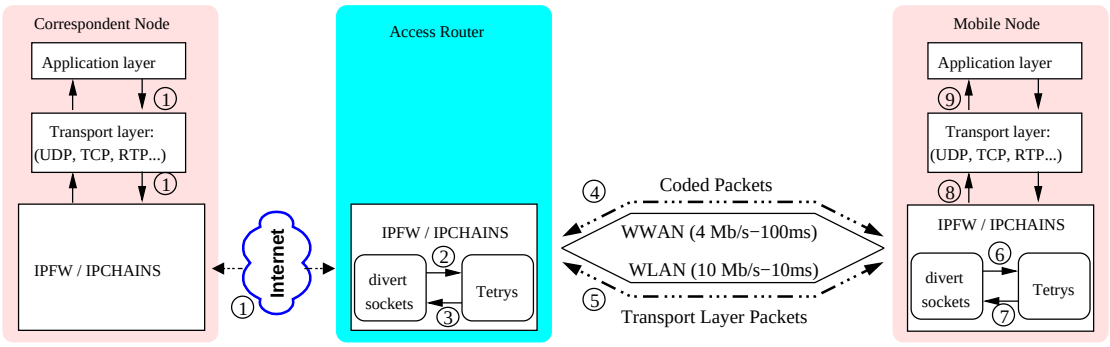


FIGURE 6.2: Comment insérer Tetrys dans la pile protocolaire.

De façon pragmatique, la figure 6.2 présente une solution possible de déploiement de Tetrys au niveau 3. Dans notre cas, nous avons utilisé l'implémentation `divert socket` de BSD également disponible sous GNU/Linux avec l'API `ipchains`.

L'envoi de segments de données entre le CN et le MN fonctionne comme suit : tout d'abord (première étape), les paquets traversent normalement la pile TCP/IP puisque le CN n'est pas impliqué dans le processus d'encodage. Les paquets qui atteignent l'AR suivent les règles de *IPFW* qui redirige alors (seconde étape) les paquets à destination du MN à l'instance de Tetrys correspondante (il y a une instance par MNs supportés par l'AR). Tetrys ajoute les paquets à sa fenêtre d'encodage et les ré-injecte (étape 3 et 4) avec un numéro de séquence et un bit dans le champs option IP permettant de différencier un paquet de redondance d'un paquet de données. Les paquets de redondance sont alors ré-injectés avec l'adresse IP nécessaire et sont envoyés via le WWAN (étape 4) ou le WLAN (étape 5) suivant si le *handover* est montant ou descendant.

La taille des paquets codés est égale à la taille maximum des paquets de données courant dans la fenêtre d'encodage de Tetrys. Les paquets qui atteignent le MN, via des interfaces différentes, sont re-dirigés vers Tetrys (étape 6) qui décode et reconstruit les paquets manquants. L'ensemble des paquets reçus par l'encodeur Tetrys à l'étape 2 est ré-injecté

(étape 7) de manière ordonnée. Enfin en étape 8, les paquets sont donnés à la couche transport de façon transparente.

Lorsque Tetrys est utilisé afin d'améliorer la qualité du lien (avec deux interfaces), les paquets de données source sont envoyés sur l'interface la plus rapide (qui est aussi celle qui accuse le plus de pertes dans notre scénario expérimental) tandis que les paquets de redondance sont envoyés sur le WWAN.

Pendant un *handover* montant [4], le PLR est estimé [5] par Tetrys et lorsqu'il dépasse un seuil donné (70% dans notre scénario), l'encodage des données Tetrys est démarré et les paquets résultants sont émis sur le WWAN. Lorsque le WLAN est hors de portée, l'émission de paquets sur le WLAN cesse. Durant un *handover* descendant, les paquets sont envoyés codés sur le WWAN et non codés sur le WLAN. Lorsque le PLR du WLAN décroit en dessous du seuil, les paquets encodés sont envoyés selon le taux de redondance calculé sur le WLAN jusqu'à ce que le PLR soit négligeable pour TCP.

6.2.1 Allocation de la redondance et interaction avec TCP

De part le fait que TCP considère les pertes comme une indication de congestion, ses performances sont médiocres en présence de pertes ayant une autre cause (comme c'est le cas pour le *handover*).

Une solution possible, couramment employée et évaluée est l'utilisation de codes bloc (de bout en bout ou entre le MN et le CN) pour corriger les pertes dues à des erreurs sur le lien.

Cependant, dans [87] [6], les auteurs montrent que l'utilisation de code en bloc de bout en bout ne permet pas de résoudre le problème lorsque le taux de perte devient significatif (supérieur à 10%). Ceci est dû au fait que TCP requière une livraison des paquets dans l'ordre et qu'il est sensible aux variations de RTT (induits par le décodage par bloc) ce qui implique des retransmissions abusives via l'expiration du RTO et donc une réduction du débit.

Le fait que Tetrys corrige l'intégralité des pertes lui profère un avantage certain comparé aux codes bloc. Néanmoins pour corriger L pertes, il requière la réception de L paquets

4. Un handover montant (*Upward handover*) décrit un handover depuis un réseau WLAN vers un réseau WWAN.

5. Les paquets de données sont reçus par Tetrys qui ajoute une entête contenant un numéro de séquence permettant la détection des pertes au niveau du récepteur. Le récepteur Tetrys envoi des acquittements reportant le taux de perte mesuré. Dans le cas d'un *hard vertical handover*, le début du handover peut être détecté par l'absence d'accusé avant une durée comprise entre un et deux RTT.

6. Dans leur implémentation, TCP a dû être modifié afin que le contrôle de congestion ignore les pertes. Dans notre cas, nous faisons l'hypothèse d'une séparation complète entre les couches réseau et transport.

de redondance. Cela signifie que si il y a eu plus de $R \cdot cwnd$ pertes en l'espace d'une fenêtre de congestion de TCP (*cwnd*), le décodeur Tetrys n'est pas capable de récupérer les paquets manquants et la connexion sera gelée tant que $\frac{L-R \cdot cwnd}{R}$ retransmissions n'auront pas été reçues. Naturellement nous avons relevé que lorsque cela se produit, TCP entre en mode `backoff` et bien souvent la connexion s'arrête.

Pour prévenir ce phénomène, en sus du paquet de redondance envoyé tous les k paquets source, nous proposons de gérer l'émission de la redondance sur la base d'une fréquence temporelle.

Le seul critère à remplir pour que les erreurs soient reconstruites dans les temps tout en minimisant la quantité de redondance émise est le suivant :

$$\frac{R \cdot cwnd}{RTT} > f_r \geqslant \frac{L - R \cdot cwnd}{4 \cdot RTT}$$

avec f_r la fréquence d'envoi de redondance minimale (si on fait l'hypothèse que le RTO correspond à 4 fois le RTT).

L peut être remplacée par une valeur comprise entre *cwnd* (tous les paquets sont perdus) et $PLR \cdot cwnd$ selon l'importance associée au coût d'envoi de la redondance et celui associé aux *spurious retransmissions*. Quelle qu'elle soit, l'émission des paquets de redondance basée sur la fréquence n'entre n'intervient que si la fréquence d'envoi d'une redondance tous les k paquets source (*i.e.* $R \cdot \frac{cwnd}{RTT}$) descend sous le seuil de f_r, autrement dit, que si la progression de la *cwnd* est bloquée par plus de $R \cdot cwnd$ pertes. Ainsi, l'utilisation de f_r ne change que très peu la quantité de redondance totale émise.

6.3 Evaluation

Notre banc de test est similaire à celui présenté en figure 6.2, excepté pour les liens WLAN et WWAN qui sont émulés avec Netem [88] au dessus de deux liens Ethernet. Le CN et AR sont connectés avec un lien à 10Mbit/s ayant un délai de transmission négligeable ($\approx 1ms$). Par défaut, la capacité du lien WLAN est de 10Mbit/s avec 10ms de RTT (4Mbit/s et 100ms pour le WWAN). Ces valeurs sont cohérentes pour un réseau cellulaire UMTS HSDPA. Le taux de redondance par défaut est de 20%.

6.3.1 Comparaison avec FEC sur un lien à pertes

Le tableau 6.1 montre le débit obtenu par TCP au dessus de Tetrys ou FEC lorsque les paquets sont envoyés via le WLAN. On observe que pour un faible taux de perte, Tetrys

PLR	0.0	0.5	2	4	5
	8	10	12	16	20
TCP/Tetrys (4,5)	7.78/0.01	7.81/0.01	7.81/0.01	7.80/0.01	7.81/0.00
	7.82/0.01	7.81/0.01	7.81/0.01	7.18/0.02	4.6/0.26
TCP/FEC (4,5)	7.79/0.02	7.83/0.00	6.48/1.2	3.01/0.6	2.68/2.24
	Timeout				
TCP/FEC (8,10)	7.81/0.01	7.78/0.05	7.81/0.03	6.3/1.48	Timeout
	Timeout				
TCP/FEC (12,15)	7.82/0.02	7.82/0.01	7.79/0.03	7.54/0.10	4.06/5.05
	Timeout				
TCP/FEC (16,20)	7.81/0.01	7.81/0.02	7.82/0.01	7.825/0.01	Timeout
	Timeout				

TABLE 6.1: Débit/Ecart type en Mb/s pour $R = 0.2$, $B_D = 10Mbps$ avec les paquets de données et de redondance envoyés uniquement sur le WLAN.

PLR	0.0	0.5	2	4	5
	8	10	12	16	20
TCP/Tetrys (4,5)	9.54/0.00	7.69/0.18	6.5/0.49	8.18/0.34	9.02/0.05
	8.55/0.7	8.66/0.3	8.45/0.57	7.10/0.31	5.12/0.2
TCP/FEC (4,5)	9.53/0.00	5.96/0.02	3.07/0.08	1.13/0.6	1.25/0.14
	0.17/0.05	Timeout			
TCP/FEC (8,10)	9.54/0.00	7.19/0.08	4.72/0.32	2.71/0.46	1.85/0.40
	Timeout				
TCP/FEC (12,15)	9.55/0.00	7.7/0.11	4.79/1.36	3.73/1.13	Timeout
	Timeout				
TCP/FEC (16,20)	9.53/0.00	7.37/0.65	5.83/0.68	3.73/1.13	0.84/1.18
	Timeout				

TABLE 6.2: Débit/Ecart type en Mb/s pour $R = 0.2$, $B_D = 10Mbps$, $B_R = 4.0Mbps$ avec les paquets de données envoyés sur le WLAN et ceux de redondance sur le WWAN.

et FEC permettent le maintient d'un débit initial de $7.8Mb/s$. On observe que Tetrys et FEC assurent un faible taux de perte (perçu par TCP) permettant le maintien d'un débit de 8Mbit/s pour TCP. De façon similaire à de précédents travaux sur TCP/FEC [87], nous observons que lorsque le taux de perte est important, le débit TCP/FEC décroit et la connexion entre en mode `backoff` et stagne. Au contraire, TCP/Tetrys n'est pas aussi sensible à ces taux de pertes et ne décroit seulement qu'à partir de $PLR = 14\%$. Il est à noter que le taux d'encodage est fixé à 0.2 durant cette expérimentation. Il n'y aurait pas de diminution du débit TCP si nous avions ajusté dynamiquement ce taux en fonction d'un PLR estimé.

La table 6.2 montre le débit obtenu par TCP au dessus de Tetrys ou FEC lorsque les paquets de données sont envoyés via le WLAN (10Mbit/s, 10ms) et la redondance sur le WWAN (4Mbit/s, 100ms). Nous pouvons observer que les résultats obtenus avec TCP/-FEC sont pires que ceux donnés sur lien unique en table 6.1. C'est également le cas pour TCP/Tetrys avec de faible PLR (pour 0.5% ou 2% par exemple). Ces résultats s'expliquent par l'assymétrie du délai des deux liens et la fenêtre de TCP qui s'adapte au produit bande passante délai (BDP) du WLAN. Lorsque des pertes se produisent, la reconstruction nécessite l'attente de paquets de redondance qui arrivent 90ms plus

tard. Pendant ce temps, aucun paquet n'atteint la destination TCP et donc aucun acquittement ne permet de faire glisser la fenêtre d'émission TCP de la source, ce qui est à l'origine de nombreux déclenchements de RTO. Le RTT perçu par TCP augmente avec le PLR et la fenêtre TCP s'accroit jusqu'à ce qu'elle atteigne le BDP correspondant au lien de plus fort délai. Ceci explique les mauvaises performances (qui peuvent être améliorées en retardant artificiellement les paquets à la vitesse de l'interface la plus lente) de TCP/Tetrys pour des petits PLR et l'amélioration observée pour des PLR plus élevés.

Ces deux points : 1) l'asymétrie de délai du lien, et 2) les pertes non-récupérées par FEC, impactent fortement le débit TCP/FEC.

Bien que nous ne testons pas ici l'aspect d'agrégation de bande passante (comme dans SCTP), ces résultats montrent que contrairement aux précédentes stratégies de codage, Tetrys permet à TCP d'être utilisé dans des conditions fortement perturbées, notamment en ce qui concerne l' assymétrie des délais qui est la caractéristique principale de l'agrégation de bande passante de plusieurs chemins.

6.3.2 Illustration de Tetrys avec un scénario de handover

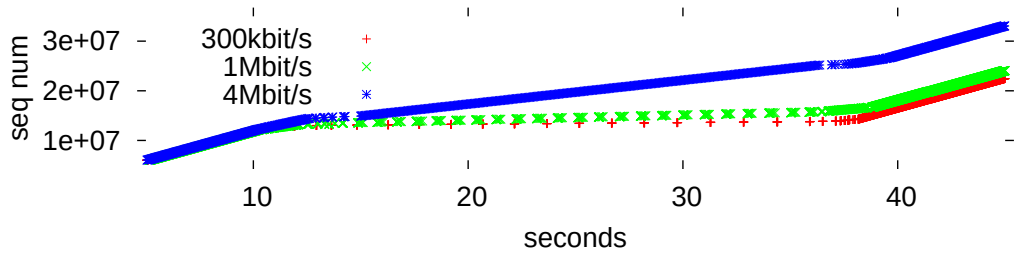


FIGURE 6.3: Numéro de séquence des paquets transmis par TCP en fonction du temps. Scénario de *handover* avec différentes capacité de WWAN(100 ms) et un WLAN fixé à ($C = 10$ Mb/s, $RTT = 10$ ms).

La figure 6.3 montre l'évolution des numéros de séquence émis par TCP (i.e. l'évolution du débit) pour différentes valeurs de la capacité du WWAN (300kbit/s, 1Mbit/s and 4Mbit/s). Il faut 0.5 seconde au lien WLAN pour passer de l'état actif à inactif (et inversement). On peut voir que le débit ne chute pas sensiblement lors du *upward vertical handover* (du WLAN vers le WWAN). En effet, durant la dégradation du lien (0.5 s), Tetrys doit attendre les paquets de redondance transmis sur le WWAN. Le délai perçu par TCP se rapproche ainsi progressivement des 100ms ce qui permet l'adaptation du RTO. De plus, lorsque le WLAN est finalement coupé, la proportion de paquets de TCP qui dépassent la bande passante disponible sur le WWAN est normalement jeté. Ce problème est bien connu et a mené à de nombreuses adaptations de TCP pour les *handovers* verticaux [82]. Dans le cas de Tetrys, les paquets de TCP en trop sont stockés

dans le buffer d'encodage puis reconstruits à la vitesse du WWAN en attendant que TCP adapte son débit. Cela explique le regroupement des numéros de séquence que l'on peut observer pour les courbes rouge et verte. En effet les paquets générés instantanément par TCP ne sont décodés que lorsque suffisamment de paquets de redondance ont été reçus sur le WWAN, générant ainsi un acquittement pour l'ensemble de la précédente fenêtre. Cela a pour effet d'augmenter la *cwnd* de TCP et donc le délai de décodage ainsi que le prochain RTT perçu par TCP.

Le cas du *downward vertical handover* est bien moins contraignant puisque Tetrys ne doit que corriger les pertes. L'utilisation de Tetrys permet donc à TCP de s'adapter à la nouvelle interface sans qu'il n'y ait un ralentissement important de la connexion. Ces observations restent valables pour différentes valeurs de RTT du WWAN.

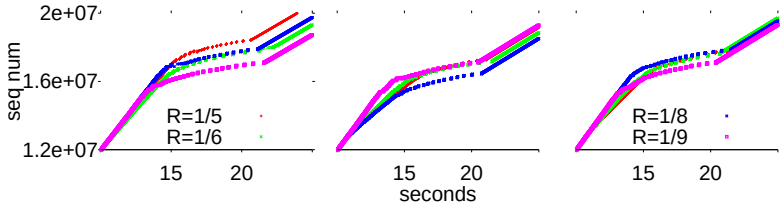


FIGURE 6.4: Scénario de *handover* avec différents taux de redondance (sur les courbes) avec ($C = 10$ Mb/s, $RTT = 10$ ms) pour le WLAN et ($C = 4$ Mb/s, $RTT = 100$ ms) pour le WWAN.

La figure 6.4 étudie l'impact du taux de redondance sur le débit de TCP durant un *handover* vertical. Les trois figures correspondent à différentes instances de simulation avec les mêmes paramètres ($10Mb/s$;$10ms$ pour le WLAN et ($4Mb/s$;$100ms$ pour le WWAN). Chacune des courbes correspond à l'évolution des numéros de séquence pour des taux de redondance de respectivement ($R = 1/5, R = 1/6, R = 1/8, R = 1/9$). Durant le *handover*, le PLR du WLAN passe de 0 à 100% en $0.5s$ et inversement 10 secondes plus tard. On peut voir qu'il n'y a pas de relation évidente entre le débit obtenu et les différents taux de redondance utilisés par TCP/Tetrys. En effet, lors d'un *handover*, ce paramètre ne nécessite pas d'adaptation puisque la redondance basée sur la fréquence minimale permet de reconstruire les pertes.

6.4 Conclusion

Une fois encore, dans ce chapitre, nous mettons en avant les propriétés avantageuses de Tetrys pour la protection des protocoles de transport lors d'un *vertical handover*. L'évaluation expérimentale illustre le fait que ce type de code peut être une stratégie complémentaire à un protocole de management de la mobilité. En sus de la simple gestion du *handover*, nous avons montré que Tetrys permet à TCP de maintenir son débit en

tirant parti des interfaces multiples sans fil présentes dans les *smartphones* actuels. Cette dernière caractéristique abonde vers une utilisation de Tetrys en tant que mécanisme de fiabilité pour permettre aux protocoles de transport actuels d'agréger la bande passante disponible dans un contexte multi-chemins ou multi-interfaces.

Troisième partie

Travaux prospectifs

Chapitre 7

Réseaux Anarchiques

Contents

Dans ce chapitre, nous nous intéressons aux réseaux dits « anarchiques » qui sont caractérisés par une absence totale de contrôle de congestion aux extrémités. Contrairement à TCP [1] qui domine les hôtes de l'Internet et qui assure une utilisation efficace et équitable du réseau via une adaptation volontaire et collaborative du débit, dans les

1. Les réseaux anarchique peuvent utiliser les format de paquet de n'importe quel protocole de transport (*e.g.* TCP, UDP...)

réseaux anarchiques les sources peuvent émettre à la vitesse maximale de leurs interfaces sans souci d'une quelconque équité. Dans ce chapitre, nous chercherons à compléter les précédents travaux sur le sujet ([89–91]) d'une part via l'implémentation d'un mécanisme de transport anarchique et d'autre part, en adressant des points tels que l'efficacité et l'équité obtenues en fonction des topologies. Enfin, nous étudierons la possibilité d'utiliser des applications temps réel sur de tels réseaux.

Nous proposons une méthode de configuration dynamique des paramètres d'encodage de Tetrys afin d'illustrer son efficacité en tant que mécanisme de transport pour réseaux anarchiques. En particulier, nous montrons que ce mécanisme est capable de concurrencer les réseaux actuels gérés par des sources à contrôle de congestion. Nous montrons entre autre que pour des RTT de l'ordre d'une centaine de milli-secondes, l'efficacité [2] obtenue est du même ordre que celle des contrôles de congestion actuels. Pour des RTT plus longs, notre étude montre que la version « anarchique » remplace avantageusement le principe du contrôle de congestion. Du point de vue de l'équité, il s'avère que contrairement à ce qui était pressenti dans de précédents travaux, la solution proposée (couplée à un mécanisme de rejet équitable des paquets sur les routeurs *Fair-Drop*) permet aux réseaux anarchiques d'approcher une meilleure allocation *max-min fair* que TCP.

7.1 Introduction

7.1.1 Topologies, Congestion Collapse et Contrôle de Congestion

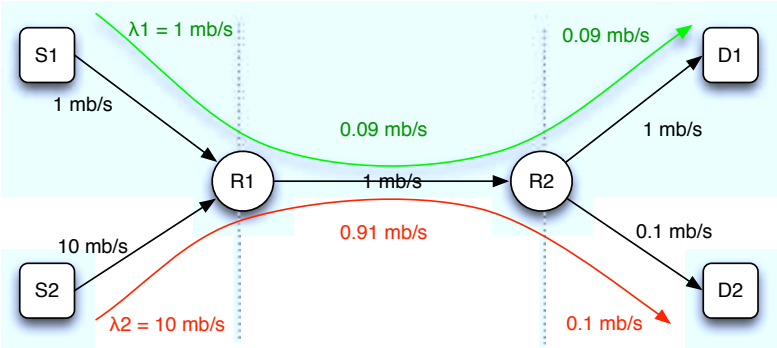


FIGURE 7.1: Illustration d'une perte d'efficacité liée à la topologie et l'absence de contrôle de congestion. Les sources S_1 et S_2 émettent respectivement à $\lambda_1 = 1mb/s$ et $\lambda_2 = 10mb/s$ et obtiennent un débit de 0.09mb/s et 0.91mb/s sur le lien R1-R2. La somme des débits à la réception est de 0.2mb/s ce qui est largement sous-optimal puisque si $\lambda_1 = 0.9mb/s$ et $\lambda_2 = 0.1mb/s$ la somme des débits à la réception serait de $1Mb/s$. Cette sous efficacité est souvent pointée du doigt comme une cause potentielle du *congestion collapse*.

2. Ici nous considérons l'efficacité comme le rapport entre la capacité maximum du réseau et la somme des débits utiles obtenus par les applications

7.1.1.1 Origine du congestion collapse

Les retransmissions : Le phénomène de *congestion collapse* est souvent considéré comme synonyme d'absence de contrôle de congestion. En effet, ce phénomène fut initialement observé avec l'utilisation de la première version de TCP [2] (1981) qui n'effectuait uniquement qu'un contrôle de flux via une fenêtre de taille fixe. Dans ce contexte, lorsque la capacité d'un lien est atteinte, l'accroissement du RTT résultant de l'augmentation de la taille des files d'attente est parfois plus rapide que le rafraichissement du RTO. Ainsi, des paquets en transit peuvent alors être considérés comme perdus. De plus, à la détection d'une perte, les paquets qui suivent sont également considérés comme perdus et retransmis (phénomène connu sous le nom de retransmission abusive ou *spurious RTO*). Ces paquets déjà reçus augmentent alors la probabilité de perte. Un mécanisme d'espacement croissant des retransmissions permet néanmoins au réseau de rester stable, bien qu'inefficace [92].

La topologie : En sus du mécanisme de retransmission, la topologie a largement été pointée du doigt comme facteur aggravant du *congestion collapse*. La figure 7.1 en est une illustration. En négligeant les effets de synchronisation des paquets, le débit du flot issu de S_2 lui permet d'accéder au lien à $\frac{\lambda_2}{\lambda_1+\lambda_2} = 0.91Mb/s$ alors qu'il ne dispose que d'un accès à $0.1Mb/s$ sur le lien $R_2 - D_2$. Les paquets du flot issu de S_2 sont donc jetés par R_2 à une vitesse de $0.81Mb/s$. Les paquets qui occupent de la bande passante avant d'être jetés plus loin dans le réseau sont considérés comme « morts » [93].

Inversement, le flot issu de S_1 n'occupe que 10% du lien (R_1, R_2) et le débit total des deux récepteurs est de $0.2Mb/s$. Le débit utile maximum que l'on peut obtenir sur cette topologie est de $1Mb/s$ si S_1 et S_2 adaptent leur débit ($\lambda_1 = 0.9$ et $\lambda_2 = 0.1$).

Définition 7.1.1. *Paquets morts : Un paquet est dit « mort » [93] si il a déjà occupé de la bande passante sur des goulots d'étranglement avant d'être jeté par un goulot d'étranglement en aval de ces derniers. Ainsi, la bande passante occupée sur le ou les précédents goulots d'étranglement se fait au détriment des autres flots qui auraient pu en bénéficier.*

Dans cet exemple, il y a une réduction du débit utile par un facteur de 5 mais notons tout de même que le réseau reste utilisable. Il existe cependant des topologies particulières (cycliques) où la proportion de paquets morts implique que le débit peut tendre vers zéro [93–95]. On retrouve ces topologies dans les réseaux *token ring* dont l'utilisation tend de nos jours à disparaitre. Certains fournisseurs d'accès à Internet ont également des topologies en anneau. Cependant, ce problème d'effondrement est prévenu par un contrôle d'accès efficace.

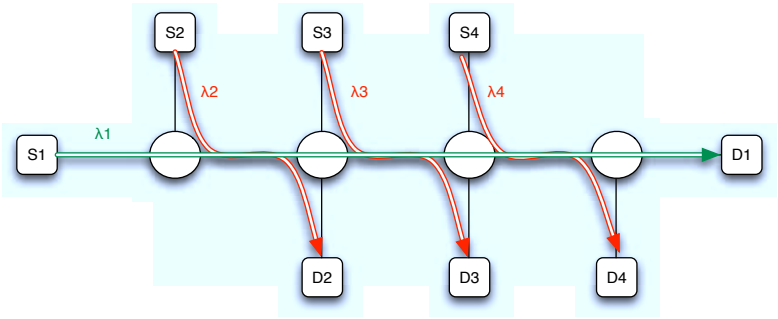


FIGURE 7.2: Exemple de topologie en parking où tous les liens ont la même capacité de $1Mb/s$. La somme des débits reçus vaut $2.2Mb/s$ si toute les sources émettent au même débit ($\lambda_{1,2,3,4} = 1Mb/s$). Cependant les débits obtenus par chacune des paires sont inégaux. Il peut être maximisé à $3Mb/s$ si $\lambda_1 = 0$ et $\lambda_{2,3,4} = 1Mb/s$. Étant donné que cela revient à couper la connexion de S_1, une autre alternative serait $\lambda_{1,2,3,4} = 0.5Mb/s$ qui allouerait équitablement la capacité au prix d'une perte d'efficacité de 1/3. Cet répartition équitable du trafic correspond au *max-min fair*.

Compromis entre efficacité et équité : Éviter la chute de la somme des débits utiles n'est pas le seul critère de qualité à retenir pour les utilisateurs. Considérons, par exemple, la topologie en « chemin de fer » ou en « parking » de la figure 7.2. Dans ce cas, maximiser le débit utile revient à rendre impossible la connexion entre S_1 et D_1. Il faut donc, en sus du débit utile total, considérer la répartition du débit entre les flots des utilisateurs. Nous nous focaliserons sur le cas de la « max-min fairness » (Cf. 7.2) qui permet la répartition équitable des débits au prix d'une perte d'efficacité globale [96]. Ce type d'allocation peut être obtenu de manière distribuée si un mécanisme de transmission basé sur une fenêtre d'émission (première version de TCP) est couplé à une gestion équitable (e.g. Fair-Queuing) des files d'attente sur les routeurs[3]. Si l'on relâche l'hypothèse du Fair-Queuing, un algorithme de gestion de la fenêtre d'émission de type AIMD (accroissement additionnel, reduction multiplicative) permet de s'approcher d'une répartition *max-min fair* avec des files d'attente de type Drop-Tail [97] .

Quelle qu'ait été l'importance de la topologie ou des retransmissions, le contrôle de congestion TCP Tahoe proposé par Van Jacobson a solutionné le *congestion collapse* assurant ainsi une meilleure robustesse d'Internet, sa pérennité et une relative équité entre les utilisateurs.

Critique du contrôle de congestion et alternatives : Néanmoins, ce dernier peut être critiqué sur deux points. D'une part, l'assignement de trafic obtenu est largement sous optimal (aussi bien en terme de débit utile qu'en terme d'équité [98]) et d'autre

3. L'utilisation de la fenêtre d'émission fixe et une politique de gestion de file d'attente type Fair-Queuing avait été initialement retenue dans des réseaux tels que le SNA d'IBM

part, cela suppose une volonté collaborative de chaque utilisateur [4] [99, 100]. C'est une des raisons pour laquelle Raghavan et Snoeren remettent en cause dans [90] la pertinence même de l'utilisation d'un contrôle de congestion. Notamment, ils proposent que chaque flot puisse émettre à une vitesse arbitraire ce qui, d'après ce que nous avons vu, devrait mener à un *congestion collapse* de part le problème des retransmissions et des paquets morts résultant de la topologie. Il proposent d'écarter ce problème via l'hypothèse de l'existence d'un mécanisme de codage qui, contrairement à TCP, permettrait que chaque paquet reçu à la destination lui soit utile (*i.e.* utilisation d'un code optimal).

Définition 7.1.2. *Code/mécanisme de transmission **optimal** : Nous parlerons de code optimal ou de mécanisme de transmission optimal lorsque chaque paquet reçu est utile au destinataire et ce, quel que soit le débit d'émission de la source [90].*

Définition 7.1.3. *Paquet inutile : Un paquet est considéré comme inutile s'il n'augmente pas la quantité d'information dont dispose le récepteur, autrement dit, si les informations qu'il contient ont déjà été préalablement reçues.*

L'allocation des ressources (*proportional fairness*, *max-min fairness* ou autre) n'est par contre pas étudiée et l'impact de la topologie (via les paquets « morts ») sur l'efficacité du réseau a été négligé. Cependant, les auteurs de [89] considèrent que le *congestion collapse* n'est pas étudié sur des topologies réalistes et ils montrent sur différents types de topologies que la perte d'efficacité serait modérée ($\approx 20\%$) si les files d'attente des routeurs sont de type Drop-Tail [5] et quasi nulle si elles sont de type Fair-Drop [6].

Définition 7.1.4. *Fair-Drop - files d'attente à pertes équitable : Une file d'attente de type Fair-Drop est une file d'attente qui, lorsqu'elle est pleine, jette les paquets de chaque flot avec une probabilité proportionnelle à leur débit d'arrivée [101]. La répartition du trafic est de type max-min-fair à la sortie d'un file d'attente Fair-Drop.*

L'utilisation de files d'attente actives (AQM) est considérée comme incontournable dans les réseaux anarchiques. En effet, pour ne pas se retrouver dans les scénarios de congestion collapse décrits aux figures 7.1 et 7.2, il faut d'une part prévenir l'apparition excessive de paquets morts. D'autre part il faut assurer une certaine équité entre les utilisateurs de manière à ce que le débit du lien d'accès au goulot d'étranglement ne soit pas le seul paramètre influant sur le débit obtenu en sortie du routeur. Le Fair Queuing et

4. En effet, un utilisateur malveillant ou mal configuré peut occuper toute la bande passante et rendre ainsi le réseau inutilisable pour les utilisateurs *congestion controlled*. Cependant, du point de vue de l'internet, les opérateurs effectuent un contrôle d'accès. Un utilisateur n'a donc un impact que sur son réseau local.

5. Avec les files d'attente de type Drop-Tail, lorsqu'un paquet arrive et que la file est pleine, il est tout simplement jeté.

6. Les files d'attentes à pertes équitables (*Fair-Drop* permettent d'assurer une bande passante égale aux flots à la sortie de la file d'attente)

ses variantes permettent d'atteindre ce but. Cependant leur complexité les rend inutilisable en pratique et, dans le contexte des réseaux anarchiques, les mécanismes à perte équitable leur sont préférés puisque bien moins consommateurs de ressources.

Les travaux de [89] et [90] posent l'hypothèse de l'existence d'un mécanisme de codage optimal ou du moins suffisamment efficace pour concurrencer l'efficacité des réseaux actuels mais ne fournissent pas de tels mécanismes. L'autre point pourtant crucial pour l'adoption d'un tel réseau est le fait qu'ils négligent totalement l'impact sur les applications temps réel et leur prise en compte. Ce sont là deux points majeurs qui sont adressés dans ce chapitre.

7.1.2 Objectifs de l'étude et challenges liés aux réseaux anarchiques

Comme l'ensemble des travaux sur les réseaux anarchiques, nous considérons que le réseau participe activement à la répartition du trafic avec des files d'attente à répartition équitable des pertes (Fair-Drop). Sous cette hypothèse nous adressons les points suivants.

7.1.2.1 Efficacité du débit reçu

Dans ces réseaux, le débit d'émission n'est borné que par la vitesse de l'interface tandis que le débit de réception est limité par la capacité disponible sur le chemin utilisé par le flot. Si il est aisé de générer des paquets à la vitesse de l'interface via la répétition ou divers mécanismes de codage, la difficulté réside dans le fait que la réception de chacun de ces paquets doit apporter de l'information supplémentaire par rapport aux précédents.

Tetrys permet une telle d'assurer une telle efficacité dès lors que la vitesse d'ajout des paquets source dans la fenêtre d'encodage correspond exactement à la bande passante disponible du récepteur. Étant donné qu'il n'est pas possible pour la source de prévoir la vitesse de réception du destinataire durant le prochain RTT, nous proposons un mécanisme d'adaptation des paramètres de codage qui réduit considérablement la probabilité qu'un paquet reçu soit inutile et permet une efficacité proche de celle de TCP.

7.1.2.2 Réduire le coût d'émission

Sur certaines topologies, le problème des paquets morts peut survenir malgré les files d'attente FairDrop. L'importance de ce problème augmente avec le débit d'émission des flots générateurs de paquets morts. Dans les réseaux anarchiques, les utilisateurs ne coopèrent pas entre eux et cherchent à réduire le coût associé au transfert (le temps

nécessaire au transfert) via l'augmentation de leur débit. Or, si un utilisateur sait qu'en émettant au delà d'un certain débit limite D il ne pourra plus améliorer son débit de réception, ce dernier n'a aucun intérêt à dépasser D (les paquets en excès seront morts).

Le coût est donc à la fois fonction du temps de transfert (débit auquel les paquets sont reçus) et du coût relatif à l'émission d'un paquet (débit d'émission).

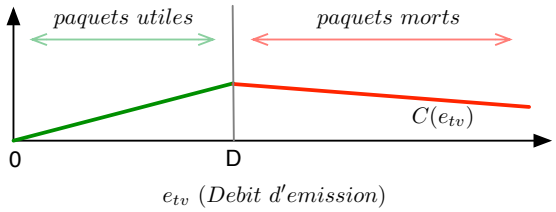


FIGURE 7.3: Efficacité (en termes de coût) du transfert en fonction du débit d'émission e_{tv}.

Un utilisateur égoïste cherchera donc à maximiser une fonction d'utilité du type

$$C(e_{tv}) = C_R \cdot r_{tv} - C_E \cdot e_{tv}$$

avec e_{tv} (resp. r_{tv}) le débit d'émission (resp. réception), C_R l'utilité associée au débit de réception et C_E le coût associé au débit émis. Comme l'illustre la figure 7.3, $C(e_{tv})$ atteint son maximum pour $e_{tv} = D$. Lorsque les files d'attente sont de type FairDrop, le débit limite D correspond au débit de la *max-min-fairness* pour le flot.

Ainsi, sur la base de cette hypothèse de minimisation de coût, nous proposons « Tetrys Min » un mécanisme dont l'utilisation conjointe avec Fair-Drop, permet de minimiser le nombre de paquets morts et de s'approcher du *max-min-fairness*. La conception d'un tel mécanisme pour d'autres types de files d'attente (Drop-Tail ou RED) dépasse le cadre de cette étude.

7.1.2.3 Supporter les applications temps réel

Le caractère opportuniste de TCP engendre bien souvent des pertes et une augmentation du délai de traversée des files d'attente [102] sur les flux UDP (potentiellement temps réel) avec lesquels ils partagent la bande passante. Bien que la cohabitation avec l'algorithme AIMD de TCP soit pénalisante, les applications temps réel de l'Internet basées sur UDP rencontrent des taux de pertes assez faibles et de faibles variations de délai ce qui constitue des conditions relativement favorables.

Dans les réseaux anarchiques [7], le taux de perte et leur répartition est par contre plus délicat à gérer et la question du support des applications temps réel dans ce contexte a été jusqu'à maintenant explicitement négligé. Émettre des paquets de redondance Tetrys à la vitesse de l'interface est certes optimal en termes de délai de reconstruction des pertes mais peut mener à une sous utilisation du réseau et à un coût d'émission inutilement élevé. Étant donné que le mécanisme de configuration développé section 3.4 supporte également des taux de pertes conséquents et des distributions variées, nous l'utiliserons pour le support des applications temps réel dans les réseaux anarchiques tout en minimisant le débit d'émission.

7.1.3 Comparaison aux autres travaux du domaine

Dans [90], Raghavan et Snoeren proposent *Achoo*, un mécanisme de transmission adapté aux réseaux anarchiques. La couche transport découpe les données émises par l'application en blocs appelés caravanes. Chaque caravane est émise au débit maximum possible. Ce débit maximum possible correspond à I_c/n avec I_c la capacité de l'interface et n le nombre de flots se partageant l'interface. L'émission de chaque caravane permet de sonder la bonne utilisation du débit de chacun des flots se partageant l'interface. Ainsi, si le débit utile d'un flot est inférieur à I_c/n (*i.e.* inférieur au débit équitable entre les différents flots se partageant l'interface), il tentera de réduire son débit d'émission sur la caravane suivante afin de favoriser les autres flots de l'interface qui peuvent éventuellement avoir une meilleur efficacité (et donc augmenter l'efficacité totale). Si cette réduction de débit à l'émission est suivie d'une réduction de débit à la réception [8], le précèdent débit est conservé. L'augmentation du débit suit le même principe et s'effectue quand le débit d'émission est le même que le débit de réception.

L'envoi des informations par caravanes implique qu'à la fin de l'émission de chacune d'entre-elles, des paquets relatifs à cette caravane sont envoyés durant un RTT alors qu'ils sont déjà intégralement reçus. Comme le montre la figure 7.4, cela peut réduire considérablement l'efficacité du mécanisme de transmission si le RTT est significatif. De plus, les auteurs ne considèrent pas non plus le cas de topologies défavorables qui peuvent générer des paquets morts. Cependant, cette proposition est adéquate pour gérer l'accès des différents flots à l'interface. De manière complémentaire, notre étude se focalisera sur la gestion d'un unique flot afin de réduire la proportion de paquets morts et d'améliorer l'équité à l'échelle du réseau plutôt que pour une interface donnée.

7. Bien entendu il n'y a des pertes que si il y a de la congestion. Dans les réseaux ante-contrôle de congestion, les flots sources ne dépassaient pas la capacité du réseau et il n'y avait donc que rarement des pertes.

8. Le récepteur émet périodiquement des acquittements qui indique notamment le débit utile

Les auteurs de [103] ont proposé l'utilisation des codes Tornado comme mécanisme de transmission pour la livraison de flux dans l'Internet. Plus récemment, le développement des LT-codes a motivé les auteurs de [91] qui proposent de les utiliser. Leur évaluation est basée sur la théorie des jeux avec laquelle ils comparent l'efficacité de TCP et des codes fontaines. La topologie utilisée dans leur modèle n'a qu'un seul goulot d'étranglement et il n'y a donc, par définition, aucune chance d'avoir des paquets morts. Dans [89], Bonald et Al. ont montré que sur la plupart des topologies de base, la perte d'efficacité est tolérable et sans commune mesure avec les préjugés sur les causes possibles du *congestion collapse*. Cependant ils ne considèrent la traversée que d'une seule des topologies de base à la fois (triangle, chaine, arbre montant ou descendant). Le coût de l'anarchie qu'ils mesurent est donc à pondérer avec le nombre d'éléments de ce type qu'il y aurait à traverser sur des topologies plus réalistes.

Comparé à [91] et [89], nous évaluons de façon réaliste un réseau anarchique par l'implémentation complète de la pile protocolaire et par le choix des topologies utilisées (c.a.d. avec un fort potentiel de paquets morts). De plus et contrairement à [91], nous aborderons les aspects d'équité. Á l'inverse, les travaux des auteurs de [89] ayant déjà abordé les problèmes de stabilité du processus stochastique relatif au nombre de flots présents dans le réseau, nous nous focaliserons que sur les performances des flots longs.

7.2 Gestion du transfert de données

Les deux fonctions devant être assurées par la couche de transport anarchique sont d'une part de maximiser le débit utile et d'autre part de réduire le coût en émettant le moins de paquets possibles. Nous abordons ces deux fonctions de manière indépendante.

7.3 Maximisation du débit utile

On considère dans cette partie que $e_{tv} = e_{iv}$, c'est à dire que l'émetteur envoie toujours au débit maximum de son interface. Tant qu'il y a des données non acquittées, le débit non utilisé par l'envoi de données est donc comblé par des paquets de redondance Tetrys ($e_{rv} = ei_v - e_{dv}$).

Une façon d'envisager un code optimal est de considérer Tetrys avec un taux de redondance instantanément égal au taux de perte afin d'assurer que chacun des paquets parvenant au récepteur lui apporte de l'information (chaque paquet de redondance est utile).

e_{dv}	Vitesse d'émission des données par la source
e_{rv}	Vitesse d'émission des paquets codés par la source
e_{tv}	Vitesse d'émission par la source
r_{dv}	Vitesse de réception des paquets de données par le destinataire
r_{rv}	Vitesse de réception des paquets codés par le destinataire
r_{tv}	Vitesse de réception des paquets par le destinataire
$e_{iv} ou I_e$	Vitesse de l'interface de la source
PLR	Taux de perte ($PLR = r_{tv}/e_{tv}$).
R	Le taux de redondance sur le débit émis ($R = e_{rv}/e_{tv}$)
c_r	$\approx e_{rv}/r_{tv}$ sachant que r_{tv} n'est connu qu'avec un FTT de retard.
SEB	Mémoire de stockage des segments de données émis par l'application vers l'encodeur Tetrys
$\|SEB\|$	Nombre de paquets contenus dans le buffer SEB à un instant donné
$maxSEB$	La capacité de SEB

TABLE 7.1: Notation

On peut naturellement s'abstraire de cette contrainte en envoyant l'ensemble du fichier via un code sans rendement (LT ou Tornado). L'efficacité obtenue serait quasi-parfaite si le fichier à transmettre est suffisamment grand. L'inconvénient est qu'il est nécessaire que la couche transport puisse conserver et manipuler l'ensemble du fichier. De plus, ceci n'est pas généralisable (par exemple, cela ne peut s'appliquer aux flots de capture vidéo *live*) et cette hypothèse est en rupture avec l'architecture actuelle des systèmes d'exploitation où les données sont passées au fur et à mesure à l'interface `socket`. De la même manière que *Achoo* dans [90], nous suivons une approche compatible avec l'architecture actuelle. La couche transport reçoit les données au fil de leur génération ou de l'espace disponible dans la `socket`. Cette approche nous permettra de conserver le même mécanisme de transport et de codage pour les applications temps réel et seul ses paramètres diffèrent.

L'utilisation des codes bloc classiques est possible mais ne permet pas de s'approcher du mécanisme optimal. Dans le cas de *Achoo*, dès que k segments de données sont disponibles, des paquets codés sont générés à partir de ces derniers et envoyés. Leur transmission ne s'arrête que lorsque la reconstruction du bloc est accusée par le destinataire. Comme l'illustre la figure 7.4, il s'écoule donc un RTT entre l'instant où un bloc est décodé par le récepteur et l'instant où la source arrête d'émettre des paquets relatifs à ce dernier.

L'efficacité ainsi atteinte est de $\frac{k/D}{k/D+RTT}$. Ceci implique que sur les réseaux à fort produit bande passante délai, de grandes valeurs de k sont nécessaires ce qui peut ne pas être possible étant donné les contraintes d'espace mémoire précédemment soulevées. Si Tetrys accepte tous les paquets jusqu'à atteindre la taille maximale de son buffer d'encodage,

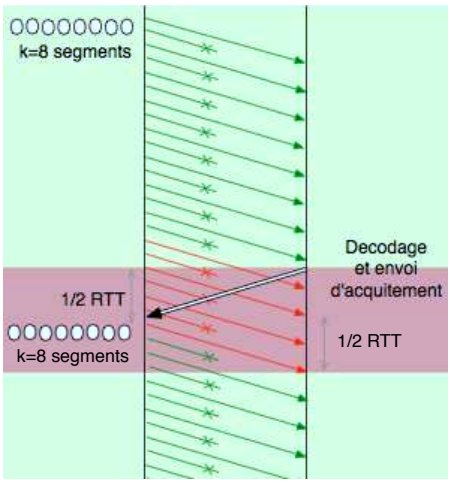


FIGURE 7.4: Illustration de la perte d'efficacité liée à l'utilisation des mécanismes tels que *Achoo*.

l'efficacité serait la même avec $\frac{SEB/D}{SEB/D+RTT}$ où SEB est la taille maximale du buffer d'encodage.

Pour minimiser la proportion de paquets inutiles envoyés après un décodage, il est nécessaire que la `socket` source accepte des paquets en permanence, et ce, à une vitesse aussi proche que possible de celle à laquelle ils sont reçus par le destinataire.

Considérons dans un premier temps que r_{tv} est constant. Sachant la valeur de r_{tv} retournée par le récepteur, le RTT et la fréquence des acquittements a_{freq}, il est possible d'obtenir $normal_{SEB}$ la taille de SEB (le buffer d'encodage) qui correspond au fonctionnement normal de Tetrys dans ces conditions.

$|SEB|$ est au moins égal au produit bande passante délai, c.a.d $RTT * r_{tv}$. Puisque les acquittements ne sont pas envoyés à chaque paquet reçu mais périodiquement, on a $(RTT + a_{freq}/2) * r_{tv}$. En considérant la probabilité $P(\overline{a})$ de perdre un acquittement on obtient :

$$normal_{SEB} = \alpha_{rec} \cdot r_{tv} \cdot (RTT + \frac{a_{freq} * (1.0 + \frac{1.0}{1.0-p(\overline{a})})}{2}) \tag{7.1}$$

avec $\alpha_{rec} > 1$ qui nous permet d'adapter l'estimation du temps de récurrence moyen en fonction de la complexité tolérée.

Lorsque que $|SEB| > normal_{SEB}$, l'événement de perte courant doit être corrigé avant que $|SEB|$ n'atteigne la taille maximale du buffer d'encodage (i.e. $(c_r \cdot T \cdot r_{tv} + |SEB|) \leq max_{SEB}$) et donc au moins $|SEB| - normal_{SEB}$ paquets de redondance doivent atteindre le récepteur (i.e. $(1 - c_r) \cdot r_{tv} \geq |SEB| - normal_{SEB}$) ce qui peut s'écrire par :

$$\frac{c_r}{1 - c_r} = \frac{max_{SEB} - |SEB|}{|SEB| - normal_{SEB}} \tag{7.2}$$

À la réception de chaque acquittement, c_r est ajusté avec la valeur de r_{tv}.

En pratique, l'apparition de nouveaux flots dans le réseau fait varier r_{tv} au cours du temps. Ainsi, lorsque Tetrys est en fonctionnement normal, $c_r = 1$. Cependant, e_{dv} courant peut être inférieur à la valeur instantanée de r_{tv} qui ne peut être connue à l'avance et donc ne pas utiliser cette bande passante pour des paquets utiles. Afin de palier à cette potentielle perte d'efficacité il faut donc prendre $c_r > 1$ à la suite de chaque décodage ce qui aura pour effet de rentrer à nouveau dans une phase de perte à corriger et donc de rendre l'ensemble des paquets reçus utiles.

Une autre source possible d'inefficacité est, comme dans le cas de *Achoo*, la durée pendant laquelle Tetrys génère des paquets avec $e_{dv} < r_{tv}$ alors qu'il n'y a aucune perte à corriger. Cette durée peut être sensiblement diminuée si dans les acquittements le récepteur indique le nombre m de paquets manquants pour décoder. Suivant m, l'émetteur sait si entre l'instant d'émission de l'acquittement et l'instant où le récepteur recevra de nouveaux paquets émis avec un taux de codage différent (RTT), le récepteur aura reçu plus de paquets de redondance qu'il ne lui manquait de paquets. Si $(e_{tv} * c_r * RTT) > m$, l'émetteur considère que la perte est corrigée et fixe c_r à sa valeur initiale.

Nous noterons que contrairement au reste du mécanisme proposé, cet algorithme n'est pas tolérant aux pertes d'acquittements et ne convient pas au cas où il y aurait plusieurs récepteurs pour le flot.

7.3.1 Minimisation du coût d'émission

La partie précédente visait à augmenter la probabilité qu'un paquet reçu soit utile et donc à réduire la partie du coût relative à la durée du transfert. Cependant, émettre à la vitesse de l'interface alors que le débit utile n'est qu'une fraction de ce dernier peut être coûteux en temps de calcul des paquets de redondance ainsi qu'en énergie utilisée pour la transmission, notamment sur des liens sans fil. Dans cette partie, le but est de réduire le débit d'émission sans influer sur le débit utile. Comme dans [89] et[90], nous faisons, pour les raisons énoncées dans l'introduction, l'hypothèse de routeurs avec files d'attente de type Fair-Drop.

Étant donné l'état courant du réseau, si la source émet au débit de l'interface ($e_{tv} = e_{iv}$), alors le débit de réception (r_{tv}) est maximal. Si le flot émet à ce même débit maximal de réception ($e_{tv} = r_{tv}$), il ne devrait pas subir de perte et le débit d'émission sera ainsi minimal sans avoir d'impact sur le débit utile.

Les conditions du réseau en termes de charge et de capacité de lien font varier r_{tv} au cours du temps. Ainsi plutôt que descendre le débit d'émission aux environs de r_{tv} (qui peut

induire une chute du débit utile), il est préférable d'utiliser un débit minimum $\alpha_{sonde} \times r_{tv}$ en deçà duquel il ne faut pas descendre. Ce débit plancher permet de l'occuper la bande passante disponible dès la réception d'un accusé notifiant l'augmentation de r_{tv}.

Le mécanisme que nous avons déployé est détaillé en figure 7.5 et fonctionne comme décrit ci-dessous. Au début de la connexion, on fixe e_{tv} à e_{iv}. Ensuite, lorsque le récepteur a reçu suffisamment de paquets pour avoir une estimation correcte [9] de r_{tv}, il réduit son débit d'émission par un facteur α_d. Si l'émetteur reçoit un accusé lui indiquant que la réduction du débit d'émission entraine une réduction du débit à la réception avant que $\beta_d \cdot RTT$ secondes ne se soient écoulées, il reprend le débit précédent et tente d'augmenter à nouveau le débit. A l'inverse, après $\beta_d \cdot RTT$ secondes, si le prochain acquittement n'indique pas de chute de r_{tv}, le débit est de nouveau réduit par un facteur α_d. Selon la vitesse d'adaptation que l'on souhaite obtenir, α_d peut être augmenté à chaque itération. Le débit d'émission e_{tv} est ainsi réduit tout les $\beta_d * RTT$ secondes jusqu'à ce qu'une diminution de r_{tv} soit observée ou jusqu'à ce que e_{cv} atteigne une valeur seuil de $\alpha_{sonde} * r_{tv}$. L'augmentation du débit suit le même principe excepté qu'elle s'arrête lorsque $e_{tv} = e_{iv}$.

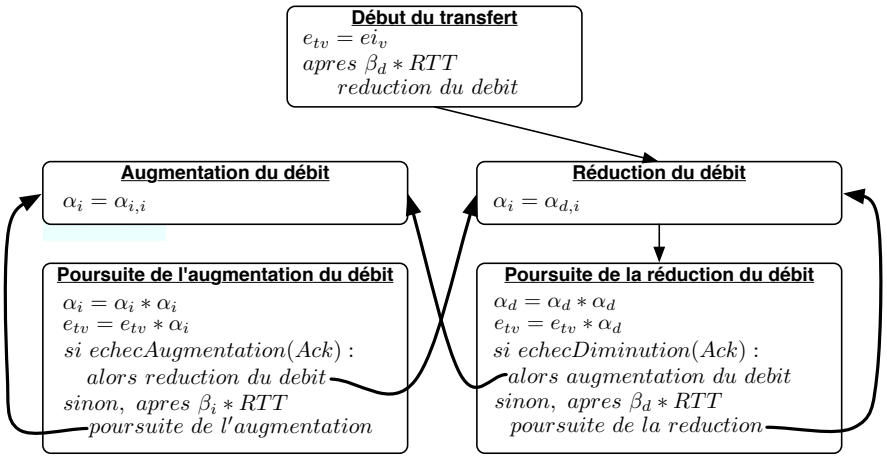


FIGURE 7.5: *Tetrys min* augmente le débit d'émission tant qu'il permet d'augmenter le débit de réception. À l'inverse, le débit d'émission est réduit tant que le débit de réception ne diminue pas.

Nous noterons qu'avec des files d'attente de type Fair-Drop, la réduction du débit atteindra la valeur seuil de $\alpha_{sonde} * r_{tv}$ avant qu'une chute du débit de r_{tv} soit observée. L'impact qu'a ce mécanisme de réduction de coût d'émission sur le débit utile est négligeable. Dans l'hypothèse où α_{sonde} ne permettrait pas d'absorber le taux de perte résiduel inhérent à la file d'attente, le compromis entre perte de débit utile et diminution du débit de transmission peut être configuré via les tests *echecAugmentation*(Ack) et

9. $\beta_d * RTT$ avec $\beta \in [1, 2]$ en fonction des paramètres de la moyenne glissante utilisés pour calculer r_{tv}.

$echecDiminution(Ack)$. En l'occurrence, si $\frac{r_{tv}(courant)-r_{tv}(precedent)}{e_{tv}(courant)-e_{tv}(precedent)} < \alpha_{eff}$, le gain sur r_{tv} est trop faible par rapport au débit requis à l'émission et donc l'augmentation du débit n'est pas justifiée. De la même façon, si $\frac{r_{tv}(courant)-r_{tv}(precedent)}{e_{tv}(courant)-e_{tv}(precedent)} > \alpha_{eff}$ lors d'une réduction du débit, la perte sur r_{tv} est trop grande par rapport à la réduction du coût associée à e_{tv}.

7.4 Gestion des applications à contrainte de temps

Le support des applications n'est pas garanti lorsque le taux de redondance nécessaire pour palier au taux de perte ne peut pas être reçu par le destinataire. A l'inverse, il est clair que d'après les propriétés de Tetrys présentées au Chapitre 3, émettre au débit de l'interface lorsque $e_{dv} << r_{tv}$ est le meilleur moyen de satisfaire les contraintes de délai de l'application.

Il se peut par contre que le coût de transmission soit inutilement élevé de part le fait que les contraintes de l'application peuvent être satisfaites avec un taux de redondance moins important. Le taux de redondance à appliquer peut être déterminé via la méthode présentée en section 3.4. Dans le cas de Fair-Drop, le taux de redondance à considérer sera celui du taux de perte "ambiant" qui correspond à la probabilité qu'un paquet trouve une file d'attente pleine malgré les pertes accusées par les autres flots [10]. En sus de ce taux de perte "ambiant" nous devons considérer l'impact que peut avoir l'arrivée d'un nouveau flot (temps réel ou FTP) sur l'un des routeurs que traverse le flot.

L'utilisation de Tetrys comme mécanisme de transmission pour les applications temps réel dans les réseaux anarchiques ne nécessite à priori pas d'autre modification que le provisionnement de la redondance en vue de l'apparition d'un nouveau flot (*i.e.* en vue de prévenir l'instabilité dans du réseau). Nous comparerons dans la section 7.5.6 les performances obtenues par TCP/UDP et Tetrys en termes d'efficacité d'utilisation du réseau et de délai lorsque les applications temps réel sont combinées à des applications de transferts de données et ce sur une topologie semblable à celle que pourrait déployer un FAI.

7.5 Évaluation

Dans cette section, nous comparons les performances des réseaux anarchiques aux réseaux conventionnels régis par le contrôle de congestion. Les topologies et les paramètres tels

10. Il est donc possible que des arrivées groupées remplissent la file d'attente. De plus, le taux de perte appliqué aux flots est calculé sur les bases d'un débit moyenné sur une certaine durée. Une modification rapide de ces derniers peut donc également remplir la file d'attente au détriment des autre flots.

que le délai, la capacité des files d'attentes et leur types sont variés de manière à adresser les principaux cas de figures et appréhender les avantages et inconvénients de ces deux solutions.

Dans notre évaluation, la notion d'efficacité est définie comme suit :

Définition 7.5.1. *Efficacité : Nous définissons l'efficacité de Tetrys ou de TCP comme le rapport entre le débit utile aux applications et la capacité du réseau, c'est à dire le débit total lorsque ce dernier est maximisé.*

Notons tout de même que la *max-min-fairness* peut être très éloignée de la capacité du réseau et que dans certains cas, il n'est pas souhaitable que l'efficacité soit proche de 1. De même, étant donné le rapport entre la charge utile et la taille des paquets (1000/1040), l'efficacité ne peut dépasser les 0.96.

Ensuite, nous évaluerons dans les sections 7.5.3 et 7.5.4 les notions d'efficacité et d'équité (via l'index de Jain [104]) sur des topologies de test de référence telles le papillon et le parking.

Définition 7.5.2. *Equité : Nous utiliserons l'index de Jain [104] pour mesurer l'équité de la répartition de la bande passante entre chacun des n flots qui partagent un goulot d'étranglement. Cet index se définit comme suit :*

$$f(f_0, f_1, ..., f_n) = \frac{(\sum f_i)^2}{n \times (\sum f_i^2)} \tag{7.3}$$

Dans certains cas, l'index de Jain ne permet pas d'identifier clairement que des flots sont lésés et nous présenterons également le débit obtenu par le ou les flots lésés.

En sus de ces métriques, nous évaluerons à la section 7.5.5 les besoins en termes de files d'attente Fair-Drop sur une topologie semblable à celle d'un FAI d'échelle nationale.

Finalement, section 7.5.6, nous évaluerons sur cette même topologie la possibilité de supporter des applications de voix sur IP.

7.5.1 Paramètres par défaut utilisés dans les simulations

Toute nos simulations ont été effectuées sous ns-2. Lorsque TCP est utilisé, SACK est activé par défaut et la taille maximum de la fenêtre TCP est de 512Ko afin que l'émetteur ne soit jamais limité par cette dernière. Il n'existe pas d'implémentation de Fair-Drop disponible sous ns-2 ; nous avons donc développé une version de Fair-Drop relativement simple qui tend à se comporter comme Drop-Tail pour des files d'attente de faible

capacité. Les performances sont donc sous-estimées pour des petites capacités de file d'attente (c-à-d < 10) et le taux de perte "ambiant" est assez significatif pour ces valeurs. Lorsque RED est utilisé c'est avec la configuration par défaut pour TCP et avec `q_weight_`= 0.5, `thresh_`= 5 et `maxthresh_=qlim_` pour Tetrys. Cette différence de traitement s'explique par le fait que RED est conçu pour réguler des flots qui implémentent un contrôle de congestion ce qui n'est pas du tout le cas pour les flots Tetrys qui remplissent constamment la file d'attente et obtiennent des performances similaires à celles de Drop-Tail. La configuration appliquée à RED permet de s'affranchir des effets de synchronisation que l'on peut avoir avec Drop-Tail, réduisant ainsi la probabilité qu'un flot trouve systématiquement la file d'attente pleine. Afin de réduire encore d'avantage la probabilité que des pertes soient dues à la synchronisation de flots Tetrys, l'émission de chaque paquet par la source est soumis à un retard choisi de façon aléatoire. Pour finir, la taille par défaut des files d'attente est de 20 paquets pour Tetrys et correspond au produit bande passante × délai pour TCP.

7.5.2 Efficacité sur un lien

Mécanisme \ Délai		5ms	100ms	300ms
Tetrys		95%	95%	93%
Achoo	$k = 500pkt$	92%	84%	73%
	$k = 2000pkt$	95%	93%	89%
TCP et files d'attente 10 paquets	$cwnd = 64$	96%	62%	34%
	$cwnd = 512$	96%	80%	57%
TCP et files d'attente de BDP paquets	$cwnd = 64$	96%	95%	41%
	$cwnd = 512$	96%	94%	92%

TABLE 7.2: Efficacité d'un seul flux Tetrys, TCP ou Achoo sur un lien.

7.5.2.1 Objectif

Dans un premier temps nous évaluons TCP, Tetrys et Achoo avec un seul flot sur un seul lien. Nous étudions l'efficacité de ces différents mécanismes dans des conditions idéales ainsi que l'impact de paramètres tels que la taille des files d'attente ou le délai.

7.5.2.2 Paramètres

Nous avons répété des transferts FTP de 100 secondes pour différents délais (5ms, 100ms et 300ms) et des files d'attente de taille 10 et BDP paquets. La bande passante du lien

est de $2Mb/s$, la charge utile des paquets est de 1000 octets (efficacité maximum de 96%).

7.5.2.3 Analyse

On peut voir sur le tableau 7.2 que pour un délai de $5ms$ tous les mécanismes sont très proches du maximum à l'exception de *Achoo* avec une taille de bloc de 500 paquets. L'augmentation du RTT accroit la période durant laquelle Achoo envoie des informations relatives à un bloc déjà décodé. Ainsi, on constate pour un délai de $100ms$ et $300ms$ que les performances d'Achoo chutent sensiblement.

Les performances de Tetrys chutent légèrement pour un délai de $300ms$. En effet, avec un tel délai, les mauvaises estimations du nombre de paquets de redondance émis par rapport au nombre de paquets manquants rallongent le temps de décodage. Ceci diminue donc le taux de codage et augmente le nombre de paquets inutiles envoyés lors du prochain décodage. Dans ce contexte, une adaptation du taux de codage par une fonction concave aurait été plus appropriée.

Pour ce qui est de TCP avec des tailles de files d'attente conformes aux recommandations (i.e. BDP du plus long flux [102]), si l'on considère que la taille de la fenêtre d'émission est au maximum de $64Ko$, les performances sont bornées par la taille de cette même fenêtre et chutent lorsque le délai augmente. En pratique la taille de la fenêtre n'est pas bornée à $64Ko$ [11] et les performances de TCP restent stables si l'on considère une fenêtre de $512Ko$. La diminution de l'efficacité observée pour un délai de $300ms$ est due au rapport entre le temps requis pour sortir du slow-start et la durée du transfert volontairement bornée à $100s$ pour illustrer ce fait. On constate cependant que TCP est sensible à la taille des files d'attente alors que pour Tetrys, cela n'a pour seul effet que d'augmenter le délai de bout en bout.

Mécanisme \ Délai		5ms	100ms	300ms
Tetrys		0.977 / 0.976	0.990 / 0.989	0.975 / 0.973
Achoo	$k = 500pkt$	0.951 / 0.950	0.874 / 0.872	0.760 / 0.758
	$k = 2000pkt$	0.986 / 0.984	0.967 / 0.966	0.924 / 0.923
TCP		1.0 / 0.961		

TABLE 7.3: Ratio entre le *goodput* et le *throughput*. La valeur affichée après la barre oblique correspond également au même ratio sauf que *throughput* inclu les acquittements

11. Dans les systèmes d'exploitation actuels l'option *windowscale* [105] est activée par défaut et permet à la fenêtre d'émission de remplir des produits bande passante délai de l'ordre du Giga-octet.

Le tableau 7.3 présente le ratio entre le entre le *goodput* et le *throughput* ainsi que le ratio *goodput* et le *throughput* incluant les acquittements. Le mécanisme de régulation de débit par la vitesse de réception des acquittements permet à TCP de s'ajuster efficacement à la bande passante disponible au prix de fréquents acquittements sur la voie retour. Le ratio qui est de 1 pour la voie aller tombe à 0.96 si l'on inclut la voie retour. Si on considère des paquets plutôt que des octets (métrique d'intérêt pour les routeurs), le ratio de paquets utiles chute à 0.5 alors que pour Tetrys ou Achoo, le trafic généré sur la voie retour est négligeable face au trafic de données.

7.5.2.4 Conclusion

Comme on pouvait s'y attendre, de grandes valeurs du RTT réduisent l'efficacité de *Achoo*. Ainsi de grande tailles de buffers sont nécessaires pour qu'il puisse concurrencer TCP ou Tetrys. Les performances de TCP sont très proches du meilleur des cas possible mais elle se dégradent lorsque le RTT augmente ou que la taille des files d'attente diminue. Tetrys est également très proche du cas optimal, tandis qu'à l'inverse de TCP, il est quasiment insensible aux variations du RTT ainsi qu'aux tailles de files d'attente. La fréquence des acquittements requise par TCP implique une charge sur la voie retour qui peut être importante si l'on considère un cout de traitement par paquet plutôt qu'en volume de trafic. L'utilisation de Tetrys comme mécanisme de transmission anarchique constitue une alternative efficace à TCP au prix cependant du coût associé au codage.

7.5.3 Topologie en papillon

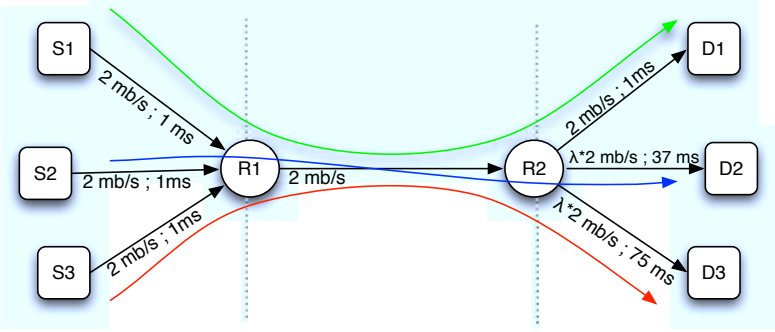


FIGURE 7.6: Topologie en papillon inspirée du guide d'évaluation de TCP [106].

7.5.3.1 Objectif

Sur cette topologie nous souhaitons étudier l'impact du partage d'un lien sur l'efficacité totale. En sus de l'efficacité globale nous évaluons le coût à l'émission ainsi que l'impact du mécanisme de minimisation du coût sur l'efficacité globale. Enfin, nous nous intéressons également à l'impact des différences de délai sur l'iniquité entre les différents flots.

7.5.3.2 Paramètres

Nous utilisons cette fois ci la topologie décrite dans la figure 7.6 qui correspond à un sous ensemble des recommandations faites par [106] pour l'évaluation de TCP. Cette topologie dite en "papillon" permet d'étudier l'interaction de plusieurs flots TCP ayant des délais différents.

Par souci de simplicité d'analyse des résultats, nous ne considérons que trois flots (f_0, f_1, f_2) qui se partagent le lien $R_1 - R_2$ à $2Mb/s$ dont nous faisons varier le délai de $5ms$, $100ms$ et $300ms$. Après $R_1 - R_2$ les flots empruntent chacun un lien différent, ayant respectivement pour délai $1ms$, $37ms$ et 75ms (C.f. [106]) avec une même bande passante de $2Mb/s$. Les conditions rencontrées par les flot avant le goulot d'étranglement sont les mêmes avec $2Mb/s$ de bande passante et $1ms$ de délai.

		Efficacité		Equité	
		Tetrys	TCP	Tetrys	TCP
Drop-Tail	5ms	93.2%	95.7%	0.99	0.73
Drop-Tail	100ms	94.0%	94.8%	0.99	0.59
Drop-Tail	300ms	94.6%	92.7%	0.99	0.97
RED	5ms	93.4%	93.9%	0.99	0.67
RED	100ms	94.0%	83.7%	0.99	0.94
RED	300ms	95.1%	64.5%	0.99	0.96
Fair-Drop	5ms	92.7%	95.9%	0.99	0.86
Fair-Drop	100ms	93.8%	94.1%	0.99	0.98
Fair-Drop	300ms	94.9%	92.9%	0.99	0.97

TABLE 7.4: Efficacité et équité de trois flots Tetrys ou TCP sur la topologie en papillon avec $\lambda = 1.0$.

7.5.3.3 Analyse

Le tableau 7.4 contient les valeurs de l'index de Jain et l'efficacité pour des files d'attente de type Drop-Tail, RED et Fair-Drop avec différentes valeurs du délai pour le goulot étranglement.

On remarque que pour des files d'attente de type Drop-Tail et Fair-Drop, l'efficacité de Tetrys et TCP est stable même si comme précédemment, on note que l'efficacité de TCP diminue avec le délai. De manière plus surprenante, on observe l'inverse pour Tetrys. En effet, par la concurrence entre les trois flots pour accéder à la file d'attente, le débit obtenu par chacun des flots est d'autant plus variable que on l'observe sur une échelle de temps courte [12]. L'estimation du nombre de paquets de redondance reçus depuis le dernier RTT est donc biaisé pour des faibles valeurs du RTT ce qui réduit l'efficacité.

La file d'attente RED ne change ni le débit ni l'équité obtenus par Tetrys. Par contre, la configuration par défaut de RED n'est pas favorable aux flots à fort produit bande passante délai et les performances de TCP sont ainsi dégradées.

Equité Comme dans les travaux de Lopez et Al. [91], nous constatons que l'équité est quasi-parfaite avec Tetrys ce qui, étant donné la topologie, est trivial.

Le débit obtenu par un flot Tetrys est insensible au délai de bout en bout tandis que pour TCP le flot de plus faible RTT obtient le plus grand débit [13]. On peut également remarquer que l'iniquité de TCP est sensiblement réduite via l'utilisation de Fair-Drop.

Minimisation du coût Dans les simulations précédentes, le mécanisme de minimisation du coût lié au débit d'émission n'est pas activé. Le tableau 7.5 compare le rapport entre le nombre de paquets émis et le nombre de paquets utiles reçus selon que le mécanisme de minimisation du coût lié au débit d'émission est activé (*Tetrys min*) ou non (*Tetrys max*) [14].

Nous pouvons observer que le mécanisme de minimisation du coût ne détériore pas l'efficacité de Tetrys et a même tendance à légèrement l'améliorer. L'explication est la même que pour l'augmentation de l'efficacité de *Tetrys max* que l'on observe lorsque le délai augmente. En effet, le débit d'émission de chaque flot étant proche du débit équitable, son débit instantané est moins variable. L'estimation du nombre de paquets émis depuis le dernier acquittement est donc plus précis ce qui réduit le nombre de paquets inutiles.

Conformément aux motivations de ce mécanisme, on constate que le nombre de paquets émis utiles est bien plus élevé ($60 \sim 70\%$) que lorsque Tetrys émet au débit de l'interface ($\sim 19\%$). On constate cependant que l'efficacité en termes de paquets émis décroît

12. Pour rappel, si Tetrys émet des paquets avec une période T, un paquet qui doit être émis au temps t sera retardé d'un délai choisi aléatoirement entre 0 et T.

13. Si l'on fait abstraction des possibles effets de synchronisation, c.f. sec.5.2 de [107].

14. Le tableau 7.5 contient également les valeurs de l'efficacité qui pour *Tetrys max* sont les même que dans le tableau précédent.

		Efficacité		paquets utiles / paquets émis	
		Tetrys max	*Tetrys min*	*Tetrys max*	*Tetrys min*
Drop-Tail	5ms	93.2%	94.7%	19.3%	72.4%
Drop-Tail	100ms	94.0%	95.2%	19.5%	69.4%
Drop-Tail	300ms	94.6%	95.2%	19.6%	62.1%
RED	5ms	93.4%	94.7%	19.3%	72.1%
RED	100ms	94.0%	95.0%	19.5%	70.0%
RED	300ms	95.1%	92.5%	19.7%	62.3%
Fair-Drop	5ms	92.7%	93.9%	19.2%	71.8%
Fair-Drop	100ms	93.8%	94.3%	19.4%	67.2%
Fair-Drop	300ms	94.9%	95.2%	19.6%	58.6%

TABLE 7.5: Évaluation du gain apporté par l'algorithme de minimisation du débit (*Tetrys min*).

lorsque le RTT augmente. Cela s'explique par le fait que $\beta_i < \beta_d$ et que e_{tv} augmente plus vite (tous les $\beta_i * RTT$) qu'il ne diminue ($\beta_d * RTT$).

Si l'efficacité n'est pas affectée et que le coût de l'émission est largement diminué, ce n'est pas le cas de l'équité. En effet, la fréquence d'augmentation ou de diminution du débit étant fonction du RTT, un des inconvénients majeur de ce mécanisme est le fait que le flot au plus court RTT obtient une bande passante supérieure si le routeur n'est pas de type Fair-Drop. L'index de Jain chute alors aux environs de 0.5 pour RED et Drop-Tail alors qu'il se maintient à 0.999 pour Fair-Drop.

7.5.3.4 Topologie di-symétrique

Jusqu'à maintenant, nous n'avons pas abordé de cas où la topologie peut générer des paquets « morts » [107]. En changeant la bande passante des liens (R_1, D_1) et (R_2, D_2) à $400Kb/s$ (initialement à $2Mb/s$), la capacité maximale demeure à $2Mb/s$ mais n'est que de $1.46Mb/s$ si les flots émettent toujours au débit de l'interface ($2Mb/s$). Le tableau 7.6 montre l'efficacité obtenue par Tetrys et TCP.

On constate que lorsque Tetrys émet au débit de l'interface, la présence de paquets morts sur le lien (R_0, R_1) réduit sensiblement l'efficacité qui chute aux alentours de 69%. Le mécanisme de minimisation du coût réduit le nombre de paquets émis par s_2 et s_3 et augmente la proportion de paquets du flot 1 qui accède au goulot d'étranglement (i.e. la proportion de paquets mort sur (R_0, R_1) diminue). On constate que cet effet de bord du mécanisme permet d'augmenter sensiblement l'efficacité et même de concurrencer TCP dans certains cas.

		Efficacité		
		Tetrys max	*Tetrys min*	TCP
Drop-Tail	5ms	68.8%	89.7%	95.6%
Drop-Tail	100ms	69.0%	89.8%	93.7%
Drop-Tail	300ms	69.3%	95.7%	89.0%
RED	5ms	68.3%	90.7%	94.2%
RED	100ms	69.0%	89.8%	85.3%
RED	300ms	69.5%	89.2%	68.5%
Fair-Drop	5ms	68.3%	87.9%	95.8%
Fair-Drop	100ms	69.0%	86.4%	94.0%
Fair-Drop	300ms	69.5%	85.9%	88.9%

TABLE 7.6: Topologie en papillon asymétrique ($\lambda = 0.2$ c.f. fig 7.6).

Comme le suggère l'étude sur la topologie symétrique, Tetrys sur Drop Tail ou RED est sujet à l'iniquité de par les différences de RTT. Ainsi, si l'on active les mécanismes de réduction de coût (*c.a.d. Tetrys-min*) avec Drop-Tail et RED, f_1 (de plus faible délai) obtient $\sim 1.4Mb/s$ alors que f_2 et f_3 n'obtiennent que $0.2Mb/s$. Avec Fair-Drop, le partage de la bande passante correspond bien à l'allocation *max-min fair* attendue et f_2 et f_3 obtiennent $0.38Mb/s$ chacun. Avec TCP, le débit obtenu par f_2 et f_3 varie entre $0.25Mb/s$ et $0.38Mb/s$ en fonction des délais et de la file d'attente.

7.5.3.5 Conclusion

Lorsque le délai de bout en bout est faible, le partage de bande passante est plus efficace avec TCP qu'avec Tetrys tandis que la tendance s'inverse (*e.g.* 300ms) lorsqu'il augmente. *Tetrys min* permet d'augmenter significativement l'efficacité des paquets envoyés (70% contre 19%) et, par effet de bord, il permet d'augmenter l'efficacité totale. Dans le cas ou les topologies peuvent générer des paquets morts, *Tetrys min* semble même être indispensable. L'équité de *Tetrys min* est quasiment parfaite lorsqu'il est combiné aux files d'attentes Fair-Drop tandis que les différences de délai influent celle de TCP qui s'éloigne significativement de la *max-min-fairness*.

7.5.4 Topologie en parking

7.5.4.1 Objectif

La topologie en parking illustrée sur la figure 7.2 est connue pour être défavorable à TCP en terme d'équité (sa capacité à atteindre la *max-min fairness*) tandis que les files d'attente Fair-Drop permettent d'obtenir de facto une répartition équitable. L'équité de la topologie en papillon 7.6 est trivialement de 1.0 lorsque tous les flots émettent au

débit de l'interface [91]. Ici, le contexte est moins favorable puisque la bande passante obtenue par le premier flot descend à $0.14Mb$ avec un taux élevé de paquets morts. Nous étudions donc dans ce contexte l'efficacité du mécanisme de minimisation du débit couplé à Fair-Drop (ou non) ainsi que l'équité obtenue entre chaque flot (notamment f_1 qui traverse l'ensemble du réseau).

		Efficacité			Equité		
		Tetrys max	*Tetrys min*	TCP	*Tetrys max*	*Tetrys min*	TCP
Drop-Tail	5ms	74%	93.3%	95.2%	0.75/0.15Mb	0.75/0.01Mb	0.75/0.02Mb
Drop-Tail	100ms	–%	91.8%	83.3%	–	0.76/0.05Mb	0.81/0.20Mb
Drop-Tail	300ms	72.8%	92.4%	78.6%	0.77/0.14Mb	0.75/0.03Mb	0.82/0.23Mb
RED	5ms	74.3%	92.6%	79.3%	0.75/0.11Mb	0.76/0.04Mb	0.87/0.42Mb
RED	100ms	74.0%	91.2%	73.1%	0.76/0.17Mb	0.76/0.05Mb	0.84/0.26Mb
RED	300ms	81.3%	91.6%	60.2%	–	0.76/0.04Mb	0.90/0.40Mb
Fair-Drop	5ms	64.5%	64.2%	84.7%	0.98/0.81Mb	0.99/0.85Mb	0.84/0.32Mb
Fair-Drop	100ms	64.1%	64.9%	83.7%	0.99/0.83Mb	0.99/0.84Mb	0.80/0.19Mb
Fair-Drop	300ms	64.0%	64.7%	68.3%	0.99/0.84Mb	0.99/0.84Mb	0.91/0.47Mb

TABLE 7.7: Topologie en parking décrite sur la figure7.2. Une efficacité à 1 équivaut à supprimer le flot f_1 et l'efficacité de la max-min-fairness n'est que de 0.63 ($0.66 * 0.96$). L'index de Jain vaut 1.0 si tous les flots obtiennent $1Mb/s$ (conformément au max-min-fairness). Le délai correspond au délai rencontré par le flot f_1 qui est 3 fois plus grand que celui rencontré par les autres flots.

7.5.4.2 Analyse

Le tableau 7.7 montre l'efficacité, l'index de Jain ainsi que le débit obtenu par le flot f_1 (qui traverse l'ensemble du réseau). Notons que l'efficacité de 1.0 n'est atteinte que si le débit du flot f_1 est réduit à zéro, ce qui ne correspond pas à l'assignement de type *max-min fair* que nous cherchons à obtenir et dont l'éfficacité n'est que de 66%. Elle n'est donc donnée qu'à titre illustratif puisque qu'il n'est pas possible d'effectuer de comparaison entre les différents scénarios. Nous avons cependant observé que l'efficacité de chaque flot considéré séparément reste du même ordre que pour les topologies précédentes. La topologie en parking ne semble pas présenter de caractéristique particulière qui influencerait Tetrys.

Comme le suggèrent les propriétés de Fair-Drop, c'est avec cette dernière que Tetrys est le plus proche de l'assignement *max-min fair* avec un débit aux environs de $\sim 0.85Mb/s$ pour f_1 (au lieu de $0.96Mb/s$ pour le *max-min fair*) et un index de Jain de ~ 0.99 lorsque Tetrys émet au débit de l'interface. Comme prévu, le débit obtenu par f_1 avec RED et Drop-Tail est aux environs de $\sim 0.14Mb/s$ ce qui, du point de vue du potentiel

initiateur du flot f_1, n'est sans doute pas acceptable et motive d'autant plus l'utilisation des files d'attente de type Fair-Drop dans ces réseaux.

Pour ce qui est de TCP, il est clair qu'avec RED et Drop-Tail, ses performances en termes d'équité sont nettement meilleures que celles de Tetrys. Cependant, le flot f_1 reste sensiblement défavorisé et ne dépasse pas les $0.47Mb/s$ et ce, quel que soit le type de file d'attente. Ce résultat demeure éloigné de l'assignement *max-min fair* attendu et peut être considéré comme insatisfaisant.

La combinaison Tetrys/Fair-Drop permet donc à la fois une efficacité satisfaisante et un assignement de trafic beaucoup plus proche que celui du *max-min fair* obtenu par TCP.

7.5.5 Topologie d'un FAI national

Bien que les topologies en cycle sur les réseaux locaux ou les réseaux d'entreprise soit assez peu répandues depuis le déclin du TokenRing, il semblerait, que les réseaux d'opérateurs d'échelle nationale contiennent des structures cycliques.

7.5.5.1 Objectif

L'objectif de cette partie est d'une part d'étudier le comportement de Tetrys sur une telle topologie (qui correspond aux conditions rencontrées par les flots une fois sortis du réseau local) et d'autre part d'étudier l'impact du type de file d'attente déployé aux différentes zones du réseau. Ce second point est primordiale puisque nous avons montré qu'à priori, l'efficacité et l'équité sont conditionnées par la présence de Fair-Drop. Or, ces dernières étant considérées comme coûteuses, leur déploiement sur des routeurs optiques serait peu envisageable.

7.5.5.2 Scénario

Afin de reproduire le réseau d'un FAI, nous proposons dans la figure 7.7 une topologie constituée d'hôtes, de DSLAM, de routeurs d'accès et de routeurs de coeurs. On peut y observer cinq cycles formés par les routeurs d'accès qui correspondent à une desserte régionale ainsi q'un cycle formé par le coeur du réseau. Chacun des cycles locaux est formé de 4 routeurs auxquels sont rattaché un DSLAM qui connecte 4 *clients adsl*. Les *cycles locaux*, que l'on nommera par la suite *pétales* par analogie à l'aspect florale de la figure 7.7, sont rattachés à un seul routeur de coeur qui servira à l'identifier. Dans le cas de topologies réelles, les paquets ne sont généralement pas perdus dans le coeur du réseau et les pertes se cantonnent aux routeurs d'accès. Cela est dû à la fois au

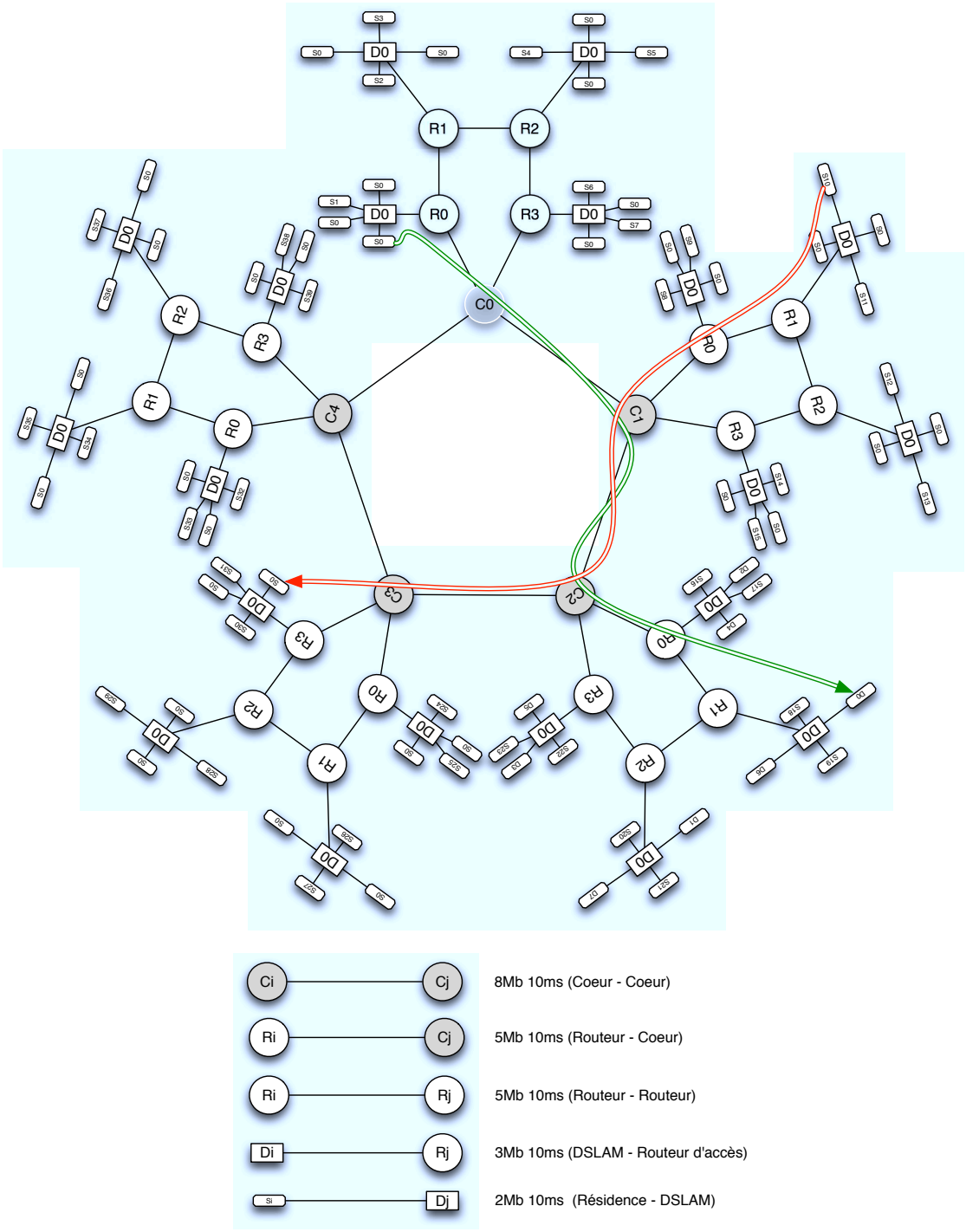


FIGURE 7.7: Topologie cyclique reproduisant un FAI national.

fait que le coeur du réseau est sur-dimensionné et au contrôle d'accès qui est effectué par les routeurs de bordures. Cependant, les fournisseurs d'accès Internet ne dévoilent généralement par leur topologie au grand public pour des raisons de sécurité évidentes. Nous ne disposons donc pas d'information sur le rapport entre la bande passante des liens qui connecte les pétales aux routeurs de coeur et les liens du routeur de coeur. Afin

de ne pas s'attacher à un cas trop optimiste, nous considérons que la congestion peut se produire sur les routeurs de coeur et nous fixons la bande passante des différents liens en conséquence. Ainsi, chacun des 40 flots subit successivement 25% de perte lors de l'accès au DSLAM, 16% des pertes sur le dernier lien de la pétale et finalement 40% dans le coeur.

Pour faciliter l'analyse des résultats, chaque flux doit avoir le même délai et être en concurrence avec le même nombre de flots. Pour ce faire, chaque paire source/destination doit traverser 3 routeurs de coeur (2 liens). De plus, si la source appartient à la partie inférieure, la destination appartient à la partie supérieure de la pétale de destination. Ainsi chaque flot traverse 9 liens et le RTT est donc de $180ms$. Sur chaque DSLAM, deux utilisateurs émettent du trafic à la vitesse de l'interface tandis que les deux autre servent de récepteurs pour du trafic émis par une autre pétale. Leurs débits passent ainsi progressivement de $2Mb/s$ à $0.5Mb/s$.

File	Efficacité			Equité		
d'attente	Tetrys min	Tetrys max	TCP	Tetrys min	Tetrys max	TCP
RED	89.2 / 70.0	46.1 / 11.6	86.8	0.50	0.92	0.98
Fair-Drop	91.0 / 71.2	90.0 / 23.5	94.0	0.99	0.99	0.99

TABLE 7.8: Efficacité / ratio entre le nombre de paquets envoyés par Tetrys et le nombre de paquets reçus au total ; et l'équité concernant la topologie 7.7.

On peut voir que sur le tableau 7.8 les efficacités de 91% et 90% obtenues par Tetrys sont du même ordre que celles obtenues sur les autres topologies. Bien que les résultats soient satisfaisants, il est clair que l'efficacité et/ou l'équité entre les flots Tetrys nécessite le déploiement de Fair-Drop.

Nous étudions dans la section suivante la possibilité de se passer du Fair-Drop dans le coeur du réseau voire de le remplacer par des politiques de gestion de files d'attente moins coûteuses en termes de ressources.

7.5.5.3 Fair-Drop dans le coeur de réseau

Nous constatons sur le tableau 7.9 qu'avec les capacités de lien choisies, la suppression de Fair-Drop sur les routeurs de coeur change sensiblement l'efficacité et la répartition du trafic entre les flots. Il semblerait même que ce soit sur ces routeurs que Fair-Drop soit le plus indispensable. Même si il semble plus probable que le coeur de réseau ne soit pas à ce point sous dimensionné par rapport au réseau d'accès, il est nécessaire de prendre en compte ce cas de figure et de trouver un palliatif à Fair-Drop dans les routeurs de coeur. Pour ce faire, nous proposons de remplacer le principe de pertes équitables par flot au profit de pertes équitables par interfaces d'entrées sur l'équipement (routeur/switch).

DSLAM	Bordure	Coeur	Efficacité			Equité		
			Tetrys min	Tetrys max	TCP	Tetrys min	Tetrys max	TCP
FD	FD	FD	91.0	90.0	94.0	0.99	0.99	0.99
FD	FD	RED	88.4	48.1	86.0	0.69	0.93	0.99
FD	RED	FD	90.8	86.4	94.2	0.99	0.99	0.99
RED	FD	FD	90.8	88.2	94.2	0.99	0.99	0.99
RED	RED	FD	91.1	82.8	94.2	0.99	0.99	0.99
FD	RED	RED	89.2	46.0	87.0	0.47	0.91	0.98
RED	RED	RED	89.2	22.9	86.8	0.50	0.92	0.98

TABLE 7.9: Efficacité et équité obtenues par Tetrys et TCP pour diverses combinaisons de files d'attente au sein du réseau.

Le nombre d'interfaces étant relativement restreint (maximum $\approx$ 128), la complexité est bornée (peu importe le nombre de flots réels) d'autant plus que la décision de jeter le paquet peut être incluse dans le processus de routage/commutation et ne nécessite aucun état supplémentaire si ce n'est la vitesse d'arrivée des paquets de chaque interface vers l'interface de sortie sur laquelle se trouve la file d'attente.

A notre connaissance, il n'est pas fait mention de file d'attente à pertes équitables se focalisant sur d'autre aspect que sur des optimisations d'implémentation de Fair-Drop [101] et/ou comme RED-PD, qui réduirait la complexité en ne conservant des états que pour les flots qui dépassent la bande passante équitable [108]. La file d'attente que nous proposons ne nécessiterait que 64 bits par interface (deux entiers codés sur 32 bits) pour estimer le débit de chaque interface vers la file d'attente et le taux de perte à appliquer. Ceci reste donc négligeable par rapport aux besoins du routage. De plus, les opérateurs pourraient à leur guise établir le compromis entre efficacité et équité plutôt que de subir la répartition du trafic de TCP qui tend à être *max-min fair* et donc à être largement sous optimale comme dans le cas de la topologie en parking.

L'objet ici n'étant pas de concevoir un nouveau type de file d'attente, nous avons simplement adapté les vitesses des liens d'accès aux routeurs afin de reproduire le fonctionnement d'une file d'attente à pertes équitables par interfaces. Ainsi, les liens entre les pétales et le coeur du réseau sont fixés à $2.1Mb/s$ (le débit est surestimé afin de limiter la sous utilisation du lien de sortie). Par exemple, les $4.2Mb/s$ issus des liens $R_0 - C_0$ et $R_3 - C_0$ partagent alors le lien $C_0 - C_1$ avec les flots émis par C_4 et le taux de perte (configurable) sur les routeurs de coeur n'est plus que de 2% ce qui limite les paquets morts dans le réseau. L'efficacité de *Tetrys min* est alors de 92.6% avec un index de Jain de 0.99 ce qui valide le fait qu'une telle file d'attente devrait suffire.

Il serait cependant nécessaire de poursuivre les investigations sur les propriétés de la file d'attente proposée, notamment dans le cas d'autres topologies.

7.5.6 Evaluation des applications à contrainte de temps

Nous avons vu que le contexte des réseaux anarchiques n'impliquait pas de changement sur la façon dont Tetrys protégeait les applications temps réel, excepté qu'il est nécessaire de prévoir de la redondance afin de palier aux pertes résultant de l'apparition de nouveaux flots dans le réseau.

File d'attente	Mécanisme	Efficacité	Délai : 95th / moyenne	PLR VoIP
FD(10)	Tetrys max	82.6	117/109ms	0%
	Tetrys min	88.2	159/115ms	
FD(10)	TCP/UDP	82.4	95/88ms	2.0%
DT (10)	TCP/UDP	82.0	91ms	2.2%
RED(10)	TCP/UDP	73.0	92/86ms	4.4%
FD(20)	Tetrys max	88.0	128/120ms	0%
	Tetrys min	90.0	180/131ms	
FDT(20)	TCP/UDP	86.8	108/94ms	1.7%
DT (20)	TCP/UDP	85.4	111/93ms	2.4%
RED(20)	TCP/UDP	84.2	98/89ms	3.5%
50 FDT	Tetrys max	84.6	168/155ms	0%
	Tetrys min	90.2	179/165ms	
FD(50)	TCP/UDP	91.9	131/106ms	0.3%
DT (50)	TCP/UDP	90.2	154/112ms	2.4%
RED(50)	TCP/UDP	84.2	105/82ms	0.2%
FD(190)	Tetrys max	82.2	356/322ms	0%
	Tetrys min	91.0	380/318ms	
FD(190)	TCP/UDP	94.8	243/182ms	0.2%
DT (190)	TCP/UDP	94.4	391/277ms	1.4%
RED(190)	TCP/UDP	82.8	105/91ms	0.08%

TABLE 7.10: Efficacité, délai et taux de perte perçu par la VoIP en fonction du type de file d'attente et de leur taille.

Le dernier point à adresser concerne la tolérance de Tetrys à la distribution des pertes induites par les réseaux anarchiques. Nous étudions donc dans cette partie le comportement de quatre flots de téléphonie sur Internet (VoIP) combinés à 36 flots de transfert de données sur la topologie représentant un FAI 7.7. Afin de générer des fluctuations de la bande passante disponible et du taux de perte, 4 flots démarrent respectivement aux instants $t = \{20, 40, 60 \text{ et } 80s\}$. Dans le cas où les sources cherchent à minimiser leurs débits d'envoi, les flots de VoIP émettent avec le plus petit taux de redondance possible leur permettant de satisfaire les besoins de l'application, en l'occurrence 95% des paquets doivent être reçus avant $400ms$. Dans le cas échéant, les flots VoIP émettent au débit de leur interface ($2Mb/s$) et ce, bien que le débit des données utiles ne soit que de $64Kb/s$. L'ensemble des paramètres demeurent ceux de la figure 7.7 chacun des flots

pouvant obtenir au minimum $500Kb/s$ (partage équitable sur le goulot d'étranglement avec Fair-Drop). Le RTT est fixé à $180ms$ pour l'ensemble des flots.

Nous comparons les performances obtenues par la version anarchique (Tetrys) et la solution actuelle à savoir TCP pour les flots FTP et UDP pour la voix sur IP. Afin de ne pas introduire de biais lié à la capacité de la file d'attente, nous répétons les mesures en fixant cette dernière à 10, 20 50 et 190 paquets correspondant à la taille idéale sur les routeur de coeurs en terme de produit bande passante délai. Les mesures sont également répétées pour TCP/UDP avec Drop-Tail et RED (en mode gentle [109]). Cependant avec Drop-Tail, nous soupçonnons un phénomène de synchronisation globale [107] entre les flux TCP qui serait à l'origine du taux de perte anormalement élevé observé sur les flots UDP [15]

Les métriques que nous utilisons sont d'une part l'efficacité du réseau en termes de paquets utiles reçus par les applications, qu'il s'agisse de la VoIP ou de FTP (les paquets de redondance reçus ne sont pas comptabilisés). Cela nous permet d'évaluer la perte d'efficacité induite par la combinaison Tetrys et VoIP. D'autre part, nous considèrons les conditions perçues par les flux de VoIP à savoir le délai de bout en bout moyen et 95^{eme} percentile ainsi que le taux de perte.

Le premier constat qui peut être fait d'après les résultats du tableau 7.10 est que dans tous les cas de figures testés, les contraintes de l'application ont été satisfaites aussi bien avec la version anarchique que dans les réseaux standards. Il est donc possible de supporter les applications à contrainte de temps telles que la VoIP sur les réseaux anarchiques.

Du point de vue du délai, il est évident que celui de *Tetrys max* est plus faible que celui de *Tetrys min* et que ces derniers augmentent avec la capacité des files d'attente. Par contre, il est intéressant de noter qu'avec TCP/UDP, le délai obtenu avec la taille de buffer préconisée pour TCP est largement au dessus de celui obtenu par Tetrys pour des files d'attente de taille 10 ou 20.

On constate également que grâce à l'algorithme de minimisation du débit, l'efficacité du réseau n'est pas impactée et reste sensiblement au même niveau que lorsqu'il n'y a que des flots FTP. Ceci est cependant à tempérer du fait que la proportion de flots de VoIP est négligeable par rapport aux flots de données. D'autre mesures sont donc à faire, en inversant cette proportion.

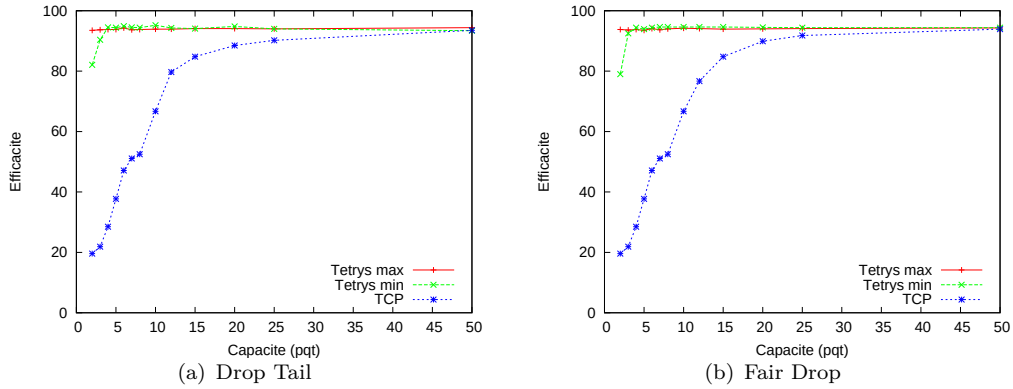
15. Suite au papier de Sally Floyd sur cet effet de synchronisation globale, des ajouts de paramètres non activés par défaut dans le simulateur ns-2 ont été proposés (i.e. `set overhead_`) afin de contrecarrer cet effet lors de l'utilisation de files d'attente DropTail. Cependant ils ne sont pas parfaits et peuvent introduire d'autres effets néfastes (comme une sous-utilisation du lien). Ainsi, la recommendation générale est d'utiliser de préférence des files d'attente RED.

7.6 Impact des tailles de files d'attente (sur topologies papillon, parking et cyclique)

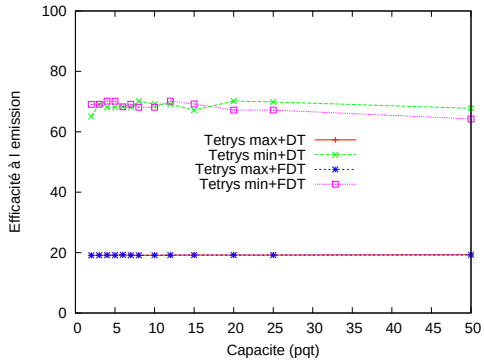
Les mécanismes de contrôle de congestion qui se basent sur les pertes pour la détection de la congestion sont particulièrement sensibles au dimensionnement des files d'attente dans le réseau. De la même manière, les protocoles basés sur le délai pour inférer la congestion tels que TCP Vegas ou TFRC requièrent des tailles de files d'attente minimum pour que le temps passé au sein de ces dernières augmente suffisamment et que la congestion soit détectée. La taille de file d'attente initialement préconisée était de l'ordre du produit bande passante délai des flots TCP [110] (ou TCP-friendly). Ceci peut donc correspondre à des millions de paquets. L'augmentation du produit bande passante délai et donc du coût de la mémoire sur les routeurs, a amené à reconsidérer ce principe de configuration. Ainsi, il a été montré dans [111] que l'utilisation du goulot d'étranglement peut être gardée proche de 100% même si les files d'attente ne font que $RTT * BP * \sqrt{M}$ et plus récemment à 80% avec des files d'attente d'une vingtaine de paquets [112]. Malgré leur inefficacité, de telles tailles de buffers se justifient par le coût des mémoires associées aux routeurs optiques et sont encore considérées comme trop élevées pour les routeurs tout optique [113]. Etant donné que les réseaux anarchiques font abstraction de tout contrôle de congestion, comparés à TCP, les flots anarchiques sont beaucoup moins sensibles à la taille des files d'attente.

Topologie en papillon La figure 7.8 montre l'évolution de l'efficacité pour des tailles de files d'attente variant de 2 à 50 paquets sur la topologie en papillon (symétrique) avec un délai de $100ms$ (c.f. section 7.5.3). Les valeurs relevées pour Tetrys avec des files d'attente de type Drop-Tail ne sont données qu'à titre indicatif. On observe qu'avec Fair-Drop, la taille de la file d'attente n'influe pas sur les performances de Tetrys qui se maintiennent au dessus de 94%. On observe également que l'efficacité de TCP ne dépasse pas les 50% lorsque la capacité de la file d'attente est inférieure à 10 puis augmente linéairement entre 20 et 50 pour finalement atteindre la même efficacité que Tetrys. Du point de vue de l'équité, on peut observer sensiblement le même phénomène sur la figure 7.9 mais on peut tout de même noter que TCP nécessite également Fair-Drop pour obtenir un index de Jain proche de 1.0.

Topologie en parking Pour la topologie en parking, la même tendance peut être observée sur la figure 7.10 et Tetrys atteint rapidement la valeur correspondant à une répartition équitable du trafic (i.e. 0.66) alors que TCP nécessite des tailles de files d'attente légèrement plus importantes. Dans ce cas, la croissance rapide de TCP s'explique par le fait que le produit bande passante délai des flots à deux sauts sont beaucoup plus

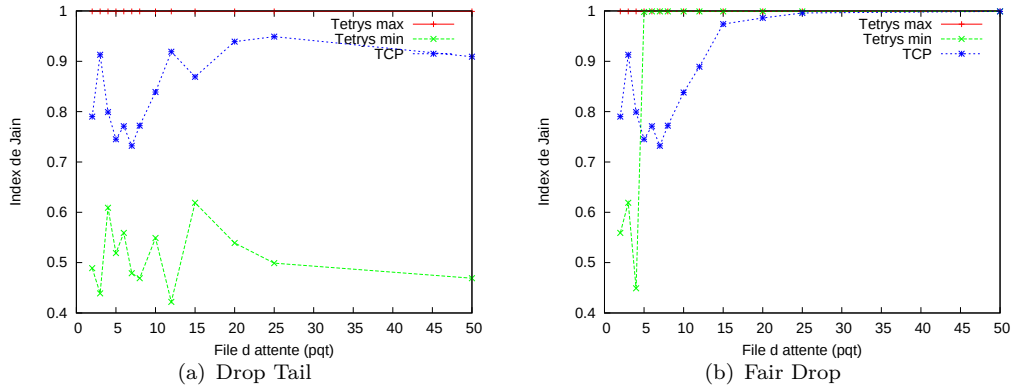


(a) Drop Tail (b) Fair Drop



(c) Coût à l'émission

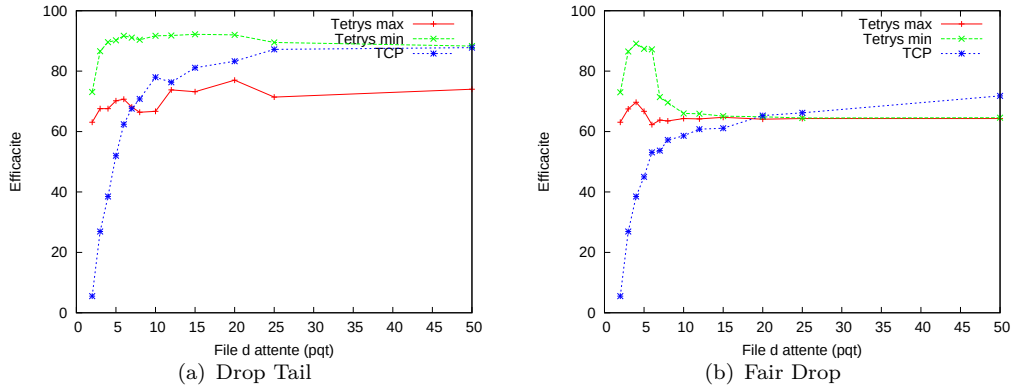
FIGURE 7.8: Efficacité de Tetrys et TCP sur la topologie en papillon (avec $\lambda = 1.0$) en fonction de la capacité des files d'attente.



(a) Drop Tail (b) Fair Drop

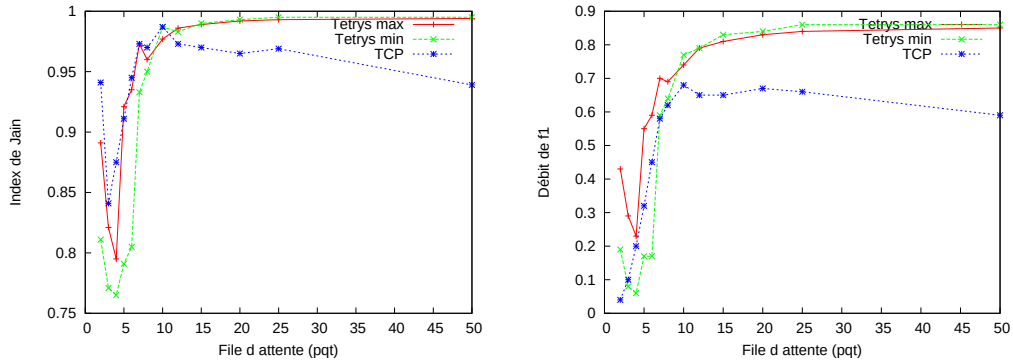
FIGURE 7.9: Index de Jain pour la topologie en papillon 7.1 ($\lambda = 1.0$) en fonction de la capacité des files d'attente.

faibles, ce qui réduit la taille de file d'attente requise. Les résultats concernant l'équité sont donnés par l'index de Jain sur la figure 7.10 accompagné de la figure 7.10 qui présente le débit obtenu par le flot f_1. On peut voir que pour des petite tailles de files d'attente, aussi bien Tetrys que TCP sont très éloignés d'une répartition équitable. Cela est dû à Fair-Drop dont les implémentations ne sont pas efficaces pour ces buffers de petites tailles. On peut malgré tout noter que lorsque la taille de file d'attente dépasse 8 paquets, le débit du flot f_1 atteint déjà $0.85Mb/s$ alors que TCP reste aux environs



(a) Drop Tail (b) Fair Drop

FIGURE 7.10: Efficacité de Tetrys et TCP sur la topologie en parking 7.2 en fonction de la capacité des files d'attente.



(a) Index de Jain (Fair Drop) (b) Débit de f_1 (Fair Drop)

FIGURE 7.11: Index de Jain et débit du flot f_1 (qui traverse l'ensemble du réseau) en fonction de la capacité des files d'attente sur la topologie en parking.

de $0.65Mb$ et ce malgré le fait que Fair-Drop soit utilisé.

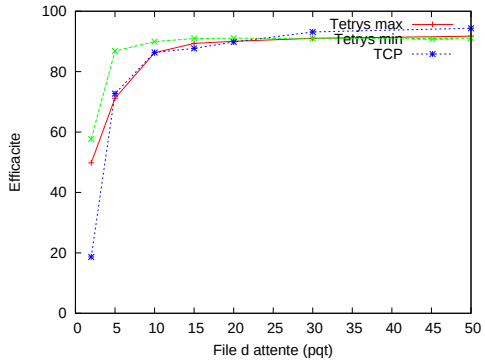


FIGURE 7.12: Efficacité de Tetrys et TCP sur la topologie d'un FAI 7.7. Notons qu'ici, le produit bande passante (500kb/s) délai (200ms) de TCP n'est que de 12 paquets et cette valeur idéale pour TCP est donc très vite atteinte.

Topologie cyclique Pour ce qui est de la topologie dite en « fleur », le produit bande passante délai de TCP est d'autant plus petit. Dans ce cas précis, cela permet d'être

performant malgré des petites tailles de files d'attente. On constate malgré tout que *Tetrys min* atteint les 90% d'efficacité à partir de taille de files d'attente de 5 paquets alors que TCP (comme *Tetrys max*) atteint cette valeur pour des files d'attente de 15 paquets pour finalement être plus efficace que Tetrys lorsque ces dernières dépassent les 30 paquets. Pour ce qui est de l'équité, tous les flots obtiennent le même débit, que ce soit avec TCP ou Tetrys et ce quelle que soit la taille de la file d'attente. Nous noterons néanmoins que cela n'aurait pas été le cas de TCP si les flots avaient des délais différents.

7.7 Conclusion

Pour que les réseaux anarchiques puissent concurrencer les performances du contrôle de congestion, il est primordial que l'efficacité soit proche de un, que le codage soit sans rendement et qu'il permette de rester en conformité avec le paradigme de l'implémentation `socket` actuelle. Nous avons proposé et évalué un mécanisme d'adaptation du taux de codage de Tetrys qui permet de maximiser la probabilité qu'un paquet reçu soit utile. Sur un lien, l'efficacité obtenue par ce mécanisme est du même ordre que celle de TCP.

Contrairement aux précédents travaux, nous avons montré que malgré l'utilisation de FairDrop, certaines topologies peuvent induire une proportion significative de paquets morts. Notre solution à ce problème reste en accord avec le principes des réseaux anarchiques. Nous supposons que les utilisateurs sont opportunistes et cherchent donc à réduire le coût du transfert, en réduisant en priorité sa durée mais également le coût induit par l'émission d'un paquet. Cette hypothèse nous a permis de proposer *Tetrys min*, un mécanisme de réduction du débit d'émission sans compromettre le débit utile obtenu par l'application.

Ceci permet d'améliorer significativement l'efficacité et de concurrencer TCP sur l'ensemble des topologies que nous avons testées. La répartition du trafic obtenue par Tetrys et Fair-Drop est par contre beaucoup plus proche de la *max-min-fairness* que ne l'est TCP, y compris pour la topologie en parking.

Nous avons de plus montré que Tetrys pouvait efficacement protéger des flots de VoIP dans un réseau occupé et chargé par des flots anarchiques de type transfert de fichier, sans réduire l'efficacité globale du réseau. Au regard de cette étude, le concept de réseaux anarchiques dans lequel Tetrys serait utilisé aussi bien pour le transfert de fichiers que pour la protection de flux temps réel semble être viable et présenter des performances similaires à celles des réseaux actuels.

Par ailleurs, nous avons succinctement évalué l'impact qu'a la capacité des files d'attente sur les performances respectives de Tetrys et TCP.

Il en résulte que pour les files d'attente de faible capacité ($\approx$ 8 paquets), notre proposition de réseau anarchique surpasse largement TCP, ce qui en fait un concept viable pour les futurs routeurs tout optique qui ne pourront stocker qu'une quantité très limitée d'information.

7.7.1 Critiques et travaux futurs

Nous avons montré à l'aide de la topologie similaire à celle d'un FAI que les réseaux anarchiques sont dépendants des files d'attente Fair-Drop, y compris dans les routeurs de coeur de réseau. Étant donné qu'un traitement et des états pour chacun des flots n'est pas possible sur les routeur de coeurs, une alternative doit être trouvée à Fair-Drop sur ces derniers [16]. Une solution serait d'avoir un mécanisme de minimisation du débit semblable à *Tetrys min* mais qui fonctionnerait également avec DropTail. Puisque l'existence même d'un tel mécanisme n'est pas triviale, nous proposons que les files d'attentes à pertes équitables par interfaces d'entrées plutôt que par flot. La complexité de cette dernière est négligeable et son implémentation sur des routeurs à haut débit serait donc possible. Le trafic obtenu avec cette file d'attente n'est pas nécessairement de type *max-min fair*, et ses propriétés sont à étudier pour des topologies de tailles réalistes. Un des avantages serait par exemple le fait que le taux de perte à appliquer peut être dérivé du contrôle d'accès. L'opérateur pourrait adapter librement la répartition du trafic afin de privilégier l'efficacité plutôt que l'équité.

Nous avons montré que les performances de Tetrys surpassent celles de TCP pour des files d'attente de faible capacité mais néanmoins supérieures à 8 paquets. En dessous de cette valeur, l'implémentation de Fair-Drop tend à perdre des paquets par dépassement de buffer plutôt que de manière équitable (*e.g.* se comporte comme un file d'attente de type Drop-Tail). Il faut donc repenser l'implémentation de Fair-Drop de manière à ce que les paquets des différents flots demeurent équitablement jetés malgré des files d'attente de capacité arbitrairement petite. Bien que dégradés, nous pensons que les résultats de *Tetrys min* seront meilleurs que ceux que nous avons pu relever.

De même, afin de palier aux effets de synchronisation des flots, nous avons introduit un temps d'attente aléatoire à l'émission de chaque paquet. En considérant une échelle de temps assez courte, le débit obtenu par un flot lorsqu'il partage un goulot d'étranglement tend à être variable. Le débit préalablement estimé est donc biaisé ce qui pénalise l'efficacité de Tetrys en particulier pour un faible RTT. De la même manière, les pertes appliquées par Fair-Drop sont uniformes. Les appliquer de façon régulière (à la façon d'un tourniquet) permettrait d'obtenir un débit beaucoup moins variable à court terme

16. Cependant, il a récemment été montré par Lawrence Roberts que même les routeurs de coeur pourraient gérer un grand nombre d'état à la volée [114]

et donc de pénaliser moindrement l'estimation du débit et du nombre de paquets manquants.

Bien que l'évaluation des applications temps réel nous permette de dire qu'il est possible de supporter ces dernières dans un réseau anarchique, elle reste sommaire et n'apporte pas d'information sur les conditions rencontrées par les flots et particulièrement sur la distribution des pertes. En effet, plus les pertes sont groupées, plus il faut que Tetrys génère de la redondance afin de les reconstruire dans le délai imparti par l'application et moins les réseaux anarchiques sont efficaces. Il semblerait malgré tout qu'avec Fair-Drop ou RED, les pertes soient uniformément distribuées.

Quatrième partie

Conclusion et travaux futurs

Chapitre 8

Résumé des contributions et perspectives

Ce chapitre résume les principales contributions de cette thèse avant de présenter les perspectives de travaux jugées les plus pertinentes.

8.1 Conclusion générale

Cette thèse présente un code à effacement appelé Tetrys. Nous avons mis en avant ses propriétés innovantes dans des contextes aussi variés que les applications temps réel ; les DTN et la gestion de la mobilité. Nous avons montré qu'il permet de générer des symboles de redondance à la volée, sans rendement et qu'il tolère des corps finis de petite taille (GF(4)). De plus, il peut être considéré comme un code fortement MDS (quasiment) et permet une fiabilité totale. Sa modélisation à permis de dériver son temps de décodage et de montrer que sa complexité en temps et en espace est abordable. La combinaison de ses diverses propriétés en font en code polyvalent avec une large palette d'application possible.

Nous nous somme concentrés sur son utilisation comme mécanisme de fiabilité de bout en bout. Dans ce contexte, sa configuration ne dépend que du taux de redondance ce qui la rend bien plus robuste que celle des codes bloc. Finalement, nous avons proposé une heuristique pour la configuration de Tetrys. Elle permet de minimiser le taux de redondance utilisé tout en satisfaisant les besoins des applications en termes de délai de reconstruction et de fiabilité.

Comparé à des codes en bloc, Tetrys permet d'améliorer la qualité d'une vidéo transmise sur un canal à effacement. Cependant cette application ne tire par profit de la fiabilité

totale de Tetrys et lorsqu'une trame est incomplète elle est (au mieux) affichée avec une erreur qui se répercutera sur les trames suivantes. Nous avons donc proposé d'adapter les codecs vidéo de façon à ce qu'ils réintègrent les trames décodées tardivement par Tetrys dans le processus de lecture. L'étude préliminaire à montré que la combinaison de Tetrys et de ces codecs permettront d'améliorer sensiblement la qualité de la vidéo.

Les applications à contraintes de délai ne sont pas le seul cas d'application de Tetrys. Dans les DTNs, nous avons montré que contrairement aux mécanismes existant, la robustesse de la configuration de Tetrys permet le support des applications de *streaming*. L'utilisation de Tetrys comme mécanisme de fiabilité de couche 3.5 permet de réduire l'impact des *handover* verticaux sur les protocoles de transports (comparé aux propositions utilisant les codes bloc).

De manière plus prospective, nous avons montré que Tetrys rassemble les propriétés requises pour la transmission dans les réseaux anarchiques. Nous avons implémenté une solution complète dont l'évaluation à permis de montrer que le concept de réseaux anarchiques peut remplacer avantageusement son équivalent à contrôle de congestion, notamment au regard de la taille des buffers requis dans le réseau. Cela reste vrai aussi bien pour l'efficacité, l'équité, l'équité et les différents types d'applications supportées (temps réel).

8.2 Résumé des contributions et travaux futurs

8.2.1 Codage à la volée

La fenêtre d'encodage de Tetrys n'est ni par bloc, ni glissante, mais élastique car elle inclut l'ensemble des paquets émis mais pas accusés par le destinataire. Cette particularité implique que Tetrys permet une fiabilité totale sans retransmission dès lors que le taux de redondance est supérieur au taux de perte. Lors de l'encodage, les coefficients sont choisis aléatoirement au sein du corps fini. Ainsi les symboles de redondance sont générés indépendamment les uns des autres et Tetrys peut être considéré comme un code sans rendement. Le décodage se fait par inversion de la matrice des coefficients de symboles source perdus. Nous avons montré qu'à partir de GF(4), la restriction des coefficients aux éléments non nuls d'un corps fini permet de rendre négligeable la probabilité que la matrice de décodage ne soit pas inversible. Il en découle que bien que Tetrys ne soit pas MDS au sens strict du terme, nous pouvons considérer dans le contexte d'une application aux réseaux de communications, que Tetrys a les mêmes propriétés de délai qu'un code convolutionnel fortement MDS alors qu'il ne nécessite qu'un corps fini de petite taille, qu'il est sans rendement et qu'il permet une fiabilité totale sans retransmission.

Nous avons proposé un modèle analytique de Tetrys à partir duquel a été dérivé le délai de décodage et la complexité du mécanisme en espace et en temps de calcul. Comme pour tout code convolutionnel fortement MDS, il en résulte que le délai de décodage n'est fonction que de la distribution des pertes et de l'écart entre le taux de redondance et le taux de pertes. La complexité de Tetrys à l'encodage est une fonction linéaire de la taille de la fenêtre d'encodage. Dans le cas des applications temps réel, on peut considérer qu'il s'agit du produit bande passante délai. Pour les applications de type *streaming*, le temps de récurrence peut être important et nous avons donc couplé Tetrys au mécanisme *sack-when-seen* qui permet également de ramener la complexité au produit bande passante délai. La complexité du décodage est linéaire en fonction du produit bande passante délai et cubique en fonction du nombre de pertes à corriger et du nombre de paquets. La composante cubique de la complexité est liée à la taille de la matrice à inverser et est donc négligeable pour les applications temps réels.

Les codes bloc étant les plus utilisés, nous avons étudié les performances de Tetrys comparé à ces derniers et notamment la robustesse de la configuration. Nous avons montré que le délai de décodage de Tetrys est du même ordre que la meilleure configuration possible d'un code bloc MDS de même taux de redondance. Ainsi, Tetrys est bien plus profitable dans des réseaux de communications soumis à des conditions fortement variables. Nous avons également proposé un mécanisme de configuration afin que Tetrys satisfasse les contraintes des applications en termes de délai et de fiabilité, tout en minimisant le taux de redondance utilisé.

Travaux futurs : Une amélioration potentielle du mécanisme de codage serait de le rendre MDS. Cependant, le gain attendu en termes de délai de décodage n'est pas significatif, d'autant plus que cette construction MDS se fera au prix d'une taille de corps fini largement supérieure. Une autre extension possible de Tetrys serait d'adapter (via une répartition adéquate du nombre de coefficients nuls) la complexité du code en fonction de la borne de délai requise par l'application. Dans ce cas, Tetrys pourrait être vu comme un code convolutionnel LDPC avec des symboles non binaires.

8.2.2 Applications de Tetrys

8.2.2.1 Application à la vidéo conférence

Nous avons comparé la qualité d'un flux video H.264/AVC dans deux cas de figures. D'une part lorsque qu'il est protégée par Tetrys et d'autre part avec un code bloc de longueurs diverses mais de taux de redondance égale à celui de Tetrys. Les résultats illustrent parfaitement la robustesse de configuration de Tetrys qui lui permet de s'adapter

au débit variable du flux vidéo induit par la différence de taille entre les trames I et P. Ainsi Tetrys permet un gain de plusieurs *dB* par rapport aux différents codes bloc.

Contrairement aux codes bloc qui peuvent être configurés de manière à ce que les paquets d'une trame soient décodés avant leurs instants de lecture, le délai de Tetrys croit avec le nombre de pertes et les paquets peuvent être reconstruits après l'instant de lecture. Bien que les trames d'un GOP aient une dépendance temporelles entre elles, aucun codec ne propose de recalculer les trames initialement affichées avec une erreur lorsque les mécanismes de transmission réseau les reconstruits tardivement. Nous avons donc fait une évaluation préliminaire de la combinaison entre Tetrys et un codec qui prend en compte les décodages tardifs. Les résultats montrent que le gain potentiel est significatif avec un surcoût de complexité tolérable pour le codec video.

Travaux futurs : *Décodage tardif* Les résultats préliminaires sur les codecs autorisant un décodage tardif sont encourageants mais ils doivent encore être validés par des expérimentations réelles. Il faudra pour cela modifier l'implémentation des principaux codecs existant et étudier leur interaction avec les mécanismes d'atténuation d'erreur des codecs.

Protection inégale : Dans un flux video, les trames ne sont pas toute de même importance. Ainsi, les paquets issus des trames I ont doivent être mieux protégés que ceux issus des trames P ou B et de nombreuses proposition de codes à effacement ont été faite dans ce sens. Plus généralement, il existe de nombreuses applications qui ont une tolérance inégale aux pertes de paquets au sein d'un flot. Tetrys peut être adapté pour fournir divers types de protection inégale :

– Plusieurs instances de codage Tetrys peuvent être appliquées à des sous ensembles des symboles source avec un taux de redondance adapté à l'importance des données. Ce schéma de codage permettrait de réduire la quantité de redondance totale émise et éventuellement la complexité puisque la fenêtre d'encodage de chaque instance de Tetrys serait, a priori, d'une taille moindre.
– Tetrys permettant de facto une fiabilité totale, la différence de niveau de protection est donc placée au niveau du délai requis pour le décodage. Dans ce contexte, il est envisageable d'utiliser une hiérarchie $C_1, C_2, ...C_m$ de codes Tetrys imbriqués ayant pour fenêtres d'encodage $W_1, W_2, ...W_m$ de symboles sources d'importance croissante. Les paquets de W_{i+1} seraient alors inclus dans W_i. Les paquets de W_m bénéficieraient d'un grand nombre de paquets de redondance puisqu'elle sont encodées dans l'ensemble des paquets de redondance. A l'inverse, des pertes qui affecteraient les paquets de faible importance n'auraient aucune influence sur le délai de décodage des paquets d'importance supérieure.

– Si l'on souhaite contrer plus particulièrement les pertes en rafales, les fenêtres d'encodage peuvent être de tailles finies. Ainsi W_i contiendrai K_i paquets avec $K_{i+1} < K_i$.

Les propriétés de l'ensemble de ces variantes restent à définir. Il en va de même pour leur interfaces avec les applications telles que la vidéo. Tetrys ouvrent donc la voie à l'ensemble de ces travaux qui nous identifions comme une perspective de recherche.

8.2.2.2 Fiabiliser les handovers verticaux

Le but est ici de réduire l'impact des *handover* verticaux sur les applications et les protocoles de transport. Nous avons considéré le cas d'un *soft vertical handover*. Notre solution utilise Tetrys comme un protocole de fiabilité qui s'intercale de façon transparente entre les couches 3 et 4 du routeur d'accès et du noeud mobile. Nous avons dû développer une méthode de gestion de la redondance adaptée aux RTO de TCP. Dans le cas où le réseau WLAN est fortement dégradé, nos résultats montrent que Tetrys permet à TCP de conserver son débit initial avec un surcoût négligeable en terme de redondance. De la même manière, lorsque le WLAN est fortement dégradé mais en même temps le WWAN le est disponible, Tetrys permet de tirer profit de la bande passante des deux interfaces.

Ces propriétés représentent une contribution majeure par rapport aux autres solutions utilisant des codes à effacement. Nous avons également montré qu'avec Tetrys, un changement instantané entre le WLAN et le WWAN ne bloque pas la connexion et ce sans aucune modification du code de TCP.

Travaux futurs : Notre évaluation à été faite via l'implémentation de Tetrys dans les *divert socket* d'OpenBsd (couche 3) et les liens WLAN et WWAN ont été simulés via *DummyNet*. La prochaine étape sera d'intégrer Tetrys dans une solution complète telle que *shim6* [115].

De manière plus générale, nous avons entamé des travaux sur la transmission multi-chemins à l'aide de Tetrys au niveau 3/4 e manière à gerer la fiabilité et le délai de décodage/lecture des données. A terme, cette proposition devrait intégrer un contrôle de congestion par chemin. Le but est de permettre au protocole de transport de niveau 4 de profiter de manière transparente de l'agrégat de la bande passante disponible sur les différentes interfaces.

Nous avons également entamé des travaux sur l'adaptation de Tetrys comme brique de fiabilité pour les applications multicast fiables à grande échelle.

8.2.3 Travaux prospectifs : Réseaux anarchiques

Nous avons identifié les propriétés de Tetrys comme étant celles des mécanismes de transmission optimaux, dont l'existence est prise comme hypothèse par les réseaux anarchiques [89, 90]. Dans ces réseaux, les utilisateurs sont égoïstes et ne cherchent qu'à réduire le coût de leur transmission de manière individuelle (ce coût s'exprime principalement en temps de transfert mais également en débit d'émission). Nous avons proposé et implémenté dans le simulateur ns-2 une pile protocolaire [1] de réseau anarchique. Elle se base sur une minimisation du coût [2] de la communication et sur l'optimisation des paramètres de codage de Tetrys pour optimiser l'utilisation qui est faite du réseau. Cette solution supporte aussi bien les applications de transfert de donnée (*e.g.* FTP) que les applications temps réel.

A l'aide de cette implémentation, nous avons montré les propriétés suivantes sur des topologies variées :
- l'efficacité des réseaux anarchiques est similaire à celle des réseaux classiques régis par le contrôle de congestion de TCP.
- lorsqu'ils sont couplés aux files d'attente Fair-Drop, l'assignement de trafic obtenu est beaucoup plus proche de la *max-min-fairness* que les réseaux à contrôle de congestion.
- L'efficacité des réseaux anarchiques est bien supérieure à celle des réseaux à contrôle de congestion lorsque les files d'attente sont de faible capacité.

8.2.3.1 Application au streaming dans les réseaux tolérant au délai

Dans les DTNs, le taux de perte, le délai et l'ordre d'arrivée des paquets sont particulièrement variables. L'utilisation de codes en bloc à été proposé mais leur configuration est délicate dans un environnement aussi versatile. L'utilisation des LT codes a également été proposée mais ces derniers requièrent un grand nombre de symboles source pour être efficace. De plus, aucun des mécanismes de transmission DTN n'est adapté aux flot de données qui produisent et consomment les données en continu. Nous avons donc montré que la robustesse de la configuration de Tetrys est également avantageuse dans ce contexte. L'évaluation indique que Tetrys reconstruit l'ensemble des bundles en un laps de temps largement inférieur à celui des codes bloc MDS dont nous avons pourtant idéalisé le fonctionnement. De plus, pour les paramètres considérés (un bundle de 200kb toute les 2 secondes), la complexité reste tolérable.

1. Y compris, l'implémentation des files d'attente à pertes équitables qui sont considérées comme indissociables des réseaux anarchiques
2. En termes de délai de transfert et du nombre de paquets émis

Travaux futurs : Dans les DTNs, le taux de réplication influe sur le niveau de remplissage des buffers et donc, influe bien évidement sur la congestion. Dans une certaine mesure, ajouter de la redondance peut donc être perçu comme contre productif. Plusieurs propositions de *network coding* dans les DTNs ont montré qu'il était possible de réduire la congestion et d'améliorer le taux de livraison des messages. Nous pensons que la combinaison du codage de bout en bout de Tetrys et d'une solution de *network coding* par flot permettrait de réduire la congestion tout en conservant les propriété de Tetrys en termes de robustesse et de délai de décodage.

Dans notre évaluation, nous avons choisi des valeurs arbitraires pour le taux de codage. En pratique il faut pouvoir estimer le taux de perte rencontré par les *bundles* sur l'ensemble du chemin.

8.3 Publications

1. Density-Aware Routing in Highly Dynamic DTNs : The RollerNet Case *(A paraitre)*. Tournoux Pierre-Ugo, Leguay Jeremie, Benbadis Farid, Conan Vania, Dias de Amorim Marcelo et Whitbeck, John. IEEE Transactions on Mobile Computing.

2. On-the-Fly Coding for Time-Constrained Applications *(En cours de revision majeure - second round)*. Pierre-Ugo Tournoux, Emmanuel Lochin, Jérôme Lacan, Amine Bouabdallah et Vincent Roca. IEEE Transactions on Multimedia.

3. A Packet Error Recovery Scheme for Vertical Handovers Mobility Management Protocols. Pierre-Ugo Tournoux, Emmanuel Lochin, Henrik Petander, Jérôme Lacan. ICST Mobiquitous 2010, Sydney, Australia, December 2010

4. Robust Streaming in Delay Tolerant Networks. Pierre Ugo Tournoux, Emmanuel Lochin, Jérémie Leguay et Jérôme Lacan. IEEE International Conference on Communications (IEEE ICC 2010) Cape Town, South Africa, May 2010

5. On-the-Fly Coding for Real-Time Applications. Pierre-Ugo Tournoux, Amine Bouabdallah, Jérôme Lacan et Emmanuel Lochin. ACM Multimedia 2009 Systems Track Beijing, China, October 2009

6. The Accordion Phenomenon : Analysis, Characterization, and Impact on DTN Routing. Pierre Ugo Tournoux, Jeremie Leguay, Farid Benbadis, Vania Conan, Marcelo Dias de Amorim et John Whitbeck. IEEE Infocom 2009. Rio, Brazil - April 2009

7. Tetrys : Un mécanisme de fiabilisation polyvalent. Pierre-Ugo Tournoux, Amine Bouabdallah, Emmanuel Lochin et Jérôme Lacan. Colloque Francophone sur l'Ingénierie des Protocoles (CFIP 2009) Strasbourg, 12-15 Octobre 2009. Mention spéciale jeune chercheur du jury CFIP 2009

Bibliographie

[1] J. Postel. User Datagram Protocol. RFC 768 (Standard), August 1980. URL `http://www.ietf.org/rfc/rfc768.txt`.

[2] J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. URL `http://www.ietf.org/rfc/rfc793.txt`. Updated by RFCs 1122, 3168.

[3] V. Jacobson. Congestion avoidance and control. In SIGCOMM '88 : Symposium proceedings on Communications architectures and protocols, pages 314–329, New York, NY, USA, 1988. ACM. ISBN 0-89791-279-9.

[4] Transport level mechanisms for bandwidth aggregation on mobile hosts. In ICNP '01 : Proceedings of the Ninth International Conference on Network Protocols, page 165, Washington, DC, USA, 2001. IEEE Computer Society.

[5] Bryan Ford and Janardhan Iyengar. Breaking up the transport logjam. In HOTNETS-VII, 2008.

[6] ITU. T-REC-G.107-200503-I, The E-Model, a computational model for use in transmission planning.

[7] Fouad A. Tobagi and Ismail Dalgic. Performance evaluation of 10base-T and 100base-T ethernets carrying multimedia traffic. IEEE Journal on Selected Areas in Communications, 1996.

[8] Jean chrysostome Bolot and Hugues Crépin. Analysis and control of audio packet loss over packet-switched networks. In IEEE Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV, pages 163–174. Springer, 1995.

[9] Amela Sadagic, Ben Teitelbaum, Jason Leigh, Magda El Zarki, and Haining Liu. A survey of network qos needs of advanced internet applications. Technical report, 2002.

[10] Jumpei Arata, Hiroki Takahashi, Phongsean Pitakwatchara, Shin'ichi Warisawa, Kozo Konishi, Kazuo Tanoue, Satoshi Ieiri, Shuji Shimizu, Naoki Nakashima, Koji

Okamura, Young Kim, Sung Kim, Joon Hahm, Makoto Hashizume, and Mamoru Mitsuishi. A remote surgery experiment between japan-korea using the minimally invasive surgical system. In R. Magjarevic, J. H. Nagel, R. Magjarevic, and J. H. Nagel, editors, World Congress on Medical Physics and Biomedical Engineering 2006, volume 14 of IFMBE Proceedings, pages 3065–3068. Springer Berlin Heidelberg, 2007.

[11] Andrew S. Tanenbaum. Computer networks : 2nd edition. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988. ISBN 0-13-162959-X.

[12] Stephen Farrell and Vinny Cahill. Evaluating LTP-T : A DTN-Friendly transport protocol. In IWSSC, Salzburg, Austria, September 2007.

[13] Michael S. Borella, Debbie Swider, Suleyman Uludag, and Gregory B. Brewster. Internet packet loss : Measurement and implications for end-to-end qos. In International Conference on Parallel Processing, pages 3–12, 1998.

[14] Vern Paxson. End-to-end internet packet dynamics. SIGCOMM Comput. Commun. Rev., 27(4) :139–152, 1997. ISSN 0146-4833. doi : http://doi.acm.org/10.1145/263109.263155.

[15] Yi Lu, Yuhui Zhong, and Bharat Bhargava. Packet loss in mobile ad hoc networks. Technical report, 2003.

[16] C. Villamizar, R. Chandra, and R. Govindan. BGP Route Flap Damping. RFC 2439 (Proposed Standard), November 1998. URL `http://www.ietf.org/rfc/rfc2439.txt`.

[17] Daniel Aguayo, John Bicket, Sanjit Biswas, Glenn Judd, and Robert Morris. Link-level measurements from an 802.11b mesh network. SIGCOMM Comput. Commun. Rev., 34(4) :121–132, 2004. ISSN 0146-4833. doi : http://doi.acm.org/10.1145/1030194.1015482.

[18] B.Ganguly, V.Subramanian, and S.Kalyanaraman. Performance of disruption-tolerant network mechanisms applied to airborne networks. IEEE MILCOM, 2007, 2007.

[19] K. Fall. A delay-tolerant network architecture for challenged internets. In Proc. ACM SIGCOMM, 2003.

[20] A. Chaintreau, P. Hui, J. Crowcroft, C. Diot, R. Gass, and J. Scott. Impact of human mobility on the design of opportunistic forwarding algorithms. In Proc. INFOCOM, 2006.

[21] J. Leguay, A. Lindgren, J. Scott, T. Friedman, and J. Crowcroft. Opportunistic content distribution in an urban setting. In Proc. CHANTS, 2006.

[22] UMassDieselNet. A bus-based disruption tolerant network. URL `http://prisms.cs.umass.edu/diesel`.

[23] P.-U. Tournoux, J. Leguay, F. Benbadis, V. Conan, M.D. de Amorim, and J. Whitbeck. The accordion phenomenon : Analysis, characterization, and impact on dtn routing. In INFOCOM 2009, IEEE, pages 1116–1124, April 2009.

[24] Jorg Kaiser Changling Liu. A survey of mobile ad hoc network routing protocols. Technical report, 2003.

[25] John Whitbeck and Vania Conan. Hymad : Hybrid dtn-manet routing for dense and highly dynamic wireless networks. In IEEE WoWMoM Workshop on Autonomic and Opportunistic Communications, 2009.

[26] A. Vahdat and D. Becker. Epidemic routing for partially connected ad hoc networks. Technical Report CS-200006, Duke University, April 2000.

[27] T. Spyropoulos, K. Psounis, and C. Raghavendra. Spray and wait : An efficient routing scheme for intermittently connected mobile networks. In Proc. WDTN, 2005.

[28] A. Lindgren, A. Doria, and O. Schelen. Probabilistic routing in intermittently connected networks. In Proc. SAPIR, 2004.

[29] C. E. Shannon. A mathematical theory of communication. SIGMOBILE Mob. Comput. Commun. Rev., 5(1) :3–55, 2001. ISSN 1559-1662. doi : http://doi.acm.org/10.1145/584091.584093.

[30] P. Elias. Coding for two noisy channels. In Information Theory, The 3rd London Symposium, pages 61–76. Buttersworth's Scientific Publications, September 1955.

[31] James Massey. Shift-register synthesis and bch decoding. IEEE Trans. Information. Theory, pages 122–127, 1969.

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

[33] J. Lacan, V. Roca, J. Peltotalo, and S. Peltotalo. Reed-Solomon Forward Error Correction (FEC) Schemes. RFC 5510 (Proposed Standard), April 2009. URL `http://www.ietf.org/rfc/rfc5510.txt`.

[34] Frédéric Didier. Efficient erasure decoding of reed-solomon codes. CoRR, abs/0901.1886, 2009.

[35] A. Soro and J. Lacan. FNT-based Reed-Solomon Erasure Codes. ArXiv e-prints, July 2009.

[36] Michael Luby. Lt codes. In FOCS '02 : Proceedings of the 43rd Symposium on Foundations of Computer Science, page 271, Washington, DC, USA, 2002. IEEE Computer Society. ISBN 0-7695-1822-2.

[37] Peter Elias. Coding for noisy channels. In IEEE Transactions on Information Theory, pages 37–46, 1955.

[38] Claude Berrou, Karine Amis Cavalec, and Alain Glavieux. Codes et turbocodes. Springer, 2007.

[39] Alberto Jiménez Feltström and Kamil Sh. Zigangirov. Time-varying periodic convolutional codes with low-density parity-check matrix. IEEE Transactions on Information Theory, 45(6) :2181–2191, 1999.

[40] Michael Lentmaier, Arvind Sridharan, Kamil Sh. Zigangirov, and Daniel J. Costello Jr. Terminated ldpc convolutional codes with thresholds close to capacity. CoRR, abs/cs/0508030, 2005.

[41] Emin Martinian and Carl-Erik W. Sundberg. Burst erasure correction codes with low decoding delay. IEEE Transactions on Information Theory, October 2004.

[42] H. Gluesing-Luerssen, J. Rosenthal, and R. Smarandache. Strongly MDS Convolutional Codes. ArXiv Mathematics e-prints, March 2003.

[43] Robert M. Gray. Toeplitz and circulant matrices : A review. Technical report, 2001.

[44] R. Comroe and D. Costello. Arq schemes for data transmission in mobile radio systems. Selected Areas in Communications, IEEE Journal on, 2(4) :472–481, 1984. URL `http://ieeexplore.ieee.org/xpls/abs\_all.jsp?arnumber=1146084`.

[45] S. De Vuyst, S. Wittevrongel, and H. Bruneel. Performance analysis of the stop-and-wait arq protocol over a channel with bursty errors. In HET-NETs.

[46] D.D. Clark. Window and Acknowledgement Strategy in TCP. RFC 813, July 1982. URL `http://www.ietf.org/rfc/rfc813.txt`.

[47] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgment options. Request For Comments 2018, IETF, October 1996.

[48] V. Paxson and M. Allman. Computing tcp's retransmission timer. Request For Comments 2988, IETF, November 2000.

[49] J. M. Wozencraft and M. Horstein. Coding for two-way channels. Technical report.

[50] C. Lott, O. Milenkovic, and E. Soljanin. Hybrid arq : Theory, state of the art and future directions. pages 1 –5, jul. 2007. doi : 10.1109/ITWITWN.2007.4318035.

[51] Emina Soljanin, Nedeljko Varnica, , and Philip Whiting. Incremental redundancy hybrid arq with ldpc and raptor codes. IEEE Trans. Inform. Theory, Sept. 2005.

[52] D. J. Costello, J. Hagenauer, H. Imai, and S. B. Wicker. Applications of error-control coding. Information Theory, IEEE Transactions on, 44(6) :2531–2560, 1998.

[53] Vijaynarayanan Subramanian and Shivkumar Kalyanaraman. Hybrid packet fec and retransmission-based erasure recovery mechanisms (harq) for lossy networks : Analysis and design. In Wireless Systems : Advanced Research and Development (WISARD), 2007.

[54] Hybrid arq for robust video streaming over wireless lans. In ITCC '01 : Proceedings of the International Conference on Information Technology : Coding and Computing, page 317, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7695-1062-0.

[55] J. K. Sundararajan, D. Shah, M. Medard, M. Mitzenmacher, and J. Barros. Arq for network coding. INFOCOM 09, July 2009.

[56] J.Lacan and E. Lochin. Rethinking reliability for long-delay networks. In IEEE International Workshop on Satellite and Space Communications, 2008. IWSSC 2008., pages 90–94, Oct. 2008.

[57] Jeff Kahn and János Komlós. Singularity probabilities for random matrices over finite fields. Combinatorics, Probability and Computing, 10 :137 – 157, October 2001.

[58] David F. Carta. Two fast implementations of the "minimal standard" random number generator. Communications of the ACM, 33(1) :87–88, 1990.

[59] Ryan Hutchinson, Roxana Smarandache, and Jochen Trumpf. On superregular matrices and mdp convolutional codes. Linear Algebra and its Applications, 428 (11-12) :2585 – 2596, 2008.

[60] Jay Kumar Sundararajan, Devavrat Shah, and Muriel Médard. ARQ for network coding. IEEE International Symposium on Information Theory (ISIT), pages 1651–1655, July 2008.

[61] David J. Aldous and James A. Fill. Reversible Markov Chains and Random Walks on Graphs. Book in preparation, `http://www.stat.berkeley.edu/~aldous/book.html`, 200X.

[62] Johannes Bloemer, Malik Kalfane, Richard Karp, Marek Karpinski, Michael Luby, and David Zuckerman. An XOR-based erasure-resilient coding scheme, 1995. Technical report TR-95-048, International Computer Science Institute, Berkeley, California.

[63] S. Lin and D. Costello. Error Control Coding : Fundamentals and Applications. Prentice-Hall, Englewood Cliffs, NJ, 1983.

[64] H.264/AVC JM Reference Software, http ://iphome.hhi.de/suehring/tml/.

[65] Stephan Wenger. H.264/AVC over IP. IEEE Trans. Circuits Syst. Video Techn, 13(7) :645–656, 2003.

[66] J. Klaue, B. Rathke, and A. Wolisz. Evalvid - A framework for video transmission and quality evaluation. In Performance Tools, pages 255–272, 2003.

[67] Yi J. Liang, Eric Setton, and Bernd Girod. Network-adaptive video communication using packet path diversity and rate-distortion optimized reference picture selection. J. VLSI Signal Process. Syst., 41(3) :345–354, 2005. ISSN 0922-5773. doi : http://dx.doi.org/10.1007/s11265-005-4157-x.

[68] Yi J. Liang, Eckehard G. Steinbach, and Bernd Girod. Real-time voice communication over the internet using packet path diversity. In MULTIMEDIA '01 : Proceedings of the ninth ACM international conference on Multimedia, pages 431–440, New York, NY, USA, 2001. ACM.

[69] S. Wee, Wai tian Tan, J. Apostolopoulos, and M. Etoh. Optimized video streaming for networks with varying delay. volume 2, pages 89 – 92 vol.2, 2002.

[70] Sushant Jain, Michael Demmer, Rabin Patra, and Kevin Fall. Using redundancy to cope with failures in a delay tolerant network. SIGCOMM Comput. Commun. Rev., 35(4) :109–120, 2005. ISSN 0146-4833. doi : http://doi.acm.org/10.1145/1090191.1080106.

[71] E. Altman and F. De Pellegrini. Forward correction and fountain codes in delay tolerant networks. In INFOCOM 2009, IEEE, pages 1899–1907, April 2009. doi : 10.1109/INFCOM.2009.5062111.

[72] Xiaolan Zhang, G. Neglia, J. Kurose, and D. Towsley. On the benefits of random linear coding for unicast applications in disruption tolerant networks. In

Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks, 2006 4th International Symposium on, April 2006.

[73] Khaled A. Harras and Kevin C. Almeroth. Transport layer issues in delay tolerant mobile networks. In IN IFIP NETWORKING, 2006.

[74] Lloyd Wood, Wesley M. Eddy, Will Ivancic, Jim Mckim, and Chris Jackson. Saratoga : a delay-tolerant networking convergence layer with efficient link utilization. In Third International Workshop on Satellite and Space Communications (IWSSC '07, 2007.

[75] A. Morton, L. Ciavattone, G. Ramachandran, S. Shalunov, and J. Perser. Packet Reordering Metrics. RFC 4737 (Proposed Standard), 2006.

[76] Sharad Jaiswal, Gianluca Iannaccone, Christophe Diot, Jim Kurose, and Don Towsley. Measurement and classification of out-of-sequence packets in a tier-1 ip backbone. IEEE/ACM Trans. Netw., 15(1) :54–66, 2007. ISSN 1063-6692.

[77] Yi Wang, Guohan Lu, and Xing Li. A study of internet packet reordering. ICOIN 2004, 2004.

[78] Laurent Dairaine, Laurent Lancérica, Jérôme Lacan, and Jérôme Fimes. Content-access qos in peer-to-peer networks using a fast mds erasure code. Computer Communications, vol. 28(15) :1778–1790, September 2005.

[79] C. Perkins. IP mobility support for IPv4. RFC 3344, 2002.

[80] R. Chakravorty, P. Vidales, K. Subramanian, I. Pratt, and J. Crowcroft. Performance issues with vertical handovers - experiences from GPRS cellular and WLAN hot-spots integration. In IEEE PERCOM, 2004.

[81] H. Petander, E. Perera, and A. Seneviratne. Multicasting with selective delivery : a safetynet for vertical handoffs. Wirel. Pers. Commun., 43(3) :945–958, 2007.

[82] S.-E. Kim and J. A. Copeland. TCP for seamless vertical handoff in hybrid mobile data networks. In IEEE Globecom, 2003.

[83] L. Daniel and M. Kojo. The performance of multiple TCP flows with vertical handoff. In ACM MobiWAC, 2009.

[84] R. Stewart et al. Stream Control Transmission Protocol. RFC 4960, 2007.

[85] M. Fiore and C. Casetti. An adaptive transport protocol for balanced multihoming of real-time traffic. In IEEE GLOBECOM, 2005.

[86] J. Wang, J., and X. Zhu. Latent handover : A flow-oriented progressive handover mechanism. Computer Communications, 31(10) :2319–2340, 2008.

[87] T. Anker, R. Cohen, and D. Dolev. Transport layer end-to-end error correcting, 2004. Technical Report - School of Engineering and Computer Science, Hebrew University, Israel.

[88] Stephen Hemminger. Network emulation with NetEm. In Martin Pool, editor, LCA 2005, Australia's 6th national Linux conference (linux.conf.au), Sydney NSW, Australia, April 2005. Linux Australia, Linux Australia.

[89] Thomas Bonald, Mathieu Feuillet, and Alexandre Proutiére. Is the "law of the jungle" sustainable for the internet ? In INFOCOM, pages 28–36, 2009.

[90] Barath Raghavan and Alex C. Snoeren. Decongestion control. In Proceedings of the 5th ACM Workshop on Hot Topics in Networks (HotNets-V), Irvine, CA, 2006.

[91] Luis López, Antonio Fernández, and Vicent Cholvi. A game theoretic comparison of tcp and digital fountain based protocols. Comput. Netw., 51(12) :3413–3426, 2007. ISSN 1389-1286.

[92] J. Nagle. Congestion Control in IP/TCP Internetworks. RFC 896, January 1984. URL `http://www.ietf.org/rfc/rfc896.txt`.

[93] Tom Kelly, Sally Floyd, and Scott Shenker. Patterns of congestion collapse, 2003.

[94] Jean-Yves Boudec. Rate adaptation congestion control and fairness : A tutorial, 2008.

[95] Sally Floyd, Senior Member, and Kevin Fall. Promoting the use of end-to-end congestion control in the internet. IEEE/ACM Transactions on Networking, 7 : 458–472, 1999.

[96] Thomas Bonald and Laurent Massoulié. Impact of fairness on internet performance. In SIGMETRICS '01 : Proceedings of the 2001 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, pages 82–91, New York, NY, USA, 2001. ACM. ISBN 1-58113-334-0.

[97] Dah-Ming Chiu and Raj Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. Comput. Netw. ISDN Syst., 17 (1) :1–14, 1989. ISSN 0169-7552.

[98] L. Massoulié and J. Roberts. Bandwidth sharing : Objectives and algorithms. In IEEE/ACM Transactions on Networking, pages 1395–1403, 1999.

[99] Stefan Savage, Neal Cardwell, David Wetherall, and Tom Anderson. Tcp congestion control with a misbehaving receiver. SIGCOMM Comput. Commun. Rev., 29(5) :71–78, 1999. ISSN 0146-4833. doi : http://doi.acm.org/10.1145/505696.505704.

[100] Rob Sherwood, Bobby Bhattacharjee, and Ryan Braud. Misbehaving tcp receivers can cause internet-wide congestion collapse. In CCS '05 : Proceedings of the 12th ACM conference on Computer and communications security, pages 383–392, New York, NY, USA, 2005. ACM. ISBN 1-59593-226-7.

[101] Rong Pan, Balaji Prabhakar, Lee Breslau, and Scott Shenker. Approximate fair allocation of link bandwidth. IEEE Micro, 23 :36–43, 2003. ISSN 0272-1732.

[102] Arun Vishwanath, Vijay Sivaraman, and George N. Rouskas. Considerations for sizing buffers in optical packet switched networks. In INFOCOM, 2009.

[103] John W. Byers, Michael Luby, Michael Mitzenmacher, and Ashutosh Rege. A digital fountain approach to reliable distribution of bulk data. SIGCOMM Comput. Commun. Rev., 28(4) :56–67, 1998. ISSN 0146-4833. doi : http://doi.acm.org/10.1145/285243.285258.

[104] Raj Jain, Dah-Ming Chiu, and W. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. CoRR, cs.NI/9809099, 1998.

[105] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for High Performance. RFC 1323 (Proposed Standard), May 1992. URL `http://www.ietf.org/rfc/rfc1323.txt`.

[106] Lachlan L. H. Andrew, Cesar Marcondes, Sally Floyd, Lawrence Dunn, Romaric Guillier, Wang Gang, Lars Eggert, Sangtae Ha, and Injong Rhee. Towards a common tcp evaluation suite. 2008. URL `http://hdl.handle.net/1959.3/44431`.

[107] Tom Kelly, Sally Floyd, and Scott Shenker. Patterns of congestion collapse. Technical report, 2003.

[108] Ravesh Mahajan and Sally Floyd. Red with preferential dropping (red-pd). Technical report, 2001.

[109] Vincent Rosolen, Olivier Bonaventure, and Guy Leduc. A red discard strategy for atm networks and its performance evaluation with tcp/ip traffic. SIGCOMM Comput. Commun. Rev., 29(3) :23–43, 1999. ISSN 0146-4833. doi : http://doi.acm.org/10.1145/505724.505728.

[110] Curtis Villamizar and Cheng Song. High performance tcp in ansnet. SIGCOMM Comput. Commun. Rev., 24(5) :45–60, 1994. ISSN 0146-4833. doi : http://doi.acm.org/10.1145/205511.205520.

[111] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. Sizing router buffers. SIGCOMM Comput. Commun. Rev., 34(4) :281–292, 2004. ISSN 0146-4833. doi : http://doi.acm.org/10.1145/1030194.1015499.

[112] Mihaela Enachescu, Yashar Ganjali, Ashish Goel, Nick Mckeown, and Tim Roughgarden. Routers with very small buffers. SIGCOMM Comput. Commun. Rev., 35(3) :83–90, July 2005. ISSN 0146-4833. doi : 10.1145/1070873.1070886. URL `http://dx.doi.org/10.1145/1070873.1070886`.

[113] Arun Vishwanath, Vijay Sivaraman, Marina Thottan, and Constantine Dovrolis. Enabling a bufferless core network using edge-to-edge packet-level fec. In INFOCOM'10 : Proceedings of the 29th conference on Information communications, pages 1433–1441, Piscataway, NJ, USA, 2010. IEEE Press. ISBN 978-1-4244-5836-3.

[114] Lawrence G. Roberts. A radical new router. IEEE Spectr., 46(7) :34–39, 2009. ISSN 0018-9235. doi : http://dx.doi.org/10.1109/MSPEC.2009.5109450.

[115] E. Nordmark and M. Bagnulo. Shim6 : Level 3 Multihoming Shim Protocol for IPv6. RFC 5533 (Proposed Standard), June 2009. URL `http://www.ietf.org/rfc/rfc5533.txt`.